IDENTIFYING POLYMORPHIC MALWARE VARIANTS USING BIOSEQUENCE ANALYSIS TECHNIQUES

By

Vijay Naidu

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
AUCKLAND UNIVERSITY OF TECHNOLOGY
AUCKLAND, NEW ZEALAND
June 2018

Abstract

Modern antivirus systems (AVSs) are not able to detect new polymorphic malware variants until they emerge, even when signatures of one or more variants belonging to a specific polymorphic malware family are known. Polymorphic malware can transform into functionally identical variants of themselves. Polymorphism changes the order of the viral code but not typically the code itself to avoid signature-based detection. Current AVSs detect malware by adopting signatures based on the most essential parts of a known virus, such as execution traces, instruction sequences, etc. Virus writers exploit the weaknesses of malware signature databases by creating new variants using the same engine employed by an already existing polymorphic malware family. In this thesis, virus detection and signature extraction techniques are presented. These techniques were developed by exploring string matching techniques traditionally employed in biosequence analysis. The main contribution of these matching techniques is to extract syntactic patterns (i.e. conserved regions/sequences) from semantically rich polymorphic hex code. These extracted syntactic patterns act as signatures and are used in the identification of polymorphic malware variants belonging to the same family. Moreover, these extracted syntactic patterns can help in identifying new variants that make simple alterations to their newly generated variants. The string matching approaches presented in this thesis may revolutionise our knowledge of polymorphic variant generation and give rise to a new era of string-based syntactic AVSs.

Table of Contents

List of Figures

List of Tables

Attestation of Authorship

"I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (expect where explicitly defined in the acknowledgements), nor material which to a substantial extension has been submitted for the award of any other degree or diploma of a university or other institute of higher learning."

Vijay Naidu

Auckland, New Zealand

June 2018

Acknowledgements

Firstly, I would like to thank my supervisors, Prof. Ajit Narayanan, Assoc. Prof. Jacqueline Whalley and Assoc. Prof. Russel Pears without their encouragement, gratitude, inspiration, and support; I would not have been able to achieve my doctoral studies.

I would like to thank my parents Lohanathan Parumal (my dad) and Susheela L. Parumal (my mum) for all their help, support and affection and without them I would not have dreamed on studying higher.

I would like to thank Saide Lo, the assistant school manager of School of Engineering, Computer and Mathematical Sciences (SECMS) and Karishma Bhat, the programme administrator of SECMS for their immense support and kind-heartedness. Also, many thanks to Terry Brydon, the school manager of SECMS and Ramon Lewis, the technician of SECMS.

Last but not least, I would like to thank my wife Shari-May Naidu and her wonderful family, my PhD colleagues and best friends: Ahmad Wedyan, Abhimanyu Garhwal Singh, Sreenivas Sremath Tirumala, Nishantha Medagoda, Ahmed Al-Sa'di and Seyedjamal Zolhavarieh for all their kind help, suggestions, support, love, and understanding.

Vijay Naidu

Auckland, New Zealand

June 2018

Chapter 1 Introduction

This chapter introduces the motivation of this research, followed by a brief introduction to the background of this research and related work including syntactic and semantic virus detection approaches. This chapter concludes by providing the thesis contribution, structure, and related publications.

1.1 Motivation

Malware development and spread have reached epidemic proportions, with millions of new variants released every year infecting computer systems. Traditional antivirus systems (AVSs) have difficulty in coping with this growth. The traditional method for dealing with viruses and worms (two of the most common types of malware) is to use AVSs to look for 'signatures' which represent critical parts of executable code in the numerical or alphanumerical form. Whereas a virus requires some action to be propagated, such as being attached to a program by the user, a worm can propagate by itself. The aim of virus and worm writers is usually to damage computer systems, hence the term 'malware'. Signatures can be calculated from a pattern of operations in the malware code or can represent the encryption algorithm used to hide the virus or worm. Signatures were originally and continue to be identified and calculated by human experts, and are typically a sequence of hexadecimal (hex) numbers intended to identify viruses and worms uniquely. There are currently no known methods for automatic generation of syntactic signatures for new malware.

The automatic extraction of malware signatures for use in AVSs remains a relatively unexplored area of cyber security, despite the urgent need to find effective solutions to the increasing number and severity of attacks (Symantec, 2014; Naidu, Whalley, & Narayanan, 2017) that now pose a global risk (WEF, 2012).

Evidence for this related shortage of research in polymorphic malware (together with its variants) detection as well as in automatic signature generation using bioinformatics techniques (mainly sequence alignment) comes from a simple sequence of searches using Google Advanced Patent Search and Google Scholar (as of April 5th, 2018) and is shown in Table 1.1. Table 1.1 shows that there is a lack of research on automatic signature generation to polymorphic malware/virus and its variants detection/identification using sequence alignment techniques.

Table 1.1: Simple sequence of searches indicating the related shortage of research using Google Advanced Patent Search and Google Scholar (as of April 5th, 2018).

Search Terms	Google Advanced Patent Search	Google Scholar
"malware"	55,000 hits	157,000 hits
"malware detection"	19,000 hits	27,100 hits
"malware identification"	478 hits	922 hits
"polymorphic malware"	1,210 hits	1,890 hits
"polymorphic virus"	951 hits	1,360 hits
"automatic signature generation"	1,580 hits	1,620 hits
"automatic generation of signatures"	3 hits	66 hits
"automatic signature extraction"	269 hits	115 hits
"automatic extraction of signatures"	328 hits	97 hits
"polymorphic malware detection"	1,050 hits	257 hits
"polymorphic malware identification"	0 hit	3 hits
"polymorphic virus detection"	874 hits	492 hits
"polymorphic virus identification"	0 hit	0 hit
"sequence alignment"	76,500 hits	917,000 hits
"polymorphic malware detection" + "automatic signature generation"	6 hits	28 hits
"polymorphic malware identification" + "automatic signature generation"	0 hit	0 hit
"polymorphic virus detection" + "automatic signature generation"	108 hits	55 hits
"polymorphic virus identification" + "automatic signature generation"	0 hit	0 hit
"polymorphic malware detection" + "automatic signature generation" + "variants"	2 hits	10 hits
"polymorphic malware identification" + "automatic signature generation" + "variants"	0 hit	0 hit
"polymorphic virus detection" + "automatic signature generation" + "variants"	51 hits	28 hits
"polymorphic virus identification" + "automatic signature generation" + "variants"	0 hit	0 hit
"polymorphic malware detection" + "automatic signature generation" + "variants" + "sequence alignment"	0 hit	5 hits
"polymorphic malware identification" + "automatic signature generation" + "variants" + "sequence alignment"	0 hit	0 hit
"polymorphic virus detection" + "automatic signature generation" + "variants" + "sequence alignment"	0 hit	1 hit
"polymorphic virus identification" + "automatic signature generation" + "variants" + "sequence alignment"	0 hit	0 hit

Early automatic extraction attempts focused on simulating the way that human experts analyse malware to generate signatures for use in AVSs (Kephart & Arnold, 1994; Huda, et al., 2017). Typically, an anomalous behaviour of a computer system leads to the identification of suspicious code that is then analysed to identify invariant code portions (syntax) or code portions that are regularly executed (semantics). Semantics are required for the process of code execution behaviour because they can capture the self-modifying feature of malware code (Debray, Coogan, & Townsend, 2008; Korczynski & Yin, 2017).

Such analysis leads to the generation of malware 'signatures' for use by AVSs when scanning network packets, user files or memory. Before such signatures can be released, they must be checked against non-malware to ensure that the number of false positives is kept acceptably low. For instance, signatures based on a malware encryption/decryption engine are likely to lead to unacceptably high false positives due to the large proportion of normal Internet traffic that also carries encryption/decryption information for integrity (e.g. hash algorithms) and authentication (e.g. certified public keys).

Relying on human expertise alone to provide manually extracted signatures is no longer feasible with the growing volume of malware. As a result, automatic signature extraction approaches have become increasingly sophisticated. Semantic approaches (Christodorescu, Jha, Seshia, Song, & Bryant, 2005; United States Patent No. US11523199, 2006; Sathyanarayan, Kohli, & Bruhadeshwar, 2008; Feng, Bastani, Martins, Dillig, & Anand, 2017), in addition to standard dynamic and execution behaviour analysis (Ellis, Aiken, Attwood, & Tenaglia, 2004; Gao, Reiter, & Song, 2005; United States Patent No. US9609015B2, 2017) now include control flow analysis (Cesare & Xiang, 2010; United States Patent No. US9817974B1, 2017), behaviour model checking (Kinder, Katzenbeisser, Schallhart, & Veith, 2005; Bailey, et al., 2007; United States Patent No. US9832211B2, 2017) and executable graph mining (Eskandari & Hashemi, 2012; Chen, Jeng, Huang, Chen, & Chou, 2017) as well as formal semantic models of analysis (Chaumette, Ly, & Tabary, 2011; Wüchner, Ochoa, & Pretschner, 2015; Shen, Hsu, & Shieh, 2017). Syntactic, or static approaches (Zhang & Reeves, 2007; Leder, Steinbock, & Martini, 2009; Griffin, Schneider, Hu, & Chiueh, 2009; Jidigam, Austin, & Stamp, 2015; Zhu, et al., 2018) on the other hand, while initially promising because of their ability to extract signatures that may apply to different minor variants of the same malware code. And to generate signatures irrespective of differences in execution paths, have not managed to keep pace with the latest polymorphic and metamorphic techniques used by hackers to obfuscate their malware (Moser, Kruegel, & Kirda, 2007; Bearden, 2017; Nguyen, 2018). Static methods typically disassemble or reverse engineer the malware executable code so that data mining of the source code is possible; techniques such as statistical analysis of parameter values, repeating strings and comparative entropy (Schultz, Eskin, Zadoc, & Stolfo, 2001; Baldangombo, Jambaljav, & Horng, 2013; United States Patent No. US20170061125A1, 2017), code feature selection (Komashinskiy & Kotenko, 2010; Huda, 2017) and feature extraction (Tabish, Shafiq, & Farooq, 2009; Sun B. L., 2017; Wang, 2017), and n-grams analysis (Abou-

Assaleh, Cercone, & Sweidan, 2004; Kolter & Maloof, 2006; Shafiq, Tabish, & Farooq, 2008; Hassen, 2017) have been used.

For polymorphic viruses:

- Where the virus decryption routine may transform, but the virus body carries the same code (Zhang, Reeves, Ning, & Iyer, 2007), static analysis can still work (Gurnani, 2017).
- This is for the reason that the various executable instances possess a common and invariant source body code whether this code is re-ordered or 'junk' instructions introduced (Kaur & Singh, 2014).

And for metamorphic viruses:

- Where each variant of the virus is structurally and syntactically distinct from the last version but semantically identical (Borello & Me, 2008), create immense challenges for static checkers (Vu, et al., 2017).
- As a result of which common metamorphic instances of the same virus may not be detected as such (Leder, Steinbock, & Martini, 2009).
- Common metamorphic processes include modifying numerical expressions to different but equivalent instructions and modifying constants into computed variables (United States Patent No. US7937764B2, 2011).
- If the morphing part is itself encoded (Sridhara & Stamp, 2013) the critical information relevant to structural change may not be obtainable.

While malware identification is not decidable from a hypothetical viewpoint (Cohen, 1987; Cohen, 1989; Adleman, 1988; Naidu & Narayanan, 2016), it is still not known whether algorithms exist that will consider a random file or script and decide accurately if it possesses particular aspects of a malicious file (Zuo & Zhou, 2004; Narayanan, Chen, Pang, & Ban, 2013). Due to rising intricacy of obscuration and advancement of unseen kinds of transformed malicious program (for instance, ransomware, spyware, adware and botnets), malware specialists are still needed to apply the latest variety of polymorphic and metamorphic malware identification methods currently known to exist (Idika & Mathur, 2007; Robiah, et al., 2009; Fukushima, Sakai, Hori, & Sakurai, 2010; Elhadi, Maarof, & Osman, 2012; Khan, Siddiqui, & Ferens, 2017).

In this thesis, the focus is mainly about polymorphic virus developed via alterations within the encoder and decoder. In the first instance the virus body of a typical polymorphic malware consists of encrypted and decrypted virus body which are mutated by its mutation engine (Chaumette, Ly, & Tabary, 2011; Ali & Soomro, 2018). Although the static components of polymorphic malware like the malware body can be employed to identify polymorphic instances of the same malware in spite of the fact that transformations in the encoder and decoder, there might be zero static components in a metamorphic malware. In comparison to identification of polymorphism via memory-dependent signature identification, metamorphic malware files are semantically examined (for instance, simulated in a virtual sandbox) such that individual instances are detected via normal functionality and behaviour (United States Patent No. US20120072988A1, 2012; Nhuong, 2014). It is hard to state whether metamorphic virus or polymorphic virus is diffcult to identify. However, viruses with functionalties of both metamorphism and polymorphism are the most difficult virus to identify and defend against (Skoudis & Zeltser, 2004; Naidu & Narayanan, 2016). In this research, the focus is mainly with regards to polymorphic virus identification so as to detect virus instances solely via syntactic approach, although, one of the chapters in this thesis focuses on the detection of a metamorphic malware family via syntactic approach. Polymorphic virus consists of three components: the malware structure, the mutation engine and the decryptor (see Figure 1.1 below). Where, L_M means the language of decryptor instead of the complete language of malware (Chaumette, Ly, & Tabary, 2011; Naidu & Narayanan, 2016). Throughout the infection procedure, a polymorphic malicious file M will replicate on its own to a new malicious file $M' \in L_M$ by choosing a randomly encoded key k, encoding the virus structure and consequently the mutation engine by means of the key k and eventually creating a new decryptor, embedding k. Because the virus structure and mutation engine are encoded through a different randomly generated key, detecting polymorphic virus generally means detecting its clear-code decryptor. While both metamorphic and polymorphic malware are encrypted to try evade AVSs, polymorphic malware (as stated earlier) usually preserves its functionality (body containing payload or instructions for infection) with static code and its encryption engine changes with each infection to avoid detection. Signatures based on the static parts of polymorphic body can then be generated for use in AVSs. Metamorphic malware, however, can generate different but functionally equivalent bodies as well as make changes in its encryption engine, making signature generation difficult (Borello J.-M. F., 2010; Mohamed & Ithnin, 2017).

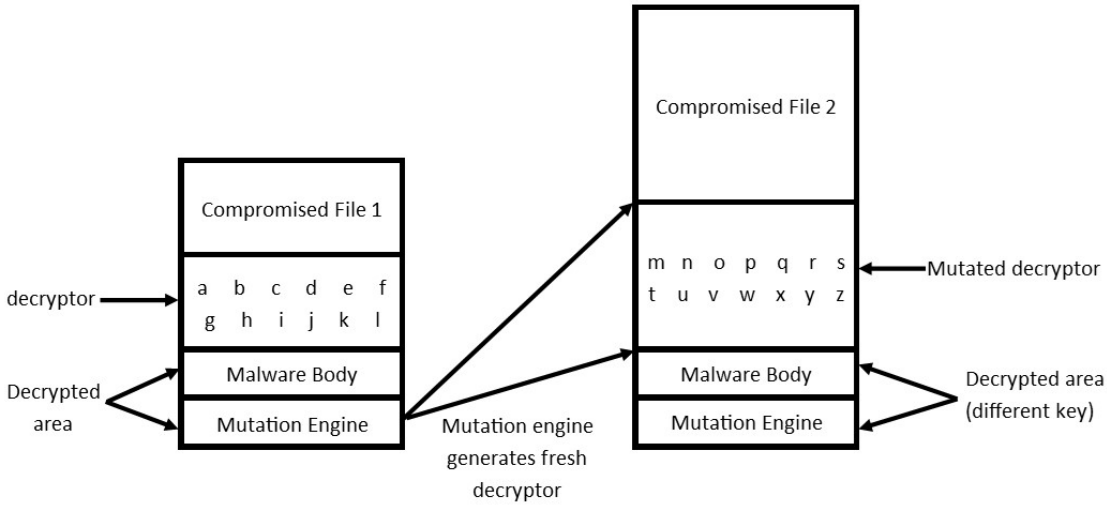

Figure 1.1: The process of polymorphic malware infection (Chaumette, Ly, & Tabary, 2011, p. 41).

1.2 Background and Related Work

Some researchers have continued to search for static malware structure analysis algorithms (Zhang & Reeves, 2007; Leder, Steinbock, & Martini, 2009; Griffin, Schneider, Hu, & Chiueh, 2009; Ye Y. , Li, Jiang, & Wang, 2010; United States Patent No. US20170193229A1, 2017) for use in automatic signature generation despite current emphasis on semantic-based approaches. Static structure analysis can reveal all possible execution paths in a scalable manner, not just those actually followed. Static checkers have, however, faced problems in disassembling the executable code and identifying complex obfuscation (Moser, Kruegel, & Kirda, 2007; Chua & Balachandran, 2018) when attempting to reconstruct the original malware code.

A potential breakthrough in static structural analysis was demonstrated (Narayanan, et al., 2012; Chen, et al., 2012a; Chen, et al., 2012b; Narayanan, et al., 2013a; Narayanan, et al., 2013b) by adopting a nature-inspired and natural computation approach. These works explored the effects of giving amino acid representation to problematic machine learning data and to evaluate the benefits of supplementing traditional machine learning with bioinformatics tools and techniques. The signatures of 60 computer viruses and 60 computer worms were converted into amino acid representations and first multiply aligned separately to identify conserved regions across different families within each class (virus and worm). This was followed by a second alignment of all 120 aligned signatures together so that non-conserved regions were identified prior to input to a number of machine learning techniques. Differences in length between virus and worm

signatures after the first alignment were resolved by the second alignment. Their first set of experiments indicated that representing computer malware signatures as amino acid sequences followed by alignment leads to greater classification and prediction accuracy. Although their experiments led to greater classification and prediction accuracy there was no comparison made with the other state of the art techniques. Their second set of experiments indicated that checking the classification results of data mining algorithms, such as Naïve Bayes, J48, LAD Tree, OneR and Perceptron from artificial virus and worm data against known proteins can lead to generalisations being made from the domain of naturally occurring proteins to malware signatures. However, they stated that further work was needed to determine the advantages and disadvantages of different representations and sequence alignment methods for handling problematic machine learning data.

Narayanan and Chen (Narayanan, et al., 2012; Chen, et al., 2012a; Chen, et al., 2012b; Narayanan, et al., 2013a; Narayanan, et al., 2013b) have shown that malware forms may share deep functional connections with naturally existing counterparts in virology, and biological knowledge could be utilised to detect malware. Some AVSs use signature detection to identify and eliminate viruses. However, by using biological knowledge from bioinformatics and by generating malware variants from scratch using knowledge of polymorphic viruses, it may be feasible to detect syntactic forms that promote to establish if a part of script possesses a malware form along with its instances adopting the techniques introduced by Narayanan and Chen (Narayanan, et al., 2012; Chen, et al., 2012a; Chen, et al., 2012b; Narayanan, et al., 2013a; Narayanan, et al., 2013b). A complicating aspect is that most malware are variants of already existing known types with known code. These variations are in the symbolic code of viruses and worms. There are currently no known techniques for mining symbolic viral and worm (and their variants) code directly. By representing such code in biological form (DNA, amino acids), it may be possible to identify the critical regions of malware code that contribute to malware function through traditional bioinformatics techniques that also attempt to identify sequences through the commonality of biological subsequences.

1.3 Syntactic and Semantic Approaches

The syntactic or static approach can be described formally as follows. Given a grammar G^m for generating malware code, defined as follows: $G^m=(N,T,S,R)$ where N is the set of non-terminal symbols, T is the set of terminal symbols, S is the start symbol and R the set

of rewriting rules, the formal language $L(G^m)$ (the set of possible malware programs generated by this grammar) = $\{x \in T | S \rightarrow x\}$, i.e. all those strings of terminal symbols *x* reachable from *S* through the rewriting rules. The signature extraction task can then be defined to be - find one or more patterns σ of length *l* for one or more $x \in T$ of length *n*, where $l \leq n$, such that σ is a 'signature' of *x* and therefore of $L(G^m)$. The definition of σ will in turn need to take into account partial matches against source code *x* of *T*. The advantage of static methods is that, given G^m, different σ can be derived for actual and possible (i.e. future) *x*. However, the problem for static methods is that, if $L(G_1^m) \gg L(G_2^m) \gg \cdots \gg L(G_k^m)$, where '$\gg$' means 'evolves through polymorphism and/or metamorphism', there will need to be a different σ for each L_i, $i=1..k$, despite the ancestry relationships in G^m.

The semantic approach can be described as follows (Preda, Christodorescu, Jha, & Debray, 2007): program *P* is infected by malware *M* if the semantics of *M* is part of the semantics of *P*, i.e. $S[\![M]\!] \subseteq S[\![P]\!]$, where *S* signifies 'semantics' or execution. If *Obf* is a program transformer that introduces obfuscation into a set of programs **P** such that *Obf*: **P** $\rightarrow$ **P**, a semantic malware detector can then be defined as *D*: **P** x **P** $\rightarrow$ {0,1}: *D*(*P*,*M*) = 1. However, if there are many different semantic interpretations (executions) possible due to obfuscation, each will need to be identified separately to determine semantic inclusion of *M* in *P*.

The fundamental difficulty for a signature extraction approach based on semantics is that a malicious infection must take place in advance before manual signature extraction. Anticipating new (unknown) polymorphic (as well as metamorphic) malware instances to make commercial AVSs prepare for undiscovered instances has continued to be a faraway aim concerning syntactic as well as semantic techniques. In this research, the research goal is about emphasising a string-based automatic signature extraction approach for use in identifying polymorphic malware.

As stated earlier (in the abstract section), syntactic techniques toward signature extraction relying on the syntactic detection of viruses are comparatively unexamined in contrast with semantic techniques. The historical rationale for this is that the same malware behaviour can be exhibited in various physical malware code forms, and therefore the rationale goes that semantic study alone will unveil commonalities amid variants of the same virus for an effective signature generation. There is some previous research on representing viral code as sentences of a viral language and identifying the 'rules' of the

language so that new sentences/viral instances can be created or new instances parsed back to the appropriate viral language. The research hypothesis in this thesis is that, it is feasible to detect syntactic forms for an existent malware that support to establish if a part of malware script possesses a kind of malware along with its malicious variants and, of so, to which family of virus (language) the code belongs. A signature, according to this approach, represents the fragments of code that exist in some or all variants of a virus family. For some simple virus families such as Cascade virus, etc., one signature may be sufficient to identify all variants (Beaucamps, 2007). For other viral/worm families such as Trident Polymorphic Engine (TPE), Code Red worm, Chameleon virus, Whale virus, etc., more than one signature may be required to capture all variants (Kim & Karp, 2004; Beaucamps, 2007). As an instance of a polymorphic string-based method, consider the structurally-related set of sentences:

The boy saw the girl

The girl was seen by the boy

We see that the boy saw the girl

We see that the girl was seen by the boy

Signature extraction is similar to looking for the two patterns 'boy saw girl' and 'girl seen boy' that will help to detect all four sentences as belonging to the same structural set. If options and alternatives are permitted, '{we see} [boy|girl] [saw|seen] [boy|girl]' is an approximate regular expression (signature) for all four sentences that will also permit derivations of new structurally related sentences not so far encountered (e.g. 'the girl saw the boy'). Nevertheless, viral signatures will need to take into account dependencies amidst non-adjacent code to deal with specific polymorphic attributes as well as possible rearrangements of code that modify the left-to-right order of signatures. Note that functionality (semantics) is excluded in these instances. For real viruses, functionality is important, but the hex code of viruses already encompasses semantic attributes because they exist in machine language. Although some polymorphic code goes deeper than mixing of word sequence patterns (as shown above) for malware such as Bistro virus, Dark Paranoid virus, etc. (Beaucamps, 2007). Based on the literature review presented in this thesis, there has been no such previous syntactic approach used in virus detection and signature generation, especially given the immense growth in the understanding of string-dependent searching techniques in the field of bioinformatics during the last 20 years.

1.4 Problem Statements, Research Objectives and Questions

This section provides the problem statement for the identification of polymorphic viruses along with their variants and the research objectives and questions.

1.4.1 Problem Statements

Commercial identification methods are presently not effective in identifying polymorphic viruses due to two major rationales. Primarily, the group of all instances belonging to a polymorphic virus might determine a formal language in non-finite state (Filiol E. , 2007; Naidu & Narayanan, 2016) and thus cannot be detected adopting regular expressions (Chaumette, Ly, & Tabary, 2011; Wagner, Rind, Thür, & Aigner, 2017). Secondarily, semantic-dependent signature extraction methods can be inadequate in dealing with unknown polymorphic instances due to alterations in their behaviours using suitable code rearrangement independent of their unaltered functionalities. The existent signature extraction process is either by a human expert or learning procedure that is, so far, unexplored. The learning procedure is based on rules that are defined beforehand for the kind of signature to be extracted (United States Patent No. US11523199, 2006). Learning sophisticated language categories, such as context-free or regular grammars, are not beneficial through positive inputs alone (Gold, 1967). Nevertheless, it is unknown what an optimal negative category of malware it shall be (for instance, malware files with their payload eliminated, random files, non-malware files, etc.). So far, commercial AVS products have practically maintained pace with new instances regardless of the attempts needed to manually develop signatures presumably for the reason that polymorphic instances to date have demonstrated low levels of complexity (Naidu & Narayanan, 2016). But the growing use of complex techniques by virus writers might rapidly make this technique uncontrollable (Chaumette, Ly, & Tabary, 2011; Naidu & Narayanan, 2016).

Also, as stated earlier in page no. 1, the automatic extraction of malware signatures for use in AVSs remains a relatively unexplored area of cyber security, despite the urgent need to find effective solutions to the increasing number and severity of attacks (Symantec, 2014; Naidu, Whalley, & Narayanan, 2017) that now pose a global risk (WEF, 2012). In this thesis, the nature of this research is to examine if string-searching algorithms, such as the SWA, be able to give rise to string-based syntactic techniques for the automatic extraction of syntactic virus signatures, not only for known (polymorphic/metamorphic) virus instances (P_k – see page no. 28 for more details) but

also for unknown future (polymorphic/metamorphic) virus instances (P_x – see page no. 28 for more details).

1.4.2 Research Objectives

The research objectives of this thesis are constructed as follows:

- To review literature on malware detection techniques employed by modern commercial AVSs and previous research into malware detection.
- Develop techniques that can extract syntactic viral signatures from malware variants and measure their performance by testing them against malware datasets.
- To integrate biosequence analysis techniques into existing approaches of automatic signature extraction for malware detection.
- To assess the performances of modern commercial AVSs by testing them against malware datasets and comparing them with proposed approaches.

1.4.3 Research Questions

The research questions related to this thesis are formulated as follows:

Q1: Can the syntactic viral signatures extracted from the process of alignment techniques detect the known (P_k) and unknown (P_x) variants of polymorphic malware families? Can such syntactic viral signatures outperform the detection capabilities of the current commercial AVSs in the detection of polymorphic malware families? (Chapter 4)

Q2a: Which process of alignment techniques (i.e. local – SWA or global alignment - NWA) would perform better in the detection of polymorphic known (P_k) variants? Can these alignment techniques generate new syntactic viral signatures for viruses that were not previously encountered? (Chapter 5 – Part-I)

Q2b: Can the process of alignment techniques with different combinations of gap penalties identify new syntactic viral signatures for both known (P_k) and unknown (P_x) variants of polymorphic malware families? (Chapter 5 – Part-II)

Q2c: Which substitution matrices, such as BLOSUM (BLOck SUbstitution Matrix), PAM (Point Accepted Mutation), can lead to new syntactic viral signatures with higher accuracies in the detection of polymorphic known (P_k) variants? (Chapter 5 – Part-III)

Q3: Which string representations (i.e. hex, DNA, aligned-DNA) produce syntactic viral signatures with higher accuracies in comparison to each other? Can these syntactic viral

signatures extracted from the process of data mining, such as NNge (Non-Nested generalised exemplars) algorithm supplementing the alignment techniques, detect the known (P_k) and unknown (P_x) variants of a polymorphic malware family? (Chapter 6)

Q4: Can the process of phylogenetics classify the syntactic viral signatures extracted from the process of biosequence analysis techniques, and can these classified syntactic viral signatures detect the known (P_k) and unknown (P_x) variants of a metamorphic malware family? (Chapter 7)

Although Chapter 5 – Part-I and III primarily focuses on identifying new syntactic viral signatures, the main aim of Chapter 5 – Part-1 is to determine whether local or global alignment performs better and that of Chapter 5 – Part-III is to determine which substitution matrices perform best with higher accuracies in the detection of malware variants. Additionally, the alignment tool adopted in this thesis to extract syntactic viral signatures puts sequence length restrictions on longer sequences (more details on page nos. 18, 78, 109 and 123). Therefore, these sub-chapters will only use known (P_k) variants of JS.Cassandra (Chapter 5 – Part-I and III) and W32.Kitti virus (Chapter 5 – Part-I) to test the efficiency, accuracy and effectiveness of the newly generated viral syntactic signatures. Efficiency, accuracy and effectiveness are measured with the help of test statistics such as sensitivity/recall, specificity, precision and F1 score (see page no. 59 for a more detailed description).

In this thesis, the terms signature, meta-signature, and super-signature indicate different kinds of syntactic viral signatures and are defined as follows. A signature is a single string that can identify a single or (in some cases) a few known (P_k) malware variants, whereas a meta-signature is a string (or a common substring/pattern) that can identify most or all known (P_k) malware variants (Naidu, Whalley, & Narayanan, 2017). A super-signature is a string (or a common substring/pattern) that can identify not only all the known (P_k) malware variants but also some or all unknown (P_x) (or new) malware variants.

1.5 Hypothesis and Proposed Approach

1.5.1 Drawbacks of Previous Approaches

It is argued that the limitations of most previous approaches are that a similar malware activity can be exhibited in various physical code structures such that only semantic analysis will unveil commonalities amid instances of a similar malware. This rationale

might be suitable for future, intricate malware types with complicated transmutation methods. The understanding amidst former and existent malware types, nevertheless, is that existing transmutation methods do easy and uncomplicated modifications to produce unseen instances and these might be sufficient to evade identification adopting semantic-dependent signatures. This thesis claims that by employing string-based syntactic approaches such as the string matching SWA and rule-based NNge classifier algorithm, these limitations can be overcome.

1.5.2 Hypothesis

The research hypothesis (as stated earlier on page no. 9) in this thesis is that, it is feasible to detect syntactic forms for an existent malware that support to establish if a part of malware script possesses a kind of malware along with its malicious variants. Similar research hypothesis can be applied on other malware types such as worms, etc. and will be analysed later as a future work. If this hypothesis fails to address uncomplicated polymorphic instances of a virus type, it is implausible that syntactic forms/patterns are identified for further complicated polymorphic instances. To test this research hypothesis, the proposed approach is applied to a more complex polymorphic malware, such as metamorphic malware in Chapter 7. Generally, metamorphic malware variants are the more evolved versions of polymorphic malware (Beaucamps, 2007).

1.5.3 Smith-Waterman Algorithm (SWA)

The string matching SWA will be used in Chapters 4 to 7 to perform the pairwise local alignment. Pairwise alignment is a process of either locally or globally aligning two sequences in order to determine the regions of similarities (or conserved regions) that may indicate structural, functional and/or evolutionary relationships between the sequences (Koyutürk, 2005). Techniques such as string matching are used to detect one or several locations inside a string/sequence where other strings/sequences know as patterns/substrings are identified. Let '$\sum$' be a letter/code (a character) that is a finite set. Traditionally, the patterns/substrings and searched string both are vectors of elements of '$\sum$'. Further, the '$\sum$' might be a general alphabet, that is, for example, the alphabets A to Z in the known Latin format. Many other techniques might adopt binary codes/numbers (that is, $\sum = \{0,1\}$) and in the field of bioinformatics/computational biology, DNA codes (that is, $\sum = \{A,T,G,C\}$) (Charras & Lecroq, 1997).

In this thesis, to extract the most commonly occurring pattern/substring from the polymorphic virus variants belonging to the same family, SWA was adopted. The

algorithm of SWA performs sequence alignment locally amidst the two sequences/strings to search identical segments/sections amidst the two sequences/strings. Those two strings/sequences can be an amino acid (protein) or a DNA (nucleotide) sequence/string. The algorithm of SWA searches the most paired/matched sub-sequences/substrings amid the pattern and search string. Despite scrutinising the entire string/sequence, the algorithm of SWA extracts segments of every possible lengths, subsequently differentiates as well as improves the rate of similarity. The algorithm of SWA can search for exact matches or replaced matches (that is, a code/character within the sequence/string can be replaced by a new code/character, together with no code/character (that is gap), inside the substring/pattern, and conversely). The algorithm of SWA is an adaptation of the NWA (that is Needleman-Wunch algorithm) (Smith & Waterman, 1981). They both are considered as dynamic programming algorithms. Typically, the algorithm of SWA is guaranteed to search the best/exact alignment locally with reference to the substitution/scoring technique that is being adopted (specifically, the substitution and gap scoring scheme). Surely there are several substitution/scoring matrices feasible and adopted by the algorithm of SWA like PAM, BLOSUM and IDENTITY (ID) substitution matrix. Nevertheless, the ID substitution matrix is adopted in most of the research experiments in this thesis to conduct exact/identical matching as it provides the most parsimonious method in that no assumptions are made as to how symbols may be related to each other. The outcomes from the process of SWA are known as 'alignments' because one or the other or both sequences/strings could be altered by the process of gap insertions to generate the best/exact pattern/substring matches. SWA is compared to NWA in Chapter 5 – Part-I to determine which dynamic programming performs well. Moreover, a traditional version of SWA (Smith & Waterman, 1981) is used in this thesis and it is guaranteed to identify the best optimal local alignment (i.e. common substrings/signatures) results (Pham, 2011).

Pairwise local alignment will be carried out using a tool called 'JAligner' (Moustafa, 2010). As stated earlier, 'JAligner' is a freely available open-source Java-based tool that adopts the algorithm of SWA.

1.5.4 NNge

A data mining rule-based classification algorithm known as NNge will be employed in Chapter 7 to generate rules, and the rules will later be adopted to extract common substring/pattern using the SWA from the polymorphic virus variants belonging to the

same family. NNge is a new algorithm that generalises exemplars without overlap or nesting. NNge is an expansion of Nge (Salzberg, 1991), which conducts generalisation by combining exemplars (Panda & Patra, 2009). A learning method based on generalised exemplars begins with a collection of *A* instances (i.e. training instances), $\{I^1, I^2, ..., I^A\}$, each one being distinguished by the values of *m* attributes (the attributes can be nominal, numerical or a combination of both) and a class label. The goal of the learning method is to create a collection of generalised exemplars (i.e. hyperrectangles), $\{G^1, G^2, ..., G^B\}$. A hyperrectangle covers a collection of instances, and each of its dimensions is defined either by a set of categorical values (i.e. nominal values) or by a set of quantitative values (i.e. numerical values).

In a specific example when a hyperrectangle covers just one instance it is regarded to be non-generalised exemplar (Zaharie, Perian, & Negru, 2011). An instance of a hyperrectangle is shown below (Martin, 1995):

class B if p1 = (2 or 4 or 6) AND

p2 = (22) AND

(p3 >= 9 AND p3 <= 32) AND

p4 = (b or c)

Within the NNge algorithm (Martin, 1995) (see below), creating the collection of hyperrectangles starting from the training collection is an accumulative procedure, where, for every instance I^n the subsequent three stages are consecutively enforced, i.e. classification, model adjustment and generalisation. The classification stage locates the hyperrectangle G^b which is nearest to I^n. The model adjustment stage divides the hyperrectangle G^b if it covers an inconsistent instance. The generalisation stage extends G^b in sequence to cover I^n, at most if the generalised instance does not overlap/cover an inconsistent instance/hyperrectangle (Zaharie, Perian, & Negru, 2011).

NNge Algorithm:

For each instance I^n in the training collection do:

*Locate the hyperrectangle G^b which is nearest to I^n /*Classification Stage*/*

IF $D(G^b,I^n) = 0$ THEN

*IF class(I^n)≠class(G^b) THEN Divide/Split(G^b,I^n) /*Adjustment Stage*/*

*ELSE G':=Extend(G^b,I^n) /*Generalisation Stage*/*

IF G' overlaps with inconsistent hyperrectangles

THEN add I^n as a non-generalised exemplar

ELSE G^b:=G'

The classification stage is formulated based on the distance *D(I,G)* between an instance $I=(I_1, I_2, \ldots, I_n)$ and a hyperrectangle *G* as shown in Eq. (3.1).

$$D(I,G) = \sqrt{\sum_{k=1}^{n} \left(w_k \frac{d(I_k, G_k)}{I_k^{max} - I_k^{min}} \right)^2} \qquad Eq.(3.1)$$

In Eq. (3.1), I_k^{min} and I_k^{max} indicates the set of numerical values across the training collection which corresponds to attribute *k*. For categorical (i.e. nominal) attributes, the length of this set is a constant value of 1. G_k is the interval $[G_k^{min}, G_k^{max}]$ if I_k is a quantitative attribute, and is a list of values if I_k is a categorical attribute. The distance between the corresponding hyperrectangle i.e. the 'side' and the attribute values is formulated based on the type of the attribute as illustrated in Eq. (3.2).

$$d_{nom}(I_k, G_k) = \begin{cases} 0, & I_k \text{ belongs to } G_k \\ 1, & otherwise \end{cases}, d_{num}(I_k, G_k) = \begin{cases} 0, & G_k^{min} \le I_k \le G_k^{max} \\ G_k^{min} - I_k, & I_k < G_k^{min} \\ I_k - G_k^{max}, & I_k > G_k^{max} \end{cases} \qquad Eq.(3.2)$$

The constant w_k signifies weights corresponding to attributes and can be regulated throughout the training procedure (Salzberg, 1991) or can be assigned to the mutual information (Zaharie, Perian, & Negru, 2011; Wettschereck & Dietterich, 1995).

The adjustment stage is implemented when a previously created hyperrectangle covers an instance associated with a different class. To circumvent the creation of nested hyperrectangles NNge regulates the current hyperrectangle so that the inconsistent instance is eliminated. This is accomplished by splitting the hyperrectangle into two or more hyperrectangles and potentially into a few isolated variants/instances. The generalisation stage comprises of modifying the 'border' of the nearest hyperrectangle

possessing the same class as the training case in order to cover it. The extension is obtained only when the newly split hyperrectangle does not overlap with hyperrectangles possessing a separate class. If the overlap is detected the training case is included in the model as a non-generalised exemplar (Zaharie, Perian, & Negru, 2011).

Rule extraction adopting the NNge classifier will be conducted with the use of an open source software product known as Weka (Waikato Environment for Knowledge Analysis) Weka is a tool that provides a collection of machine learning algorithms for performing data mining tasks (Frank, Hall, & Witten, 2016).

1.5.5 Limitations of Proposed Approach and Possible Solutions

There are three limitations related to the approaches proposed in this thesis. One is that the string matching search using the SWA finds only the most conserved regions, from left to right, between two variants in DNA representations resulting in meta-signatures (syntactic viral signatures) based solely on syntactic structural commonalities. Such meta-signatures are not likely to successfully identify all unknown (P_x) variants. While meta-signatures obtained using this approach have been shown to successfully identify unknown (P_x) variants of the Win32.Kitti and Win32.Cholera polymorphic viruses with high precision and recall (see Table 5.10 – page no. 101), it is unlikely that such an approach would completely identify all the unknown (P_x) variants of a particular family. However, it was discovered in this research that in the case of JS.Cassandra signatures generated using this approach gave high precision for the randomly generated file but a low precision for non-malicious (P_u) files (see Figures E.1-E.3 – page no. 229). The reason for these results is that all of the JS.Cassandra non-malicious files (P_u) were still executable and as a result, still contain a potential threat. A rule-based approach can be used to overcome the limitation of signatures generated in left to right order. Such an approach tends to use either a divide and conquer (top-to-bottom) approach or alternatively, separate and conquer (bottom-to-top) approach. These methods have the potential to find common substrings with deep syntactic structural commonalities and ensures that every instance (i.e. every polymorphic variant) of the original training set (the variants in DNA representation) is covered by at least one (single) rule, thereby reducing/nullifying the false positive and false negative rates (Witten, 2014; Koklu, Kahramanli, & Allahverdi, 2015).

The primary objective of data mining is to search and extract meaningful and valuable information, sometimes as rules that represent the patterns and clusters in data, from a

vast selection of data (Koklu, Kahramanli, & Allahverdi, 2015; Heikki, 1997). Classification is one of the most common tasks in data mining that uses machine learning techniques, where, considering two or more different classes of sample data, a classifier needs to be constructed by the learner to distinguish between the different classes. Once a classifier has been trained it provides a model that can be used to anticipate the class/group of unseen data. For large datasets, a common approach involves merging association rule mining and classification to produce more efficient and accurate classification systems. Association rule mining helps to identify important relationships and association in data. The benefits of this combined rule-based classification method are the rules that are derived from decision trees, and both the trees and the rules are simple to interpret. Furthermore, new occurrences can be classified quickly (Koklu, Kahramanli, & Allahverdi, 2015; Datta & Saha, 2011). Rule-based common substrings (meta-signatures) obtained in this way might potentially capture knowledge which makes the identification of new variants possible. Thus, the rule-based NNge approach is explored in this research and detailed in Chapter 6 (see page no. 121).

The second limitation worth noting is that a string matching search using the SWA is 'pairwise' and only allows alignment of two sequences at a time. Only the regions of similarity in these sequences are considered in extracting the signatures rather than using knowledge of all the known (P_k) variants. A rule-based data mining approach allows three or more sequences at a time to be used to extract signatures through (single) rules. This means a rule-based approach is capable of identifying variation (and commonality) in patterns of residue i.e. 'conserved regions' for all the variants employed in the training phase.

Third (limitation), using the SWA gives 'out of memory' error and eventually crashing or freezing the software product when aligning larger (DNA/protein) sequences. This means that the time and potential for error are increased because to avoid these memory issues - meta-signatures must be extracted by performing several separate alignments. Larger sequences (greater than 300,000 sequence length) had to be broken down into several smaller sub-sequences and then pairwise alignments run on those sub-sequences. There are currently no openly available online tools or software products that can perform pairwise alignment of larger sequences using the ID substitution matrix (and/or are based on the SWA) except for JAligner, and even JAligner cannot cope with all the sequences which represent malware variants used in this research. The proposed rule-based method

allows several DNA sequences to be input at once and overcomes the issues associated with alignment and long sequences. By generating (single) rules that identify deep structural commonalities from the input DNA strings effective signatures can be extracted (using alignment techniques) which are shorter and require fewer runs of pairwise alignment in subsequent steps.

1.6 Thesis Description

Coming forward to the four research objectives and questions (discussed in Section 1.4), the essential motivation underlying those research objectives and questions have been discussed. The main objective of this thesis is to examine if string searching algorithms, like the SWA, be able to give rise to syntactic techniques for the automatic extraction of syntactic virus signatures/substrings/patterns – not only for known (P_k) malicious variants but also for unknown future (P_x) malicious variants. In the concluding section (i.e. Section 8.5) of this thesis, whether string searching algorithms of greater sophistication has given any advantage is evaluated. The following subsections (i.e. Subsections 1.6.1-1.6.3) will explain the contribution of this thesis, summarise the organisation of the rest of the thesis as well as the list of related publications.

The work in this thesis uses malware datasets that are originally 5-11 years old but their variants were generated during the course of this research and were generated on a Windows 10 environment. These datasets were used in Chapters 4-7. These chapters focused on these old datasets because of the following reasons:

- The experiments from Chapters 4, 5, 6 (Part-II) and 7 show that modern AVSs still cannot completely detect the variants belonging to these virus datasets (Naidu & Narayanan, 2016; Naidu, Whalley, & Narayanan, 2017).
- There is a lack of malware samples (Dumitraş & Neamtiu, 2011) with the capability of generating new variants.
- Furthermore, the aim of this study is to detect the known (P_k) and unknown (P_x) variants of a polymorphic malware family. Finding a new polymorphic (or any other) malware family with proper documentation on its implementation process (Preda & Maggi, 2017), variant generation and original source code (Upchurch & Zhou, 2013) is difficult.

However, a more recent malware dataset is used in Chapter 8. This dataset belongs to a metamorphic malware family. Its variants were also generated on a Windows 10

environment. Two of the four virus families employed in this thesis are a proof of concept malware, namely, Transcriptase metamorphic virus (Musale, Austin, & Stamp, 2015; Troia, Visaggio, Austin, & Stamp, 2016) and JS.Cassandra polymorphic virus (SPTH, 2004; SPTH, 2015).

Some of the malware nowadays are hybrid and may not be purely viruses or worms (Fosnock, 2005). The work in this thesis purely focuses on polymorphic (Chapters 5-7) and metamorphic (Chapter 8) viruses and doesn't deal with hybrid or mixed types of malware. The main aim of this thesis is to detect the known (P_k) and unknown (P_x) variants of polymorphic and metamorphic malware through syntactic signatures. Analysis of hybrid malware will be considered as a future work.

The key differences between the previous work (Narayanan, et al., 2012; Chen, et al., 2012a; Chen, et al., 2012b; Narayanan, et al., 2013a; Narayanan, et al., 2013b) and the work that is conducted in this thesis are as follows:

- Previous work used pre-existing malware signatures for their experiments and current work used the entire hexadecimal (hex) dumps from malware variants.
- Previous work applied biosequence analysis techniques on the malware signatures and current work applied biosequence analysis techniques on the hex dumps of the malware variants.
- Previous work generated JavaNNS/PRISM rules which distinguished viruses from worms. Current work generated viral syntactic signatures that were used to detect known (P_k) and unknown (P_x) variants of a malware family.
- Previous work adopted a different representation approach for converting hex data into amino acid in comparison to current work.
- Previous work used global alignment (i.e. adopting Needleman-Wunsch algorithm) to align their sequences. Current work used both local (i.e. Smith-Waterman algorithm) and global (i.e. Needleman-Wunsch algorithm) alignment but mainly used local alignment to align its sequences.
- Previous work used sequence alignment techniques with fixed substitution matrices (i.e. GONNET and BLOSUM or None) and default gap penalties to align the malware signatures and did not extract the substrings for signature testing purposes, in the same way as is done by the work conducted in this thesis. Current work used sequence alignment techniques with different substitution matrices (but mainly used ID

substitution matrix) and explored the effects of using different gap open and gap extend penalties.

- Previous work did not use the consensus (a by-product of sequence alignment) to extract signatures (i.e. common substrings) for signature testing purposes but the current work did. Previous work used the consensus for data mining and classification purposes.
- Previous work did not compare their results with the other state of the art techniques. But current work compared its results with the other state of the art AVSs.
- Previous work did not measure its efficiency and effectiveness of their techniques but current work did by calculating the test statistics, such as precision, sensitivity, specificity, etc.

1.6.1 Thesis Contribution

The contribution of this thesis to add to the existing knowledge can be outlined as follows:

1. A comprehensive literature review of previous work and malware detection techniques adopted by the state of the art AVSs are highlighted in Chapter 2.

2. Chapter 3 contributes to this thesis by discussing the research method adopted in this study.

3. A detailed analysis of the syntactic process of a string-based method for identifying several polymorphic virus variants is described in Chapter 4. The proposed string-based syntactic approach to the extraction of syntactic virus meta-signatures (common substrings) automatically is shown to identify each and every known (existing) polymorphic virus variants (P_k) belonging to JS.Cassandra virus family.

4. In Chapter 5, further detailed analyses of the syntactic process of a string-based method for identifying several polymorphic virus variants supplemented by the following three sub-chapters (parts) are described:

   a. By adopting the two different dynamic programming approaches (Part-I)

   b. By studying the effects of gap open and gap extend penalties using SWA (Part-II), and

   c. By adopting SWA with several different substitution matrices (Part-III).

This chapter demonstrates that through the proposed string-based syntactic approaches, it is possible to extract meta-signatures/super-signatures after applying data mining classification techniques such as PRISM to the extracted signatures. Such meta-signatures/super-signatures can, in turn, be employed as rule-based string templates for creating more specific, variant-oriented polymorphic malware signatures for detecting polymorphic malware (and its known (P_k) variants) belonging to the same family. In other words, the work presented in this chapter has shown how to progress from viral code consensus identification for a set of executables for the same virus (training set) to the generation of signatures in either regular expression or rule format for identification of other known (P_k) variants of the same virus (test set).

5. Chapter 6 addresses some of the limitations of previous work demonstrated in Chapters 4 and 5 (Naidu & Narayanan, 2016a; Naidu & Narayanan, 2016b; Naidu & Narayanan, 2016c). This chapter demonstrates how representing polymorphic malware as sequences of DNA allows traditional data mining and sequence alignment approach to extract rule-based meta-signatures that help to identify known/unknown (P_k/P_x) variants of that particular polymorphic malware family.

6. Chapter 7 focuses on applying string-based syntactic detection method on detecting metamorphic malware. Previous work mainly focused on detecting the known (P_k) and unknown (P_x) variants of several polymorphic malware families. There was no attempt made to test the capability of string-based syntactic detection method on detecting metamorphic malware and this chapter makes an attempt to do so. This chapter demonstrates the detection of known (P_k) and unknown (P_x) variants of a metamorphic malware family using syntactic viral signatures extracted adopting the techniques proposed in previous chapters. This chapter also distinguishes the syntactic viral signatures.

The intention of this thesis is to aid the global fight against cybercrime through understanding the mechanisms leading to new polymorphic variants so that appropriate automatic signature extraction techniques can be developed to help reduce their impact.

1.6.2 Thesis Structure

The remainder of this thesis is structured into the following six chapters:

Chapter 2: A detailed introduction to classification of malware, such as virus is supplied in this chapter. A review of previous research into malware malware detection is dicussed.

The classification of viruses by their masking strategies is discussed. Subsequently, the discussion in depth about polymorphism, classification of polymorphism, levels of polymorphism, mutation engines, and the polymorphic decryptors. Also, discussion about malware detection techniques adopted by the state of the art AVSs and also, emphasises on the limitations of those detection techniques. The chapter concludes by discussing the history of malware and validation of tool that is adopted in this thesis.

Chapter 3: This chapter covers sections such as the research design and method that are adopted in this thesis, the design of the proposed approach, discussion of results and evidence, and analysis and evaluation of approaches.

Chapter 4: The aim of the research described in this chapter will be to explore effective and efficient syntactic (string-based) approaches (without reverse engineering) of sequence matching algorithm for the detection of all new (unknown) or some (known) polymorphic virus variants automatically adopting the SWA with ID substitution matrix. SWA is used extensively in bioinformatics for sequence alignment (finding common subsequences or consensuses among a set of variable length sequences).

Chapter 5: The aim of the research described in this chapter is to explore in depth, the efficient and effective (string-based syntactic) bioinformatics approaches (without reverse engineering) of sequence matching algorithm for the detection of all new (unknown) or some (known) polymorphic virus variants automatically using the SWA, in three parts.

- In Part-I, two different dynamic programming approaches, that is, the Needleman-Wunsch algorithm (NWA) and the SWA are adopted. The aim here is to compare the two different dynamic programming approaches and to determine which one performs better. Only two commonly used traditional alignment techniques (Lal & Verma, 2017; Prasad & Jaganathan, 2018) are considered in this part for the purpose of global and local alignments as other existing alignment techniques are basically implementations of these techniques (Dohi, Benkrid, Ling, Hamada, & Shibata, 2010; Geers, Çağlayan, & Heij, 2013; Tucci, O'Brien, Blott, & Santambrogio, 2017; Patel, Gandhi, & Bhatti, 2017) that produce similar results (as shown in Section 2.4 – see page no. 48) and are not considered in this part.
- In Part-II, the approach from Chapter 4 is further examined to see whether string searching algorithms of greater sophistication, that is, the SWA with different

combinations of gap open and gap extend penalties, be able to give rise to syntactic techniques for the extraction of syntactic virus signatures automatically – not only for known polymorphic virus variants (P_k) but also for unknown future (P_x) virus variants.

- In Part-III, the approach from Chapter 4 is further refined by adopting the SWA with several different substitution matrices and to determine which one performs better with higher accuracies and generates new syntactic viral signatures.

Chapter 6: Automatic signature generation has concentrated mainly on semantics rather than syntax because of the structural variety shown by viruses in general and polymorphic viruses in particular. Initial work (demonstrated in Chapters 5 and 6) (Naidu & Narayanan, 2016a; Naidu & Narayanan, 2016b; Naidu & Narayanan, 2016c) on string-based syntactic approach using the SWA of representing the hexadecimal dumps of a polymorphic malware and its known (P_k) variants into DNA sequences was promising in a number of ways. The aim of the research described in this chapter is to address some of the limitations of that work and to investigate a syntactic structure approach to automatic signature extraction using data mining algorithm (and sequence alignment approaches). Non-nested generalised exemplars (NNge) are used (as a data mining algorithm) to extract rule-based meta-signatures for the detection of all new (unknown) or some (known) polymorphic virus variants belonging to the same family automatically. Three different sets of experiments are conducted in this chapter.

Chapter 7: A phylogenetic-inspired signature-based syntactic detection method is proposed in this chapter. The proposed method follows the concept of automatic signature extraction from previous work (Chapters 4-6) of generating malware patterns (syntactic viral signatures) in the form of "regions of similarities" (i.e. conserved regions) using sequence alignment techniques. Syntactic viral signatures are extracted from the results of sequence alignments by applying alignments on a small group of metamorphic malware variants belonging to the Transcriptase family. The same group of metamorphic malware variants are again used to generate a phylogenetic tree to determine its metamorphic relationships. The result of phylogenetic tree analysis is later used in the classification of syntactic viral signatures in order to detect the variants of Transcriptase family both systematically and thoroughly.

Chapter 8: The conclusion is described in this chapter, followed by the overview of the work performed in this thesis. Lastly, the objectives for future work are described.

1.6.3 Publications

The five research papers below have been composed and published throughout the doctoral study of this research thesis (Naidu & Narayanan, 2016a; Naidu & Narayanan, 2016b; Naidu & Narayanan, 2016c; Naidu, Whalley, & Narayanan, 2017; Naidu, Whalley, & Narayanan, 2018).

1. Naidu, V., and Narayanan, A., A syntactic approach for detecting viral polymorphic malware variants, Intelligence and Security Informatics (Springer – LNCS 9650), Eleventh Pacific Asia Workshop (PAISI 2016), Auckland, New Zealand, pp. 146-165, April 19, 2016 (Naidu & Narayanan, 2016a). No CORE ranking is available for PAISI workshop. Although the H-Index of Lecture Notes in Computer Science (LNCS) is 251, which was obtained from the Scientific Journal Rankings (SJR) – SCImago. The work reported in chapter four contributes to this paper.

2. Naidu, V., and Narayanan, A., Using Different Substitution Matrices in a String-Matching Technique for Identifying Viral Polymorphic Malware Variants, in the Proceedings of IEEE World Congress on Computational Intelligence (WCCI – CEC 2016), Vancouver, Canada, July 24-29, 2016 (Naidu & Narayanan, 2016b). The CORE2018 conference ranking of IEEE CEC conference is rank B. The work reported in chapter five – part three contributes to this paper.

3. Naidu, V., Whalley, J., and Narayanan, A. (2017). Exploring the Effects of Gap-Penalties in Sequence-Alignment Approach to Polymorphic Virus Detection. Journal of Information Security (JIS), 296-327. The Google-based Impact Factor of JIS journal is 2.43. The work reported in chapter five – part two contributes to this paper.

4. Naidu, V., and Narayanan, A., Needleman-Wunsch and Smith-Waterman Algorithms for Identifying Viral Polymorphic Malware Variants, in the Proceedings of the Fourteenth IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC 2016), Auckland, New Zealand, August 8-12, 2016 (Naidu & Narayanan, 2016c). The CORE2018 conference ranking of IEEE DASC conference is rank C. The work reported in chapter five – part one contributes to this paper.

5. Naidu, V., Whalley, J., and Narayanan, A. (2018). Generating Rule-Based Signatures for Detecting Polymorphic Variants Using Data Mining and Sequence Alignment Approaches. Journal of Information Security (JIS), 265-298. The Google-based

Impact Factor of JIS journal is 2.43. The work reported in chapter six contributes to this paper.

There is now a clear statement about my contribution to these publications in 1.6.3:

“For all five papers, I extracted and prepared all the data, ran all the relevant software, recorded all the output and analysed all the results with supervision from my two PhD supervisors. In all cases, I provided the first full draft, responded to all supervisor comments, submitted the articles as first named author, corresponded with the editor on responses to referees and helped to prepare the final version for submission. For the three conference papers, I made the oral presentations. My contribution to each of these publications is 80% or over.”

Chapter 2 Malware, Polymorphic Malware, and their Detection Approaches

This chapter provides literature review of classification of malware together with their masking strategies and previous research into malware detection. This chapter also discusses in depth about polymorphism, classification of polymorphism (based on both syntactic and semantic reconstruction), levels of polymorphism, mutation engines, polymorphic decryptors and metamorphism. Lastly, the state of the art malware detection approaches along with their limitations are discussed. Additionally, information regarding the materials and tools, such as malware datasets, alignment tool, etc. used in thesis can be found in Appendix C (see page no. 224) and are not discussed in this section.

2.1 Classification of Malware and Recent Research into Malware Detection

Based on a malware's mode of operation, it can be divided into several classes. There are three features of malware, and they are as follows (Aycock, 2006):

1. **Self-reproducing malware:** Malware with this kind of feature when active tries to generate by making new instances, or duplicates of itself (Aycock, 2006).

2. **Population expansion of malware:** This feature of a malware reports the overall variation in the number of new occurrences due to self-reproduction. Malware without this self-reproducing feature will on all occasions have zero population expansion, but malware with a zero population expansion may self-reproduce.

3. **Parasitic malware:** For an existing malware with this kind of feature, it needs some additional executable code. 'Executable' in this instance can be whatever that can be executed, like applications with binary code, decoded code, a disk with a boot block code, etc.

'Virlock', has been identified the first polymorphic ransomware that self-replicates but also reacts as a parasitic malware. Thus, 'Virlock' has all three of the features defined by Aycock (2006). There are only six variants of 'Virlock' known to have been detected by security experts (Paganini, 2014).

In this section, the focus is primarily on viruses as this malware type is employed for experimental purposes in this thesis in order to evaluate a proposed syntactic approach to malware detection.

2.1.1 Virus

A virus is a self-reproducing malware and has the capability of population expansion and is also a parasitic malware. A virus, when executed, attempts to reproduce itself into additional executable code; when it advances, the code is reported to be infected. A virus is a malware. The infected code, when in the process, can infect fresh code one after the other. The key determining a feature of a virus is this self-reproduction into executable code that already exists (Aycock, 2006). The virus called '1260' is the first notorious polymorphic virus, which came into existence, in 1989. The virus scanners at that time could not cope with the '1260' virus because this virus could not be detected using a simple string searching approach (Szőr, 2005). 'Tequila' is the first universal polymorphic virus that came into existence in 1991 (Sharman, Krishna, Rao, & Upadhyaya, 2006). 'ACG' is the first DOS-based metamorphic virus that came into existence in 1998 (Rad, Masrom, & Ibrahim, 2012).

Three types of polymorphic virus families and a metamorphic virus family are used in this thesis and are discussed in depth in Section 2.5, respectively.

2.1.2 Previous Research into Malware Detection

The aim of this thesis is to apply a purely syntactic exploration of the possibility of generating signatures automatically from malware source code without the need for semantic analysis or information.

The task of a syntactic learning system for signature generation of polymorphic malware using hex code only (i.e. no execution traces) is specified below (see Figure 2.1):

(a) From the code of a set of seen variants P_s, automatically generate signatures to identify unseen variants P_u, where P_s and P_u form currently known variants P_k.
(b) From the code of a set of known variants P_k, automatically generate signatures to identify unknown variants P_x for cross-validation. In this case, P_x are code variants that have not been seen before for either training or testing purposes.

The learning task is to maximise true positive rates and minimise false positive and false negative rates in both cases above. It is currently not known whether matching techniques that work well for (a) will continue to work well for (b), or whether bioinformatics and data mining techniques that look for patterns in underlying structure are required to allow generalisation to unknown (P_x) variants.

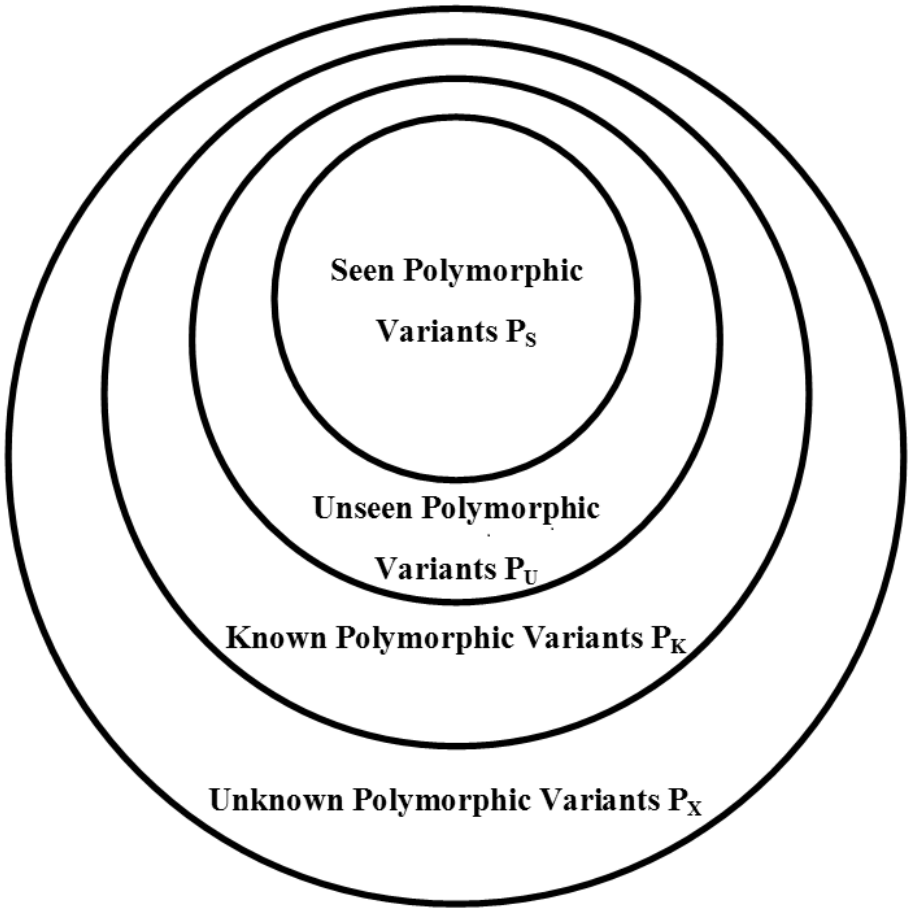

Figure 2.1: Distribution of Polymorphic Malware Variants.

The main body of research over the last fifteen years has concentrated on malware detection adopting semantic-based approaches and only a few adopting syntactic-based approaches. A list of approaches to automatic signature generation is presented in Table 2.1. Most of the previous approaches deal with only a restricted set of variants (i.e. small datasets) (Devesa, Santos, Cantero, Penya, & Bringas, 2010; Lindorfer, Kolbitsch, & Comparetti, 2011; McLaughlin, et al., 2017) either belonging to the same malware family or multiple malware families and it is currently not known how generalisable these approaches are for trapping other variants of the same family, either unseen P_u or unknown P_x (Naidu, Whalley, & Narayanan, 2017).

Table 2.1: Related research to the automatic signature generation in malware detection.

Researchers/Application	Type of Malware	Type of Approach	Description
Wespi, et al. (1999)	Intrusions	Semantic	Variable length patterns from training data consisting of system call traces of commands under normal execution were analyzed by a sequence-based algorithm called Teiresias for intrusion detection.
Honeycomb (Kreibich & Crowcroft, 2004), Autograph (Kim & Karp, 2004) and EarlyBird (Singh, Estan, Varghese, & Savage, 2004)	Worms	Syntactic	Generate signatures that constitute individual adjoining byte strings (tokens).
Polygraph (Newsome, Karp, & Song, 2005)	Polymorphic worms	Syntactic	Identifies an array of tokens, a subsequence of tokens and Bayes signatures based on probabilistic methods to detect polymorphic worms.
Nemean (Yegneswaran, Giffin, Barford, & Jha, 2005)	Worms	Semantic	Focus on identifying signatures that defend against worms.
PAYL (Wang, Cretu, & Stolfo, 2005)	Worms	Semantic	Produces subsequence signature tokens that associate ingress/egress payload notifications to detect the initial replication of worms.
Hamsa (Li, Sanghi, Chen, Kao, & Chavez, 2006)	Polymorphic worms	Semantic	Produces a set of signature tokens that can deal with polymorphic worms by investigating their invariant activity.
ShieldGen (Cui, Peinado, Wang, & Locasto, 2007)	Worms	Semantic	Generates network signatures for unseen vulnerabilities (worms) that are based on protocol-aware for instance.
AutoRE (Xie, et al., 2008)	Botnets	Semantic	Produces a spam signature creation architecture from spam emails that use botnets to detect them.
Coull and Szymanski (2008)	Masquerade	Semantic	Sequence alignment was used to identify masquerade detection by comparing 'audit data' with legitimate user signatures extracted from their actual command line entries.
Scheirer, et al. (2008)	Polymorphic worms	Syntactic and Semantic	Identification of many polymorphic worms and uses intrusion detection techniques such as sliding window schemes and instruction semantics.
Wurzinger, et al. (2009)	Botnets	Semantic	Identifies botnets that are under the influence of botmaster (malicious body) using network signatures by examining the response from a compromised host to a received command and by generating detection models.
Botzilla (Rieck, Schwenk, Limmer, Holz, & Laskov, 2010)	Malware binaries	Semantic	Produces signatures for the malicious activities (traffic) created by a malware binary executed several times within a controlled domain.
Zhao, et al. (2013)	General malware datasets	Semantic	Generates signatures through an inverse transcoding method by converting the malware sequential information, such as system call sequences, propagation dataflow, etc. into amino acid sequences and then aligning them using multiple sequence alignment tool.
ProVex (Rossow & Dietrich, 2013)	Botnets	Semantic	Generates signatures to identify botnets that use encrypted command and control (C&C) systems after being given the keys and decryption routine employed by the malware be derived using binary code reuse strategy.
FIRMA (Rafique & Caballero, 2013)	Botnets	Semantic	Detects C&C systems but does not produce signatures for those.
Ki, et al. (2015)	Worms, Trojans, etc.	Semantic	Generates sequences that are typical API call sequence motifs of malicious activities belonging to several malware samples and employed multiple sequence alignment tool to align those malware samples to extract signatures.
MalGene (Kirat & Vigna, 2015)	Evasive malware samples	Semantic	Uses sequence alignment techniques on two sequences of system call events belonging to two different analysis environments: One environment in which the malware evades the AVS, and the other in which it exhibits the malicious activities. These events are used to construct an 'evasion signature' using sequence alignment.
Fazlali & Khodamoradi (2018)	Metamorphic malware	Semantic	Extracts set of opcode features from disassembled malware code and applies data mining algorithms, such as J48, Random Forest, NB-Tree, LAD Tree, etc. to classify and generate decision trees. Decision trees are created based on statistics of every opcode feature that act as signatures and are used to detect malware. They claim that their proposed approach can lead to an extension of a detection system for generating an automatic signature for a specific malware class.

Some other related and selected previous work that primarily focuses on malware detection using data mining and bioinformatics approaches are shown in Table 2.2. Some research has been undertaken using data mining and bioinformatics approaches for the identification of polymorphic virus and its unseen P_u variants, let alone its unknown P_x variants. Nonetheless, data mining with a novel approach to rule extraction for malware detection was first demonstrated by Chen, et al. (2012a; 2012b).

In contrast to much of the work shown in Tables 2.1 and 2.2, a purely syntactic approach is independent of any prior semantic knowledge. Previous use of sequence alignment and data mining has for the most part been semantic in nature, depending on system behavior patterns (Kirat & Vigna, 2015) or using *n*-grams of bytes (Shim, Kim, & Im, 2015) instead of code or structural patterns for the detection of malware. Also, because of their semantic nature, the generalizability of the results to new P_u variants generated through polymorphism is unknown. A purely syntactic-oriented approach, on the other hand, is based on the intuition that most new P_u (polymorphic) variants are simple syntactic variations of existing malware. The complicating aspect is variable length variations. The 'expressive power' of signatures can be estimated by detecting how well these signatures generalize to unseen P_u and unknown P_x variants of the same family, all obtained through polymorphic (structural) alterations to the code. The benefit of a syntactic approach is that no semantics is needed. More importantly, as will be shown in this thesis, the number of malware training instances required to extract signatures for use against unseen P_u test instances is exceptionally small given the sequence alignment and data mining approaches adopted in the experiments.

Table 2.2: Some related and selected previous work in the detection of malware using data mining and bioinformatics approaches.

Researchers	Data Mining	Data Set	Type of Malware	Type of Approach
Chen, et al. (2012)	Data Mining Classifiers Algorithms i.e. ANNs (Artificial Neural Networks) i.e. JavaNNS and Symbolic Rule Extraction i.e. J48 classifier	60 malicious files, 30 belonging to virus group and 30 belonging to worm group.	One family, with a total of 60 malicious samples, 30 each for virus and worm categories.	Extraction of hexadecimal (hex) sequences from viral and worm malicious files. Multiple sequence alignment using T-Coffee was applied on the extracted hex sequences for data mining process.
Prabha & Kavitha (2012)	Data Mining Classifier Algorithms i.e. J48, KNN (K-Nearest Neighbours), Naïve Bayes.	100 binaries out of which 90 were benign and 10 were malware binaries.	15 subfamilies, with a total of 1,056 malicious viral samples.	Extraction of hexadecimal dumps/Extraction of byte sequences in terms of *n*-grams of different sizes
Kumar & Mishra (2013)	Data Mining Classifier Algorithms i.e. IBK (k-nearest neighbours classifier)	Existing dataset: 323 malicious files with a combination of viruses and worms. New upcoming dataset: 323 malicious files with a combination of viruses and worms.	Virus and Worm.	Extraction of hexadecimal (hex) sequences from viral files and conversion of hex sequences into ASCII sequences. Applies multiple sequence alignment on the converted ASCII sequences for data mining process.
Srakaew, Piyanuntcharatsr, & Adulkasem (2015)	Data Mining Classifier Algorithm i.e. J48 by generating decision trees.	Reference Data Set: 1,200 files in total out of which 900 are malicious, and 300 are non-malicious. Application Data Set: 3,251 files in total out of which 2,951 are malicious, and 300 are non-malicious.	Reference Data Set: Allapple, Podhuha and Virut viral families each containing 300 malicious samples. Application Data Set: Allapple, Podhuha and Virut viral families with 890, 8 and 2,053 malicious samples, respectively.	Statistical Features Approach: Conversion of malicious and non-malicious files into hexadecimal sequences for extracting statistical aspects using *n*-grams of bytes. Abstract Assembly Approach: Conversion of malicious and non-malicious files into assembly instructions for extracting selected instructions using *n*-grams of interesting opcodes.
Vu, et al. (2017)	Data Mining Classifier Algorithms like Multilayer Perceptron (MLP), Gaussian Naïve Bayes (NB) and Support Vector Machine (SVM).	Dataset: 9,690 samples in total out of which 9,390 were malware samples of Locker, Mediyes, Winwebsec, Zbot and Zeroaccess.	Metamorphic malware samples.	Detects metamorphic malware by analysis of Portable Executable (PE) with the "Longest Common Sequence" (LCS). Extracts four different features such as API calls, PE header, code sequences and DLLs import from malware executables. Applies data mining techniques to classify and separate the extracted features.

2.1.3 Classification of Viruses by Masking Strategies

One other way of classifying viruses from both an AVS and a computer user's perspective is in the ways viruses mask (or hide) their presence (Aycock, 2006). This type of strategy commonly adopted by viruses is simply known as masking. From this section onwards, the focus is primarily on simple polymorphism and complex polymorphism (i.e. metamorphism) strategies as the malware types that are employed in the experiments in this thesis use polymorphism (and metamorphism) as their masking strategy. More information on other types of masking strategies are discussed in Appendix A (see page no. 208).

2.1.4 Polymorphism

Many computer viruses use the approach of polymorphism to hide or obfuscate their existence (Hosmer, 2008). Polymorphism is an obfuscating technique that modifies the virus code and forms new mutations or variants of the virus. The mutation aims to hide the fact that the code contains a virus while still keeping its malicious functions intact. The viral code mutates using a polymorphic engine that changes the viral decryption routine using an obfuscation method such as inserting garbage code (see Figure 2.2). Each time the polymorphic engine is executed, it maintains its original algorithm but changes its code.

Figure 2.2: The structure of a polymorphic malware (SANS Institute, 2003, p. 5).

Hiding the existence of a computer virus is accomplished by encryption. When the polymorphic code gets executed, a decryption function inside the computer virus code decrypts the code thereby allowing its tasks to be executed. As soon as the tasks are successfully executed, the encryption function encrypts the decrypted code back to its original form to avoid AVS detection. Polymorphism does not work just because of the encryption. It works largely due to the modification of both the encryption and the decryption code, thereby creating a different form of a computer virus each time it replicates (Hosmer, 2008; Gragido & Elisan, 2012; Ye Y. , Li, Adjeroh, & Iyengar, 2017).

The first widespread computer virus that used a polymorphic engine was called 'Tequila' and was first discovered in 1991 (Sikora & Zelinka, 2017). The second-most commonly known polymorphic computer virus was written in 1992 by the hacker 'Dark Avenger' whose main intention was to avoid the technique of pattern recognition utilised by most of the antivirus programs (Sikora & Zelinka, 2017). 'Virut', another polymorphic computer virus, is a file infector that has been operating since 2006 and recently in 2013 a Polish organisation stopped its malicious activities (Hosmer, 2008; Gragido & Elisan, 2012; Nataraj, et al., 2011; Schwartz, 2013).

Aycock's 2006 classification of viruses by their masking strategies is presented in Table 2.3.

Table 2.3: Virus Classification by their masking strategies collated from Aycock (2006, pp. 34-48).

Virus Masking Strategy	Description	Examples
No Masking	No concealment at any moment is a camouflage strategy which is exceptionally easy to apply in a computer generated virus.	Ply
Stealth	Conceals the virus body and also effectively takes actions to hide the infection itself.	Regin
Encryption	The virus body structure, which largely contains a payload, infection, and trigger, is encrypted making it difficult to detect.	CryptoLocker
Strong Encryption	Uses two different approaches by either retrieving the decryption key from external or internal source of an infected machine for the process of encryption.	RMNS, Dichotomy, CryptoLocker
Metamorphism	When executed produces a logically identical variant of its original source code. It constantly produces machine code and never its original source code.	Simile, Zmist
Oligomorphism (semi-polymorphic)	An encoded virus which has a tiny, fixed amount of several decryptor routines at its disposal.	Whale, Memorial
Polymorphic	Employs an obfuscating technique that modifies the viral code by mutation. Each time the polymorphic engine is executed, it maintains the function of the original infectious algorithm but changes its code.	1260, Virut, JS.Cassandra

2.1.5 Classification of Polymorphism

Polymorphic viruses can be further classified into three different classes based on their obfuscation techniques, namely: self-identification, syntactic reconstruction, and semantic reconstruction. These obfuscation techniques are discussed in depth in Appendix C. A summary outlining the classification of polymorphic viruses based on obfuscation approach is given in Table 2.4. Of these approaches, the syntactic obfuscation approach is a method commonly used by polymorphic viruses to bypass byte level identification and classification – an approach used by many AVSs. The syntactic

obfuscation approach is explored in this thesis, and the viruses that use this approach and are employed in this thesis are JS.Cassandra, W32.CTX and W32.Kitti viruses.

Table 2.4: Classification of polymorphic viruses based on obfuscation method details sourced from Aycock (2006, pp. 38-46) and Cesare (2010, pp. 26-31).

Obfuscation Method	Description	Examples
Self-Identification	• When a malware mutates to infect a new file, it can also modify any form of itself that it encounters. • Malware that is able to identify itself or any of its variants and not infect itself uses self-identification obfuscation.	Zperm
Syntactic Reconstruction	Syntactic reconstruction involves modifying the virus's syntactic structure without altering its semantics.	Tequila, Pathogen, Dark Avenger Mutation Engine (DAME), Virus Creation Laboratory (VCL), JS.Cassandra virus, W32.CTX virus, W32.Kitti virus
Semantic Reconstruction	• Semantic reconstruction is an extension to the syntactic reconstruction where the new instance is a procured creation of the primary malware. • Semantic modification of a malware happens due to the malware writers changing the primary source code or the malware functionality.	TridenT Polymorphic Engine (TPE), Blaster worm variants

2.1.6 Levels of Polymorphism

Based on the complication of code in the malware decryption technique, antivirus software writers place the polymorphic malware into ordered levels. These levels were first defined by Dr. Alan Solomon and then later refined by Vesselin Bontchey. These levels of polymorphism as reported by Ferris (2006) are detailed in Table 2.5.

Table 2.5: Levels of Polymorphism (Ferris, 2006; Belcebu, n.d.).

Polymorphism level	Attributes	Examples
Level 1 Virus semi-polymorphic or oligomorphic	Level 1 virus have a set of decryptors involving interchangeable virus code, choosing one during infection process.	Slovakia Cheeba Whale
Level 2 Polymorphic Virus	The decryptor of a level 2 virus includes one or several interchangeable instructions, simultaneously keeping the rest of its virus code changeable.	JS.Cassandra
Level 3 Polymorphic Virus	The decryptor of a level 3 virus constitutes of 'junk' or non-operating functions like CLI, NOP, STI or XOR.	JS.Cassandra
Level 4 Polymorphic Virus	• Level 4 virus adopts the process of instruction mixing which involves code modification and transposable instructions. • Decryptor algorithm of a level 4 virus stays unchangeable.	W32.Kitti
Level 5 Polymorphic Virus	• Level 5 virus have every aspect described in level 1 to level 4. • The decryptor algorithm of level 5 virus can be changeable. • Possibility of semi-encoding of the decryptor script and constant encoding of the virus script.	TridenT Polymorphic Engine (TPE)
Level 6 Polymorphic Virus	• Level 6 viruses are also called as permuting viruses. And their codes can be decrypted. • While infecting, they are separated into chunks that are placed in irregular order. • Regardless of that, the malware survives to be able to deploy. Such malware may be decoded but generally are not.	JS.Cassandra, AOD.385.B, O97M.Cybernet.Gen, W32.Finaldo.B@mm, TridenT Polymorphic Engine (TPE)
Level 7 Polymorphic Virus	Level 7 virus have every aspect described in level 1 to level 6 along with Heuristic, Entry Point Obfuscation (EPO) as well as Goat and Emulator counter-measure methods (Hamm & Johnson, 2002; Schiffman, 2010).	W32.CTX/W32.Cholera (in this case uses EPO method only)

This thesis deals with levels 2-4 and 6-7, respectively. Three different polymorphic malware families are used in this thesis, namely, JS.Cassandra, W32.Kitti and W32.CTX virus and adopts the levels of polymorphism as shown in Table 2.5. JS.Cassandra virus

adopts levels 2, 3 and 6. W32.Kitti virus adopts level 4 and W32.CTX virus adopts level 7, respectively.

2.1.7 Mutation Engine

A mutation engine is a malicious code; that can be connected or combined with a normal program or malware that will perform the following actions (Ferris, 2006; Li, Loh, & Tan, 2011):

1. Encode itself and also the program that is connected to it.
2. Generate a decryptor that will execute first before the execution of the main program.
3. All the decryptors generated have a unique malware signature.

In a polymorphic malware, the malware body and the mutation engine are both encoded. When a user executes a program compromised by a polymorphic malware, first the decryption routine takes charge of the system, then decodes both the mutation engine and the malware body. Later, the decryption routine transports the control of the system to the malware, which finds a fresh program to compromise (Ferris, 2006; Li, Loh, & Tan, 2011).

At this moment, the malware creates a clone of both the mutation engine and of itself. The clones are created in the Random Access Memory (RAM). The malware then employs the mutation engine, which arbitrarily creates the decryption routine that decodes the malware, yet holds slight or no similarity to any previous versions of the decryption routine. Later, the malware encodes this new clone of the mutation engine and the malware body. Eventually, the malware attaches this fresh decryptor routine together with the mutation engine and the newly encoded virus, to a fresh program. Therefore, not only is the malware body encoded but also the malware decryption routine is different from one infection to the other (Carlin, O'Kane, & Sezer, 2017). This confuses the antivirus software scanner systems looking for the set of bytes i.e. the malware mask (malware signature is sometimes called the malware mask), which detects a particular decryption routine. With no constant decryption routine and no constant malware signature to scan for, no two file infections appear the same (United States Patent No. US5826013A, 1998; Ferris, 2006).

An instance of a polymorphic malware employing a generic mutation engine is shown in Figure 2.3 (Ferris, 2006):

```
MOV DX, 10 ;Actual section of the decryptor!
MOV SI, 4567 ;junk code
AND AX, [SI+4567] ;junk code
CLD ;junk code
MOV DI, cluttered_data ;Actual section of the decryptor!
TEST [SI+4567], BL ;junk code
OR AL, CL ;junk code
primary_loop: ADD SI, SI ;junk instruction, actual loop!
XOR AX, 4567 ;junk code
```

Figure 2.3: An example of a polymorphic malware using a generic Mutation Engine (Ferris, 2006).

The above code suggests that it is purely a set of insignificant instructions to confound antivirus researchers and malware analysts (Ferris, 2006).

2.1.8 Polymorphic Decryptor (The decryption routine)

An instance of a partly polymorphic decryptor with a sequence of instructions is shown in Figure 2.4 below. The purpose of this set of instructions is such that no individual byte of the malware or its decryptor remains consistent while compromising various files (Ferris, 2006; Li, Loh, & Tan, 2011).

```
MOV reg_1, count ; reg_1, reg_2, reg_3 are chosen from
MOV reg_2, key ; AX, BX, CX, DX, SI, BP, DI
MOV reg_3, _offset ; count, key, _offset may also be changeable
_LOOP:
Xxx byte ptr [reg_3], reg_2 ; xor, add, sub
DEC reg_1
Jxx _loop ; ja, jnc
; encoded malware code and contents comes after
```

Figure 2.4: An example of part of a polymorphic decryptor with a sequence of instructions (Ferris, 2006).

Many of the polymorphic malware employ far more intricate algorithms for their decryption technique than the example provided in Figure 2.4. The instructions or their equivalents are diluted by not modifying any instructions such as STC, NOP, DEC, STI, CLC, and the unused instructions like XCHG, etc. (Ferris, 2006).

Polymorphic malware of full strength employs more complex algorithms which lead to any number of various arbitrary instructions such as XOR, SUB, ROL, ADD, ROR, in any arbitrary sequence and in the malware decryptor. All the encryption combinations can also be achieved by an arbitrary construction package that may possess essentially every instruction available in the Intel or AMD processors (Ferris, 2006) with every potential addressing function.

There is a set of obviously useless instructions and compositions that are not dismantled by debugging software products of some commercial companies. For instance, the CS:NOP and CS:CS compositions. So in reality, if the malware writers input some untouched instructions and junk together with these useless instruction compositions such as CS:CS and CS:NOP, antivirus researchers and malware analyst will have a difficult job attempting to crack open the decryption system of that particular malware (Ferris, 2006; Kolesnikov & Lee, 2004).

2.1.9 Metamorphism

Viruses based on metamorphism technique, when executed produces a logically identical variant of its original source code (see Figure 2.5 below), where, G_1-G_n are the current generations, V_1-V_n are the virus bodies and $File_1$-$File_n$ are the infected files (Berg, 2011). It constantly produces machine code and never its original source code. The aim of metamorphism is to avoid getting detected by AVSs (Irshad, al-Khateeb, Mansour, Ashawa, & Hamisu, 2018). Computer viruses based on metamorphism converts their binary code into an interim representation, modifying by themselves the interim representation and then converting the modified pattern back over to machine code (Musale, Austin, & Stamp, 2015). Basically, by following that procedure, no area of the virus remains the same, as the process of metamorphism alters itself, unlike polymorphism that cannot change its original source (Choudhary & Vidyarthi, 2015; Troia, Visaggio, Austin, & Stamp, 2016).

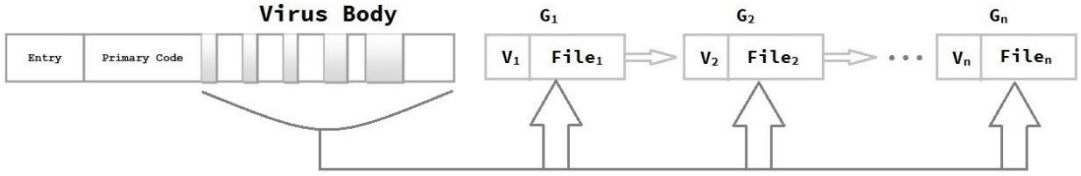

Figure 2.5: The structure of a metamorphic malware (SANS Institute, 2003, p. 5; Berg, 2011, p. 10).

The code-changing approaches utilised by polymorphic viruses all relate to metamorphic viruses. Both use a mutation engine, except on every infection a polymorphic virus does not need to alter its engine because it can reside in the encoded section of the virus. On the other hand, a metamorphic virus needs to self-modify its mutation engine after every infection (Attaluri, McGhee, & Stamp, 2009; Irshad, al-Khateeb, Mansour, Ashawa, & Hamisu, 2018). The mutation engines inside metamorphic viruses whose output and input are machine code must be in the position to restructure and destructure machine code.

Metamorphism is comparatively easy to execute in viruses that transmit in the form of source code, like macro viruses. A virus may depend on the tools available in the infected machine for the process of metamorphism. For example, 'Apparition', is programmed in Pascal and manages its private source code; if a compiler is located on an infected machine, the virus introduces junk material into its source code and recompiles on its own (Aycock, 2006).

Metamorphic malware is a complex version of polymorphic malware, where the complete inner content is altered (Musale, Austin, & Stamp, 2015). Metamorphic malware is occasionally referred as "body polymorphism". For well-structured metamorphic malware, the technique of encoding is not essential, or even beneficial (Musale, Austin, & Stamp, 2015; Naidu & Narayanan, 2016). The virus so-called 'Win95/Regswap', which emerged in 1998, is commonly considered as being the original instance of metamorphic malware (Szőr, 2005).

'Simile' discovered in 2002, is a computer virus that uses the approach of metamorphism that targets Microsoft based Windows machines. The 'Simile' computer virus is written in assembly code (Marinescu, 2003; Konstantinou & Wolthusen, 2008). 'Zmist', also known as 'Zombie.Mistfall' discovered in 2002, is a computer virus that uses the method of metamorphism and was written by a Russian virus programmer called 'Zombie' (Konstantinou & Wolthusen, 2008).

2.2 Malware Detection Techniques

The first attempt to detect and prevent computer malware were undertaken by malware analysts and antivirus engineers who developed specific decryption routines aimed at capturing individual polymorphic malware. Manually, they computed specific programs designed to identify numerous series of computer code understood to be employed by a mutation engine to decode a malware's body (Szőr, Advanced Code Evolution Techniques and Computer Virus Generator Kits, 2005). This virus detection technique was time-consuming, lengthy, expensive, and unrealistic because a mutation engine can generate supposedly arbitrary programs which can systematically execute decryption and thus potentially produce trillions and trillions of variants (Ferris, 2006; United States Patent No. US7337471B2, 2008). Virus detection using this technique also tends to wrongly detect one polymorphic malware as another.

Many polymorphic malware employ the same available mutation engine; credit goes to malware authors like 'Dark Avenger' and others, who distribute these programs freely, sometimes with the source code, to the public. This assists antivirus researchers and malware analysts considerably as they have the mutation engine's decryption program beforehand. However, distinct engines employed by specific polymorphic malware in many cases produce identical decryption algorithms, which makes any detection based solely on decryption algorithms totally untrustworthy. These loopholes helped antivirus engineers and malware analysts to create generic decryption methods that deceive a polymorphic malware into decoding and disclosing itself (Ferris, 2006; Kaushal, Swadas, & Prajapati, 2012).

When an antivirus group obtains a fresh malware, they extract the binary pattern of the malicious file and place it in a database known as the VPF (Virus Pattern File). Maintaining the VPF library is a standard practice for all the antivirus software companies. During the scanning process, the binary pattern within the VPF database is checked with the patterns of the files on the system, and if it is true, then the file is said to be compromised with a malware (Ferris, 2006; United States Patent No. US8935788B1, 2015).

Antivirus researchers employ four measures, to identify a fresh malware strain (Ferris, 2006; Vinod, Laxmi, & Gaur, 2009):

1. **Measure 1:** If feasible identify the malware signature (malware mask).

2. **Measure 2:** Malware detection with the aid of malware mask after eliminating the 'junk' unused instructions.

3. **Measure 3:** Initiate to decrypt the algorithmic virus.

4. **Measure 4:** If it is not feasible to locate the malware signature, the antivirus researcher would then deploy the 'Striker' (Heller, 1996) (also known as the emulator) to force the malware decode itself and disclose the malware binary pattern to the antivirus engineer (Nachenberg, 1996; Gualtieri, 2002).

Because of the varied algorithm in the decryption routine and the quantity of 'junk', the decryptor employs antivirus engineers sometimes are unable to locate the malware mask. Level 6 category malware such as 'AOD.385.B', 'O97M.Cybernet.Gen', and

‘W32.Finaldo.B@mm’ are extremely polymorphic and without a doubt would require Measure 4 identification (Ferris, 2006; Szőr, 2005).

Some of the different kinds of existing state of the art AVS approaches for polymorphic malware (and general malware) detection are discussed below:

2.2.1 Machine Learning/Data Mining Approach

Malware identification with machine learning approaches has now gained acceptance. Mitchell (1997) describes machine learning as the analysis of computer-based algorithms that enhances via evaluation. Moskovitch, Elovici, & Rokach (2008) suggested identification of malicious files based on observing computer activity. They reported that employing classification algorithms using only 20 attributes the mean identification precision surpassed 90%. The benefit of employing machine learning approaches is that it will not only identify known (P_k) malware but may also play a major role in the identification of unknown (P_x) malware. Well-known machine learning approaches used by the researchers for the identification of the second generation based malicious programs are Hidden Markov Models (HMMs), Naïve Bayes, Neural Networks, and Decision Trees (Sharma & Sahay, 2014). This thesis compares the syntactic signatures obtained from the process of biosequence analysis techniques to the state of the art AVSs adopting these machine learning approaches for the purpose of malware detection. The state of the art AVSs such as McAfee, Symantec, Bitdefender and ESET adopt several machine learning approaches to detect the malicious files. For instance, McAfee uses “machine infection characteristics for behavior-based detection” (United States Patent No. US8266698B1, 2012), “cloud-based” machine learning technique (United States Patent No. US15283238, 2016), “Hidden Markov process for outbreak pathology inference” (United States Patent No. US9679140B2, 2017), etc. to detect malicious files. Symantec uses “sequencing and timing information of behavior events in machine learning” (United States Patent No. US8401982B1, 2013), artificial neural network (ANN) (United States Patent No. US8775333B1, 2014) as a threat classifier, etc. to detect/determine malicious files/process. Bitdefender uses “cascading classifiers” (United States Patent No. US20160335432A1, 2016) technique adopting machine learning algorithms to detect malicious files and ESET uses “Augur Machine Learning engine” (Kubovič, 2017) to detect malicious files.

2.2.2 Normalisation Approach

The malware created by high-level construction sets (toolkits) like UPX (Raphel & Vinod, 2015) are very hard to identify. For the identification of such malicious files, normalisation approaches can be employed to enhance the identification rate of known (P_k) malware. In this approach, the normaliser receives an obscure variant of a malware and deletes the malicious activity function performed on the program and then generates the normalised executable form. After the process of normalisation, the malware (normalised form) signature is excavated and checked against the signature of a non-malware file. Christodorescu, et al. (2005) developed a normaliser for malware that manages three general malware concealment activities which are, junk insertion, code reordering, and code packing. Subsequently, Armoun, et al. (2013) implemented a universal normaliser for malware which can stock malware concealment algorithms in the form of automata compositions and employ them for normalising the metamorphic viruses. Currently, a standard normaliser for malware has been implemented in the form of automata compositions for normalizing metamorphic viruses, which has an identification rate of nearly 81% (Sharma & Sahay, 2014).

2.2.3 Scan Engine (Signature based Approach)

The scan engine is a major part of any antivirus software product, and the true measure of its calibre. It is the part of an antivirus software program that scans all the files and identifies malware (Ferris, 2006; United States Patent No. US8813222B1, 2014).

It does not matter how good the user interface of an antivirus software program is because it is the scan engine that ultimately decides how good the program is at detecting malware. When an antivirus software scans a local directory or volume drives, it transmits the files one after the other to the scan engine which compares each file with the VPF (Virus Pattern File) (United States Patent No. US8935788B1, 2015). An exceptional scan engine takes some time, hardly utilising the system resources (United States Patent No. US6851058B1, 2005; Ferris, 2006).

Scanners inbuilt in antivirus programs performs by employing scan strings. They search for a series of bytes in a constant position and a constant string. This constant position and constant string are known as the malware mask or malware signature. Antivirus scanners also utilise ‘Variable Scan Sequences’ technique. During the process of searching for malicious files, the antivirus file scanners search for sequence/string bytes in a separate position but within a definite sequence/string. All of which is performed

inside a virtual machine or an emulator known as “State-based cache for antivirus software” (United States Patent No. US5854916A, 1998), and is one of many approaches adopted by virus scanners. When an antivirus program is deployed it begins to search/scan for malicious files, as each file is included to the scanner, the scanner executes the file inside a Random-access memory (RAM) created emulator. The files included inside this emulator perform in a similar manner as it does on an actual machine. Antivirus scanner verifies as well as handles the executed file in a similar way as it performs within the emulator/virtual machine. A malware executing within the virtual machine can cause no threat for the reason that it is secluded from the real machine. When a scanner places a file compromised by a polymorphic malware into this virtual machine, the malware decryption routine runs and decodes the encoded malware body. This reveals the malware body to the scanner, which can then look for signatures within the malware body that exactly detects the malware strain. If the scanner places a file that is not compromised, there is no malware activity to reveal and quarantine. With regards to the non-malware activity, the scanner immediately halts the process of executing the file within the virtual machine, gets rid of the file from the virtual machine, and continues to scan the following file. This process is called generic decryption as it makes the encrypted file to forcefully decrypt on its own terms. The major drawback with the procedure of generic decryption is the slow process. It has no practical purpose if the procedure of generic decryption takes hours awaiting for a polymorphic virus file to decrypt inside an emulator. In contrast, provided the procedure of generic decryption stops beforehand, it may dodge a polymorphic malware before it is possible to disclose much of itself for the scanner to find a substring/signature. Hence, to defeat the problem, the procedure of generic decryption adopts the process of heuristics (which will be discussed in detail below) - a set of generic instructions that help to differentiate malicious files from non-malicious files. Most antivirus scanners have inbuilt heuristic programs (Ferris, 2006; United States Patent No. US20090013405A1, 2009).

In this thesis, the syntactic signatures are compared with the state of the art AVSs such as Symantec (United States Patent No. US7130981B1, 2006), Bitdefender (United States Patent No. US8813222B1, 2014), etc. that adopts scan engine technology.

2.2.4 Cryptanalysis

Cryptanalysis is the process of decoding coded messages and exploration of ciphers, codes and encoded text, such as a malware mutation engine (Uto, 2013). The main aim

of cryptanalysis is to identify loopholes in a specific program and to decipher the code employed to encode the contents unaware of the key to the code (Filiol E. , 2002). It is (basically) cracking, but in this case, the antivirus engineers are cracking open a malware which employs a particular key (which only the malware writers knows) to decode the malware. There are four fundamental measures (Ferris, 2006) in a classic cryptanalysis process (Uto, 2013) which are as follows:

1. **Measure 1:** To discover the programming language employed to write that specific malware.

2. **Measure 2:** To discover the type of computer system used by that specific malware. This can be a time-devouring phase in the method and comprises of calculating character frequency, implementing statistical tests, and looking for recurrent patterns.

3. **Measure 3:** To modify particular keys of the system (Singh, Troia, Corrado, Austin, & Stamp, 2016).

4. **Measure 4:** Modification of the plain text, this measure occurs at the same time as the process of measure 3.

In this thesis, the syntactic signatures are compared with the state of the art AVSs such as Symantec (United States Patent No. US5826013A, 1998; United States Patent No. US6357008B1, 2002), Bitdefender (United States Patent No. US8813222B1, 2014), McAfee (United States Patent No. US7234167B2, 2007; United States Patent No. US7346781B2, 2008), etc. that adopts the concept of cryptanalysis.

2.2.5 Heuristic Approach

The method aims to detect viruses based on a signature generic to the family of the virus or by an inexact match to an existing signature. This approach permits the AVS author to modify the antivirus scanner by modifying a malware probability. For instance, a genuine non-malware program will, by all means, employ the outcomes from math calculations it performs as it executes within an emulator. Similarly, a polymorphic malware may conduct similar math calculations; yet eliminates the outcomes as those outcomes are inapplicable to the malware. In reality, a polymorphic malware may carry out such calculations simply to make it look like a genuine program in an attempt to evade the malware scanner. In such cases, a heuristic based approach plays a major role. Heuristic based generic decryption searches for such odd activity (United States Patent No.

US20090013405A1, 2009; United States Patent No. US20120266244A1, 2011). An odd behaviour raises the chances of infection and alerts the scanner that depends on the process of heuristic based instructions to increase the time duration a suspicious file runs within the virtual machine, granting a possibly compromised file sufficient time to decode itself and reveal a hidden malware (Ferris, 2006; United States Patent No. US20090013405A1, 2009).

Generic decryption is dependent on a group of antivirus engineers being able to examine billions of possible malware variants, find common areas that all viruses in a family share and contain. While AVS vendors do not make it public knowledge exactly how their systems operate it is believed that these generic signatures contain fragments of unique code from a number of areas in the infected file. These areas are unique to the virus family, and it is from these areas a single generic signature can be created (United States Patent No. US9858414B2, 2018).

Below is an instance of how a heuristic-based scanner performs (Ferris, 2006):

1. **Promoter Instructions:** If a NOP instruction is detected, then increment the malware probability by 0.5%. If the contents of a register are destroyed before scanning, then increment the malware probability by 10%.

2. **Inhibitor Instructions:** If the program generates DOS interrupts, then decrement the malware probability by 15%. If the program does no memory writes among 100 instruction runs, then decrement the malware probability by 5%.

The scan engine assumes that each file has a 10% chance of malicious behaviour. If the malware probability is greater than zero, the process of emulation proceeds. The malware probability is changed throughout the scan each time a file is scanned as the heuristic rules monitor malware-like or non-malware-like activity (Ferris, 2006).

The pitfall of a heuristic-based approach is that it needs progressive evaluation and modification. Heuristic-based rules may have adjusted to detect 300 malware, but for instance, may ignore 15 of that malware when changed to detect ten new malware. Heuristic based rules can also be modified to target any program file, which possesses features of being a possible malware, which in return, increases the time duration it acquires to scan that specific program (Ferris, 2006).

In this thesis, the concept of heuristic approach adopted by the state of the art AVSs such as Symantec (United States Patent No. US6357008B1, 2002; United States Patent No. US7418729B2, 2008), Bitdefender (United States Patent No. US9460284B1, 2016; United States Patent No. US9531735B1, 2016), McAfee (United States Patent No. US7917955B1, 2011), etc. are compared with the approaches proposed here, that is, using the syntactic viral signatures.

2.3 History of Malware – Timeline

1971: Creeper virus – first virus to ever infect computers that were connected to ARPANET network (LAVASOFT, 2013).

1986: Brain virus – first IBM PC based compatible virus and also the first MS-DOS based computer virus.

1987: Vienna virus – first "direct-action" virus to infect Macintosh computers through floppy disk. Stoned virus – first boot sector based computer virus (LAVASOFT, 2013).

1989: 1260 virus – first virus to adopt polymorphism technique to evade AVS detection.

1990: Chameleon virus – first polymorphic virus.

1991: Tequila virus – first widespread polymorphic virus (HubPages, 2014; Johnston, 2014). Dark Avenger Mutation Engine – first well-known virus construction kit and also the first polymorphic generator.

1995: Concept virus – first macro virus that infected Microsoft word documents.

1999: Melissa worm – first widespread worm to combine the techniques of macro viruses with mass-email bombs.

2001: Nimda worm – first widespread worm to employ mass-emailing techniques, attack network drives and exploit web servers based on Internet Information Services (IIS).

2003: JS.Cassandra virus – first open source JavaScript polymorphic virus (VX Heavens, 2006).

2010: Stuxnet worm – first powerful worm to infect Supervisory Control, and Data Acquisition (SCADA) systems (Norton, 2010; Whigham, 2016).

2013: CryptoLocker ransomware – first widespread ransomware to use Trojan to attack Microsoft Windows-based computer systems, which spreads via a compromised email attachment or via a botnet that already exists. This ransomware adopts the approach of public-key cryptography based on the RSA (Rivest-Shamir-Adleman) algorithm (Rivest, Shamir, & Adleman, 1978). Transcriptase virus – first proof of concept and open source JavaScript metamorphic virus (Musale, Austin, & Stamp, 2015; Troia, Visaggio, Austin, & Stamp, 2016).

In this thesis, JS.Cassandra polymorphic virus family first discovered in 2003, W32.CTX/W32.Cholera polymorphic virus family first discovered in 2000, and W32.Kitti polymorphic virus family first discovered in 2011 are employed in Chapters 4 to 6 for experimental purposes. And Transcriptase metamorphic virus family first discovered in 2013 is utilised in Chapter 7. The above-mentioned virus families are discussed in depth in the Appendix C section.

2.4 Tool Validation

This thesis adopts the concept of biosequence analysis techniques. One of the techniques that is majorly used in this thesis is an alignment tool. This alignment tool is used to align two sequences in order to extract the longest common substrings (i.e. syntactic viral signatures) between two malware variants and this process is known as pairwise alignment. The tool that is commonly used to achieve this sort of alignment in this thesis is known as 'JAligner' (Moustafa, 2010). 'JAligner' implements the SWA algorithm (as discussed in page no. 14) and is employed in this thesis to conduct the process of pairwise local alignment. More information about this tool is discussed in Appendix C (see page no. 226). In this section, the tool is validated using the standard predictive validation and by conducting the triangulation approach. For all the alignment processes that are conducted in this section using alignment tools, such as 'JAligner' (Moustafa, 2010), 'Geneious' (Kearse, et al., 2012) and 'EMBOSS Water' (Rice, Longden, & Bleasby, 2000) as demonstrated below, an ID substitution matrix with a gap open penalty of 10.0 and gap extension penalty of 1.0 is adopted.

2.4.1 Predictive Validation

In this section, predictive validation is conducted using two DNA sequences (as shown below) for validation purposes. Predictive validation is a measure developed to reliably predict the future results. This can be accomplished by building a strong relationship amidst scores on the criterion measure and new measure (Salkind, 2010). Two different

processes are conducted in this section in order to achieve a higher degree of predictive validity for the 'JAligner' tool. Process1 (criterion measure) manually conducts pairwise local alignment adopting the technique demonstrated in the literature (Smith & Waterman, 1981a; Smith & Waterman, 1981b; CLC bio, 2007) and process2 (new measure) conducts pairwise local alignment using 'JAligner' tool. Firstly, the sequences are aligned manually and the outcome of the alignment is presented. Secondly, the longest common substring is extracted. The same process is applied using the 'JAligner' tool and the results from the two processes are then compared in order to determine if they produced similar results which in turn will help to validate the 'JAligner' tool.

DNA Sequence 1: `ACTCTG`

DNA Sequence 2: `AGTTCTG`

Process1 - Result of alignment process conducted manually: An alignment matrix table is created through which an optimal alignment score is calculated and the longest common substring is extracted. A match score of 1 and mismatch score of -1 is adopted in this process. Figure 2.6 shows the local alignment matrix table of two DNA sequences generated manually, where, diagonal arrow indicates either match or mismatch, left arrow indicates deletion and up arrow indicates insertion.

SEQ2 / SEQ1	A	G	T	T	C	T	G	NULL
A	3	2	2	2	0	0	0	0
C	1	2	3	3	1	0	1	0
T	1	2	3	4	2	1	0	0
C	1	2	1	2	3	1	0	0
T	0	0	1	1	1	2	0	0
G	0	0	0	0	0	0	1	0
NULL	0	0	0	0	0	0	0	0

Figure 2.6: The manually generated local alignment matrix table of two DNA sequences.

Given, Sequence 1 = `ACTCTG` and Sequence 2 = `AGTTCTG`. Match (a, a) = +1 and mismatch (a, b) = -1 are the scoring functions, therefore, the optimal local alignment is as follows:

```
 (AC)TCTG
      ||||
(AGT)TCTG
```

Giving an optimal alignment score of 4.

Figure 2.6 also gives the following optimal local alignment adopting the trace back analysis, which is achieved by tracing back from the maximum alignment score in the matrix:

```
DNA Sequence 1: 3 TCTG 6
DNA Sequence 2: 4 TCTG 7
```

From Figure 2.6, it can be seen that the optimal alignment score of process1 is 4 (the cell highlighted in orange with the highest score). The longest common substring extracted from this process is TCTG (the cells highlighted in orange). The length of the longest common substring is 4 (TCTG). The identity and similarity percentages are 100%, respectively, as the length of the longest common substring is 4 for both the sequences. The gap score is 0 as no gaps were introduced in this experiment.

Process2 - Result of alignment process conducted using JAligner: The alignment result of two DNA sequences generated using 'JAligner' tool is presented below:

```
Sequence #1: DNA Sequence 1
Sequence #2: DNA Sequence 2
Length #1: 4
Length #2: 4
Matrix: IDENTITY
Gap open: 10.0
Gap extend: 1.0
Length: 4
Identity: 4/4 (100.00%)
Similarity: 4/4 (100.00%)
Gaps: 0/4 (0.00%)
Score: 4.00

DNA Sequence 1      3 TCTG      6
                      ||||
DNA Sequence 2      4 TCTG      7
```

From the above result, it can be seen that the optimal alignment score of process2 is 4. The longest common substring extracted from this process is TCTG. The length of the longest common substring is 4. The identity and similarity percentages are 100%, respectively. The gap score percentage is 0% as no gaps were introduced in this experiment.

The experiments conducted in this section show that both the alignment processes produced similar results/scores, indicating a strong correlation between process 1 and process 2 therefore providing a higher degree of predictive validity for the 'JAligner' tool.

2.4.2 Triangulation Approach

In this section, triangulation approach is achieved by conducting three different experiments from three different sources. Triangulation approach is a powerful cross-verification process where similar experiments from two or more sources are conducted in order to validate the tool (Rothbauer, 2008) used in this thesis (in this case, 'JAligner'). Source1 conducts pairwise local alignment using 'JAligner' tool. Source2 conducts pairwise local alignment using 'Geneious' (Kearse, et al., 2012) tool. Source3 conducts pairwise local alignment using 'EMBOSS Water' (Rice, Longden, & Bleasby, 2000) tool. All of the above experiments are conducted using the same DNA sequences as the experiments conducted in the predictive validation section. The results from the three different sources are then compared for the experimental study of the same condition, in this case, the pairwise local alignment.

Source1 - Result of alignment process conducted using JAligner: The alignment result of two DNA sequences generated using 'JAligner' tool is presented below:

```
Sequence #1: DNA Sequence 1
Sequence #2: DNA Sequence 2
Length #1: 4
Length #2: 4
Matrix: IDENTITY
Gap open: 10.0
Gap extend: 1.0
Length: 4
Identity: 4/4 (100.00%)
Similarity: 4/4 (100.00%)
Gaps: 0/4 (0.00%)
Score: 4.00

DNA Sequence 1      3 TCTG      6
                      ||||
DNA Sequence 2      4 TCTG      7
```

Source2 - Result of alignment process conducted using Geneious: The alignment result of two DNA sequences generated using 'Geneious' tool is presented below:

```
>pairwise alignment - Alignment of 2 sequences: DNA Sequence 1, DNA
Sequence 2

Score = 4.0, Identities = 4/4 (100%),
Positives = 4/4 (100%), Gaps = 0/4 (0%)

DNA Sequence 1   3 TCTG 6
                   TCTG
DNA Sequence 2   4 TCTG 7
```

Source3 - Result of alignment process conducted using EMBOSS Water: The alignment result of two DNA sequences generated using 'EMBOSS Water' tool is presented below:

```
########################################
# Program: water
# Commandline: water
#    -auto
#    -stdout
#    -asequence emboss_water-p1m.asequence
#    -bsequence emboss_water-p1m.bsequence
#    -datafile IDENTITY
#    -gapopen 10.0
#    -gapextend 1.0
#    -aformat3 pair
#    -snucleotide1
#    -snucleotide2
# Align_format: pair
# Report_file: stdout
########################################

#=======================================
# Aligned_sequences: 2
# 1: DNA Sequence 1
# 2: DNA Sequence 2
# Matrix: IDENTITY
# Gap_penalty: 10.0
# Extend_penalty: 1.0
#
# Length: 4
# Identity:       4/4 (100.0%)
# Similarity:     4/4 (100.0%)
# Gaps:           0/4 (0.0%)
# Score: 4.0
#=======================================
DNA Sequence 1      3 TCTG      6
                      ||||
DNA Sequence 2      4 TCTG      7

#---------------------------------------
#---------------------------------------
```

From the above results, it can be seen that the optimal alignment score of sources1-3 is 4. The longest common substring extracted from all the three sources is TCTG. The length

of the longest common substring from all the three sources is 4. The identity and similarity percentages are 100%, respectively, for all the three sources. The gap score percentage is 0% as no gaps were introduced in these experiments.

The experiments conducted in this section show that all the alignment sources produced similar results/scores, indicating the successful validation of the 'JAligner' tool adopting triangulation approach.

2.5 Summary

The classes of malware were discussed. In particular, virus was outlined which have two potential capabilities and are diagnostic of polymorphic malware. That is, they both have self-replicating and population expansion (variant generation) capabilities. Three kinds (or families) of viruses that are polymorphic are employed in the experiments detailed in Chapters 4 to 6. As stated earlier, the polymorphic viruses used are JS.Cassandra virus, W32.CTX/W32.Cholera virus and W32.Kitti virus. Furthermore, in Chapter 7, a metamorphic virus family known as Transcriptase virus is employed.

Classification of viruses based on the type of masking strategy they employ was discussed. Different masking approaches, such as stealth masking, oligomorphism, metamorphism, and encryption, were explored and are detailed in the appendix section (see page no. 208).

Classification of polymorphism was discussed. This chapter also outlined seven different levels of polymorphism. Then the process of generation of polymorphic variants was discussed focusing on the mutation engine and decryption routines employed by polymorphic malware. Malware detection techniques and the history of malware timelines were discussed in this section.

This chapter also presented a discussion on previous research into malware detection.

The existing state of the art malware detection approaches, including signature-based, cryptanalysis and heuristic approaches adopted by commercial AVSs were presented along with a discussion of their limitations in page nos. 43-45. These limitations along with a lack of research in syntactic-based approaches for malware detection as indicated in page nos. 29-32, provide the rationale for the research undertaken in this thesis.

Chapter 3 Research Design

Chapters 1 and 2 identified a gap in the literature that of a lack of automatic syntactic signature generation methods and the inability of commercial AVSs to identify malware not previously encountered. The use of string searching algorithms, such as the SWA for the automatic extraction of viral syntactic signatures was proposed. Moreover, Chapter 2 focused on the state of the art malware detection approaches and highlighted that these methods are not currently adequate for identifying polymorphic (as well as metamorphic) malware and its variants. Hence, the first research question (as stated earlier in page no. 11) is introduced:

Can string searching algorithms, such as the SWA lead to string-based syntactic techniques for the extraction of syntactic virus signatures automatically - not only for known (polymorphic and metamorphic) virus variants (P_k) but also for unknown (polymorphic and metamorphic) virus variants (P_x)?

Epistemology in a scientific research is a part of philosophy that deals with the origin of knowledge (Crotty, 1998). The origin of knowledge (Audi, 2002) can be classified into four categories as follows:

- Intuitive knowledge (when determining the initial concept for research)
- Authoritarian knowledge (when reviewing the literature)
- Logical knowledge (when interpreting findings)
- Empirical knowledge (when conducting experiments that induces these findings)

This thesis combines all of these origins of knowledge stated above. There are five classes/paradigms of epistemology (Scotland, 2012) and they are as follows:

- Positivism (discover truth that's out there)
- Constructivist/Interpretivism (develop truth on the basis of social interaction)
- Pragmatism (ideal method that resolves problems)
- Subjectivism (where every knowledge is simply an element of perspective)
- Critical (where social reality is initiated)

In this thesis, the author adopts a pragmatic epistemology (Armitage, 2007) as it is an ideal method that clarifies problems by combining different perspectives to help understand the data.

The analytical feature of any research method, given the primary research question (as stated above) and lack of preliminary work in the space covered by the research question, it was deemed a requirement to construct a hypothesis, design the experiment, collect the results, and analyse the results, report the outcomes and then if necessary reconstruct the hypothesis, and so on (Garhwal, 2018). This set of procedures is reiterated until, preferably, the ideal solution is established for the ideal test requirements or, as is true of in this thesis, not much time is left. When not much time is left, the benefits of what has been attained requires being assessed both on its importance and innovation as well as the likelihood of extension (Liu, 2014). This thesis will come back to reviewing the benefits of what has been attained in Chapter 8.

This chapter discusses the research design and method adopted in this thesis.

3.1 Research Design

The type of research that is conducted in this thesis is quantitative research. There are two types of quantitative research widely used by researchers, namely, experimental and non-experimental (Marczyk, DeMatteo, & Festing, 2005). This thesis adopts an experimental approach. Furthermore, the experimental approach is divided into three different categories, namely, true experimental, quasi-experimental and pre-experimental/nonexperimental (Marczyk, DeMatteo, & Festing, 2005; Creswell, 2014). In this thesis, the type of research design that is adopted is a quasi-experimental design (Marczyk, DeMatteo, & Festing, 2005; Johanson & Williamson, 2013) as this study conducts a series of individual experiments. The type of quasi-experimental design that is adopted in this thesis is a "non-equivalent comparison-group – post-test only" design (Marczyk, DeMatteo, & Festing, 2005; Privitera & Ahlgrim-Delzell, 2018). This design is selected in this study as it is the most widely adopted quasi-experimental design (Marczyk, DeMatteo, & Festing, 2005). Furthermore, this design can produce valid conclusions with careful evaluation and interpretation (Marczyk, DeMatteo, & Festing, 2005). The main significance of this sub-design ("non-equivalent comparison-group") is that it compares the dependent variables obtained from the treatment group to the non-equivalent control group (Privitera & Ahlgrim-Delzell, 2018). In this thesis, the syntactic viral signatures (dependent variables) from the proposed approach (treatment group) are compared with the modern AVSs (non-equivalent control group) after the experiments/treatments (i.e. a post-test study).

This thesis follows the concept of "Scientific Method" (SM) (Kothari, 2004; Marczyk, DeMatteo, & Festing, 2005; Creswell, 2014) as its research method to address each research objective and question. The scientific method relies on the empirical method (Yanow & Schwartz-Shea, 2015). The empirical method is an evidence-based method that depends on the information obtained from experimentation and thorough observation (Marczyk, DeMatteo, & Festing, 2005). Scientific decisions are made based on the information obtained from experimentation and thorough observation (Kothari, 2004). The empirical method is the perfect guiding principle responsible for all research performed in correspondence with the SM (Marczyk, DeMatteo, & Festing, 2005; Creswell, 2014). Standard steps of the scientific method (Marczyk, DeMatteo, & Festing, 2005) are as follows:

1. Define questions
2. Construct hypothesis
3. Perform experiments
4. Analyses
5. Draw conclusions
6. Replication

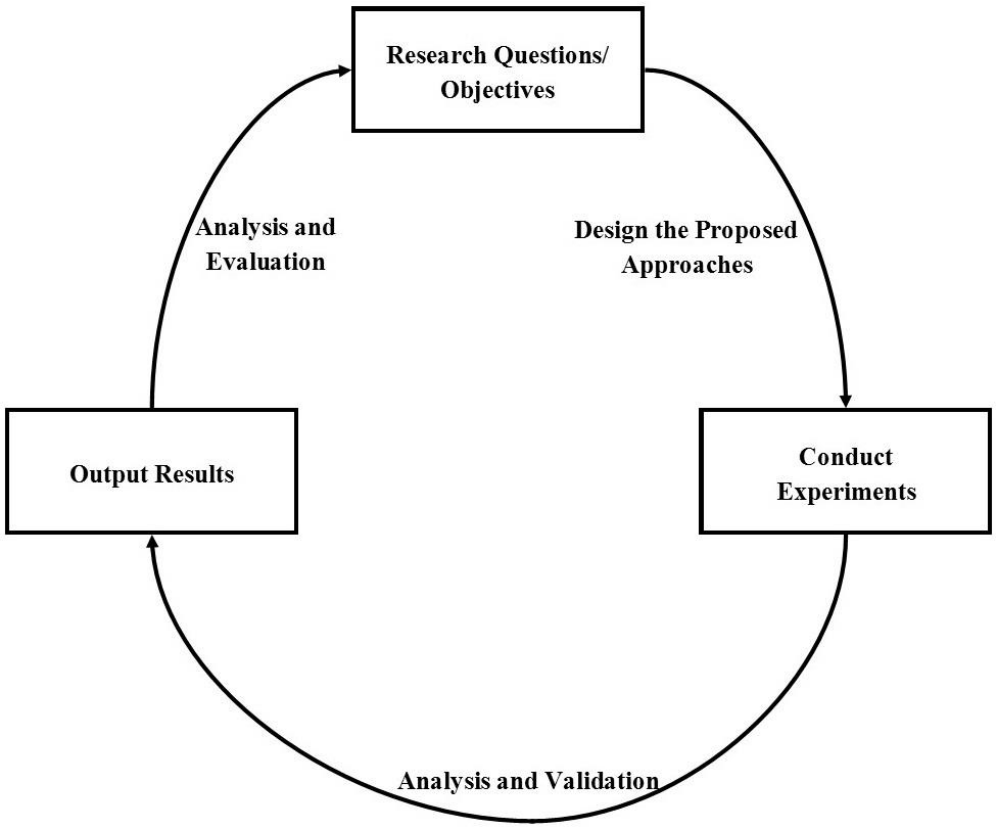

Figure 3.1: SM cycle.

An overview of the SM cycle adopted in this thesis is demonstrated in Figure 3.1. It demonstrates that the research method of this thesis can be divided into following four steps:

1. Identifying and analysing the problem
2. Defining research objectives and questions
3. Designing the proposed approach and conducting experiments
4. Analysis and evaluation

3.2 Identifying and analysing the problem

In this step, the addressed problems determined from a literature review are explained. The gap in research has been identified and defined to frame the problem statement as indicated in Chapter 1 (see page no. 10) of this thesis.

3.3 Defining research objectives and questions

In this step, the research objectives and questions of this study were framed from the current problem addressed from previous related research. These research objectives and questions help to define the hypothesis of this research. The aim is to propose a desirable solution to address the issues indicated through the research questions. The research objectives and questions are discussed in Chapter 1 (see page no. 11).

3.4 Designing the proposed approach and conducting experiments

This step is one of the most crucial segments of this research. The proposed approach along with the limitations of previous approaches are discussed in Chapter 1 (see page no. 12).

In Chapters 4 to 7, for the string-based syntactic approaches adopting the SWA and the NNge, a four-stage experimental method is designed (see Figure 3.2). These major stages will be common for all the experiments that will be conducted in Chapters 4 to 7. Depending on the experiments extra steps may be required and added where appropriate – these will be detailed in the respective chapters.

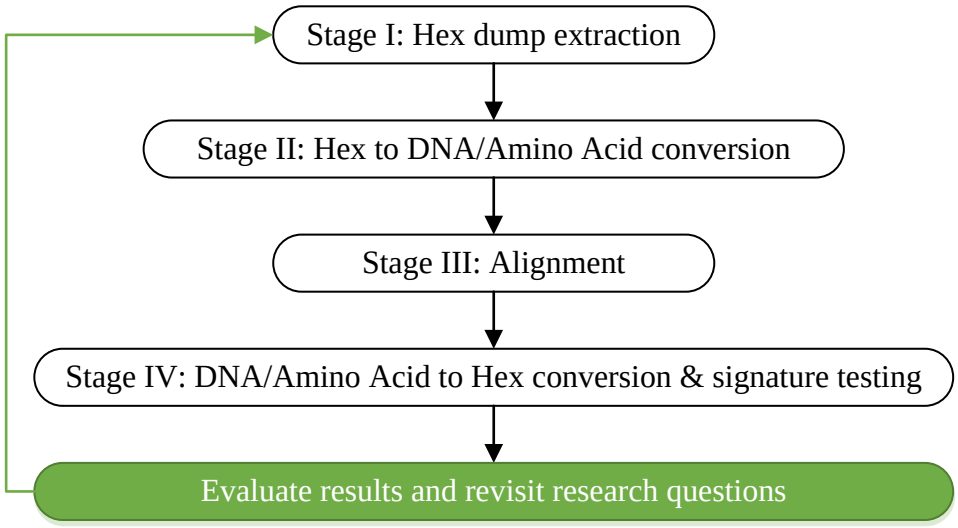

Figure 3.2: The four major stages that will drive this research in Chapters 4 to 7.

Stage I – Hex Dump Extraction: Hex (i.e. hexadecimal) dumps of the polymorphic malware along with its malicious variants of a particular family will be extracted using 'sigtool' which is available from the ClamAV website (ClamAV, 2018).

Stage II – Hex to DNA/Amino Acid Conversion: In this stage, the hex dumps obtained in Stage I will be converted into DNA/amino acid sequences.

Stage III – Alignment: DNA/amino acid sequences from Stage II will be pairwisely aligned using 'JAligner' (Moustafa, 2010). Common substrings (i.e. meta-signatures) will be extracted at this stage. This stage also includes identification and extraction of consensuses and longest common substrings. Also, in this stage, the experiments from Chapters 5-7 perform both multiple and pairwise sequence alignments in some of their steps to extract syntactic viral signatures. Furthermore, Chapter 7 performs the process of phylogenetics for the classification of syntactic viral signatures.

Stage IV – DNA/Amino Acid to Hex Conversion and Signature Testing: In this stage, common substrings extracted in Stage III will be converted into hexadecimal format. After the conversion, the common substrings in their hexadecimal format will be tested against that particular polymorphic family for malicious detection using ClamAV (the 'clamscan' antivirus scanner) (ClamAV, 2018).

3.5 Discussion of Results and Evidence

In this research, the analysis and validation will be conducted by comparing the string-based syntactic approaches with the commercial state of the art AVSs. The comparison will be conducted using one or more polymorphic malware family along with their known (P_k) and unknown (P_x) variants. The results will be presented in a tabular format which will contain the detection performance (test statistics) measures as discussed in the next section.

3.6 Analysis and Evaluation

The effectiveness of the string-based syntactic approaches for each of the polymorphic (and metamorphic) malware families is measured based on test statistics using the following metrics: true positive rate (sensitivity/recall), true negative rate (specificity), positive predictive value (precision), detection ratio (along with accuracy) and F1 score (the harmonic mean of the positive predictive value and true positive rate). F1 score is needed in this thesis as it seeks a balance between precision and recall.

The following formulae will be used to calculate these performance measures (Baratloo, Hosseini, Negida, & El Ashal, 2015; Naidu, Whalley, & Narayanan, 2017):

$$Sensitivity\ (recall) = TP \div (TP + FN)$$

$$Specificity = TN \div (TN + FP)$$

$$Precision = TP \div (TP + FP)$$

$$Accuracy = (TP + TN) \div (TP + TN + FP + FN)$$

Where, TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and, lastly, FN signifies the total number of false negatives.

The Detection Ratio is computed using the following formula:

$$Detection\ Ratio = \frac{total\ number\ of\ malicious\ files\ detected}{total\ number\ of\ files\ scanned}$$

And the F1 score (Sebastián, Rivera, Kotzias, & Caballero, 2016) is calculated by the following formulae:

$$F1\ Score = (2 \times Precision\ \times Recall) \div (Precision + Recall)$$

More specifically, in this thesis,

- true positives are the total number of malicious files (i.e. malware) correctly detected as malicious (i.e. malware),
- true negatives are the total number of non-malicious files (i.e. non-malware) correctly detected as non-malicious (i.e. non-malware),
- false positives are the total number of non-malicious files (i.e. non-malware) incorrectly detected as malicious (i.e. malware) and
- false negatives are the total number of malicious files (i.e. malware) incorrectly detected as non-malicious (i.e. non-malware)

Furthermore, recall signifies "how many malware were spotted (True Positives) among the files found in the test set (True Positives + False Negatives)?". Specificity signifies "how much non-malware were spotted (True Negatives) among the files found in the test set (True Negatives + False Positives)?". Precision signifies "how many files are actual malware (True Positives) among the files that are considered as malware (i.e. True Positives + False Positives)?". High recall (i.e. fewer false negatives) signifies that the detected (malicious) files are correctly spotted as malware. High specificity (i.e. fewer false positives) signifies that the detected (non-malicious) files are inauthentic non-malware. High precision (i.e. fewer false positives) signifies that the detected (malicious) files are authentic malware (Aniello, 2016; Narudin, Feizollah, Anuar, & Gani, 2016).

3.7 Overview of thesis

Chapters 4-7 are the experimental chapters in this thesis. Chapter 4 determines whether it is possible to extract syntactic virus signatures adopting sequence alignment techniques. The main contribution of this chapter is to detect all the known and unknown variants of polymorphic malware families using the newly extracted syntactic virus signatures. Additionally, other contribution is to compare the proposed approach with the other commercial antiviruses to determine their detection capabilities against these polymorphic malware families.

Chapter 5 is divided into three sub-parts. Chapter 5 – Part I compares the two standard sequence alignment techniques, namely, global and local alignment in order to determine which one perform better. The main contribution of this part of the chapter is to extract never seen before syntactic virus signatures adopting global and local alignments and simultaneously determining which ones perform better by testing the newly extracted

signatures against the polymorphic malware family. Chapter 5 – Part II explores different combinations of gap open and gap extend penalties to determine how generalisable the newly extracted signatures are in the detection of known and unknown variants of the polymorphic malware families. This experiment is necessary as the previous chapter (i.e. Chapter 4) only extracts signatures adopting a fixed combination of gap open and gap extend penalties. The main contribution of this part of the chapter is to find the optimal combination of gap open and gap extend penalties that are successful in detecting the known and unknown variants completely and extract new syntactic virus signatures that are not extracted in the previous chapters. Additionally, a comparison is made amidst the newly extracted virus signatures and state of the art antivirus products in order to determine their detection capabilities. Chapter 5 – Part III compares different substitution matrices to determine which one perform better as previous chapters only adopt ID substitution matrix. The other aim is to determine which substitution matrix produce new and more effective syntactic virus signatures in the detection of variants belonging to polymorphic malware families. The main contribution of this chapter is to determine the better performing substitution matrix with better accuracies in the detection of polymorphic malware families and extract new effective signatures.

Chapter 6 addresses some of the limitations of previous chapters adopting sequence alignments (see page nos. 17-19). This chapter extracts further new syntactic virus signatures by combining a data mining algorithm with sequence alignment techniques. This chapter generates malware rules which are then used in the extraction of new signatures for the detection of known and unknown variants of a polymorphic virus family. The contributions of this chapter are as follows:

- Adopting a data mining algorithm, to generate rule-based signatures automatically from real malware data.
- Comparing variable length data mining algorithm to equal length data mining algorithm using NNge on malware source code by conducting three different experiments (Experiments I-III).
- Distinguishing malicious variants from non-malicious with the help of rules generated using the data mining algorithm, NNge.
- Testing the derived rule-based signatures against real malware data and comparing the results to other commercial AVSs.

- Comparing the overall performance metrics such as true positive rate, false positive rate, precision, recall, etc. with other related work on malware detection using data mining algorithms.

Chapter 7 aims to extract syntactic virus signatures adopting biosequence analysis techniques in the detection of metamorphic virus family. This chapter focuses on the detection of variants belonging to a metamorphic virus family. Previous chapters focus on detecting the variants belonging to polymorphic virus families. This chapter mainly focuses on a metamorphic virus family which is a complex version of polymorphic virus (Musale, Austin, & Stamp, 2015). Another aim is to classify the extracted syntactic virus signatures adopting phylogenetics. The contributions of this chapter are as follows:

- Classifying viral signatures acquired from the metamorphic Transcriptase malware family adopting biosequence analysis techniques.
- Distinguishing Transcriptase malware variants adopting phylogenetics.
- Generating syntactic variable-length viral signatures from Transcriptase malware family adopting sequence alignment techniques.
- Testing the classified viral signatures against two different Transcriptase malware datasets and comparing the test results against seven individual commercial antivirus products.
- Testing the classified viral signatures against benign datasets for false positives.

3.8 Summary

This chapter presents a research design in the context of sequence analysis, which adds knowledge to the existing techniques of automatic signature generation for malware detection. It is an ideal strategy to concentrate on the research procedure and systematise the research by constructing and stating the research problem and extracting conclusions that contemplate the real world.

The research design forms a critical inference, which is that our contribution of inspiration from biology will be commenced not from the very beginning of the research but at a point in the research where it is most preferable to do so. Chapter 2 shows that there is a wealth of available literature concerning malware detection methods. Although, these traditional detection methods are well accepted and, in most instances commercially established, they represent a compromise and cannot in all cases completely and successfully detect the known (existing) polymorphic malware variants (P_k), let alone

future (new/unknown) variants (P_x). This does not mean that this study have to begin with a clean slate as such an approach would be throwing away useful and valuable existing foundations which this research may be built on. The objective of this thesis is to explore those features of signature extraction that are well accepted and relevant to this research, and structural detection of polymorphic malware using previously unexplored syntactic approaches rather than semantic approaches.

Along with the research objectives and questions presented in Section 1.4.2 (see page no. 11), a recommended approach is suggested. This approach is devised in order to see how much further this study can reach with modern approaches, extending them where appropriate to make them more desirable and useful for polymorphic malware detection. Once it is determined that this research study have advanced as far as feasible with these conventional approaches, further research investigates how syntactic approaches to signature extraction methods (Chapters 4 to 7) can contribute to malware detection. This work is motivated, as discussed in Section 1.1 (see page no. 1), by the need for automatic signature generation methods and effective approaches for the identification of polymorphic malware variants belonging to that particular family.

The main objective of the next chapter is to see whether it is possible to extract syntactic patterns from semantically rich (polymorphic) hex code using string searching algorithms. An investigation into whether current string searching algorithms, such as the SWA, be able to give rise to string-based syntactic techniques to the extraction of polymorphic syntactic virus signatures automatically is presented in an attempt to answer research question 1.

Chapter 4 A String-Based Method for Syntactically Identifying Polymorphic Virus Variants

This chapter focuses on the identification of polymorphic malware and its variants adopting the Smith-Waterman algorithm (SWA). SWA is adopted in this chapter for the following reasons:

- It is commonly adopted in the field of bioinformatics for performing local sequence alignment (Zahid, Hasan, Khan, & Ullah, 2015). Local alignments are more advantageous for dissimilar sequences that are expected to contain areas of similarities (Moreland, 2006). This chapter conducts local sequence pairwise alignments adopting SWA on dissimilar variable length sequences belonging to polymorphic malware families.
- It finds areas of high similarities between two or more variable length sequences (Xu, et al., 2017). In this chapter, the aim is to find areas of high similarities between polymorphic malware variants.
- It is the most accurate algorithm for conducting local sequence alignment (Xu, et al., 2017) and optimal in identifying local sequence alignments (Zahid, Hasan, Khan, & Ullah, 2015).
- It is a well-known algorithm that finds the longest common substring between two or more variable length sequences (Chen, Wan, & Liu, 2006). This chapter focuses on identifying the longest common substrings (i.e. syntactic patterns) which are employed as syntactic virus signatures in the detection of polymorphic malware variants.

The main purpose of the research presented in this chapter is to see whether it is possible to extract syntactic patterns from semantically rich (polymorphic) hex code and whether the extracted syntactic patterns can be employed for the complete identification of polymorphic malware variants including some or all new variants. Further, the results of the detection capabilities of the proposed approach are compared with that of the commercial AVSs for the detection of known (P_k) and unknown (P_x) variants belonging to the corresponding polymorphic virus families. The results are compared by generating their test statistics, such as accuracy, precision, recall, specificity, etc. as discussed in the previous chapter (see page no. 59).

4.1. Introduction

The goal of this chapter will be to investigate if existent string-based algorithms supported over its growing ability on how to heuristically implement the string-based algorithm for maximal effects, (like the SWA), be able to give rise to syntactic methods to the extraction of polymorphic syntactic virus signatures automatically not only for the variants already seen (P_s) but also for the unseen (P_u) and unknown (P_x) variants. JS.Cassandra virus along with its known (P_k) virus variants belonging to Cassandra polymorphic virus family are used for the experiments conducted in this chapter. Once the method was established for signature testing using the JS.Cassandra family, the method was later tested using the W32.CTX and the W32.Kitti families.

4.2. String-Based Syntactic Detection of Polymorphic Malware Variants Method: An Overview

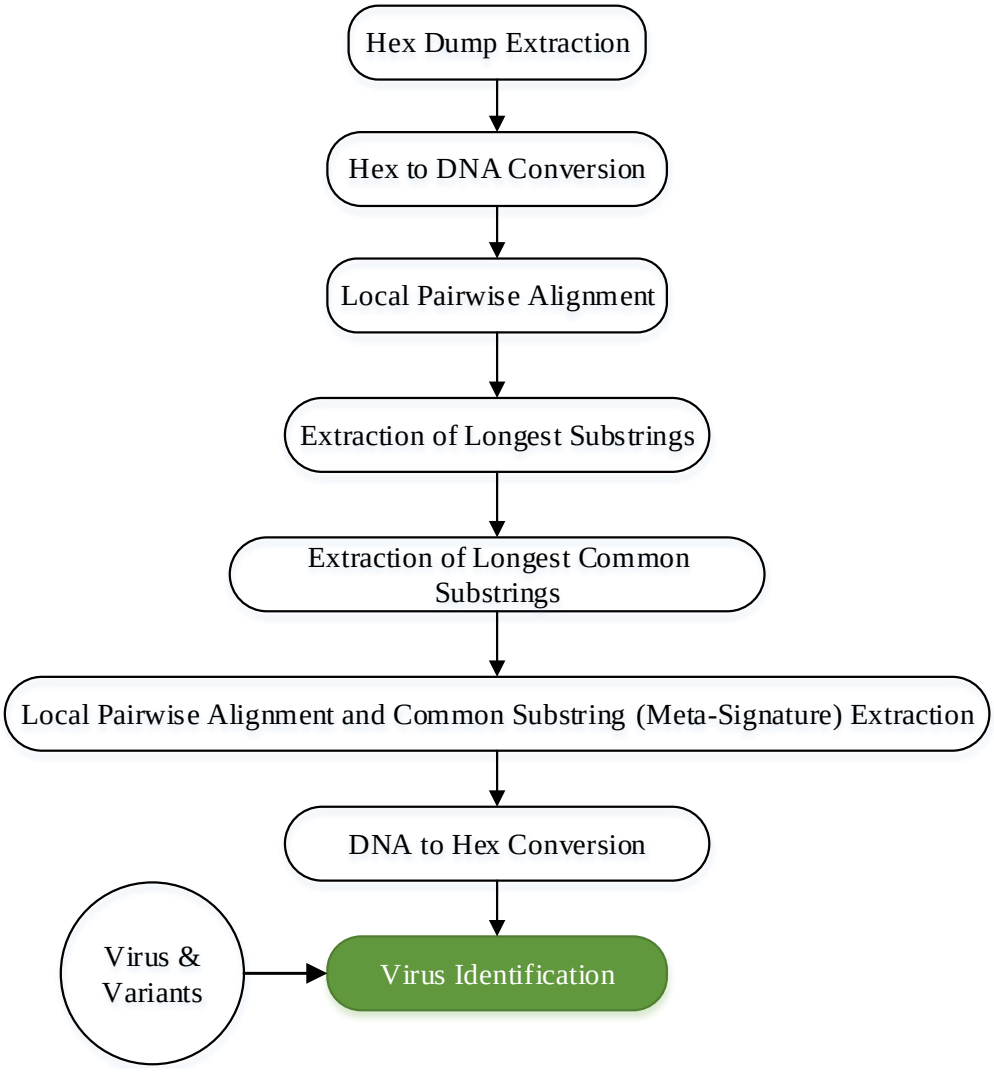

Figure 4.1: The seven steps in the String-Based Syntactic Detection of Polymorphic Malware Variants method.

The method adopted here consists of seven steps (see Figure 4.1). Obtaining the original virus (P_s) and its known (P_k) variants, together with extraction of their hex (hexadecimal) dump and the testing process were carried out on an isolated system in order to avoid any other unexpected system infections. Services of network connectivity is only adopted during the testing process but is carried out adopting 'Oracle VM VirtualBox' together with a previously installed Linux operating system (VirtualBox, 2018).

4.3. String-Based Syntactic Detection of Polymorphic Malware Variants Method: Systems and Methods

An overview of the experimental method, in four stages, was presented in Figure 3.2 and briefly discussed in Section 3.4 (see page no. 57). In this experiment, these stages where appropriate are further divided into clear experimental steps in order to provide a reproducible method (Figure 4.1). The first three stages of the general method presented in Section 3.4 (see page no. 57) are used in this experiment. Firstly in Stage I, 'sigtool' is used to extract hexadecimal dumps of the virus and its variants. In Stage II these hexadecimal sequences are converted to binary format and then into DNA representation using the rules detailed in the next section. In Stage III common substrings are extracted – this stage is divided here into four steps. First, a local pairwise alignment between all of the DNA sequences using the SWA implemented in the JAligner tool was undertaken (Moustafa, 2010). Next, the longest substring from very pairwise (local) alignment was identified and extracted, and from those longest substrings, the longest common substring that captures all the variants in the family of malware tested is identified in Step-5. Finally, in Step-6, a second process of pairwise alignment using the SWA is performed this time between the DNA sequences of the virus and its variants and the longest common substring to extract a meta-signature obtained in Step-5.

In Stage IV signature testing is undertaken. This stage consists of two distinct steps namely DNA to hexadecimal conversion and finally in Step-7, the effectiveness of the signature, using 'clamscan', for identifying the virus and its known (P_k) variants is evaluated.

The next section describes the seven steps in detail. JS.Cassandra and its sources, as well as the software tools used in this experiment, are detailed in the materials and tools section (i.e. Appendix C section – see page no. 224).

4.3.1 Hex Dump Extraction

Step-1: Forty-two variants of the 351 JS.Cassandra known (P_k) variants and the original JS.Cassandra virus (P_s) were selected for this experiment. This gives a sum of 43 malicious files (P_k) and is considered in this chapter as the training set. A new set of 43 non-malicious (P_u) JS.Cassandra files (training set) with no payload were generated by taking out their fundamental polymorphic engines manually. However, most of these newly created non-malicious (P_u) JS.Cassandra files possessed active polymorphic functions. An additional new set of 43 files (test set) were randomly created using RDFC which creates binary files of random size up to a limit of 150 KB. The output RDFC executable files (*.exe) were converted into JavaScript files (*.js). These files are now in the correct format for inclusion in the set of test files and are used to check for correct false positive and false negative rates.

The uniqueness of the 43 malicious (P_k) virus variants together with 43 non-malicious (P_u) variants and 43 randomly generated files was double-checked by generating a "CRC32b hash" value for each of the variants (see Appendix D section – page no. 231). The hash values proved that each variant was indeed unique as no two files had the same hash value, and none of them had the same file size. These files (43 non-malicious – P_u and 43 malicious – P_k variants) were further checked using 'VirusTotal' (VirusTotal, 2018) to determine whether their viral payloads were taken out in the 43 non-malicious (P_u) variants and maintained in the 43 malicious (P_k) variants of JS.Cassandra virus. Furthermore, the randomly created 43 Java files were verified adopting 'VirusTotal' in order to confirm that these files did not already exist in the VirusTotal database. None of the randomly generated files were recognised suggesting that these generated variants are unique.

Also, in this step, the processes of hex dump extraction were carried out on the 43 non-malicious (P_u) variants and 43 malicious (P_k) variants adopting 'sigtool'.

4.3.2 Hex to DNA Code Conversion

Step-2: In this step, after the hex dump extraction, the hex dumps were converted/translated into binary code and subsequently into DNA strings/sequences. This step is necessary because in the field of bioinformatics the string matching algorithms do not merely look for the absence or presence of codes/bases in specific locations but at the same time also manage the strings in such a manner that deletion and insertion of codes/bases are allowed in order to increase the count of matching codes/bases.

Scoring/substitution matrices during the sequence alignment process are also utilised in order to enable a match amidst unmatched codes/bases provided there is a possibility of mutation/transformation to another code/base. Similar substitution matrices could be produced experimentally through former string matches/alignments or a priori based on a fixed substitution rules (Narayanan, et al., 2012; Chen, et al., 2012a; Chen, et al., 2012b; Narayanan, et al., 2013a; Narayanan, et al., 2013b). In this research, a different approach is taken. That is translation of malicious virus code into a suitable biological representation/encoding is adopted prior to the process of sequence matching/alignment, with translation back to hexadecimal code for the process of signature testing.

Translation of hexadecimal sequence into binary code sequence was conducted adopting the subsequent rules:

0 → 0000	4 → 0100	8 → 1000	c → 1100
1 → 0001	5 → 0101	9 → 1001	d → 1101
2 → 0010	6 → 0110	a → 1010	e → 1110
3 → 0011	7 → 0111	b → 1011	f → 1111

Subsequent conversion of the bits into nucleotide bases and thus binary code into DNA sequences for input to JAligner was conducted adopting the following rules:

00 → A; 01 → C; 11 → T and 10 → G

An in-house generated macro was developed in order to carry out the conversion from hexadecimal representation to DNA representation, via binary representation, using EmEditor (Professional edition, 64-bit version) (EmEditor, 2018). All of the 43 extracted hexadecimal dumps of the malicious (P_k) files and 43 hexadecimal dumps of the non-malicious (P_u) JS.Cassandra files were translated into DNA codes adopting this in-house macro. These sequences were then retained for their use in step-3 to step-7. Demonstration of the translation of a 32-bit binary sequence into a 16-bit nucleotide (DNA) sequence is presented as follows:

01010010100100101010010111101101 (32-bit binary code)
CCAGGCAGGGCCTGTC (16-bit nucleotide bases)

4.3.3 Process of Pairwise Local Sequence Alignment

String matching algorithm of SWA is adopted in order to extract the most commonly occurring pattern/substring from the 43 polymorphic malicious variants.

Step-3: In this sequence alignment step, a process of pairwise sequence alignment between two of the generated DNA sequences was conducted locally adopting the SWA with the Identity matrix and 'JAligner'. For example, in this process imagine that there are four variants represented in DNA sequences, namely, D1, D2, D3, and D4, then a process of pairwise sequence alignment will be conducted locally amidst D1 and D2, then D2 and D3 and finally amidst D3 and D4. This procedure was adopted to perform local pairwise sequence alignment on the 43 translated malicious DNA codes.

Step-4: Following the procedure of sequence alignment, extraction of longest substrings from the resulting pairwise sequence alignments were carried out, emanating in 42 longest malicious substrings resulting from the 43 malicious DNA sequences.

Step-5: In this extraction step, amid the 42 longest malicious substrings, the longest common malicious substring is extracted. The extraction of such longest common malicious substring signifies the longest common pattern/substring (encoded in DNA) within the 'polymorphic family' consisting of 43 malicious JS.Cassandra variants. Remainder of the 41 longest malicious substrings were retained for their usage in the process of Step-6.

Step-6: In this second sequence alignment step, a process of pairwise sequence alignment was carried out locally amidst the 43 malicious sequences encoded in DNA (acquired from the process of Step-2) and longest common malicious substring encoded in DNA (acquired from the process of Step-5 above) one after the other. It shows that there is a common syntactic pattern/substring that is same for every polymorphic virus variants belonging to the same family. Such syntactic common substring is the syntactic meta-signature that is employed in order to identify the known (P_k) polymorphic malicious variants of the JS.Cassandra family. In any case, if the first acquired longest common syntactic substring does not provide the optimal common syntactic substring, in that case, the procedure of Step-6 is repeated employing the second retained longest common syntactic substring, otherwise, in that case, the procedure of Step-6 is repeated again employing the third retained longest common syntactic substring, and so on. In total, one

meta-signature (syntactic virus signature) was extracted in this step for the JS.Cassandra virus family.

4.3.4 Meta-Signature Virus Testing

Step-7: In the last step of signature translation and testing, the optimal common syntactic substring encoded in DNA codes is translated back into hexadecimal code. The translated syntactic hex (malicious) meta-signature was verified on the 42 known (P_k) polymorphic malicious variants and JS.Cassandra original (P_s) virus adopting 'clamscan' (ClamAV, 2018) for the sole purpose of their identification. The translated syntactic hex (malicious) meta-signature of 40 characters long that was acquired in this step, is presented as follows:

537472696e672e66726f6d43686172436f646528

Steps 3 to 7 processes were conducted on the 43 JS.Cassandra non-malicious (P_u) sequences encoded in their DNA representation. The syntactic common non-malicious substring (that is, the syntactic meta-signature) is acquired during the process of Step-7. This syntactic non-malicious meta-signature is exactly similar to the one acquired in previous steps from the 43 JS.Cassandra malicious variants. Steps 1 to 7 were also applied on the other two polymorphic virus families and two meta-signatures were extracted, one for each polymorphic virus family. Except in this case only two variants were selected randomly as the training set (P_k) from the individual polymorphic virus family and no non-malicious (P_u) files were generated. Steps 1 to 7 were applied on the corresponding two variants and the meta-signatures were extracted in a similar way as JS.Cassandra virus.

4.4. Experimental Results

In this section, the 43 files belonging to three groups, namely, malicious (P_k), non-malicious (P_u) and random, respectively, were scanned using the 12 commonly used AVSs and meta-signatures obtained in this chapter. These experiments were performed in order to determine the detection capabilities of AVSs and the meta-signatures against the polymorphic virus families. The scan results of those experiments are detailed below.

The 43 files from the three separate groups were scanned individually employing the 12 commercial AVSs. This experiment was conducted to determine the detection capability of these AVSs against these files. Appendix D (see page no. 232) provides the scan results and test statistics of some of the AVSs tested against these files.

The syntactic meta-signature acquired from the process of Step-7 was scanned against the three individual groups. A signature database file is developed in .ndb signature file format (that is, ClamAV Extended Signature File) for validating the syntactic meta-signature adopting the scanner tool called 'clamscan' antivirus scanner belonging to ClamAV (ClamAV, 2018). The format of signature database for scanner tool 'clamscan' is specified using the standardised structure, that is, "MalwareName: TargetType: Offset: HexSignature" (Naidu & Narayanan, 2016). More information regarding the clamscan database file can be found in Appendix D (see page no. 228). The screenshots of the clamscan scan results for the three individual groups utilising the syntactic meta-signature acquired through the proposed seven-step method are presented in Appendix D (see page no. 229).

The scan results show that 43 of the 43 JS.Cassandra malicious (P_k) variants, 43 of the 43 JS.Cassandra non-malicious (P_u) variants along with 0 of the 43 randomly created Java programs were fully identified as infected/malicious with the help of 'clamscan' scanner tool involving the syntactic meta-signature in under 0.3 seconds, respectively. The detection accuracy of the 'clamscan' scanner tool utilising the syntactic meta-signature with regards to the original polymorphic (P_s) JS.Cassandra virus file together with its set of 43 known (P_k) malicious as well as 43 non-malicious (P_u) variants, was altogether 100%. Further experiments were conducted in this section and the results of those experiments are presented in Appendix D (see page no. 233). These experiments were conducted to determine whether it was possible to generate variants from the JS.Cassandra non-malicious (P_u) files. The same meta-signature obtained from the above experiments was tested against these newly generated variants. The results are presented in Appendix D (see page no. 235).

Finally, the original polymorphic (P_s) JS.Cassandra virus together with its set of 351 known (P_k) malicious variants (test set) were all-around tested for the process of malicious identification employing commercial AVSs such as 'Microsoft', 'clamscan' and 'ESET'. Both the Microsoft and the ESET scanners were installed on a Microsoft Windows operating system (Windows 10 Professional edition). Installation of the 'clamscan' scanner tool was done on a Linux-based operating system employing ClamAVs built-in database and adopting the database created in this research incorporating the syntactic meta-signature encoded in hex code. The signature databases of the three AVSs were comprehensively up to date together with the most recent updates, at the time of the

experiments, installed. The scan results show that the original polymorphic (P_s) JS.Cassandra file together with its set of 351 known (P_k) malicious variants were fully identified as malicious/infected in well under 0.995 sec by the 'clamscan' utilising the meta-signature (see Appendix D for scan result – page no. 235). Only the 'Microsoft' antivirus tool and identification utilising the syntactic meta-signature could fully detect each and every malicious variants with an identification ratio of 352 out of the 352 malicious files and a detection accuracy of exactly 100% (Table 4.1).

Identification of the other two polymorphic viruses together with their unknown (new) malicious variant files (P_x) were also tested employing the newly generated syntactic meta-signature: the scan results were altogether 100% (see Table 4.1). Even though some commercial AVS products could fully detect each and every malware variants they were uncongenial in fully detecting the polymorphic viruses as well as their known (P_k) and unknown (P_x) variants (Table 4.1). For instance, 'Microsoft' scanner be able to merely detect 80 of the 1106 unknown (P_x) malicious 'Win32.Kitti' variants with an identification accuracy of 7% (overall) but at the same time be able to fully detect each and every malicious variants belonging to other two polymorphic virus types demonstrating the inconsistency of its detection algorithm.

The syntactic meta-signature acquired through the proposed seven-step syntactic method for the same polymorphic family of JS.Cassandra virus (P_s) together with its known (P_k) malicious variants was decrypted into a text and signifies a JavaScript function – '`String.fromCharCode(`'. Such function is usually a Java code function within the original source code belonging to the polymorphic JS.Cassandra virus (P_s) along with its known (P_k) malicious variants. While such viruses do not have a readily available source code due to which malware analysts require to reverse engineer the malware files in order to retrieve the source code adopting a very sophisticated procedure (Naidu & Narayanan, 2016). Provided, in this thesis, the source code of JS.Cassandra (P_s) virus along with its known (P_k) malicious variants was readily available, although commonly, viruses created using JavaScript scripts are either password protected or enciphered. As noted earlier, the scan engines of AVS products adopt the technique of traditional "Variable Scan Sequences". The crucial problem with this approach is that it is too tedious (Ferris, 2006).

Table 4.1: Test statistics and time interval to the identification of three individual polymorphic viruses as well as their known (P_k) and unknown (P_x) malicious variants adopting 'clamscan' scanner tool, ESET, Windows Microsoft Defender and the Syntactic Meta-Signature (extracted from the proposed seven-step approach).

Virus	**JS.Cassandra** together with its 351 known (P_k) malicious variants (Test set)				**Win32.Kitti Virus** together with its 1105 unknown (P_x) malicious variants (Test set)			
AVS	Microsoft Defender	ESET	clamscan	Meta-Signature	Microsoft Defender	ESET	clamscan	Meta-Signature
Detection rate	352/352 (100%)	296/352 (84%)	340/352 (97%)	**352/352 (100%)**	80/1106 (7%)	1106/1106 (100%)	1/1106 (0.09%)	**1106/1106 (100%)**
Sensitivity/Recall	100%	84%	97%	**100%**	7%	100%	0.09%	**100%**
Specificity	0.0%	0.0%	0.0%	**0.0%**	0.0%	0.0%	0.0%	**0.0%**
Precision	100%	100%	100%	**100%**	100%	100%	100%	**100%**
F1 Score	100%	91%	98%	**100%**	13%	100%	0.18%	**100%**
Detection time	unknown	4 seconds	30.613 seconds	**0.995 seconds**	unknown	43 minutes and 23 seconds	39.378 seconds	**14.003 seconds**

Virus	**2 x Win32.Cholera** together with its 198 unknown (P_x) malicious variants (Test set)			
AVS	Microsoft Defender	ESET	clamscan	Meta-Signature
Detection rate	200/200 (100%)	200/200 (100%)	67/200 (33%)	**200/200 (100%)**
Sensitivity/Recall	100%	100%	33%	**100%**
Specificity	0.0%	0.0%	0.0%	**0.0%**
Precision	100%	100%	100%	100%
F1 Score	100%	100%	50%	**100%**
Detection time	unknown	23 seconds	34.009 seconds	**1.008 seconds**

4.5. Summary

In this chapter, the efficient and effective syntactic approach of using string matching algorithms such as the SWA for the automatic extraction of signatures for the detection of some or all new polymorphic variants was examined. The experiments reported in this chapter demonstrate to some extent that current and modern state of the art AVS products cannot completely and successfully detect the known (existing – P_k) and future (new/unknown – P_x) polymorphic malware variants. By downloading the original polymorphic JS.Cassandra virus together with its known (P_k) malicious variants in its original JavaScript code as well as generating new (unknown – P_x) variants of the other two polymorphic malware using information contained in documents concerning polymorphic versions, the authenticity of the polymorphic variants was assured. By removing the payload from each of the JS.Cassandra malicious variants and checking all 86 variants (43 malicious – P_k and 43 non-malicious – P_u variants) against a number of AVS systems (using the 'VirusTotal' online tool), it has been verified that these variant files (and the code they represent) represent malicious (P_k) and non-malicious (P_u) variants. All the (pairwise) alignments were conducted with the ID matrix rather than biologically plausible mutation matrices (such as BLOSUM and PAM), addressing the concern that biological bias was being introduced into the alignment results. All the (pairwise) alignments were conducted using a fixed combination of gap open penalty (i.e. 10) and gap extend penalty (i.e. 1). By using an in-house macro tool, this chapter has shown that a natural computation approach of projecting polymorphic malware (hexadecimal) code onto biological representational space (i.e. DNA) and vice-versa and then using bioinformatics algorithms (i.e. pairwise alignment and SWA) can successfully automate signature extraction using the proposed (seven-step) string-based syntactic approach developed in this research.

In this chapter, all the syntactic meta-signature (common substring) testing against the malicious variants of polymorphic virus families was conducted adopting the 'clamscan' scanner tool, which belongs to a module in the ClamAV AVS product (ClamAV, 2018). Experimental results from this testing process are shown in Table 4.1. Based on Table 4.1, this chapter demonstrates how the proposed string-based syntactic method adopting the traditional algorithm of string matching SWA can fully detect the previously known (existing) malicious variants (P_k) belonging to the same family of original JS.Cassandra polymorphic virus (P_s) and outperform the detection capabilities of commercial AVSs. In-depth, it was demonstrated that it is possible to extract syntactic patterns from

semantically rich (polymorphic) hex code and that the extracted syntactic patterns (i.e. common substrings/meta-signatures) can be used for the complete identification of polymorphic malware variants. This proposed string matching approach adds knowledge to the automatic signature generation contributing to understanding the polymorphic variant and signature extraction and be able to give rise to a modern era of string-dependent syntactic AVSs.

The next chapter investigates whether further refined string searching algorithms, such as the SWA and NWA, can lead to syntactic techniques for the automatic extraction of new polymorphic viral signatures. Experiments are reported that are designed to address the second research question and its sub-questions (**Q2a** to **Q2c**).

Chapter 5 Exploring Advanced Sequence Alignment Techniques in a String-Based Syntactic Method for Identifying Malicious Variants of Polymorphic Virus Families

In the previous chapter, the efficient and effective syntactic based technique employing the algorithms of string matching for the detection of all new or some malicious variants of polymorphic virus families using the automatically extracted syntactic signatures was investigated. In this chapter, this work is extended. The chapter is presented in three parts. Part-I (addressing research sub-question – Q2a) introduces two different dynamic programming methods i.e. Needleman-Wunsch algorithm (NWA) and Smith-Waterman algorithm (SWA) as methods for improving the detection of viral polymorphic malware variants. Both approaches are evaluated for two different polymorphic malware and their known (existing – P_k) and unknown (new – P_x) variants. More detailed information on why this part focuses only on NWA and SWA is outlined on the next page. Part-II (research sub-question – Q2b) examines the effects of different combinations of gap open and gap extend penalties using the SWA. In Chapter 5 the approach employed a fixed combination of gap open penalty (i.e. 10) and gap extend penalty (i.e. 1). The approaches developed in this section will be demonstrated using three different polymorphic malware and their known (P_k) and unknown (P_x) variants. Part-III (research sub-question – Q2c) will adopt SWA with six different substitution matrices. In this part, the process of the first pairwise sequence alignment will be conducted for 71 different pairwise alignments with 71 different substitution matrices. The research in this thesis reported in Chapter 4 conducted pairwise alignment using the ID substitution matrix solely. The work in this part will be demonstrated using one polymorphic malware and its known (P_k) variants.

Part-I: Comparing Needleman-Wunsch and Smith-Waterman Algorithms for Identifying Viral Polymorphic Malware Variants

In this set of experiments the focus is on the string matching algorithm and the NWA and SWA will be compared to see which performs best for the extraction of signatures for identifying polymorphic variants of viral malware. This part mainly focuses on conducting experiments using two different sequence alignment techniques, namely, global and local alignments, and are also the two commonly-used standard techniques of sequence alignment (Troy, et al., 2003). Global alignment aligns the entire sequence from beginning to end in order to find the best potential alignment between two or more sequences (Kumar & Filipski, 2007), whereas, local alignment finds the "local regions with highest level of similarities" between two or more sequences (Vijan & Mehra, 2011). The standard algorithm to perform a global alignment is NWA and local alignment is SWA (Zarka, Cordier, Egyed-Zsigmond, Lamontagne, & Mille, 2013; Ghayyur, Aleem, & Islam, 2018). Further, NWA and SWA are the only two standard dynamic programming algorithms of sequence alignment (Xia, 2007). Therefore, for these exact reasons, the current part only focuses on comparing NWA and SWA in the extraction of new syntactic viral signatures for their use in the detection of polymorphic virus variants.

5.1. Introduction

Initial work (Chapter 4) exploring string-based approaches for the automatic extraction of signatures for the detection of some or all new polymorphic variants was promising. However, that initial work was limited by a number of experimental features. One such aspect was that only one sequence alignment method was considered – SWA. The research question (in detail) investigated is "Do dynamic programming approaches (i.e. the NWA and SWA) in bioinformatics for conducting task such as sequence alignment produce consensuses that not only 'fit' the known (P_k) variants (training set) but also generalise well to unknown (P_x) variants (test set)" (Q2a). The goal of the research addressed in this part of the chapter is to explore the effects of using NWA and SWA (both refined by dynamic programming) in string-based algorithms for the automatic extraction of syntactic signatures in order to detect all new or some malicious variants of polymorphic virus families.

The aim of this thread of research is to analyse if modern string matching methods, such as the NWA and SWA, can lead to innovative syntactic techniques for the automatic extraction of syntactic virus signatures not only for known (P_k) malicious variants

belonging to polymorphic virus families but also for unknown (new/future) viral variants (P_x). As stated earlier, the initial research (Chapter 4, also published in Naidu and Narayanan (2016)) was largely focused on the detection of polymorphic malware variants using the SWA (i.e. by performing local pairwise sequence alignment), but the current aim is to compare the two different dynamic programming approaches i.e. the NWA and SWA for the identification of polymorphic malware variants, by adopting both pairwise and multiple sequence alignments. The JS.Cassandra and the W32.Kitti viruses are employed along with their known (P_k) and unknown (P_x) polymorphic viral variants for experimental purposes. This part of the current chapter only focuses on these two polymorphic virus families due to sequence length restrictions (Kim & Pramanik, 1994; Yu, Bundschuh, & Hwa, 2002; Chakraborty & Bandyopadhyay, 2013) placed by the alignment tool (more details on page no. 18) adopting NWA (global alignment). The third polymorphic virus family was not considered in this part of the chapter due to its overly long sequences.

5.2. Comparing NWA and SWA for the Detection of Polymorphic Malware Variants Method: An Overview

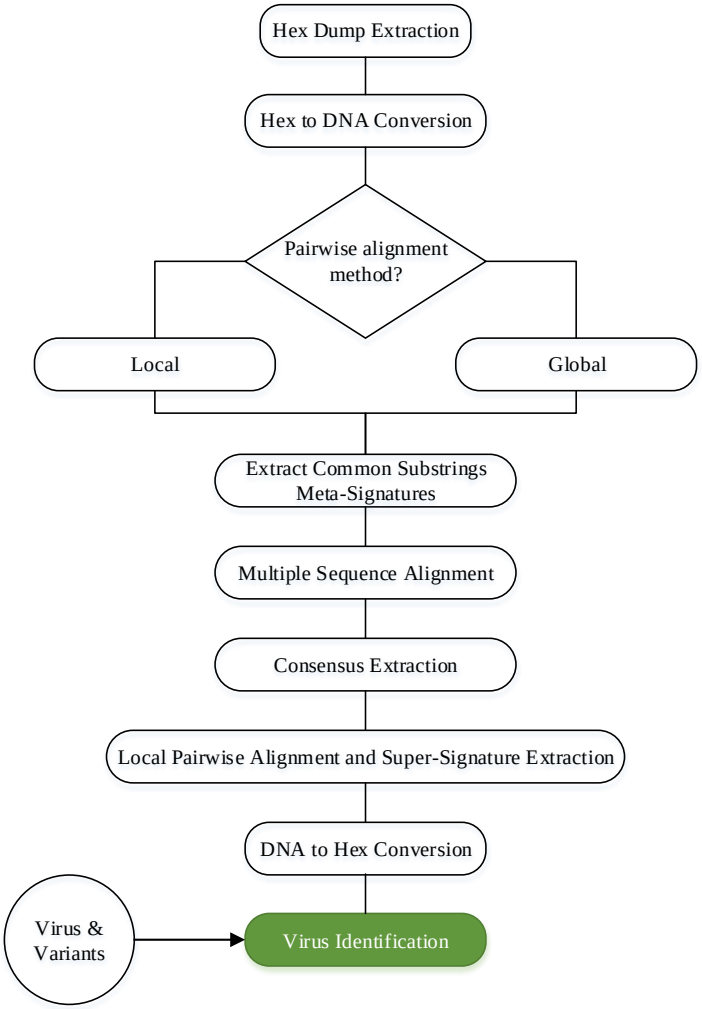

Figure 5.1: Eight-step method for comparing the Identification of Polymorphic Malware Variants by NWA and SWA.

The method in this part consists of eight steps (see Figure 5.1). As for the earlier research reported in this thesis, obtaining families of polymorphic viruses along with its known (P_k) malicious variants, generation of unknown (P_x) malware variants together with extraction of hex (hexadecimal) dump as well as the process of testing, was carried out on an isolated system in order to avoid any other unexpected system infections. Services of network connectivity is only adopted during the testing process as indicated in the previous chapter.

The method in this part employs two polymorphic malware – JS.Cassandra virus and W32.Kitti virus.

5.3. Comparing NWA and SWA for the Identification of Polymorphic Malware Variants Method: Systems and Methods

A detailed outline of the method is supplied below; a complete description follows the method.

1. **Step-1:** Extract hexadecimal dumps from polymorphic malware and its variants of the same family using 'sigtool' (ClamAV, 2018) – this step belongs to Stage I as presented in Figure 3.2.
2. **Step-2:** Convert the extracted hexadecimal dump sequences into a binary form and subsequently into DNA bases – this step belongs to Stage II as presented in Figure 3.2.
3. **Step-3:** Perform pairwise sequence alignments (global – NWA and local - SWA) using 'JAligner' (Moustafa, 2010) between the two converted polymorphic DNA sequences obtained in Step-2 – this step belongs to Stage III as presented in Figure 3.2.
4. **Step-4:** Extract common substrings (meta-signatures) after the process of first pairwise (global and local) sequence alignment conducted in Step-3 – this step belongs to Stage III as presented in Figure 3.2.
5. **Step-5:** Perform multiple sequence alignment using T-Coffee (Notredame, Higgins, & Heringa, 2000) on the extracted meta-signatures obtained in Step-4 – this step belongs to Stage III as presented in Figure 3.2.
6. **Step-6:** Extract consensuses after the process of multiple sequence alignment conducted in Step-5 – this step belongs to Stage III as presented in Figure 3.2.
7. **Step-7:** Perform pairwise (local - SWA) sequence alignments using 'JAligner' (Moustafa, 2010) between the extracted consensuses (retrieved in Step-6) and the polymorphic DNA sequences (retrieved in Step-2) belonging to the polymorphic malware (and its variants) of the same family one by one. This step will give us the common substrings (super-signatures) which will be employed to detect the polymorphic malware and all its known (P_k) and unknown (P_x) variants of that particular family – this step belongs to Stage III as presented in Figure 3.2.
8. **Step-8:** Convert the meta-signatures and super-signatures from their current representations in DNA format into the hexadecimal sequence format and then test the converted hex meta-signatures and hex super-signatures against the groups of polymorphic viruses along with its known (P_k) and unknown (P_x) malicious variants

belonging to the same family using ClamAV (ClamAV, 2018) – this step belongs to Stage IV as presented in Figure 3.2.

5.3.1 Hex Dump Extraction

Step-1: In this experiment, one variant v_000.js was selected from the 351 known (P_k) variants of JS.Cassandra virus along with the original (P_s) JS.Cassandra virus. Variant v_000.js and original (P_s) JS.Cassandra virus were selected as previous work reported in Chapter 4 showed that the same common substring (meta-signature) being found in all the alignments between the first 43 variants.

Table 5.1: File Identification Information for JS.Cassandra and its "v_000.js" variant.

Filename	File Identification Information	
JS.Cassandra.js (Original (P_s) Virus)	MD5	dc5f0d63fee16e7897aa47cce9b098f4
	SHA1	e2d9e58a81cbe19d0ccc2ae6961f5c6053fd0a9c
	SHA256	256b7d22a3475e70e9ca443f9c7357b496bac38db54244b1af0b6a2dc3e1a962
	ssdeep	192:s1c9NzNywp1N/zUQzVFsUlE4BcFEOsNpQS4oL/P08/8HnjOfRN+tMBG5k:McDNEEIE3NYoH8Hj+CeBGe
	File Size	7.6 KB (7767 bytes)
	File Type	Text
	Magic Literal	ASCII text, with very long lines, with CRLF line terminators
	TrID	Unknown!
v_000.js (Variant 1 – P_k)	MD5	994ee689fcc1c13fe100d909a4a17b3c
	SHA1	29d232725a388962d40d1a9cf5172fd733171cc9
	SHA256	3176a9e5b01ef9b3109d43bf93c090e2d1c28df4bd09cdec0be3fd61966d4e14
	ssdeep	96:ZrIhhqY/QQ2p0yazCBU93sW5StZ6luh7aZxgf1N33fSUJxPMyVwjp60PhVpYcfKu:Zr4Ypu7rvAh7qW1FrC6KhVCuKJs
	File Size	8.1 KB (8324 bytes)
	File Type	C++
	Magic Literal	ASCII C++ program text, with very long lines, with CRLF line terminators
	TrID	Unknown!

The aim of this chapter is also to find more than one meta-signature (syntactic viral signatures) using only the two files with the help of global and local alignments. As for previous experiments, the uniqueness of these two malicious programs was cross-verified by creating their file identification information using VirusTotal. File identification information was obtained from 'VirusTotal' (VirusTotal, 2018) and the results are given in Table 5.1.

Table 5.2: Analysis and Detection Ratio based on the 55 AVSs acquired from the 'VirusTotal' for the Two Malicious Files of JS.Cassandra Polymorphic Virus.

Antivirus	Js.Cassandra	Variant 2 (v_000.js)
ALYac	**Yes (JS.Cassandra.A)**	No
AVG	**Yes (JS/Cassa)**	**Yes (JS/Cassa)**
AVware	No	**Yes (Trojan.JS.Cassan.a (v))**
Ad-Aware	**Yes (JS.Cassandra.A)**	No
AegisLab	**Yes (JS.Cassa!c)**	**Yes (Script.Troj.Agent!c)**
Yandex	**Yes (JS.Crassan.A)**	No
AhnLab-V3	No	No
Alibaba	No	No
Antiy-AVL	**Yes (Virus/JS.Cassa)**	No
Arcabit	**Yes (JS.Cassandra.A)**	No
Avast	**Yes (JS:Cassa-D [Wrm])**	No
Avira (no cloud)	No	No
Baidu-International	No	No
Bitdefender	**Yes (JS.Cassandra.A)**	No
Bkav	No	No
ByteHero	No	**No**
CAT-QuickHeal	No	No
CMC	**Yes (Generic.Win32.dc5f0d63fe!MD)**	No
ClamAV	**Yes (JS.Cassandra.A)**	**Yes (JS.Cassa)**
Comodo	**Yes (Virus.JS.Cassa)**	**Yes (UnclassifiedMalware)**
Cyren	**Yes (JS/Cassan.A)**	**Yes (JS/Cassa.A.gen)**
DrWeb	**Yes (JS.Cassandra)**	No
ESET-NOD32	**Yes (JS/Cassa.A)**	**Yes (JS/Cassa.B)**
Emsisoft	**Yes (JS.Cassandra.A (B))**	No
F-Prot	**Yes (JS/Cassan.A)**	**Yes (JS/Cassa.A.gen)**
F-Secure	**Yes (JS.Cassandra.A)**	No
Fortinet	**Yes (JS/Cassa)**	**Yes (JS/Moat.CA237F0E!tr)**
GData	**Yes (JS.Cassandra.A)**	**Yes**
Ikarus	**Yes (Virus.JS.Cassa)**	**Yes (Trojan.JS.Cassa)**
Jiangmin	**Yes (Trojan/JSCassa.a)**	No
K7AntiVirus	No	**Yes (Exploit (04c561931))**
K7GW	No	**Yes (Exploit (04c561931))**
Kaspersky	**Yes (Virus.JS.Cassa)**	No
Malwarebytes	No	No
McAfee	**Yes (JS/Cassan)**	**Yes (JS/Cassan)**
McAfee-GW-Edition	**Yes (BehavesLike.JS.Downloader.zm)**	**Yes (JS/Cassan)**
eScan	**Yes (JS.Cassandra.A)**	No
Microsoft	**Yes (Trojan:JS/Cassa.A)**	**Yes (Trojan:JS/Cassa.B.gen)**
NANO-Antivirus	**Yes (Trojan.Script.Cassa.fvxo)**	No
Panda	**Yes (JS/Cassa.A)**	No
Qihoo-360	**Yes (Malware.Radar01.Gen)**	No
Rising	No	No
SUPERAntiSpyware	No	No
Sophos	**Yes (Mal/Generic-B)**	**Yes (JS/Cassan-A)**
Symantec	**Yes (JS.Casra)**	No
Tencent	**Yes (Js.Virus.Cassa.Wstz)**	**Yes (Js.Virus.Cassa.Htmc)**
TotalDefense	**Yes (JS/Cassa.A)**	No
TrendMicro	**Yes (JS_CASRA.A-O)**	No
TrendMicro-HouseCall	**Yes (JS_CASRA.A-O)**	No
VBA32	**Yes (Virus.JS.Cassa)**	No
VIPRE	No	**Yes (Trojan.JS.Cassan.a (v))**
ViRobot	No	No
Zillya	**Yes (Virus.Cassa.JS.1)**	No
Zoner	No	No
nProtect	**Yes (JS.Cassandra.A)**	No
Detection Ratio	**39/55**	**19/55**

Table 5.2 provides the results of attempting to detect these two variants using 55 commonly used AVSs. The original (P_s) JS.Cassandra virus was successfully detected by 39 of the AVSs while the v_000 variant was correctly identified by only 19 of the 35 AVSs.

Hex dumps are subsequently extracted followed by the AVS scan tests from the two malicious programs adopting the ClamAV (ClamAV, 2016) software tools openly accessible on their webpage, which utilises an application known as ‘sigtool’, in order to create hexadecimal dumps.

5.3.2 Hex to DNA Code Conversion

Step-2: In this step of code conversion, following the extraction process, the two extracted malicious hex dumps belonging to JS.Cassandra virus files were translated into DNA (nucleotide/nucleic acid) sequences adopting the DNA encoding method as shown in Section 4.3.2 (see page no. 66).

5.3.3 First Pairwise (Global and Local) Sequence Alignment and Meta-Signature Extraction

The string matching NWA and SWA were employed to extract the most common substrings/patterns from the two JS.Cassandra files. The NWA conducts sequence alignment globally amidst the two biologically represented strings in order to obtain the optimal matching segments, whereas, the process of SWA performs sequence alignment locally. For use in the NWA and the SWA, the two strings are normally represented either as proteins (chains of amino acids) or nucleotide sequences (DNA/RNA). Both the NWA and SWA identify the most matched substrings between the search string and pattern. Rather than identifying the complete sequence, the NWA and SWA extract the sections of all possible length, then compares and enhances the similarity rate. The NWA and SWA can verify for identical matches or substituted matches (i.e. a character in the string can be replaced by a different character, together with no character (gap), in the pattern, and vice versa). The SWA is assured to identify the optimal local alignment and the NWA to identify the optimal global alignment with reference to the scoring scheme being adopted (i.e. the gap scoring and the substitution strategy). There are several substitution matrices available and utilised by the NWA and SWA like BLOSUM, ID, and PAM matrix. However, fixed match/mismatch scoring strategy was adopted in these experiments to carry out exact matching. Match score was given a value of 2, whereas, mismatch score was given a value of -1, respectively. The outcomes of the NWA and SWA are known as ‘alignments’ since either or both strings can be altered with gap insertions to produce optimal pattern matches.

Step-3: In this step, pairwise (global and local) alignment was performed adopting the NWA and the SWA with a fixed match/mismatch scoring strategy between the DNA

sequence representations of JS.Cassandra and its v_000 variant using 'JAligner'. In total, two pairwise local alignments were performed in this step, one for NWA and one for SWA, respectively.

Step-4: After the phase of global and local alignment, all the feasible common substrings from the two pairwise alignments (NWA and SWA) were extracted, resulting in 37 substrings (meta-signatures), 16 from NWA and 21 from SWA. Table 5.3 presents the sequence lengths of all the 37 extracted meta-signatures in their DNA representation obtained in this step. The minimum and maximum sequence lengths of NWA meta-signatures extracted for JS.Cassandra virus were 30 and 104, respectively, with a mean (sum, median and standard deviation of 830, 36 and 28.25, respectively) of 51.875 for 16 meta-signatures in their DNA representation. The minimum and maximum sequence lengths of SWA meta-signatures obtained for JS.Cassandra virus were 29 and 104, respectively, with a mean (sum, median and standard deviation of 1139, 46 and 27.65, respectively) of 54.2381 for 21 meta-signatures in their DNA representation. The minimum and maximum sequence lengths of SWA meta-signatures obtained for W32.Kitti virus were 28 and 3736, respectively, with a mean (sum, median and standard deviation of 7331, 36.5 and 543.07, respectively) of 135.7593 for 54 meta-signatures in their DNA representation.

5.3.4 Multiple Sequence Alignment and Consensus Extraction

Step-5 (Multiple sequence alignment): In this step, a multiple sequence alignment was performed on the meta-signatures obtained in Step-4 using T-Coffee (Notredame, Higgins, & Heringa, 2000) with alignment again confined to the ID substitution matrix. This means that alignment is carried out through matching of nucleic acids in particular positions rather than the usual bioinformatics approach of using biologically informed mutation rates. Two separate multiple alignments were performed, one for the 16 meta-signatures obtained using the NWA and one for the 21 meta-signatures obtained from the SWA.

Table 5.3: Sequence Lengths for Meta-signatures extracted from DNA representations from JS.Cassandra and its v_000 variant, where, MS is the meta-signature.

NWA		SWA	
Meta-Signature	**Sequence Length**	**Meta-Signature**	**Sequence Length**
MS1	104	MS1	46
MS2	100	MS2	100
MS3	30	MS3	104
MS4	44	MS4	48
MS5	60	MS5	29
MS6	32	MS6	61
MS7	30	MS7	49
MS8	56	MS8	33
MS9	30	MS9	56
MS10	32	MS10	29
MS11	100	MS11	56
MS12	36	MS12	32
MS13	36	MS13	30
MS14	30	MS14	101
MS15	80	MS15	101
MS16	30	MS16	37
-	-	MS17	37
-	-	MS18	39
-	-	MS19	30
-	-	MS20	91
-	-	MS21	30

Step-6 (Consensus generation and extraction): In this step, the consensus was generated after the procedure of multiple sequence alignment, and the procedure was repeated two times, one for each dynamic programming algorithm (NWA and SWA). No threshold of common occurrence of a DNA character (nucleic acid) in a specific position was chosen in this step. Overall, two consensuses were extracted in this step.

5.3.5 Second Pairwise Local Sequence Alignment and Super-Signature Extraction

Step-7: In this step, a pairwise (local) sequence alignment was performed adopting the SWA with ID matrix amidst the consensus and the converted DNA sequence of the original virus employing 'JAligner'. SWA with the ID matrix was used as previous experiments reported in Chapter 4 gave a 100% accuracy for known (P_k) polymorphic malware variants. Altogether, two individual pairwise local alignments were conducted, one for each dynamic programming algorithm (NWA and SWA). A suitable gap open penalty of 10 and a gap extend penalty of 1 was adopted as previous experiments demonstrated in Chapter 4 gave a 100% detection rate for existing polymorphic variants adopting these penalties. Four super-signatures for JS.Cassandra were extracted in this step, two arising from the NWA and two from the SWA approach.

5.3.6 DNA to Hex Conversion as well as Meta-Signature and Super-Signature Testing

Step-8: In this last step, the 37 meta-signatures (retrieved in Step-4) and four super-signatures (obtained in Step-7) in their DNA format were converted into hexadecimal format. The converted hex meta-signatures and hex super-signatures were scanned against JS.Cassandra and all known (P_k) variants using clamscan.

5.4. Experimental Results

Table 5.4 gives the detection rates (with accuracy) for the detection of JS.Cassandra polymorphic malware and its known (P_k) variants employing 'clamscan' by testing the 37 meta-signatures extracted in Step-4.

Table 5.4: Test Statistics for the Detection of JS.Cassandra Polymorphic Malware and its known (352) variants (P_k) employing 'clamscan' by testing the 37 Meta-Signatures acquired in Step-4 from NWA and SWA.

Dynamic Programming Algorithm	Meta-Signature	Detection Rate (Accuracy)	Sensitivity /Recall	Specificity	Precision	F1 Score
NWA	MS1	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS2	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS3	**352/352 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
	MS4	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS5	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS6	0/352 (0.00%)	0.0%	0.0%	100%	0.0%
	MS7	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS8	196/352 (55.68%)	55.68%	0.0%	100%	71.53%
	MS9	0/352 (0.00%)	0.0%	0.0%	100%	0.0%
	MS10	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS11	0/352 (0.00%)	0.0%	0.0%	100%	0.0%
	MS12	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS13	0/352 (0.00%)	0.0%	0.0%	100%	0.0%
	MS14	17/352 (4.83%)	4.83%	0.0%	100%	9.21%
	MS15	**352/352 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
	MS16	52/352 (14.77%)	14.77%	0.0%	100%	25.74%
SWA	MS1	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS2	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS3	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS4	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS5	0/352 (0.00%)	0.0%	0.0%	100%	0.0%
	MS6	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS7	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS8	0/352 (0.00%)	0.0%	0.0%	100%	0.0%
	MS9	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS10	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS11	196/352 (55.68%)	55.68%	0.0%	100%	71.53%
	MS12	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS13	**352/352 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
	MS14	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS15	0/352 (0.00%)	0.0%	0.0%	100%	0.0%
	MS16	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100%**	**98.26%**
	MS17	0/352 (0.00%)	0.0%	0.0%	100%	0.0%
	MS18	22/352 (6.25%)	6.25%	0.0%	100%	11.76%
	MS19	32/352 (9.09%)	9.09%	0.0%	100%	16.66%
	MS20	84/352 (23.86%)	23.86%	0.0%	100%	38.53%
	MS21	52/352 (14.77%)	14.77%	0.0%	100%	25.74%

* Where MS is the meta-signature.

Most of the meta-signatures extracted using the NWA identified the variants as well as the original (P_s) JS.Cassandra virus (Table 5.4). Of the NWA meta-signatures six had an accuracy of 96.57% and two signatures resulted in 100% accuracy. Four of the NWA generated meta-signatures failed to detect the virus and its variants. Among the meta-signatures extracted using SWA four failed to detect any of the variants, including the original (P_s) virus, and only one meta-signature detected 100% viral files tested. Eleven SWA meta-signatures gave a detection rate of 96.57%.

The four super-signatures acquired from Step-7 were tested in the same way that the meta-signatures were tested. The results (see Figure 5.2) indicate that using either of the NWA super-signatures, the JS.Cassandra variants along with the original (P_s) virus, were successfully identified as infected by clamscan with an accuracy of 96.59% in 0.285 seconds and 0.313 seconds. The first SWA super-signature gave the same detection accuracy in 0.297 seconds. None of the viral variants were detected as infected using the second SWA super-signature.

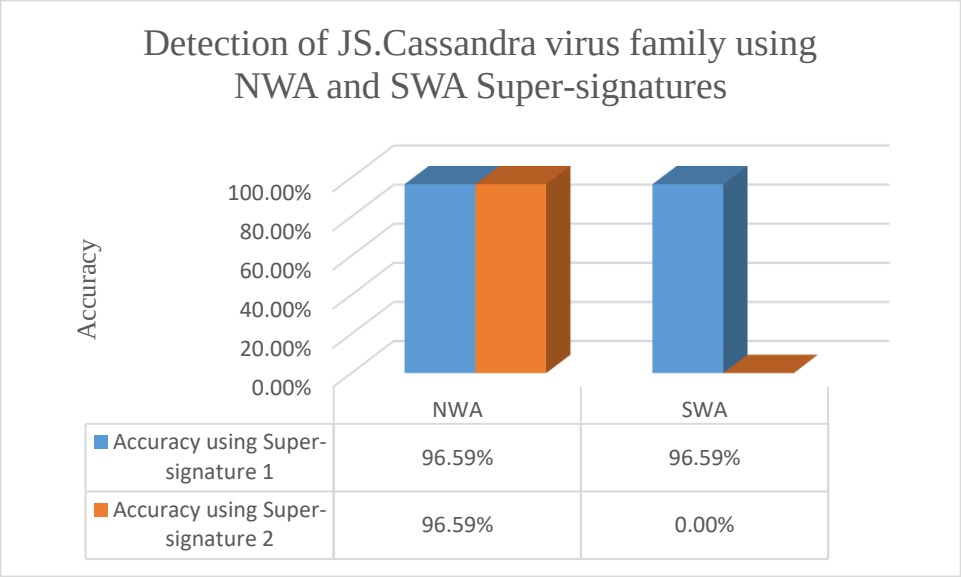

Figure 5.2: Bar graph demonstrating the detection results of JS.Cassandra virus family using the super-signatures of NWA and SWA.

Table 5.5: Identification of W32.Kitti and its 1,105 unknown (P_x) variants using the 54 meta-signatures extracted using the SWA, where, MS is the meta-signature.

Meta-Signature	Detection Rate (Accuracy)	Sensitivity /Recall	Specificity	Precision	F1 Score
MS1	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS2	438/1106 (39.6%)	39.6%	0.0%	100%	56.73%
MS3	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS4	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS5	572/1106 (51.72%)	51.72%	0.0%	100%	68.2%
MS6	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS7	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS8	0/1106 (0.00%)	0.0%	0.0%	100%	0.0%
MS9	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS10	0/1106 (0.00%)	0.0%	0.0%	100%	0.0%
MS11	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS12	0/1106 (0.00%)	0.0%	0.0%	100%	0.0%
MS13	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS14	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS15	0/1106 (0.00%)	0.0%	0.0%	100%	0.0%
MS16	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS17	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS18	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS19	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS20	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS21	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS22	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS23	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS24	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS25	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS26	972/1106 (87.88%)	87.88%	0.0%	100%	93.55%
MS27	0/1106 (0.00%)	0.0%	0.0%	100%	0.0%
MS28	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS29	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS30	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS31	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS32	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS33	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS34	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS35	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS36	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS37	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS38	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS39	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS40	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS41	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS42	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS43	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS44	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS45	0/1106 (0.00%)	0.0%	0.0%	100%	0.0%
MS46	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS47	0/1106 (0.00%)	0.0%	0.0%	100%	0.0%
MS48	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS49	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS50	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS51	0/1106 (0.00%)	0.0%	0.0%	100%	0.0%
MS52	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS53	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**
MS54	**1106/1106 (100.00%)**	**100%**	**0.0%**	**100%**	**100%**

Step-1 to Step-8 were repeated for the W32.Kitti virus. Fifty-four meta-signatures were extracted in DNA format using SWA. One multiple alignment was performed in Step-5**,** and a consensus of sequence length 5321 was extracted in Step-6. Two super-signatures/common substrings of sequence lengths 137 and 3472 were obtained in Step-7 from SWA in their DNA format. In the last step (i.e. Step-8), the 54 meta-signatures and two super-signatures were converted to the hexadecimal format and tested using 'clamscan'.

Table 5.5 provides the results of the detection rates (with accuracy and sequence lengths of the DNA and Hex representations) for the identification of W32.Kitti and its unknown (P_x) variants using 'clamscan' by testing the 54 SWA generated meta-signatures. Most of the meta-signatures identified the unknown (P_x) variants, and the original (P_s) polymorphic malware, with the exception of eight meta-signatures that failed to identify any of the files as infected. These meta-signatures are called "unknown signatures" in this research and the exact reason for its failure to detect the virus variants remains unknown. Further work is required to determine the cause of this and is not considered in this research. Although it may be the fact that those meta-signatures capture variants that either belong to a small set of special variants that were generated using a different obfuscation method or are the ones that have not yet encountered (unknown – P_x).

Figures 5.3 – 5.4 displays the clamscan results for the two SWA super-signatures. Figure 5.3 shows that 100% of the W32.Kitti viral variants together with the original (P_s) virus, were successfully identified as infected by 'clamscan' using the first SWA super-signature, in 17.249 secs. None of 1106 W32.Kitti viral variants (along with the original (P_s) virus) were successfully identified as infected using the second SWA super-signature (Figure 5.4).

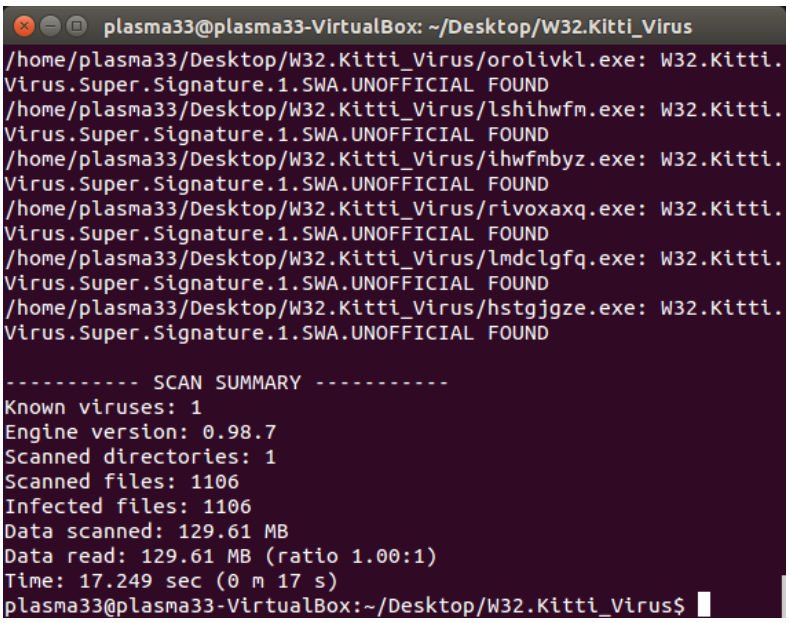

Figure 5.3: Screenshot of the clamscan result for the first SWA super-signature for the W32.Kitti virus.

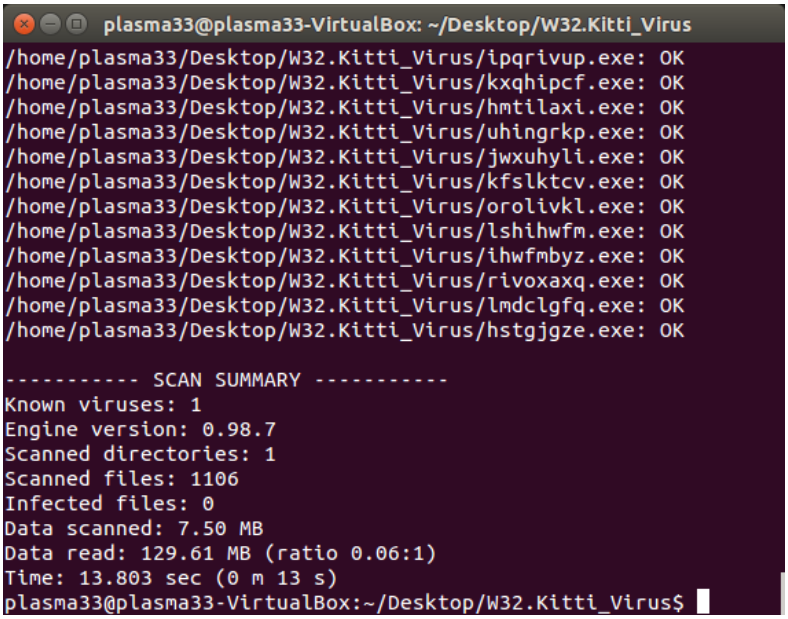

Figure 5.4: Screenshot of the clamscan result for the second SWA super-signature for the W32.Kitti virus.

5.5. Summary

In this chapter, the use of advanced sequence alignment techniques was explored. A syntactic structure approach was taken – conducting sequence alignments with the help of a fixed match/mismatch scoring scheme to automate signature extraction using both the NWA and the SWA. Interestingly, both the NWA and the SWA identified the same meta-signature, MS3 (NWA) and MS13 (SWA) that captured all the known (P_k) variants of JS.Cassandra virus.

It was anticipated that shorter signatures would identify more variants than the longer signatures as the chances of finding a short sequence repeated is likely to be higher than for longer sequences. But in this case, for JS.Cassandra virus, the MS3 NWA meta-signature with a sequence length of 30 and the MS15 NWA meta-signature with a sequence length of 80 both gave 100% accuracies. Furthermore, MS1, MS2 NWA meta-signatures and MS2, MS3, and MS14 SWA meta-signatures with lengths of under 105 resulted in malware detection accuracies of over 96%. The shortest variant file has a length of 15,534 hexadecimal characters, and the longest of the most effective signatures is 105 hexadecimal characters long or 0.7% of the shortest variant file. Thus, the signatures are remarkably small relative to the variant files in general and support the theory that shorter files are more likely to detect the variants. However, if the signatures are too short, it is likely that all files might contain the pattern or sequence defined by the signature. It was for this reason that random files were tested in the experiments in Chapter 4 to ensure that the signatures were specific enough only to detect the malware variants. Similar observations were also seen for W32.Kitti virus. Because there is not a significant variation in the lengths of the meta-signatures when compared to the lengths of the variant files, it is unlikely that the length of the meta-signatures generated using this method is an important factor in detection.

Meta-signatures: In total, 37 new syntactic viral signatures (i.e. meta-signatures) were extracted from this research, 16 from NWA and 21 from SWA. Based on the detection rates for JS.Cassandra variants, 25% NWA meta-signatures, and 19% SWA meta-signatures identified none of the known (P_k) variants, respectively. Although, 50% NWA meta-signatures and 57.14% SWA meta-signatures identified over 96% of the known (P_k) variants, respectively. This shows that the SWA meta-signatures are more effective than the NWA meta-signatures indicating that the SWA-based local syntactic viral signatures perform better than the NWA-based global syntactic viral signatures. **Super-signatures:** Based on the detection rates of JS.Cassandra variants, two of the two NWA super-signatures and one of the two SWA super-signatures identified over 96% of the known (P_k) variants, respectively. Also, one of the two super-signatures identified 100% of the unknown (P_x) variants of W32.Kitti virus. It is difficult to say whether the NWA or the SWA super-signatures are better as SWA super-signature identified all the unknown (P_x) variants (test set) of W32.Kitti virus. Moreover, it cannot be said whether meta-signatures or super-signatures are better, as meta-signatures tend to work on some or all the known (P_k) set but would not be completely effective against the unknown (never seen before)

(P_x) set. Super-signatures, on the other hand, would be completely (or in some cases, comparatively) effective against the unknown (P_x) set (see Figure 5.3).

The next part of this chapters investigates whether further refined string searching algorithms, such as the SWA with different combinations of gap open and gap extend penalties, can lead to syntactic techniques to the automatic extraction of polymorphic syntactic viral signatures. Experiments are reported that are designed to address the second part of the second research question (Q2b). More information will follow in the succeeding part.

Part-II: The Effects of Gap Open and Gap Extend Penalties in a String-Based Approach for Detecting Polymorphic Malware Variants

The experiments reported so far have used a fixed gap open and a fixed gap extend penalty. The next set of experiments was designed to assess how different gap open and gap extend penalty combinations affect the outcome of the virus identification process: whether or not optimal penalties can be established, whether or not these optimal penalties can be applied in general to different polymorphic viruses; and if the same penalties can be used for detecting unknown (P_x) variants. These experiments were designed to answer research question **Q2b** in more detail: "Do gap open and extend facilities produce consensuses that not only 'fit' the known (P_k) variants (training set) but also generalise well to unknown (P_x) variants (test set)".

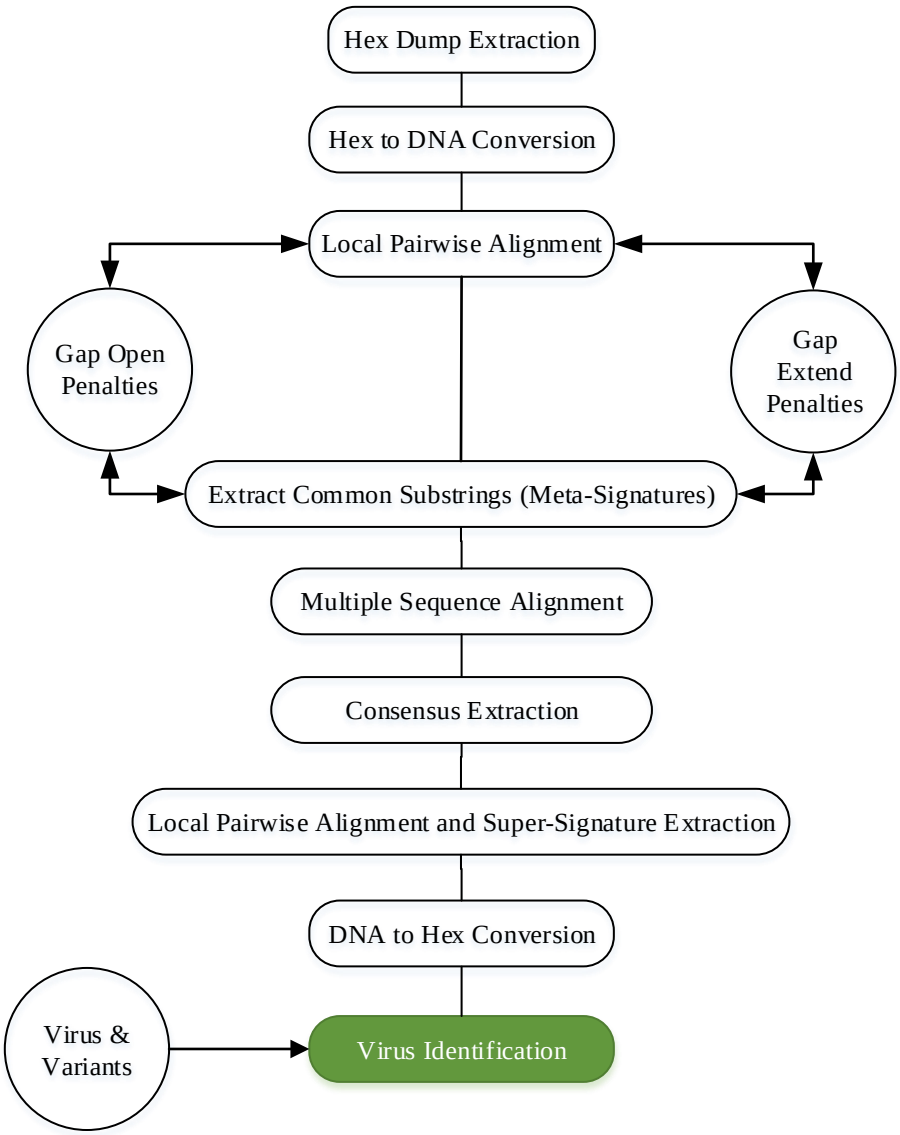

Figure 5.5: Eight-step method for the Effects of Gap Penalties in a String-Based Approach for Detecting Polymorphic Malware Variants.

5.6. Effects of Gap Penalties in a String-Based Approach for Detecting Polymorphic Malware Variants Method: An Overview

This method is comprised of eight steps (see Figure 5.5). The effects of gap open and gap extend penalties adopting SWA will be explored in relation to the detection of three different polymorphic viruses (JS.Cassandra, W32.Kitti, and W32.CTX/W32.Cholera) and their known (P_k) and unknown (potential future – P_x) variants. All the three polymorphic virus families are used in this part of the research as there is no sequence length restrictions (Kim & Pramanik, 1994; Yu, Bundschuh, & Hwa, 2002; Chakraborty & Bandyopadhyay, 2013) placed by the alignment tool (more details on page no. 18) adopting SWA with different combinations of gap penalties.

5.7. Effects of Gap Penalties in a String-Based Approach for Detecting Polymorphic Malware Variants Method: Systems and Methods

Steps 1-8 are the same as those used previously in this research except for Step-3 in which the different combinations of gap penalties are introduced (See Section 5.3 – page no. 79).

5.7.1 Hex Dump Extraction

Step-1: As for previous experiments (Chapter 4 and Chapter 5 – Part-I): six variants (the original (P_s) plus five known (P_k) variants for each of the viruses tested) were chosen for testing.

Table 5.6: Generated CRC32b hash values and file sizes for the 18 malicious files.

Polymorphic Malware	Filename	CRC32b Hash Value	File Size (bytes)
JS.Cassandra	JS.Cassandra.js (original – P_s)	26489347	7,767
	v_000.js (P_k)	848562f1	8,324
	v_002.js (P_k)	7c4ea313	9,938
	v_003.js (P_k)	bd3b9fdc	8,759
	v_004.js (P_k)	9904ef9c	8,392
	v_005js (P_k)	511621c7	9,400
W32.CTX/W32.Cholera	W32.CTX.Cholera.Virus.10853.exe (original – P_s)	c99df9b3	45,147
	actmovie.exe (P_x)	26a6ed27	13,837
	cisvc.exe (P_x)	c6f2560a	15,453
	dcomcnfg.exe (P_x)	2387735d	16,968
	forcedos.exe (P_x)	1231e17d	17,473
	MRT.exe (P_x)	6f66d56d	17,473
W32.Kitti	OC.exe (original – P_s)	a6d1f306	124,416
	absdmfcj.exe (P_x)	33867b60	124,416
	adehsjud.exe (P_x)	3106fb13	124,416
	crilunah.exe (P_x)	8d9b920b	124,416
	nafybgho.exe (P_x)	dedbcdce	124,416
	nalgjahg.exe (P_x)	19e70a50	124,416

As for the previous experiments in this thesis the 18 files were checked for uniqueness by generating CRC32b hash value for each variant (Table 5.6). The 18 files were also checked using 'VirusTotal' to confirm that malicious functionality was preserved (Table 5.7).

Table 5.7 provides the detection rate for each of the 18 variants (Table 5.6) of 55 common AVS products. Only 53.69% of the AVSs successfully detected the 15 malicious variants and 73.3% of the three original polymorphic viruses.

Table 5.7: Detection Ratio Based on the 55 State-of-the-Art AVS Products obtained from the 'VirusTotal' Website for the 18 Malicious Variants.

Polymorphic Malware 1	**Filename**	**Detection Ratio**
JS.Cassandra Virus	JS.Cassandra.js (Original Virus – P_s)	39/55
	v_000.js (Variant 1 – P_k)	19/55
	v_002.js (Variant 2 – P_k)	21/55
	v_003.js (Variant 3 – P_k)	15/55
	v_004.js (Variant 4 – P_k)	17/55
	v_005.js (Variant 5 – P_k)	17/55
Polymorphic Malware 2	**Filename**	**Detection Ratio**
W32.CTX/W32.Cholera Virus	W32.CTX.Cholera.Virus.10853 (Original Virus – P_s)	38/55
	actmovie.exe (Variant 1 – P_x)	41/55
	cisvc.exe (Variant 2 – P_x)	42/55
	dcomcnfg.exe (Variant 3 – P_x)	37/55
	forcedos.exe (Variant 4 – P_x)	39/55
	MRT.exe (Variant 5 – P_x)	39/55
Polymorphic Malware 3	**Filename**	**Detection Ratio**
W32.Kitti Virus	OC.exe (Original Virus – P_s)	44/55
	absdmfcj.exe (Variant 1 – P_x)	41/55
	adehsjud.exe (Variant 2 – P_x)	41/55
	crilunah.exe (Variant 3 – P_x)	44/55
	nafybgho.exe (Variant 4 – P_x)	12/55
	nalgjahg.exe (Variant 5 – P_x)	18/55

Hex dumps were then extracted from the 18 malicious variants using 'sigtool' ready for conversion to DNA (nucleotide) and protein (amino acid) representation.

5.7.2 Hex to DNA and Amino Acid Conversion

Step-2: In this step, the extracted hex dump sequences were converted into DNA and amino acid sequences. Two different representational methods (i.e. DNA and amino acid) were used to investigate the effectiveness of the string-based approach. In the case of DNA, the files were converted using the DNA representational method shown in Section 5.3.2 (see page no. 82).

Conversion of hexadecimal into amino acid sequences for input to JAligner was carried out adopting the rules shown in Table 5.8. A short example of the conversion of 16-bit hexadecimal code into 16 amino acid characters is shown below:

4d5a800001000000 (16-bit hexadecimal code)

KDLAQGGGGHGGGGGG (16 amino acid characters)

Table 5.8: Rules for converting hexadecimal into amino acid characters.

Hexadecimal	Amino Acid	Hexadecimal	Amino Acid
0	G	8	Q
1	H	9	P
2	I	a	A
3	R	b	B
4	K	c	C
5	L	d	D
6	M	e	E
7	N	f	F

The six extracted hex dumps for the JS.Cassandra malicious variants were converted into DNA sequences. The remaining 12 extracted hex dumps for the W32.CTX and W32.Kitti malicious variants were converted into amino acid sequences.

5.7.3 First Pairwise Local Sequence Alignment and Meta-Signature Extraction

Step-3: In this step, a pairwise (local) alignment was performed adopting the SWA with an ID substitution matrix using JAligner.

Ten different combinations of gap open and gap extend penalties were used to conduct the pairwise local alignments. The gap open penalty is the penalty for opening a gap in the alignment, whereas gap extend penalty is the penalty for extending a gap by one residue (Clustal, 2012). In this case, six variants, i.e. V1, V2, V3, V4, V5 and V6 (where V1 is the original (P_s) virus and V2-V6 are its polymorphic variants). So between V1 and V2, ten different combinations of gap open and gap extend penalties were applied, which then led to ten different pairwise local alignments. The same procedure was applied to the remaining four pairs i.e. on V2 and V3, V3 and V4, V4 and V5, and V5 and V6, respectively. In total, 150 pairwise local alignments were carried out in this step, 50 for each of the three polymorphic malware. Other combinations such as V1 and V3, V1 and V4, etc. were not considered in this step as the purpose of this research is to examine the feasibility of adding more sophisticated search facilities. In the case of the W32.Kitti

virus, only the first 46,000 amino acid characters were aligned due to longer lengths of the amino acid sequences belonging to its six variants. In the case of amino acid sequences, JAligner allows pairwise alignment of two sequences to a maximum combined sequence length of 92,000 amino acids. To cope with such long sequences JAligner requires an initial memory allocation of 13,312 MB and a maximum heap memory allocation of 15,360 MB to be assigned to JAligner. And in the case of nucleotide sequences (DNA), JAligner (Moustafa, 2010) allows pairwise alignment of two larger sequences to a maximum combined sequence length of 225,500.

Step-4: Common substrings, or meta-signatures, from the pairwise local alignments which had the highest percentage of identities and similarities, were extracted. A threshold of greater than or equal to 85% was applied to extract twelve common substrings from the 23 pairwise local alignments for JS.Cassandra virus, 17 from the ten pairwise local alignments for the W32.CTX/W32.Cholera virus and 30 from the 25 pairwise local alignments for the W32.Kitti virus. These common substrings are the meta-signatures (i.e. syntactic viral signatures) that were used to identify all the known (P_k) polymorphic variants of each virus family. Detection was carried out using the same process as detailed in Part-I.

The longest JS.Cassandra DNA meta-signature contained 397 bases and the shortest 14 amino acids. The mean sequence length for the twelve JS.Cassandra meta-signatures was 152, the median 131.5 and they have a standard deviation of 108.9. For W32.CTX virus the longest and shortest amino acid meta-signatures were 1069 and 30 amino acids long, respectively. These 17 meta-signatures in amino acid representation have a median of 276, a mean of 436 and a standard deviation of 397.7. The minimum and maximum sequence lengths of the W32.Kitti amino acid meta-signatures were 790 and 1868, respectively. These meta-signatures have a mean length of 1689 amino acids, a median of 1868 and a standard deviation of 407.706.

5.7.4 Multiple Sequence Alignment and Consensus Extraction

Step-5 (Multiple sequence alignment): Once again, multiple alignment was performed using T-Coffee with alignment constrained to the ID matrix.

Step-6 (Consensus generation and extraction): As per previous experiments, see Step-6 in Section 5.3.4 (page no. 83), three consensuses were extracted using T-Coffee. One for each of the three virus families tested.

5.7.5 Second Pairwise Local Sequence Alignment and Super-Signature Extraction

Step-7: Pairwise (local) alignment amidst the consensus and the sequence of the original (P_s) virus was performed adopting the SWA with an ID matrix using JAligner and the optimal penalty gaps (of gap open 10 and gap extend one) were chosen. As stated earlier, these optimal penalty gaps were chosen as previous experiments demonstrated in Chapter 4 gave a 100% accuracy for existing polymorphic variants adopting these penalties. There should be a single common substring, a super-signature, for each virus and its variants. If the first common substring is not a super-signature, this step is repeated with the sequence of a variant of the same family. If this does not result in a super-signature, then this step is repeated using another variant, and so on. A JS.Cassandra super-signature was obtained in two iterations from variant 1. For W32.CTX/W32.Cholera and W32.Kitti viruses the super-signature was extracted from the original (P_s) virus in a single iteration.

In total, three JS.Cassandra super-signatures, three W32.CTX/Cholera super-signatures and five W32.Kitti super-signatures were extracted.

5.7.6 DNA and Amino Acid to Hex Conversion as well as Meta-Signature and Super-Signature Testing

Step-8: In this last step, the extracted meta-signatures and super-signatures were converted back to hexadecimal format. Each of these signatures was tested against the relevant viruses and their known (P_k) and unknown (P_x) variants using clamscan.

5.8. Experimental Results

Table 5.9 provides the results of the pairwise local alignments that were performed in Step-3. Only the desired pairwise local alignment results with the highest percentage of identities and similarities, and that produced effective meta-signatures are shown in Table 5.9. Full results are presented in Appendix G (see Table G.1 – page no. 247).

The percentages of identities and similarities for JS.Cassandra were higher than 85%, indicating that there were higher percentages of the amino acid or DNA (malicious) residues conserved in the biologically represented sequences.

Table 5.9: Results of the pairwise local alignments that were performed in Step-3.

Polymorphic Malware	Pairwise Alignment	Gap Open Penalty	Gap Extend Penalty	Identity Percentage	Similarity Percentage	Gaps Percentage	Alignment Length	Alignment Score
JS.Cassandra Virus	Original and Variant 1	20	1	98.51%	98.51%	1.49%	269	242.00
		25	0.5	98.51%	98.51%	1.49%	269	238.50
		25	1	98.51%	98.51%	1.49%	269	237.00
	Variant 1 and Variant 2	15	1	89.30%	89.30%	10.70%	430	310.00
		20	1	89.30%	89.30%	10.70%	430	300.00
		25	0.5	89.30%	89.30%	10.70%	430	312.00
		25	1	89.30%	89.30%	10.70%	430	290.00
	Variant 2 and Variant 3	15	1	85.33%	85.33%	14.67%	450	262.00
		20	1	85.33%	85.33%	14.67%	450	242.00
		25	0.5	85.33%	85.33%	14.67%	450	253.00
		25	1	85.33%	85.33%	14.67%	450	222.00
	Variant 3 and Variant 4	10	1	95.22%	95.22%	4.78%	418	360.00
		15	0.5	95.22%	95.22%	4.78%	418	359.00
		15	1	95.22%	95.22%	4.78%	418	350.00
		20	0.5	95.22%	95.22%	4.78%	418	349.00
		20	1	95.22%	95.22%	4.78%	418	340.00
		25	0.5	95.22%	95.22%	4.78%	418	339.00
		25	1	95.22%	95.22%	4.78%	418	330.00
	Variant 4 and Variant 5	10	1	100.00%	100.00%	0.00%	397	397.00
		15	1	100.00%	100.00%	0.00%	397	397.00
		20	1	100.00%	100.00%	0.00%	397	397.00
		25	0.5	100.00%	100.00%	0.00%	397	397.00
		25	1	100.00%	100.00%	0.00%	397	397.00
W32.CTX/W32.Cholera Virus	Original and Variant 1	25	1	99.29%	99.29%	0.71%	1553	1507.00
	Variant 1 and Variant 2	5	1	96.15%	96.15%	3.85%	2309	2015.00
	Variant 2 and Variant 3	10	1	96.41%	96.41%	3.59%	2060	1804.00
	Variant 3 and Variant 4	5	1	94.40%	94.40%	5.60%	2017	1707.00
	Variant 4 and Variant 5	10	1	100.00%	100.00%	0.00%	736	736.00
		15	1	100.00%	100.00%	0.00%	736	736.00
		20	0.5	100.00%	100.00%	0.00%	736	736.00
		20	1	100.00%	100.00%	0.00%	736	736.00
		25	0.5	100.00%	100.00%	0.00%	736	736.00
		25	1	100.00%	100.00%	0.00%	736	736.00

Polymorphic Malware	Pairwise Alignment	Gap Open Penalty	Gap Extend Penalty	Identity Percentage	Similarity Percentage	Gaps Percentage	Alignment Length	Alignment Score
W32.Kitti Virus	Original and Variant 1	5	1	86.35%	86.35%	13.65%	3297	2061.00
		10	1	100.00%	100.00%	0.00%	1868	1868.00
		15	1	100.00%	100.00%	0.00%	1868	1868.00
		20	1	100.00%	100.00%	0.00%	1868	1868.00
		25	1	100.00%	100.00%	0.00%	1868	1868.00
	Variant 1 and Variant 2	10	1	100.00%	100.00%	0.00%	1868	1868.00
		15	1	100.00%	100.00%	0.00%	1868	1868.00
		20	1	100.00%	100.00%	0.00%	1868	1868.00
		25	1	100.00%	100.00%	0.00%	1868	1868.00
	Variant 2 and Variant 3	5	1	88.12%	88.12%	11.88%	3266	2130.00
		10	1	100.00%	100.00%	0.00%	1868	1868.00
		15	1	100.00%	100.00%	0.00%	1868	1868.00
		20	1	100.00%	100.00%	0.00%	1868	1868.00
		25	1	100.00%	100.00%	0.00%	1868	1868.00
	Variant 3 and Variant 4	5	1	88.18%	88.18%	11.82%	3265	2129.00
		10	1	100.00%	100.00%	0.00%	1868	1868.00
		15	1	100.00%	100.00%	0.00%	1868	1868.00
		20	1	100.00%	100.00%	0.00%	1868	1868.00
		25	1	100.00%	100.00%	0.00%	1868	1868.00
	Variant 4 and Variant 5	5	0.5	87.03%	87.03%	12.97%	3285	2349.00
		5	1	90.51%	90.51%	9.49%	3225	2217.00
		10	1	100.00%	100.00%	0.00%	1868	1868.00
		15	1	100.00%	100.00%	0.00%	1868	1868.00
		20	1	100.00%	100.00%	0.00%	1868	1868.00
		25	1	100.00%	100.00%	0.00%	1868	1868.00

In the case of W32.Kitti, the percentage of identities and similarities was 100%. For the W32.CTX virus, the percentages of identities and similarities was over 94% and in some cases 100%. Not unsurprisingly, the gap percentage increases with lower gap open penalties (see Columns 'Gap Open Penalty' and 'Gaps Percentage' in Table 5.9), indicating that the number of insertions or deletions required to maximise the number of matches was also lower. In previously adopted methods a fixed combination of a gap open of 10 and gap extend of 0.5 penalties was used and considered optimal. It can be seen from the results in Table 5.9 that the percentages of identities and similarities were higher when both the gap open and gap extend penalties were higher than employed previously, indicating that the (pairwise local) alignments were compact, thereby restricting the amount of gaps (with lower gap percentages) and increasing their importance (see Columns 'Gap Open Penalty', 'Gap Extend Penalty' and 'Gaps Percentage' in Table 5.9).

The best gap open and gap extend penalties are 10 and 1, and 20 and 1, respectively, as both gap penalty combinations consistently have identities and similarities of over 85%. In most cases of W32.Kitti virus and some cases of W32.CTX virus, these gap penalty combinations have identities and similarities of 100%. In one or two overall cases, the best gap open and gap extend penalties are 5 and 1, and 25 and 1, respectively, with identities and similarities of over 88%.

Table 5.10 provides the detection performance results for the identification of the three polymorphic malware along with their known (P_k) and unknown (P_x) polymorphic variants. In total, 59 meta-signatures were tested, but only the results using the most effective meta-signatures are provided in Table 5.10. The detection was carried out using 'clamscan' and the most effective meta-signatures – those that detected over 90% of the polymorphic variants.

As for all virus detection methods developed in this research, the performance of this proposed method was compared with that of the top five commercial products available at the time of this research in 2016. The results are presented in Table 5.10.

Table 5.10: Detection rates for detection of three polymorphic malware using the best performing meta-signatures.

Polymorphic Malware 1	AVS Product/Pairwise Alignment	Detection Method	Detection Ratio (with Accuracy) and Statistical Measures	
JS.Cassandra Virus	AntiVirus Ranked No. 1	Bitdefender Antivirus	**Detection Ratio (Accuracy)**	1/352 (0.2841%)
			Sensitivity/Recall	0.2841%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	0.5666%
	AntiVirus Ranked No. 2	Kaspersky Anti-Virus	**Detection Ratio (Accuracy)**	1/352 (0.2841%)
			Sensitivity/Recall	0.2841%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	0.5666%
	AntiVirus Ranked No. 3	McAfee AntiVirus	**Detection Ratio (Accuracy)**	152/352 (43.18%)
			Sensitivity/Recall	43.18%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	60.31%
	AntiVirus Ranked No. 4	Norton Security	**Detection Ratio (Accuracy)**	5/352 (1.42%)
			Sensitivity/Recall	1.42%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	2.80%
	AntiVirus Ranked No. 5	F-Secure Anti-Virus	**Detection Ratio (Accuracy)**	1/352 (0.2841%)
			Sensitivity/Recall	0.2841%)
			Specificity	0.00%
			Precision	100.00%
			F1 Score	0.5666%
	Original and Variant 1	MS1	**Detection Ratio (Accuracy)**	340/352 (96.59%)
			Sensitivity/Recall	96.59%
			Specificity	0.00%
			Precision	100%
			F1 Score	98.26%
	Variant 1 and Variant 2	MS4	**Detection Ratio (Accuracy)**	339/352 (96.31%)
			Sensitivity/Recall	96.31%
			Specificity	0.00%
			Precision	100%
			F1 Score	98.12%
	Variant 2 and Variant 3	MS5	**Detection Ratio (Accuracy)**	339/352 (96.31%)
			Sensitivity/Recall	96.31%
			Specificity	0.00%
			Precision	100%
			F1 Score	98.12%
		MS6	**Detection Ratio (Accuracy)**	325/352 (92.33%)
			Sensitivity/Recall	92.33%
			Specificity	0.00%
			Precision	100%
			F1 Score	96.01%
		MS7	**Detection Ratio (Accuracy)**	340/352 (96.59%)
			Sensitivity/Recall	96.59%
			Specificity	0.00%
			Precision	100%
			F1 Score	98.26%
		MS8	**Detection Ratio (Accuracy)**	339/352 (96.31%)
			Sensitivity/Recall	96.31%
			Specificity	0.00%
			Precision	100%
			F1 Score	98.12%
	Variant 3 and Variant 4	MS10	**Detection Ratio (Accuracy)**	325/352 (92.33%)
			Sensitivity/Recall	92.33%
			Specificity	0.00%
			Precision	100%
			F1 Score	96.01%

Polymorphic Malware 2	AVS Product/Pairwise Alignment	Detection Method	Detection Ratio (with Accuracy) and Statistical Measures	
W32.CTX/W32.Cholera Virus	AntiVirus Ranked No. 1	Bitdefender Antivirus	**Detection Ratio (Accuracy)**	176/200 (88.00%)
			Sensitivity/Recall	88.00%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	93.62%
	AntiVirus Ranked No. 2	Kaspersky Anti-Virus	**Detection Ratio (Accuracy)**	86/200 (43.00%)
			Sensitivity/Recall	43.00%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	60.14%
	AntiVirus Ranked No. 3	McAfee AntiVirus	**Detection Ratio (Accuracy)**	27/200 (13.50%)
			Sensitivity/Recall	13.50%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	23.79%
	AntiVirus Ranked No. 4	Norton Security	**Detection Ratio (Accuracy)**	177/200 (88.50%)
			Sensitivity/Recall	88.50%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	93.89%
	AntiVirus Ranked No. 5	F-Secure Anti-Virus	**Detection Ratio (Accuracy)**	191/200 (95.50%)
			Sensitivity/Recall	95.50%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	97.69%
	Variant 1 and Variant 2	MS4	**Detection Ratio (Accuracy)**	183/200 (91.50%)
			Sensitivity/Recall	91.50%
			Specificity	0.00%
			Precision	100%
			F1 Score	95.56%
	Variant 2 and Variant 3	MS7	**Detection Ratio (Accuracy)**	189/200 (94.50%)
			Sensitivity/Recall	94.50%
			Specificity	0.00%
			Precision	100%
			F1 Score	97.17%
	Variant 3 and Variant 4	MS12	**Detection Ratio (Accuracy)**	189/200 (94.50%)
			Sensitivity/Recall	94.50%
			Specificity	0.00%
			Precision	100%
			F1 Score	97.17%
		MS13	**Detection Ratio (Accuracy)**	200/200 (100.00%)
			Sensitivity/Recall	100.00%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	100.00%
		MS14	**Detection Ratio (Accuracy)**	192/200 (96.00%)
			Sensitivity/Recall	96.00%
			Specificity	0.00%
			Precision	100%
			F1 Score	97.96%
		MS15	**Detection Ratio (Accuracy)**	200/200 (100.00%)
			Sensitivity/Recall	100.00%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	100.00%
		MS16	**Detection Ratio (Accuracy)**	200/200 (100.00%)
			Sensitivity/Recall	100.00%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	100.00%
	Variant 4 and Variant 5	MS17	**Detection Ratio (Accuracy)**	200/200 (100.00%)
			Sensitivity/Recall	100.00%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	100.00%

Polymorphic Malware 3	AVS Product/Pairwise Alignment	Detection Method	Detection Ratio (with Accuracy) and Statistical Measures	
W32.Kitti Virus	**AntiVirus Ranked No. 1**	Bitdefender Antivirus	**Detection Ratio (Accuracy)**	324/1106 (29.29%)
			Sensitivity/Recall	29.29%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	45.31%
	AntiVirus Ranked No. 2	Kaspersky Anti-Virus	**Detection Ratio (Accuracy)**	1106/1106 (100.00%)
			Sensitivity/Recall	100.00%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	100.00%
	AntiVirus Ranked No. 3	McAfee AntiVirus	**Detection Ratio (Accuracy)**	293/1106 (26.49%)
			Sensitivity/Recall	26.49%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	41.88%
	AntiVirus Ranked No. 4	Norton Security	**Detection Ratio (Accuracy)**	450/1106 (40.69%)
			Sensitivity/Recall	40.69%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	57.84%
	AntiVirus Ranked No. 5	F-Secure Anti-Virus	**Detection Ratio (Accuracy)**	333/1106 (30.11%)
			Sensitivity/Recall	30.11%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	46.28%
	Original and Variant 1	MS1-MS6	**Detection Ratio (Accuracy)**	1106/1106 (100.00%)
			Sensitivity/Recall	100.00%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	100.00%
	Variant 1 and Variant 2	MS7-MS10	**Detection Ratio (Accuracy)**	1106/1106 (100.00%)
			Sensitivity/Recall	100.00%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	100.00%
	Variant 2 and Variant 3	MS11, MS13-M16	**Detection Ratio (Accuracy)**	1106/1106 (100.00%)
			Sensitivity/Recall	100.00%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	100.00%
		MS12	**Detection Ratio (Accuracy)**	1105/1106 (99.91%)
			Sensitivity/Recall	99.91%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	99.95%
	Variant 3 and Variant 4	MS17, MS19-MS22	**Detection Ratio (Accuracy)**	1106/1106 (100.00%)
			Sensitivity/Recall	100.00%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	100.00%
		MS18	**Detection Ratio (Accuracy)**	1105/1106 (99.91%)
			Sensitivity/Recall	99.91%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	99.95%
	Variant 4 and Variant 5	MS23, M25, MS27-MS30	**Detection Ratio (Accuracy)**	1106/1106 (100.00%)
			Sensitivity/Recall	100.00%
			Specificity	0.00%
			Precision	100.00%
			F1 Score	100.00%

* Where MS is the meta-signature generated using the proposed two-phase alignment with the two viral variants detailed in Column 'AVS Product/Pairwise Alignment' and signature extraction method detailed in this section. The detection was carried out using 'clamscan'.

Fifty-seven of the 59 meta-signatures tested detected some or all of the polymorphic variants (Table 5.10). Two meta-signatures failed to detect any of the variants (the results for these two signatures are not shown in Table 5.10). The results of these virus detection tests also show that in the majority of cases the top five commercially available AVSs evaluated could not detect all of the variants. Moreover, in the case of JS.Cassandra three of the commercial AVSs, Bitdefender, Kaspersky, and F-Secure, were only able to detect the original (P_s) virus. The one exception was the Kaspersky anti-virus tool which was found to successfully detect all of the unknown (P_x) polymorphic variants of the W32.Kitti virus.

For JS.Cassandra virus, detection rates of over 92% were observed when detection using seven of the 12 novel automatically generated syntactic meta-signatures developed in this research were employed. In the case of W32.Kitti virus, for 26 of the 28 most effective meta-signatures the detection rates were 100% and for the remaining two, the detection rates were over 99% (Table 5.10). A similar performance was noted for the W32.CTX virus, where four of the eight most effective meta-signatures achieved detection rates of 100% and for the remaining four, the detection rates were over 91% (Table 5.10).

The eleven super-signatures were also tested on the three polymorphic malware variants employing 'clamscan'. All 352 JS.Cassandra viral variants (together with the original (P_s) virus) were successfully identified as infected by 'clamscan' using one of the new automatically generated super-signatures (Figure 5.6). Of the two remaining JS.Cassandra super-signatures one performed well detecting 96.58% of the variants (including the original (P_s) virus) but the other performed poorly detecting 15 of the 352 variants (4.26%). All 200 of the W32.CTX viral variants (including the two original (P_s) viruses) were successfully identified by the 'clamscan' using one of its super-signatures (Figure 5.7). The remaining two W32.CTX super-signatures resulted in a reasonable accuracy of 94.5% and a very poor accuracy of 9.5%. Figure 5.8 shows that all 1106 of the W32.Kitti variants (together with the original (P_s) virus) were successfully identified by three of the five super-signatures. The remaining two super-signatures failed to detect any of the 1106 variants. As stated earlier (see page no. 88), these signatures are considered in this research as "unknown signatures" and the exact reason for its failure to detect any virus variants remains unknown. Although it may be that some of the super-signatures performed poorly or failed to detect any was due to the fact that those super-signatures capture variants that either belong to a small set of special variants that were

generated using a different obfuscation method or are the ones that have not yet encountered. None of the scans took longer than 15 seconds, with most the majority of the scans taking under three seconds.

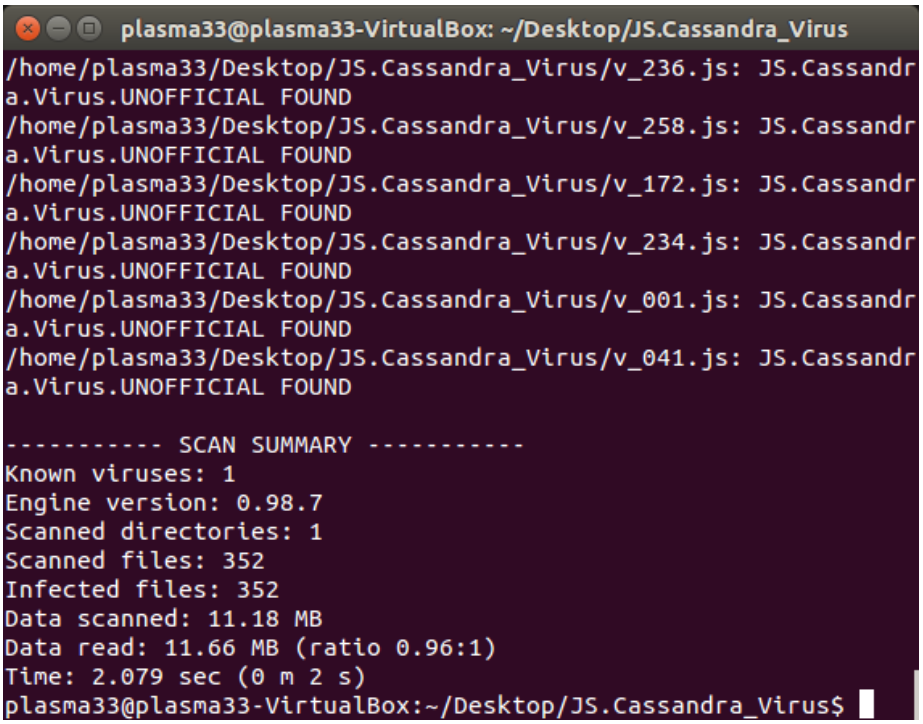

Figure 5.6: Clamscan screenshot for JS.Cassandra and known (P_k) variants using the best performing super-signature.

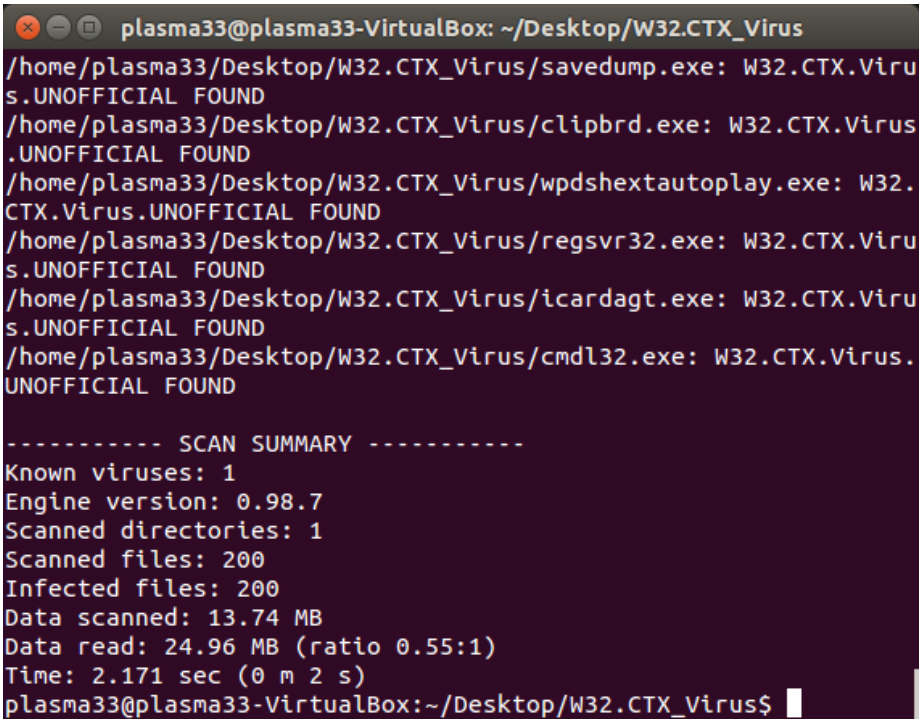

Figure 5.7: Clamscan screenshot for W32.CTX and unknown (P_x) variants using the best performing super-signature.

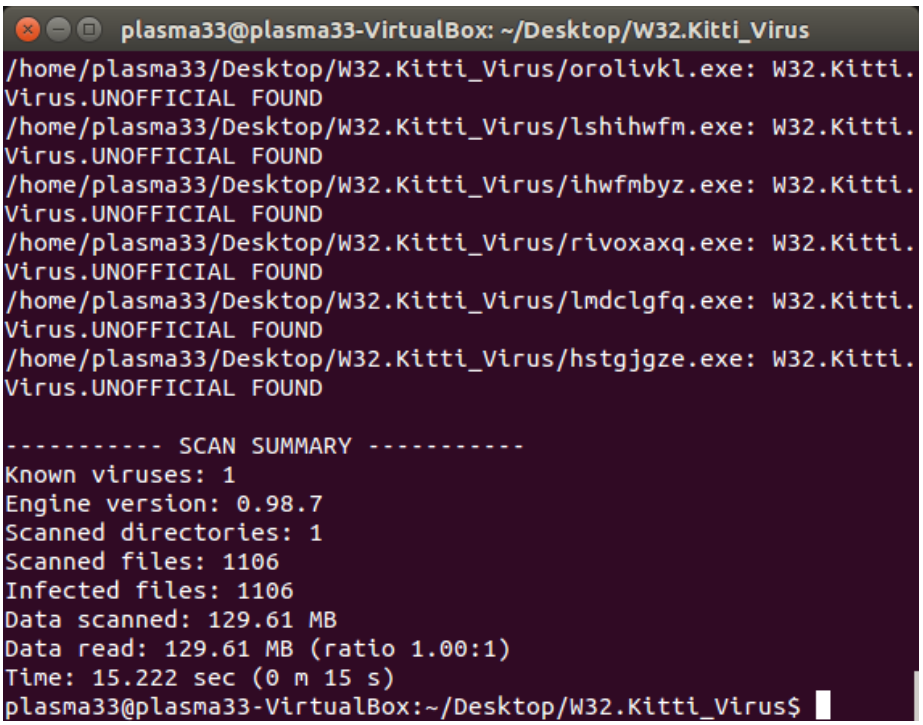

Figure 5.8: Clamscan screenshot for W32.Kitti and unknown (P_x) variants using the best performing super-signature.

5.9. Summary

In this chapter, advanced sequence alignment techniques were explored by investigating the effects of different combinations of gap open and gap extend penalties not only for the identification of known (P_k) polymorphic variants (training set) but also for the identification of unknown (P_x) polymorphic variants (test set). For the process of all the first (pairwise) sequence alignments, ten different combinations of gap open and gap extend penalties were used. Experimental results from the first alignment process are shown in Table 5.9. The process of all the second (pairwise) sequence alignments were conducted using a fixed optimal combination of gap open penalty (i.e. 10) and gap extend penalty (i.e. 1). Using an in-house macro tool and the proposed eight step string based approach, that takes a natural computation approach of projecting polymorphic malware (hexadecimal) code onto biological representational space (i.e. DNA and amino acid) and using bioinformatics algorithms (i.e. pairwise/multiple sequence alignments and SWA) has been demonstrated to be a successful method for automating malware signature extraction. The effectiveness of these extracted meta-signatures and super-signatures in the detection of polymorphic viruses was then evaluated. The vast majority of these signatures used in conjunction with ‘clamscan’ were shown to outperform the popular AVSs used to benchmark the proposed malware detection method developed in this research.

Experimental results from Table 5.9 show that optimal gap penalty combinations of a gap open of 10 and gap extend of 1 as well as gap open of 20 and gap extend of 1, in some cases have identities and similarities of over 85%, and, in most cases have 100%. Meta-signatures generated from these gap penalty combinations not only detected known (P_k) variants with accuracies of over 92% but also detected unknown (P_x) variants with accuracies of over 94% (Table 5.10). Selected super-signatures generated from the optimal combinations of a gap open of 10 and gap extend of 1 detected all the known (P_k) as well as unknown (P_x) variants (Figures 5.6-5.8).

The next section investigates further refinement of the string searching algorithms using a different type of substitution matrix. Experiments are reported that are designed to address the third part of the second research question (Q2c). More information will follow in the succeeding part.

Part-III: Using Different Substitution Matrices in a String-based Syntactic Approach for Identifying Viral Polymorphic Malware Variants

In this work JS.Cassandra virus and its variants are used. Only one polymorphic virus family is used in this research due to sequence length restrictions (Kim & Pramanik, 1994; Yu, Bundschuh, & Hwa, 2002; Chakraborty & Bandyopadhyay, 2013) placed by the alignment tool (more details on page no. 18) adopting many different types of substitution matrices. The same experimental steps will be employed as previous research, but each experiment will employ a different substitution matrix for the SWA. In previous work the ID matrix was used and this approach is the benchmark for these experiments. This work aims to answer research question Q2c in more detail: "Does using frequency-based substitution matrices produce new and more-effective signatures for detecting new polymorphic malware variants?". ID matrix is a simple non-biological substitution square matrix in which all the components of the primary diagonal are ones, and all other components are zeros. The other well-known matrices, such as BLOSUM (Block Substitution Matrix) and PAM (Point Accepted Mutation) are complex biological substitution/mutation matrices, all of which are used in this part as frequency-based approaches for experimental purposes.

5.10. Introduction

The SWA which is used in this research as a step in a proposed approach to automatic signature extraction from polymorphic viruses depends on gap parameters and on a substitution matrix or score matrix. Previous work reported in this chapter investigated the critical gap parameters. This work investigate frequency scoring matrices (e.g. BLOSUM (Henikoff & Henikoff, 1992)), explicit evolutionary model-based scoring matrices (e.g. PAM (Dayhoff, Schwartz, & Orcutt, 1978)), simple scoring matrix (e.g. MATCH), scoring matrix for DNA (e.g. EDNAFULL (Lowe, 1992)), scoring matrix inspired from PAM250 (e.g. GONNET (Prlić, Domingues, & Sippl, 2000)), and a newly transformed mutation matrix in correspondence to PAM250 (e.g. DAYHOFF (Dayhoff, Schwartz, & Orcutt, 1978; Gonnet, 1998)). The 71 frequency scoring matrices used in this part are as follows: BLOSUM30, BLOSUM35, BLOSUM40, BLOSUM45, BLOSUM50, BLOSUM55, BLOSUM60, BLOSUM62, BLOSUM65, BLOSUM70, BLOSUM75, BLOSUM80, BLOSUM85, BLOSUM90, BLOSUM100, BLOSUMN, PAM10, PAM20, PAM30, PAM40, PAM50, PAM60, PAM70, PAM80, PAM90, PAM100, PAM110, PAM120, PAM130, PAM140, PAM150, PAM160, PAM170,

PAM180, PAM190, PAM200, PAM210, PAM220, PAM230, PAM240, PAM250, PAM260, PAM270, PAM280, PAM290, PAM300, PAM310, PAM320, PAM330, PAM340, PAM350, PAM360, PAM370, PAM380, PAM390, PAM400, PAM410, PAM420, PAM430, PAM440, PAM450, PAM460, PAM470, PAM480, PAM490, PAM500, DAYHOFF, GONNET, IDENTITY, EDNAFULL, and MATCH.

The substitution matrix assigns a score for aligning any possible pair of residues in the pairwise sequence alignment. Typically, for bioinformatics sequence alignment and pattern recognition using the SWA a frequency based matrix is used not an ID matrix. Examples of such substitution matrix generation algorithms include BLAST (Altschul, 1991), BLOSUM.

In BLOSUM the score is calculated as the log-ratio of the observed probability of substitution of one amino acid by another divided by the probability of the substitution occurring due to chance (Pearson, 2013). BLOSUM is based on the conservation of domains in proteins. PAM, on the other hand, is based on the rate of divergence between sequences or evolutionary distances. DAYHOFF matrix is a frequency based mutation matrix like the PAM matrix and has a close resemblance to the PAM250 matrix. As stated earlier, GONNET is also inspired from the PAM250 matrix. MATCH matrix is a simple scoring matrix like the ID matrix and is used in this part as it provides the most parsimonious frequency based method in that no assumptions are made as to how symbols may be related to each other. EDNAFULL matrix is used in this part for nucleotides as this frequency based scoring matrix is purely made for scoring alignments of DNA sequences.

This experiment also evaluates an approach using PRISM (Cendrowska, 1987) a rule induction algorithm which converts a decision tree based on Quinlan's ID3 algorithm (Quinlan, 1979) into a rule set. PRISM is used in this part to generate rules. These rules are then employed to generate super-signatures by performing alignments.

5.11. Using Different Substitution Matrices in a String-Based Syntactic Approach for Detecting Polymorphic Malware Variants Method: Systems and Methods

This method is comprised of eight steps (see Figure 5.9). The effects of using different substitution matrices will be explored in relation to the detection of JS.Cassandra polymorphic virus and its known (P_k) variants.

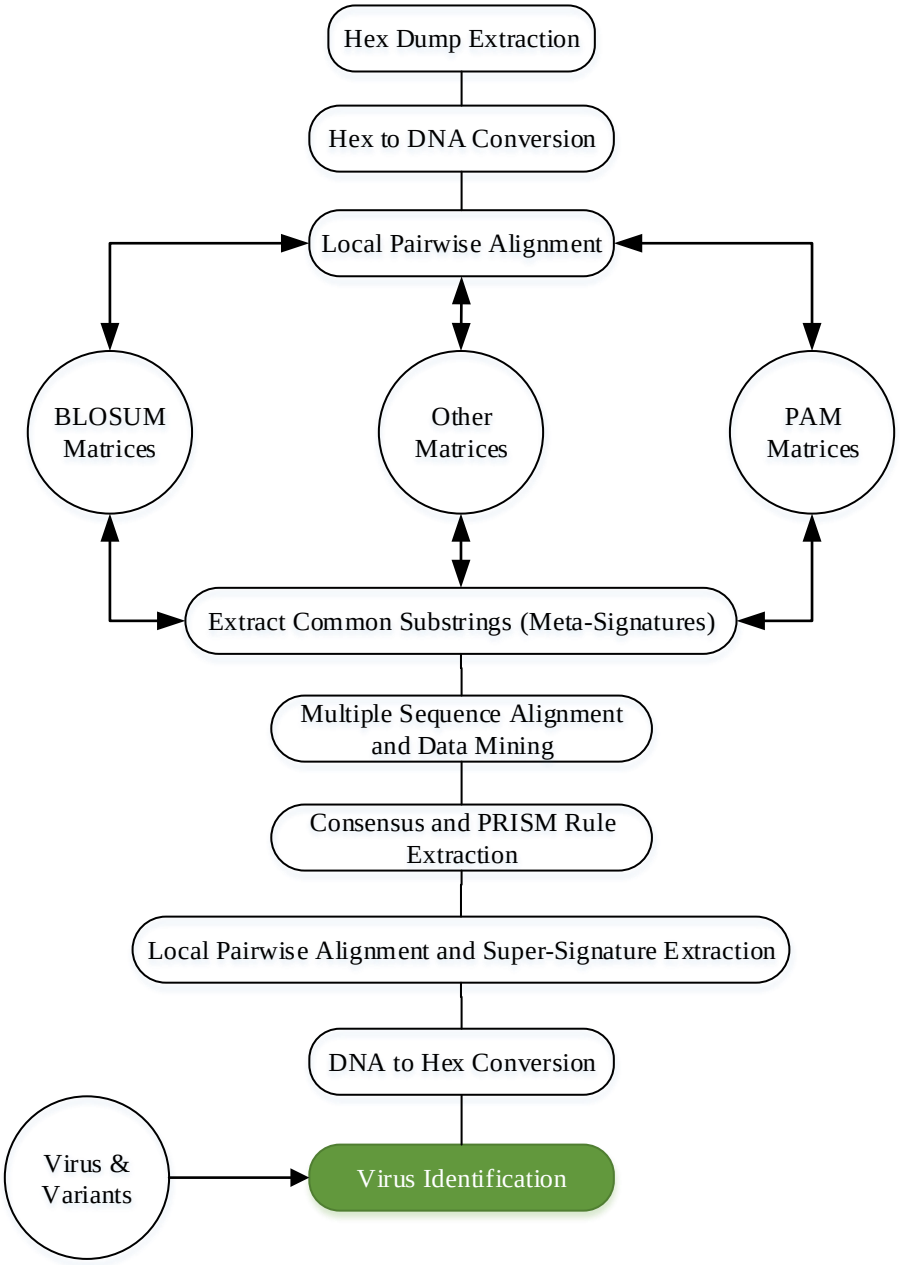

Figure 5.9: Eight-step method for using Different Substitution Matrices in a String-Based Syntactic Approach for Detecting Polymorphic Malware Variants.

Steps 1-8 are the same as those used previously (Part-I) in this research except for Step-3 in which the different substitution matrices are introduced (see Section 5.3 – page no. 79). Also, in Step-6 consensuses have been extracted after performing multiple sequence alignment (Step-5) on the newly generated meta-signatures. Next, rule induction (data mining) on the meta-signatures is performed (Step-5) to extract a set of (single) rules (Step-6). The next step involves several pairwise local sequence alignments using SWA and 'JAligner' (Step-7). Thus, Step-7 results in two sets of super-signatures (more details in Section 5.11.6). As for the previous experiments, the meta-signatures and super-

signatures are converted back to hexadecimal format and then tested using ‘clamscan’ (Step-8).

5.11.1 Hex Dump Extraction

Step-1: Two variants of JS.Cassandra, v_004.js and v_005.js, were used in this experiment. Hex dumps of each were extracted using ‘sigtool’ and the files tested with 56 AVSs retrieved from ‘VirusTotal’ (Table 5.11). Only 20 of the AVSs could detect variant 1 and 21 could detect variant 2.

Table 5.11: Analysis and detection ratio using the 56 AVSs retrieved from the ‘VirusTotal’ website for the two JS.Cassandra variants in hexadecimal format.

Antivirus	Variant 1 (v_004.js)	Variant 2 (v_005.js)
Ad-Aware	No	No
AegisLab	**Js.Cassa.B!c**	**Script.Troj.Agent!c**
AhnLab-V3	No	No
Alibaba	No	No
ALYac	No	No
Antiy-AVL	No	No
Arcabit	No	No
Avast	**JS:Cassa-A [Wrm]**	**JS:Cassa-A [Wrm]**
AVG	**JS/Cassa**	**JS/Cassa**
Avira (no cloud)	**JS/Cassa.B.8392**	No
AVware	**Trojan.JS.Cassan.a (v)**	**Trojan.JS.Cassan.a (v)**
Baidu	**JS.Virus.Cassa.b**	**JS.Virus.Cassa.b**
Baidu-International	No	No
Bitdefender	No	No
Bkav	No	No
CAT-QuickHeal	No	No
ClamAV	**Win.Trojan.Cassa-1**	**Win.Trojan.Cassa-1**
CMC	No	No
Comodo	**UnclassifiedMalware**	**UnclassifiedMalware**
Cyren	**JS/Cassa.A!Eldorado**	**JS/Cassa.A!Eldorado**
DrWeb	No	No
Emsisoft	No	No
eScan	No	No
ESET-NOD32	**JS/Cassa.B**	**JS/Cassa.B**
Fortinet	No	No
F-Prot	**JS/Cassa.A!Eldorado**	**JS/Cassa.A!Eldorado**
F-Secure	No	No
GData	No	**Script.Trojan.Agent.U7IEZA**
Ikarus	**Trojan.JS.Cassa**	**Trojan.JS.Cassa**
Jiangmin	No	No
K7AntiVirus	**Exploit (04c555e01)**	**Exploit (04c555e01)**
K7GW	**Exploit (04c555e01)**	**Exploit (04c555e01)**
Kaspersky	No	No
Kingsoft	No	No
Malwarebytes	No	No
McAfee	**JS/Cassan**	**JS/Cassan**

Antivirus	Variant 1 (v_004.js)	Variant 2 (v_005.js)
McAfee-GW-Edition	**BehavesLike.JS.Downloader.x**	**BehavesLike.JS.Downloader.zm**
Microsoft	**Trojan:JS/Cassa.B.gen**	**Trojan:JS/Cassa.B.gen**
NANO-Antivirus	No	No
nProtect	No	No
Panda	No	No
Qihoo-360	**Script/Virus.8c8**	**Script/Trojan.617**
Rising	No	No
Sophos	**JS/Cassan-A**	**JS/Cassan-A**
SUPERAntiSpyware	No	No
Symantec	No	No
Tencent	No	**Js.Virus.Cassa.Ajle**
TheHacker	No	No
TrendMicro	No	No
TrendMicro-HouseCall	No	No
VBA32	No	No
VIPRE	**Trojan.JS.Cassan.a (v)**	**Trojan.JS.Cassan.a (v)**
ViRobot	No	No
Yandex	No	No
Zillya	No	No
Zoner	No	No
Detection Ratio	**20/56**	**21/56**

5.11.2 Hex to DNA conversion

Step-2: In this step, extracted hex dump sequences were converted into binary and then into DNA format adopting the DNA encoding method shown in Section 4.3.2 (see page no. 66).

5.11.3 First Pairwise Local Alignment and Meta-Signature Extraction

Step-3: In this step, several pairwise (local) alignments were performed using the SWA with the usual gap open and gap extend penalties of 10 and 1 respectively. In total, 71 pairwise local alignments were performed in this step. Only the alignment results with the highest combinations of identity and similarity percentages were retained in this step. Based on this criterion, alignment results of BLOSUM40, DAYHOFF, IDENTITY, MATCH, PAM100, and PAM350 were selected, that is, in total six alignment results were chosen in this step. Table 5.12 shows selected results from the six pairwise local alignments that were performed in Step-3. Full results obtained in this step are presented in Appendix E (Table E.1 – see page no. 236).

Step-4: After the procedure of local alignment, the common substrings (meta-signatures) were extracted. In total, 161 common substrings were extracted: 34 using BLOSUM40, 24 using DAYHOFF, one using IDENTITY, 24 using MATCH, 31 using PAM100 and 47 using the PAM350 matrix.

Table 5.12: Selected results of the six pairwise local alignments performed in Step-3.

Substitution Matrix	Gap Open Penalty	Gap Extend Penalty	Identity Percentage	Similarity Percentage	Gaps Percentage	Alignment Length	Alignment Score
BLOSUM40	10	1	52.83%	57.81%	27.22%	41181	177285.00
DAYHOFF	10	1	49.41%	62.32%	21.70%	39868	112237.00
IDENTITY	10	1	100.00%	100.00%	0.00%	397	397.00
MATCH	10	1	63.57%	63.57%	18.81%	6083	1192.00
PAM100	10	1	51.60%	62.28%	25.74%	40656	91799.00
PAM350	10	1	47.05%	69.15%	24.62%	40532	165839.00

5.11.4 Multiple Sequence Alignment and Data Mining

Step-5: In this step, a multiple alignment was performed on all the meta-signatures retrieved in Step-4 adopting T-Coffee with alignment being restricted to the ID substitution matrix. Overall, five independent multiple alignments were performed (i.e. on 34, 24, 24, 31 and 47 meta-signatures, respectively), one for each of the five substitution matrices. IDENTITY matrix was not considered in this step as multiple sequence alignment requires two or more sequences for its process – ID matrix generated only one substring in Step-4. But the ID matrix meta-signature will be considered in this sub-step of data mining (more details below). As stated earlier, the main aim of multiple alignment here is to produce consensuses (more details in Step-6).

Also, in this step, a data mining classification algorithm known as PRISM (Cendrowska, 1987) was applied on all the 161 extracted meta-signatures (obtained in Step-4, including the meta-signature obtained from the ID matrix) to extract rules. The purpose of data mining in this step was important as multiple alignment did not take into account all the 161 meta-signatures. Rule extraction using the supervised PRISM learning (with the training set option) was performed with the help of an open source software product called Weka (Waikato Environment for Knowledge Analysis). As stated earlier, Weka is a tool that provides a collection of machine learning algorithms for performing data mining tasks (Frank, Hall, & Witten, 2016). Some of the PRISM rules for BLOSUM40 (with 62% accuracy) obtained in this step are as follows (observe that '*pos*' and '*B40*' are acronyms for position and BLOSUM40 in the rules shown below):

If pos30 = A then B40
If pos3 = C and pos18 = G then B40
If pos22 = A and pos7 = T then B40
If pos66 = A and pos2 = T then B40

5.11.5 Extraction of Consensuses and PRISM Rules

Step-6: As stated earlier, T-Coffee, similar to other alignment tools, produces a consensus sequence that signifies the most common residues (nucleotide) in each position of the sequences after alignment. In this step, the consensus was stored, and the procedure was followed five times, one for each of the five substitution matrices. No threshold of common occurrence of a nucleotide in a certain position was adopted/set in this step. Overall, five consensuses were extracted in this step.

Also, in this step (after the process of data mining using PRISM in Step-5), a total of five single PRISM rules (one for each of the five selected substitution matrices) were extracted with an accuracy of 62%. Single PRISM rule was obtained by merging all the nucleotide characters from all the positions for that specific substitution matrix. For instance, from the few PRISM rules given above for BLOSUM40, the following single rule is obtained: *ACGATAT*.

5.11.6 Second Pairwise Local Alignment and Super-Signature Extraction

Step-7: In this step, two sets of pairwise (local) alignments, using the usual gap open and gap extend penalties of 10 and 1 respectively, were performed adopting the SWA with ID matrix (see Chapters 4 and 5 – Part-II). One set between the consensuses and the original (P_s) JS.Cassandra virus and variant 1. The other set between the single PRISM rules and the original (P_s) JS.Cassandra virus and variant 1, both using 'JAligner'. Overall, 20 individual pairwise local alignments were performed in this step, 10 for each consensus and 10 for each single PRISM rule. Forty-seven super-signatures were extracted from the two sets, 21 from the consensuses-based pairwise alignments and 26 from the single PRISM rules-based pairwise alignments.

5.11.7 DNA to Hex Conversion as well as Meta-Signature and Super-Signature Testing

Step-8: Again all the 161 meta-signatures and 47 super-signatures were converted back into a hexadecimal format before testing their effectiveness using 'clamscan'.

5.12. Experimental Results

Table 5.13 supplies the results of the detection rates (with accuracy) for the detection of JS.Cassandra polymorphic malware and its known (P_k) variants adopting 'clamscan' using the meta-signatures. In total, 161 meta-signatures were tested.

Table 5.13: Detection rates for the detection of JS.Cassandra polymorphic malware and its known (351) variants (P_k) employing 'clamscan' by testing the 161 meta-signatures acquired in Step-4.

BLOSUM40						DAYHOFF					
Meta-Signatures with their Sequence Lengths in Hex (x25)	**Detection Rate (Accuracy)**	**Sensitivity/ Recall**	**Specificity**	**Precision**	**F1 Score**	**Meta-Signatures with their Sequence Lengths in Hex (x24)**	**Detection Rate (Accuracy)**	**Sensitivity/ Recall**	**Specificity**	**Precision**	**F1 Score**
MS1 – 22	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS35 – 22	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS2 – 42	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS36 – 50	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS3 – 11	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	MS37 – 53	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS4 – 50	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS38 – 38	**352/352 (100%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
MS5 – 12	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS39 – 23	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS6 – 14	**333/352 (94.60%)**	**94.60%**	**0.0%**	**100.00%**	**97.22%**	MS40 – 50	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS7 – 23	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	MS41 – 24	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS8 – 13	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	MS42 – 10	**322/352 (91.50%)**	**91.50%**	**0.0%**	**100.00%**	**95.56%**
MS9 – 40	**352/352 (100%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**	MS43 – 25	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS10 – 52	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	MS44 – 44	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS11 – 52	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS45 – 29	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS12 – 12	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS46 – 14	**352/352 (100%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
MS13 – 12	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	MS47 – 26	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS14 – 26	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS48 – 32	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS15 – 28	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS49 – 25	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS16 – 24	**326/352 (92.61%)**	**92.61%**	**0.0%**	**100.00%**	**96.16%**	MS50 – 45	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS17 – 22	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	MS51 – 42	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS18 – 28	314/352 (89.20%)	89.20%	0.0%	100.00%	94.29%	MS52 – 14	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS19 – 25	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	MS53 – 30	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS20 – 14	**352/352 (100%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**	MS54 – 16	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS21 – 96	**328/352 (93.20%)**	**93.20%**	**0.0%**	**100.00%**	**96.5%**	MS55 – 14	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS22 – 18	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS56 – 26	**331/352 (94.03%)**	**94.03%**	**0.0%**	**100.00%**	**96.92%**
MS23 – 32	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS57 – 28	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS24 – 25	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	MS58 – 16	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS25 – 44	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	-	-	-	-	-	-

BLOSUM40						IDENTITY					
Meta-Signatures with their Sequence Lengths in Hex (x9)	**Detection Rate (Accuracy)**	**Sensitivity/ Recall**	**Specificity**	**Precision**	**F1 Score**	**Meta-Signatures with their Sequence Lengths in Hex (x1)**	**Detection Rate (Accuracy)**	**Sensitivity/ Recall**	**Specificity**	**Precision**	**F1 Score**
MS26 – 42	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS59 – 198	14/352 (4.0%)	4.0%	0.0%	100.00%	7.69%
MS27 – 15	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	**PAM100**					
MS28 – 30	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	**Meta-Signatures with their Sequence Lengths in Hex (x21)**	**Detection Rate (Accuracy)**	**Sensitivity/ Recall**	**Specificity**	**Precision**	**F1 Score**
MS29 – 16	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS84 – 22	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS30 – 14	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS85 – 28	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS31 – 26	**331/352 (94.03%)**	**94.03%**	**0.0%**	**100.00%**	**96.92%**	MS86 – 10	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS32 – 28	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	MS87 – 10	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS33 – 16	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS88 – 25	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS34 – 24	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS89 – 18	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MATCH						MS90 – 16	**337/352 (95.74%)**	**95.74%**	**0.0%**	**100.00%**	**97.82%**
Meta-Signatures with their Sequence Lengths in Hex (x13)	**Detection Rate (Accuracy)**	**Sensitivity/ Recall**	**Specificity**	**Precision**	**F1 Score**	MS91 – 38	**352/352 (100%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
MS60 – 22	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS92 – 53	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS61 – 32	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS93 – 52	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS62 – 82	309/352 (87.80%)	87.80%	0.0%	100.00%	93.5%	MS94 – 10	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS63 – 34	255/352 (72.44%)	72.44%	0.0%	100.00%	84.02%	MS95 – 12	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS64 – 22	293/352 (83.24%)	83.24%	0.0%	100.00%	90.85%	MS96 – 50	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS65 – 32	132/352 (37.5%)	37.5%	0.0%	100.00%	54.54%	MS97 – 24	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS66 – 24	**336/352 (95.45%)**	**95.45%**	**0.0%**	**100.00%**	**97.67%**	MS98 – 24	**326/352 (92.61%)**	**92.61%**	**0.0%**	**100.00%**	**96.16%**
MS67 – 18	280/352 (79.54%)	79.54%	0.0%	100.00%	88.6%	MS99 – 22	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS68 – 18	288/352 (81.82%)	81.82%	0.0%	100.00%	90.001%	MS100 – 28	314/352 (89.20%)	89.20%	0.0%	100.00%	94.3%
MS69 – 22	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS101 – 25	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS70 – 16	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS102 – 14	**352/352 (100%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
MS71 – 28	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS103 – 96	**328/352 (93.20%)**	**93.20%**	**0.0%**	**100.00%**	**96.5%**
MS72 – 18	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS104 – 26	311/352 (88.35%)	88.35%	0.0%	100.00%	93.81%

MATCH						PAM100					
Meta-Signatures with their Sequence Lengths in Hex (x11)	**Detection Rate (Accuracy)**	**Sensitivity/ Recall**	**Specificity**	**Precision**	**F1 Score**	**Meta-Signatures with their Sequence Lengths in Hex (x10)**	**Detection Rate (Accuracy)**	**Sensitivity/ Recall**	**Specificity**	**Precision**	**F1 Score**
MS73 – 52	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS105 – 30	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS74 – 198	14/352 (4.0%)	4.0%	0.0%	100.00%	7.69%	MS106 – 44	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS75 – 50	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	MS107 – 42	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS76 – 16	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS108 – 14	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS77 – 65	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	MS109 – 16	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS78 – 60	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	MS110 – 14	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS79 – 52	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS111 – 26	**331/352 (94.03%)**	**94.03%**	**0.0%**	**100.00%**	**96.92%**
MS80 – 16	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS112 – 28	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS81 – 26	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS113 – 16	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS82 – 28	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS114 – 26	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS83 – 50	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	-	-	-	-	-	-
PAM350											
Meta-Signatures with their Sequence Lengths in Hex (x13)	**Detection Rate (Accuracy)**	**Sensitivity/ Recall**	**Specificity**	**Precision**	**F1 Score**	**Meta-Signatures with their Sequence Lengths in Hex (x13)**	**Detection Rate (Accuracy)**	**Sensitivity/ Recall**	**Specificity**	**Precision**	**F1 Score**
MS115 – 22	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	**MS139 – 8**	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS116 – 9	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	**MS140 – 26**	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS117 – 10	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	**MS141 – 32**	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS118 – 7	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS142 – 25	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS119 – 7	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	**MS143 – 6**	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS120 – 6	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	**MS144 – 6**	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS121 – 50	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS145 – 45	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS122 – 14	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	**MS146 – 42**	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS123 – 23	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	MS147 – 25	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS124 – 13	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	**MS148 – 6**	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS125 – 40	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	**MS149 – 6**	**343/352 (97.44%)**	**97.44%**	**0.0%**	**100.00%**	**98.7%**
MS126 – 53	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	**MS150 – 8**	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS127 – 52	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**	**MS151 – 14**	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**

PAM350											
Meta-Signatures with their Sequence Lengths in Hex (x11)	**Detection Rate (Accuracy)**	**Sensitivity/ Recall**	**Specificity**	**Precision**	**F1 Score**	**Meta-Signatures with their Sequence Lengths in Hex (x10)**	**Detection Rate (Accuracy)**	**Sensitivity/ Recall**	**Specificity**	**Precision**	**F1 Score**
MS128 – 52	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	MS152 – 7	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS129 – 8	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	**MS153 – 30**	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS130 – 28	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	**MS154 – 16**	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS131 – 8	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	**MS155 – 14**	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS132 – 8	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	**MS156 – 6**	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**
MS133 – 6	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	**MS157 – 26**	**331/352 (94.03%)**	**94.03%**	**0.0%**	**100.00%**	**96.92%**
MS134 – 8	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	**MS158 – 6**	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS135 – 38	**352/352 (100%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**	MS159 – 28	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS136 – 24	**339/352 (96.31%)**	**96.31%**	**0.0%**	**100.00%**	**98.12%**	**MS160 – 16**	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
MS137 – 7	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%	MS161 – 6	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
MS138 – 8	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%						

* Where MS is the meta-signature.

From Table 5.13, it can be seen that 123 out of the 161 meta-signatures identified JS.Cassandra and its known (P_k) variants. 112 of the meta-signatures (~70%) gave a detection accuracy of over 96%. Seven meta-signatures accurately detected all the variants (100%) and 38 (~ 24%) of the signatures identified none of the known (P_k) variants of JS.Cassandra. The PAM350 substitution matrix resulted in the highest number of successful meta-signatures with 38 out of the 47 meta-signatures (about 80.85%) identifying at least 96% known (P_k) variants of the JS.Cassandra virus. That is, on an average, the meta-signatures (or the cells) that are highlighted in bold and within the PAM350 category (see Table 5.13) identified 96% known (P_k) variants of the JS.Cassandra virus. Moreover, PAM350 substitution matrix gave the highest number of new meta-signatures in comparison to other substitution matrices.

Table 5.14 supplies the results of the identification of JS.Cassandra and its known (P_k) variants employing ‘clamscan’ using the 47 extracted super-signatures. More details on page no. 119.

Table 5.14: Detection rates for the detection of JS.Cassandra polymorphic malware and its known (351) variants (P_k) employing 'clamscan' by testing the 47 super-signatures acquired in Step-7.

		Super-Signatures with their corresponding Substitution Matrices	Detection Rate (Accuracy)	Sensitivity/ Recall	Specificity	Precision	F1 Score
Super-Signatures (x21) obtained from Consensuses-based Pairwise Alignments	**From Original Virus (x12)**	SS1 - BLOSUM40	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
		SS2 - BLOSUM40	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
		SS3 - DAYHOFF	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS4 - IDENTITY	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
		SS5 - IDENTITY	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
		SS6 - IDENTITY	53/352 (15.06%)	15.06%	0.0%	100.00%	26.2%
		SS7 - IDENTITY	43/352 (12.21%)	12.21%	0.0%	100.00%	21.76%
		SS8 - MATCH	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
		SS9 - MATCH	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS10 - PAM100	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
		SS11 - PAM100	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
		SS12 - PAM350	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
	From Variant 1 (x9)	SS13 - BLOSUM40	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS14 - BLOSUM40	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS15 - DAYHOFF	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS16 - IDENTITY	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS17 - MATCH	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS18 - MATCH	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS19 - PAM100	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
		SS20 - PAM100	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS21 - PAM350	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
Super-Signatures (x26) obtained from single PRISM Rules-based Pairwise Alignments	**From Original Virus (x19)**	SS22 - BLOSUM40	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
		SS23 - BLOSUM40	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
		SS24 - DAYHOFF	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
		SS25 - DAYHOFF	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
		SS26 - IDENTITY	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
		SS27 - IDENTITY	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
		SS28 - IDENTITY	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
		SS29 - IDENTITY	53/352 (15.06%)	15.06%	0.0%	100.00%	26.2%
		SS30 - IDENTITY	43/352 (12.21%)	12.21%	0.0%	100.00%	21.76%
		SS31 - MATCH	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
		SS32 - MATCH	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
		SS33 - MATCH	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
		SS34 - MATCH	53/352 (15.06%)	15.06%	0.0%	100.00%	26.2%
		SS35 - MATCH	43/352 (12.21%)	12.21%	0.0%	100.00%	21.76%
		SS36 - PAM100	**340/352 (96.59%)**	**96.59%**	**0.0%**	**100.00%**	**98.26%**
		SS37 - PAM100	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
		SS38 - PAM350	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS39 - PAM350	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS40 - PAM350	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
	From Variant 1 (x7)	SS41 - BLOSUM40	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%
		SS42 - DAYHOFF	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS43 - DAYHOFF	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS44 - IDENTITY	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS45 - MATCH	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS46 - PAM100	**352/352 (100.00%)**	**100.00%**	**0.0%**	**100.00%**	**100.00%**
		SS47 - PAM350	0/352 (0.00%)	0.0%	0.0%	100.00%	0.0%

* Where SS is the super-signature.

From Table 5.14, it can be seen that 34 of 47 super-signatures (~ 72%) correctly identified the JS.Cassandra variants. Sixteen of the 47 super-signatures had an accuracy of 100%, and 12 had an accuracy of 96.59%. Thirteen out of the 47 super-signatures (~28%) did not identify any of the variants. Super-signatures obtained from the pairwise alignments between consensuses and variant 1 as well as between single PRISM rules and variant 1 had the highest number of successful super-signatures with 12 out of the 47 super-signatures (~ 26%) identifying 100% of the known (P_k) variants. On the other hand, super-signatures obtained from the pairwise alignments between consensuses and the original (P_s) virus and between single PRISM rules and the original (P_s) virus had the lowest number of successful super-signatures (~ 8.51%) identifying all known (P_k) variants but had the highest number of successful super-signatures (~26%) identifying 96.59% known (P_k) variants of the JS.Cassandra virus.

5.13. Summary

In this chapter, sequence alignment techniques for automatically extracting polymorphic malware synatctic signatures were explored in more depth. The effects of using different substitution matrices (Part-III) were evaluated as well as the effects of gap penalties (Part-II). It was found that better super-signatures for JS.Cassandra could be generated from variant 1 using the PRISM single rule-based method (see Table 5.14) while the PAM350 substitution matrix gave the highest number of new and successful meta-signatures (see Table 5.13).

Because the PRISM approach to extracting super-signatures seems promising the set of experiments detailed in the next chapter further investigate a syntactic structure approach to the automatic generation of polymorphic viral signatures using a data mining (NNge) approach for extracting rules. Experiments are reported that are designed to address the third research question (**Q3**). More information will follow in the succeeding chapter.

Chapter 6 Identifying Viral Polymorphic Malware Variants using Data Mining Rule-Based NNge Classification Algorithm

Initial work (presented in Chapters 4 and 5) employed an approach which involved the conversion of hexadecimal dumps of polymorphic malware (and variants) into biological sequences before employing a sequence alignment algorithm to extract signatures was promising but also had some limitations. In an attempt to address these limitations, discussed in Section 1.5.5 (see page no. 17), non-nested generalised exemplars (NNge) are used in an attempt to further improve this proposed approach to the automatic identification of polymorphic malware. By representing polymorphic malware as DNA (or base sequences), it enables not only sequence alignment approaches but also data mining methods to be used. It is likely that rule induction techniques, well established in data mining, can be used to extract rule-based virus meta-signatures. The key differences between the previous proposed approaches (Chapters 4 and 5) and current chapter are as follows:

- Previous work adopted left-to-right string matching techniques to find the most optimally-conserved meta-signatures. The work presented in this chapter adopts a rule-based or top-down approach that attempts to find underlying patterns.
- Previous work generated equal length consensuses using sequence alignment techniques, whereas the current chapter generates variable length consensuses adopting a variable length data mining technique (NNge).
- Previous work adopted pairwise alignment techniques for extracting signatures which only allowed alignment of two viral sequences at a time taking into account only the information available in the sequence pair. This work allows all sequences to be used to extract signatures and so takes into account all the information in all the sequences at the same time, including both family generic and variant specific information.

6.1. Introduction

In this chapter, the focus is on polymorphic malware generated through modification in the encryptor and decryptor. Whereas the previously explored syntactic approaches (Chapters 4 and 5) had the aim of generating string templates for signatures, the aim of this work is to investigate whether such string templates are themselves rule-based and could, therefore, be derived directly using data mining algorithms in conjunction with

sequence alignment algorithms. If it is possible to generate a rule-based signature automatically then it should be possible to automatically create a signature that can detect entirely new variants that have not previously been encountered. In other words, it is possible to construct 'pre-emptive' AVSs that already know, to some extent, what future variants of a virus may look like based on encountering known (P_k) variants of that virus. Only JS.Cassandra virus family – known (P_k) and unknown (P_x) variants are used in this chapter due to sequence length restrictions put by the alignment tool (see page no. 18 for more details) as some of the experiments in this chapter adopt sequence alignment to extract the meta-signatures. Other two polymorphic virus families are not considered in this chapter due to its overly long sequences.

6.2. Objectives of this chapter

Three sets of experiments were performed for two main reasons. Firstly, to examine the effectiveness of the rule-based classifier (NNge) employed here, all the three sets of experiments were solely performed by using the rule-based classifier (NNge) as the primary approach to obtain more divergent, meaningful and valuable (single) rules each time. Secondly, to test the capability of this rule-based classifier (NNge) to handle larger sequences, the length of sequences in each set of experiments was increased either by using a different more verbose representational approach (i.e. DNA) or by multiple sequence alignment.

In the first set of experiments, hexadecimal (hex) dumps of JS.Cassandra and its known variants – P_k (training set) were loaded into Weka, whereas, in the second set of experiments, hex dumps of JS.Cassandra virus and its known variants – P_k (training set) were represented as DNA sequences, thereby increasing the length of (DNA) sequences by two times (from its original hex length) and then loaded into Weka (Frank, Hall, & Witten, 2016), exclusively to obtain (NNge) rule-based meta-signatures in both sets of experiments. In the third set of experiments, hex dumps of JS.Cassandra virus and its known variants – P_k (training set) were represented as DNA sequences and then multiply-aligned using MAFFT - a multiple sequence alignment online tool (Katoh, Misawa, Kuma, & Miyata, 2002; Katoh & Standley, 2013; Katoh, 2018) thereby increasing the length of the sequences by six-fold. Multiple sequence alignment was used for two main reasons. Firstly, because most data mining rule-based approaches assume fixed length sequences (Xinguang, et al., 2009) and all of the generated hex dumps of JS.Cassandra virus were of variable sequence lengths. Pairwise sequence alignment was not used in

this stage because pairwise alignment does not produce fixed length sequences, each run of pairwise alignment produces a variable length sequence. Secondly, as the first set of experiments (using hex dumps) generated NNge rules with 100% accuracy, and the second set of experiments (using DNA representation) generated NNge rules with 100% inaccuracy, DNA represented sequences from the second set of experiments were multiply-aligned, primarily to obtain NNge rules with 100% accuracy (third set of experiments). The three sets of experiments discussed here were performed with a matching number of unchanged instances (i.e. 22 instances) in every set of experiments. For every set of experiments discussed here, the process of pairwise sequence alignment was employed at a later stage. Pairwise sequence alignment was employed primarily to extract common substrings/meta-signatures from the (NNge) single rules.

A - First Set of Experiments

6.3. Experiment I - Identification of viral variants using NNge rule extraction from variants in hexadecimal format: Systems and Methods

The method for Experiment I consists of six steps (see Figure 6.1). As usual, the steps involving download of JS.Cassandra polymorphic virus and variants and hexadecimal dump extraction were performed on a stand-alone system. Network connectivity was used at the testing stage as the previous approaches.

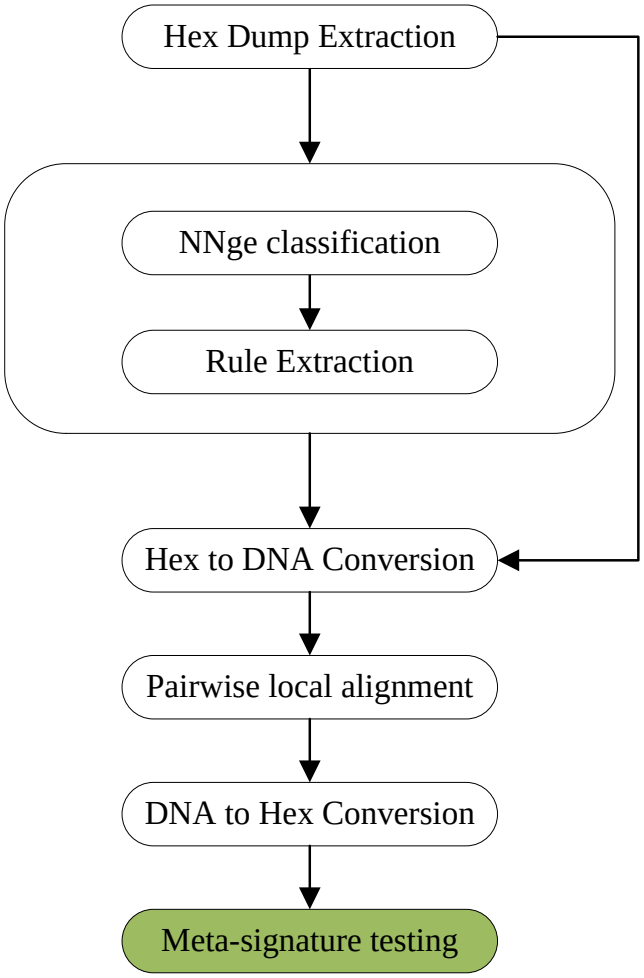

Figure 6.1: The 'Identifying Viral Polymorphic Malware Variants using Data Mining Rule-Based NNge Classification Algorithm (Experiment I)' method comprising of six steps.

A complete description of the method is supplied below.

6.3.1 Hex Dump Extraction

Step-1: As for previous experiments, 11 variants of JS.Cassandra were extracted including the original virus (P_s). In this case, since most data mining approaches assume

two or more classes, an additional 11 non-malicious/non-payload (P_u) files were generated by eliminating their main polymorphic engines. As for earlier experiments, the uniqueness of all variants was verified by generating their CRC32b hash value (see Appendix H – page no. 250). 'VirusTotal' was used to ensure that malicious activity was maintained in the 11 malicious (P_k) variants and eliminated in the 11 non-malicious (P_u) variants.

Table 6.1 gives the detection ratio, based on the 56 AVSs using 'VirusTotal'. Only four of the 56 AVSs detected one or more of the non-malicious (P_u) variant files (Table 6.1). It is likely that variants 1, 3, 4 and 9 were detected as malicious as they still had their polymorphic functions in place but no payload.

Table 6.1: Detection Ratio for each JS.Cassandra variant based on the 56 AVSs in 'VirusTotal'.

Malicious (P_k) Filename	**Detection Ratio**	**Non-Malicious (P_u) Filename**	**Detection Ratio**
JS.Cassandra.js (Original Malicious Virus – P_s)	40/56	JS.Cassandra_NP.js (Non-Malicious Virus – P_u)	0/56
v_000.js (Malicious Variant 1 – P_k)	19/56	v_000_NP.js (Non-Malicious Variant 1 – P_u)	1/56
v_001.js (Malicious Variant 2 – P_k) – for use in Step-4 and Step-5	12/56	v_002_NP.js (Non-Malicious Variant 2 – P_u)	0/56
v_002.js (Malicious Variant 3 – P_k)	22/56	v_003_NP.js (Non-Malicious Variant 3 – P_u)	1/56
v_003.js (Malicious Variant 4 – P_k)	19/56	v_004_NP.js (Non-Malicious Variant 4 – P_u)	1/56
v_004.js (Malicious Variant 5 – P_k)	22/56	v_005_NP.js (Non-Malicious Variant 5 – P_u)	0/56
v_005.js (Malicious Variant 6 – P_k)	21/56	v_006_NP.js (Non-Malicious Variant 6 – P_u)	0/56
v_006.js (Malicious Variant 7 – P_k)	21/56	v_007_NP.js (Non-Malicious Variant 7 – P_u)	0/56
v_007.js (Malicious Variant 8 – P_k)	20/56	v_008_NP.js (Non-Malicious Variant 8 – P_u)	0/56
v_008.js (Malicious Variant 9 – P_k)	20/56	v_009_NP.js (Non-Malicious Variant 9 – P_u)	1/56
v_009.js (Malicious Variant 10 – P_k)	20/56	v_010_NP.js (Non-Malicious Variant 10 – P_u)	0/56
v_010.js (Malicious Variant 11 – P_k)	20/56	–	–

Hex dumps were then extracted using 'sigtool'.

6.3.2 Data Mining

Step-2: A rule-based classification algorithm known as NNge was employed in this step to extract rules from the variants in hexadecimal format. As stated earlier, NNge first introduced by Martin (1995) is a nearest neighbour algorithm which generalises by merging exemplars and forming hyperrectangles in feature space that represent conjunction rules (if-then rules) with internal disjunction. The learning is incremental;

each example is first classified and then generalised by joining the example to its nearest neighbour, either a single instance or a hyperrectangle, in the same class. Each hyperrectangle is converted into a production rule. NNge was chosen for this research because the exclusive generalised exemplars produced result in a useful set of rules and has been proven to reduce classification time without sacrificing accuracy.

Rule generation using the NNge classifier was conducted using its implementation in Weka.

An ARFF (Attribute-Relation File Format) file was created which contained the hex dump sequences for the 22 JS.Cassandra variants. Since most data mining approaches assume fixed length sequences (Xinguang, et al., 2009), the variable length 22 hex dump sequences were converted into fixed length sequences by adding the letter 'x' at the end of short sequences. In total, the ARFF file consisted of 24,565 attributes and two classes (malicious and non-malicious). The file size of the ARFF file was 2.49 MB. The NNge classifier was trained on the full dataset. No cross-validation was performed in this step as the sole purpose of this chapter is to extract (single) consequential rules and hence, only the training on the full dataset was conducted. Figures F.1 and F.2 (see Appendix F – page no. 238) are screenshots of the pre-process panel, the classifier model and evaluation information (with/on full training set) within the classifier panel obtained from Weka during the generation of NNge rules in this step. Figure F.3 (see Appendix F – page no. 240) is a screenshot of the visualize panel showing 275 individual plot matrices between pos1-pos25 and pos13633-pos13643, where, blue circles are malicious ('m') and red circles are non-malicious ('nm'). Two NNge rules (one for each class) were generated with an accuracy of 100%. A partial segment of two NNge (hex) rules obtained in this step for the malicious (m), and 11 non-malicious (nm) hex sequences are shown below:

Malicious (m) *- class m IF : pos1 in {2,6} ^ pos2 in {0,3} ^ pos3 in {6,7} ^ pos4 in {a,b,1,2,3,7,9} ^ pos5 in {6,7} ^ pos6 in {a,e,1,2,3,5,6,7,9} ^ pos7 in {6,7}...and so on.*

Non-Malicious (nm) *- class nm IF : pos1 in {2,6,7} ^ pos2 in {f,6} ^ pos3 in {2,6,7} ^ pos4 in {f,1,5} ^ pos5 in {2,6,7} ^ pos6 in {e,0,2} ^ pos7 in {2,6,7}...and so on.*

6.3.3 Rule Extraction

Step-3: In this step, two strings in hex format (one each for malicious and non-malicious) were extracted from the two NNge rules. For example, for (m) the first substring at pos 1 becomes the first substring in the new NNge rule extracted string resulting as follows:

'*260367ab1237967ae123567967...*'. The length of the malicious string was 246,676 hex characters, whereas, the length of the non-malicious string was 74,498 hex characters. Since the two generated NNge rules were in the form of associations with position numbers, only hex data (by excluding the letter 'x') from the two NNge rules were extracted. The extracted NNge hex data were then converted into DNA sequences for the process of pairwise (local) sequence alignment.

6.3.4 Hex to DNA Conversion

Step-4: The extracted hex dump sequences were converted into binary. Conversion of hexadecimal into binary code was accomplished as per previous experiments (see Section 4.3.2 for the conversion rules – page no. 66). Three of the variants JS.Cassandra.js , v_000.js and v_001.js, were then converted into DNA sequences using the previously established conversion rules (see Section 4.3.2 – page no. 66). A short instance of the conversion of 16-bit binary code into 8 DNA characters is presented below:

1001110100101010 (16-bit binary code)

GCTCAGGG (8 DNA characters)

6.3.5 Pairwise Local Sequence Alignment

Step-5: The two converted NNge DNA sequences were pairwisely-aligned with polymorphic (converted) DNA sequences of original (P_s) JS.Cassandra virus (JS.Cassandra.js) as well as its malicious variant 1 (v_000.js) and malicious variant 2 (v_001.js), successively, using SWA and ID substitution matrix. The processes of pairwise alignments were primarily performed to extract the common substrings/meta-signatures, which will be used to detect the original (P_s) malware and all its available variants (P_k and P_x) of the JS.Cassandra polymorphic family. In total, nine meta-signatures in DNA representation were extracted in this set of experiments (see Table 6.2 for more details).

Four meta-signatures from the alignment with the DNA sequence extracted using the NNge rule for malicious (P_k) variants and five meta-signatures from the DNA sequence extracted from the NNge rule for the non-malicious (P_u) variants. Table 6.2 gives the sequence lengths of these meta-signatures. where, (1) denotes that the meta-signatures were obtained from the first set of experiments. The minimum and maximum sequence lengths of meta-signatures obtained from the first set of alignments involving malicious

NNge sequence were 12 and 125, respectively, with a mean (sum, median and standard deviation of 339, 101 and 52.411, respectively) of 84.75 for four signatures in their DNA representation. The minimum and maximum sequence lengths of signatures obtained from the second set of alignments involving non-malicious NNge sequence were 12 and 59, respectively, with a mean (sum, median and standard deviation of 151, 18 and 22.543, respectively) of 30.2 for five signatures in their DNA representation (Table 6.2).

Table 6.2: Sequence lengths of all the nine extracted meta-signatures (i.e. common substrings) in its DNA representation obtained in Step-5.

Class	Pairwise Alignment	Meta-Signature (1)	Sequence Length
Malicious (*m*)	Malicious DNA NNge Rule and Original Malicious JS.Cassandra (JS.Cassandra.js) DNA Sequence	MS1 (1)	121
	Malicious DNA NNge Rule and JS.Cassandra Malicious Variant 1 (v_000.js) DNA Sequence	MS2 (1)	12
		MS3 (1)	125
	Malicious DNA NNge Rule and JS.Cassandra Malicious Variant 2 (v_001.js) DNA Sequence	MS4 (1)	81
Non-Malicious (*nm*)	Non-Malicious DNA NNge Rule and Original Malicious JS.Cassandra (JS.Cassandra.js) DNA Sequence	MS5 (1)	50
	Non-Malicious DNA NNge Rule and JS.Cassandra Malicious Variant 1 (v_000.js) DNA Sequence	MS6 (1)	12
		MS7 (1)	59
	Non-Malicious DNA NNge Rule and JS.Cassandra Malicious Variant 2 (v_001.js) DNA Sequence	MS8 (1)	12
		MS9 (1)	18

6.3.6 DNA to Hex Conversion and Meta-Signature Testing

Step-6: In this final step, the nine meta-signatures (obtained in Step-5) in DNA sequence representation were converted back into hexadecimal format.

6.4. Summary – First Set of Experiments

To summarise the first set of experiments, 351 polymorphic malicious (P_k) variants of JS.Cassandra virus along with the original (P_s) JS.Cassandra virus (a total of 352 polymorphic malicious files) were downloaded from the official virus author's web page (SPTH, 2004; SPTH, 2015). 11 out of the 352 polymorphic malicious (P_k) files (including the original (P_s) JS.Cassandra virus) were selected for the experiments (see Table 6.1 for the list of selected malicious variants). One of the additional polymorphic malicious (P_k) file (i.e. v_001.js) out of the 352 malicious (P_k) files was selected for use in Step-4 and Step-5. An additional 11 non-malicious/non-payload (P_u) files (ignoring the additional polymorphic malicious file) were produced by eliminating their main polymorphic engines, making a total number of 22 files. Hex dumps were extracted from all the 22

files (including the additional polymorphic malicious – P_k file for use in Step-4 and Step-5), and they were all of the variable sequence lengths. All the 22 variable length hex sequences (ignoring the additional polymorphic malicious – P_k file) were converted into fixed length sequences by adding the letter 'x' towards the end of each sequence until all the 22 sequences were of equal length. As stated earlier, this conversion was necessary as most of the rule-based data mining classifiers accept fixed length sequences (Xinguang, et al., 2009). All the converted fixed length hex sequences were used to create an ARFF file (with two classes – one for malicious and the other one for non-malicious) and then later fed into Weka (see Figure F.1) (Frank, Hall, & Witten, 2016). Using the fed ARFF file as the training set, NNge rule-based classifier was conducted, which generated two long NNge rules with 100% accuracy (see Figure F.2). One NNge rule was generated for the malicious class, whereas, the other NNge rule was generated for the non-malicious class (see Figure F.2). Since the two generated NNge rules were in the form of associations with position numbers, only hex data (by excluding the letter 'x') from the two NNge rules were extracted. The extracted NNge hex data were then converted into DNA sequences for the process of pairwise (local) sequence alignment. Furthermore, 11 of the polymorphic malicious hex sequences, most importantly, JS.Cassandra original (P_s) virus and its malicious variant 1 (plus the additional polymorphic malicious variant 2) were converted into DNA sequences, also for the process of pairwise sequence alignment. As stated earlier, this conversion was essential as the pairwise alignment of hex data cause issues to string matching algorithms due to algorithmic complications in managing numeric data and code. The two converted NNge DNA sequences were pairwisely-aligned with polymorphic (converted) DNA sequences of original (P_s) JS.Cassandra virus (JS.Cassandra.js) as well as its malicious variant 1 (v_000.js) and malicious variant 2 (v_001.js), successively, using SWA and ID substitution matrix. The processes of pairwise alignments were primarily performed to extract the common substrings/meta-signatures, which will be used to detect the original (P_s) malware and all its available (P_k and P_x) variants of the JS.Cassandra polymorphic family. In total, nine meta-signatures in its DNA representations were extracted in this set of experiments (see Table 6.2 for more details). These meta-signatures (in its DNA representations) were converted back into a hexadecimal format for testing against the JS.Cassandra polymorphic malware family (see Table 6.6 and Figures 6.4 to 6.6 in Section 6.9 for more details – page nos. 141, 144-145). This conversion of DNA into hex was necessary as all the malware (and non-malware) files can be represented in hex, as it is the traditional human-decipherable representation of each byte's value.

B - Second Set of Experiments

6.5. Experiment II - Identification of viral variants using NNge rule extraction from variants in DNA format: Systems and Methods

The method for Experiment II consists of six steps (see Figure 6.2).

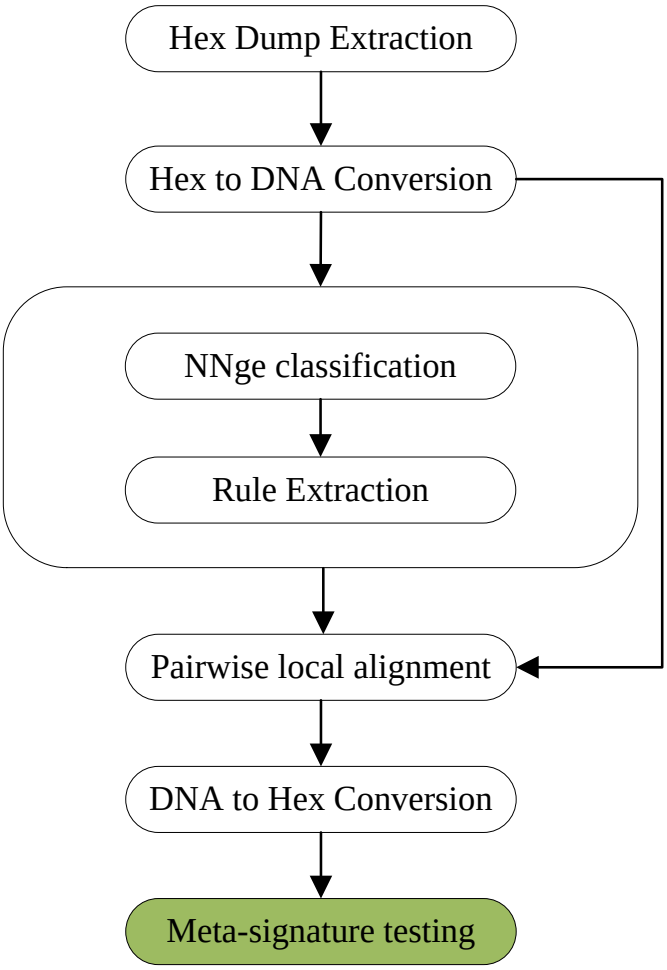

Figure 6.2: The 'Identifying Viral Polymorphic Malware Variants using Data Mining Rule-Based NNge Classification Algorithm (Experiment II)' method comprising of six steps.

The same procedure as Experiment I was used along with the same JS.Cassandra variants were used the only difference is that the variants were converted into DNA format prior to NNge rule generation. The conversion to DNA format was undertaken as normal using the DNA representational method as detailed in Section 4.3.2 (see page no. 66).

As for Experiment I, fixed length sequences were here created by adding the letter 'X' at the end of each sequence to the length of the longest variant. In total, the resultant 3.87 MB ARFF file contained 49,129 attributes and two class labels (malicious and non-

malicious). The final and error-free version of ARFF file was loaded into Weka, and NNge classification undertaken using all the data as the training set. After the first iteration, two NNge rules (one for each class) was generated in under seven minutes. Figures F.4 and F.5 (see Appendix F – page no. 241) are screenshots of the preprocess panel and the classifier model and evaluation information (with/on full training set) within the classifier panel obtained from Weka during the generation of NNge rules in this step (Step-3). Figure F.6 (see Appendix F – page no. 243) is a screenshot of the visualize panel showing 275 individual plot matrices between pos1-pos25 and pos36663-pos36673, where, blue circles are malicious ('M'), and red circles are non-malicious ('NM'). A partial segment of the two NNge (DNA) rules are shown below:

Malicious (M) *- class M IF : pos1 in {A,C} ^ pos2 in {G} ^ pos3 in {A} ^ pos4 in {A,T} ^ pos5 in {C} ^ pos6 in {T,G} ^ pos7 in {A,G,C} ^ pos8 in {T,G,C}...and so on.*

Non-Malicious (NM) *- class NM IF : pos1 in {A,C} ^ pos2 in {T,G} ^ pos3 in {T,C} ^ pos4 in {T,G} ^ pos5 in {A,C} ^ pos6 in {T,G} ^ pos7 in {A,T,C}...and so on.*

Rules were extracted in the same way as for Experiment I from these two NNge rules as in Experiment I the sub-sequences in each position were concatenated as illustrated here for the Malicious class: '*ACGAATCTGAGCTGC...*'.

The sequence length of the malicious NNge DNA string was 132,103 bases, whereas, the sequence length of non-malicious NNge DNA string was 41,670 bases. Pairwise local alignment was then performed using SWA and the ID matrix in a process similar to that described for Experiment I. In this case alignment was conducted between the NNge DNA string for the malicious class rule and the malicious JS.Cassandra variants in DNA format one by one. And this was again repeated for the NNge DNA string created based on the NNge rule for non-malicious files.

Overall, 14 common substrings (i.e. meta-signatures) were obtained in this step (Step-5). Nine meta-signatures were obtained from the alignment between the sequence extracted from the malicious DNA NNge rule and the three malicious DNA sequences (i.e. original (P_s) JS.Cassandra, variant 1 (P_k) and variant 2 (P_k) sequences as for Experiment I), whereas, five meta-signatures were obtained from the alignment between non-malicious extracted DNA NNge rule and the three malicious DNA sequences. Table 6.3 shows the sequence lengths of all the 14 extracted meta-signatures obtained in this step (Step-5), where, (2) denotes that the meta-signatures were obtained from the second set of

experiments. The minimum and maximum sequence lengths of meta-signatures obtained from the first set of alignments involving malicious NNge sequence were 12 and 125, respectively, with a mean (sum, median and standard deviation of 404, 14 and 49.604, respectively) of 44.89 for nine signatures in their DNA representation. The minimum and maximum sequence lengths of signatures obtained from the second set of alignments involving non-malicious NNge sequence were 12 and 59, respectively, with a mean (sum, median and standard deviation of 152, 18 and 22.345, respectively) of 30.4 for five signatures in their DNA representation (Table 6.3).

Table 6.3: Sequence lengths of the extracted meta-signatures in DNA representation.

Class	Pairwise Alignment	Meta-Signature (2)	Sequence Length
Malicious (*M*)	Malicious DNA NNge Rule and Original Malicious JS.Cassandra (JS.Cassandra.js) DNA Sequence	MS1 (2)	14
		MS2 (2)	12
		MS3 (2)	121
	Malicious DNA NNge Rule and JS.Cassandra Malicious Variant 1 (v_000.js) DNA Sequence	MS4 (2)	14
		MS5 (2)	12
		MS6 (2)	125
	Malicious DNA NNge Rule and JS.Cassandra Malicious Variant 2 (v_001.js) DNA Sequence	MS7 (2)	13
		MS8 (2)	12
		MS9 (2)	81
Non-Malicious (*NM*)	Non-Malicious DNA NNge Rule and Original Malicious JS.Cassandra (JS.Cassandra.js) DNA Sequence	MS10 (2)	50
	Non-Malicious DNA NNge Rule and JS.Cassandra Malicious Variant 1 (v_000.js) DNA Sequence	MS11 (2)	59
	Non-Malicious DNA NNge Rule and JS.Cassandra Malicious Variant 2 (v_001.js) DNA Sequence	MS12 (2)	13
		MS13 (2)	12
		MS14 (2)	18

In the final step (Step-6), the 14 meta-signatures (Table 6.3) in DNA sequence representation were converted back into hexadecimal format. The converted nine hex meta-signatures were tested against the JS.Cassandra original (P_s) virus and all its 351 known (P_k) polymorphic malware variants using 'clamscan'.

6.6. Summary – Second Set of Experiments

To summarise the second set of experiments, 351 polymorphic malicious variants of JS.Cassandra virus along with the original (P_s) JS.Cassandra virus (a total of 352 polymorphic malicious files) were retained from the first set of experiments. All the 22 extracted hex dumps and the subsequent 22 converted DNA sequences of JS.Cassandra virus (including DNA sequence of the additional polymorphic malicious file i.e. v_001.js for use in Step-5) were also retained from the first set of experiments, for the process of

data mining and pairwise sequence alignment. As stated earlier, this conversion to DNA was essential as the pairwise alignment of hex data cause issues to string matching algorithms due to algorithmic complications in managing numeric data and code. All the 22 converted variable length DNA sequences (ignoring the additional polymorphic malicious file) were converted into fixed length DNA sequences by adding the letter 'X' towards the end of each sequence until all the 22 DNA sequences were of equal length. As stated earlier, this conversion was necessary as most of the rule-based data mining classifiers accept fixed length sequences (Xinguang, et al., 2009). All the converted fixed length DNA sequences were used to create an ARFF file (with two classes – one for malicious and the other one for non-malicious) and then later fed into Weka (see Figure F.4) (Frank, Hall, & Witten, 2016). Using the fed ARFF file as the training set, NNge rule-based classifier was started, which generated two long NNge rules (see Figure F.5). One NNge rule was generated for the malicious class, whereas, the other NNge rule was generated for the non-malicious class (see Figure F.5). Since the two generated NNge rules were in the form of associations with position numbers, only DNA data (by excluding the letter 'X') from the two NNge rules were extracted for the process of pairwise (local) sequence alignment. The two extracted NNge DNA (data) sequences were pairwisely-aligned with polymorphic (converted) DNA sequences of original (P_s) JS.Cassandra virus (JS.Cassandra.js) as well as its malicious variant 1 (v_000.js) and malicious variant 2 (v_001.js), successively, using SWA and ID substitution matrix. The processes of pairwise alignments were primarily performed to extract the common substrings/meta-signatures. In total, 14 meta-signatures in its DNA representations were extracted in this set of experiments (see Table 6.3 for more details). These meta-signatures (in its DNA representations) were converted back into a hexadecimal format for testing against the JS.Cassandra polymorphic malware family (see Table 6.6 and Figures 6.4 to 6.6 in Section 6.9 for more details – page nos. 141, 144-145). This conversion of DNA into hex was necessary as all the malware (and non-malware) files can be represented in hex, as it is the traditional human-decipherable representation of each byte's value.

C - Third Set of Experiments

6.7. Experiment III - Identification of viral variants using NNge rule extraction from multiply aligned variants in DNA format: Systems and Methods

The method for Experiment III consists of seven steps (see Figure 6.3).

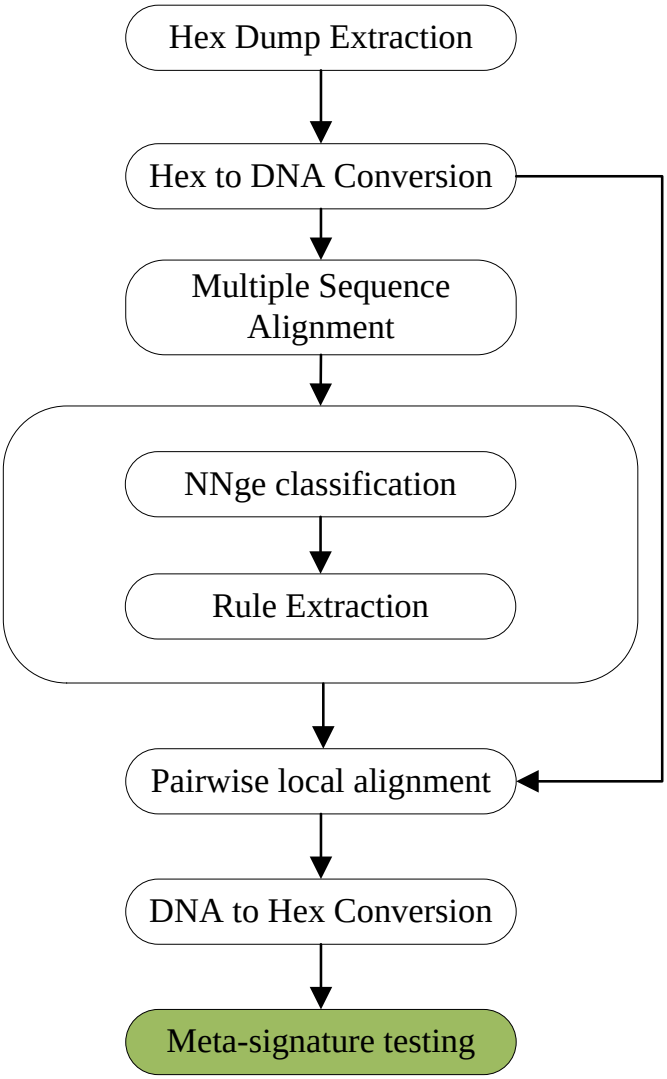

Figure 6.3: The 'Identifying Viral Polymorphic Malware Variants using Data Mining Rule-Based NNge Classification Algorithm (Experiment III)' method comprising of seven steps.

The same 22 hexadecimal dumps, 11 malicious (P_k) variants and 11 non-malicious (P_u) variants of JS.Cassandra were employed in this experiment as were used in Experiments I and II (see Section 6.3.1 – page no. 124). As for Experiment II, these 22 hexadecimal dumps were converted into DNA sequences (Step-2) (see Section 6.3.4 – page no. 127).

This experiment takes a different approach than Experiments I and II, to dealing with the need for fixed length sequences in order to generate rules using a data mining approach. Multiple sequence alignment is undertaken prior to NNge rule generation to convert the variable length sequences into fixed length sequences by inserting gaps. In Step-3, a multiple sequence alignment using MAFFT (Katoh, Misawa, Kuma, & Miyata, 2002; Katoh & Standley, 2013; Katoh, 2018) was conducted on the 22 DNA sequences. All the gaps introduced at this stage were substituted by the letter 'X'. Figures F.7 and F.8 (see Appendix F – page no. 244) are screenshots of the preprocess panel and the classifier model and evaluation information (with/on full training set) within the classifier panel obtained from Weka during the generation of NNge rules in this step (Step-4). Figure F.9 (see Appendix F – page no. 246) is a screenshot of the visualize panel showing 275 individual plot matrices between pos1-pos25 and pos47087-pos47097, where, blue circles are malicious ('M'), and red circles are non-malicious ('NM').

In **Step-4**, the same NNge classification was undertaken using Weka. The data was converted into Weka's ARFF file format. The 7.38MB file consisted of 93,438 attributes and two classes malicious and non-malicious in this ARFF file. Three NNge rules (one for the malicious class and two for the non-malicious class) were generated with an accuracy of 100% in under 33 minutes. A partial segment of each of these NNge rules are shown below:

Malicious (M) *- class M IF : pos1 in {A,X} ^ pos2 in {G,X} ^ pos3 in {A,X} ^ pos4 in {A,X} ^ pos5 in {C,X} ^ pos6 in {T,G,X} ^ pos7 in {A,G,X} ^ pos8 in {T,G,C,X} ^ pos9 in {C,X} ^ pos10 in {T,G,X}...and so on.*

Non-Malicious 1 (NM1) *- class NM IF : pos1 in {X} ^ pos2 in {X}… ^ pos96 in {T,X} ^ pos97 in {A,X} ^ pos98 in {G,X} ^ pos99 in {A,X} ^ pos100 in {A,X} ^ pos101 in {C,X} ^ pos102 in {T,G,X} ^ pos103 in {G,C,X}...and so on.*

Non-Malicious 2 (NM2) *- class NM IF : pos1 in {X} ^ pos2 in {X}… ^ pos1294 in {X} ^ pos1295 in {C} ^ pos1296 in {A} ^ pos1297 in {G} ^ pos1298 in {T} ^ pos1299 in {C} ^ pos1300 in {A} ^ pos1301 in {T}...and so on.*

In **Step-5**, three strings in DNA format were constructed based on each of these NNge rules. The process of extraction of strings from the rules is the same as detailed in Experiments I and II and any 'X' string extension characters were ignored. An example of this string extract process from the rules for NM1 is: '*TAGAACTGGC*...'. The sequence

length of the resultant malicious DNA string was 161,495, whereas, the sequence lengths of the non-malicious DNA strings were 59,740 (NM1) and 11,860 (NM2). Next, local pairwise sequence alignment between these DNA sequences extracted from each of the NNge rules and the polymorphic DNA sequences (obtained in Step-2) was performed using SWA and the ID matrix, as per Experiments I and II. In this step (Step-6), common substrings that are the meta-signatures for JS.Cassandra were extracted. In total 48 meta-signatures were obtained: 31 from the alignment between the DNA sequence extracted from the malicious NNge rule and the malicious DNA sequences, nine from the alignment of the NM1 NNge rule and eight from the NM2 NNge rule (see Table 6.4).

The minimum and maximum sequence lengths of meta-signatures obtained from the first set of alignments involving malicious NNge sequence were 12 and 117, respectively, with a mean (sum, median and standard deviation of 1315, 32 and 27.201, respectively) of 42.42 for 31 signatures in their DNA representation. The minimum and maximum sequence lengths of signatures obtained from the second set of alignments involving the first non-malicious NNge sequence (NM1) were 35 and 162, respectively, with a mean (sum, median and standard deviation of 566, 41 and 42.165, respectively) of 62.89 for nine signatures in their DNA representation. The minimum and maximum sequence lengths of signatures obtained from the third set of alignments involving the second non-malicious NNge sequence (NM2) were 29 and 120, respectively, with a mean (sum, median and standard deviation of 475, 45 and 38.127, respectively) of 59.37 for eight signatures in their DNA representation (Table 6.4).

In the final step (Step-7), the meta-signatures were converted from DNA sequence format into the hexadecimal format and then tested against JS.Cassandra and its known (P_k) variants using five AVSs.

Scan results for AVG, AntiVir, and F-Prot were obtained from an open source online tool known as ‘Gary’s Hood’ (Hood, 2016). ‘Gary’s Hood’ was used as it allows multiple files to be scanned at once using these AVSs. As previously discussed in earlier chapters, ESET AVS was installed on a private machine with a Windows operating system while ‘clamscan’ was installed on a private machine with a Linux Mint operating system. The tests were run using the ClamAV database and using the own generated (.ndb) databases containing the meta-signatures (see Appendix D – page no. 228 for more details). As usual, the databases of all the AVSs had the latest updates installed at the time of the experiments.

Table 6.4: Sequence lengths of all 48 extracted meta-signatures in DNA representation.

Class	Pairwise Alignment	Meta-Signature (3)	Sequence Length
Malicious (*M*)	Malicious DNA NNge Rule and Original Malicious JS.Cassandra (JS.Cassandra.js) DNA Sequence	MS1 (3)	21
		MS2 (3)	50
		MS3 (3)	102
		MS4 (3)	20
		MS5 (3)	79
	Malicious DNA NNge Rule and JS.Cassandra Malicious Variant 1 (v_000.js) DNA Sequence	MS6 (3)	21
		MS7 (3)	59
		MS8 (3)	110
		MS9 (3)	117
	Malicious DNA NNge Rule and JS.Cassandra Malicious Variant 2 (v_001.js) DNA Sequence	MS10 (3)	50
		MS11 (3)	29
		MS12 (3)	33
		MS13 (3)	32
		MS14 (3)	32
		MS15 (3)	15
		MS16 (3)	22
		MS17 (3)	32
		MS18 (3)	66
		MS19 (3)	33
		MS20 (3)	35
		MS21 (3)	28
		MS22 (3)	33
		MS23 (3)	12
		MS24 (3)	31
		MS25 (3)	35
		MS26 (3)	68
		MS27 (3)	26
		MS28 (3)	32
		MS29 (3)	32
		MS30 (3)	35
		MS31 (3)	25
Non-Malicious1 (*NM1*)	Non-Malicious (*NM1*) DNA NNge Rule and Original Malicious JS.Cassandra (JS.Cassandra.js) DNA Sequence	MS32 (3)	88
	Non-Malicious (*NM1*) DNA NNge Rule and JS.Cassandra Malicious Variant 1 (v_000.js) DNA Sequence	MS33 (3)	162
	Non-Malicious (*NM1*) DNA NNge Rule and JS.Cassandra Malicious Variant 2 (v_001.js) DNA Sequence	MS34 (3)	45
		MS35 (3)	35
		MS36 (3)	36
		MS37 (3)	81
		MS38 (3)	39
		MS39 (3)	41
		MS40 (3)	39
Non-Malicious2 (*NM2*)	Non-Malicious (*NM2*) DNA NNge Rule and Original Malicious JS.Cassandra (JS.Cassandra.js) DNA Sequence	MS41 (3)	120
		MS42 (3)	29
	Non-Malicious (*NM2*) DNA NNge Rule and JS.Cassandra Malicious Variant 1 (v_000.js) DNA Sequence	MS43 (3)	120
		MS44 (3)	29
	Non-Malicious (*NM2*) DNA NNge Rule and JS.Cassandra Malicious Variant 2 (v_001.js) DNA Sequence	MS45 (3)	38
		MS46 (3)	46
		MS47 (3)	44
		MS48 (3)	49

6.8. Summary – Third Set of Experiments

To summarise the third set of experiments, 351 polymorphic malicious variants of JS.Cassandra virus along with the original (P_s) JS.Cassandra virus (a total of 352 polymorphic malicious files) were retained from the first set of experiments. All the 22 extracted hex dumps and the subsequent 22 converted DNA sequences of JS.Cassandra virus (including DNA sequence of the additional polymorphic malicious file i.e. v_001.js for use in Step-5) were also retained from the first set of experiments, for the process of data mining and pairwise sequence alignment. As stated earlier, this conversion to DNA was essential as the pairwise alignment of hex data cause issues to string matching algorithms due to algorithmic complications in managing numeric data and code. All the 22 converted variable length DNA sequences (ignoring the additional polymorphic malicious file) were converted into fixed length DNA sequences by the process of multiple sequence alignment using MAFFT (Katoh, Misawa, Kuma, & Miyata, 2002; Katoh & Standley, 2013; Katoh, 2018). As stated earlier, this conversion was necessary as most of the rule-based data mining classifiers accept fixed length sequences (Xinguang, et al., 2009). All the converted fixed length DNA sequences were used to create an ARFF file (with two classes – one for malicious and the other one for non-malicious) and then later fed into Weka (see Figure F.7) (Frank, Hall, & Witten, 2016). Using the fed ARFF file as the training set, NNge rule-based classifier was started, which generated three long NNge rules with 100% accuracy (see Figure F.8). One NNge rule was generated for the malicious class, whereas, the other two NNge rules were generated for the non-malicious class (see Figure F.8). Since the three generated NNge rules were in the form of associations with position numbers, only DNA data (by excluding the letter 'X') from the two NNge rules were extracted for the process of pairwise (local) sequence alignment. The three extracted NNge DNA (data) sequences were pairwisely-aligned with polymorphic (converted) DNA sequences of original (P_s) JS.Cassandra virus (JS.Cassandra.js) as well as its malicious variant 1 (v_000.js) and malicious variant 2 (v_001.js), successively, using SWA and ID substitution matrix. The processes of pairwise alignments were primarily performed to extract the common sub-strings/meta-signatures, which will be used to detect the original (P_s) malware and all its available variants (P_k and P_x) of the JS.Cassandra polymorphic family. In total, 48 meta-signatures in its DNA representations were extracted in this set of experiments (see Table 6.4 for more details). These meta-signatures (in its DNA representations) were converted back into a hexadecimal format for testing against the JS.Cassandra polymorphic malware

family (see Table 6.6 and Figures 6.4 to 6.6 in Section 6.9 for more details – page nos. 141, 144-145). This conversion of DNA into hex was necessary as all the malware (and non-malware) files can be represented in hex, as it is the traditional human-decipherable representation of each byte's value.

6.9. Experimental Results

6.9.1 Comparison of the data mining results obtained from three sets of experiments as well as from other related and selected previous work

Table 6.5 presents the results of Experiments I, II and III and compares those results with the virus detection results presented in previously published works. In the case of the work by Chen et al., 2012b only the percentage of correctly detected and incorrectly detected instances were reported (as for J48 method) and in the case of Prabha and Kavitha (2012) no performance metrics were reported. In the case of Srakaew, Piyanuntcharatsr, and Adulkasem (2015) other overall performance metrics such as true positive rate, false positive rate, precision, recall and F1 score were not reported but instead reported metrics for the four individual datasets. These results are not presented here.

Experiments I and III gave results which outperformed those previously reported achieving 100% correctly classified instances and thus 0% incorrectly classified instances. Although Experiment II reported achieving 100% incorrectly classified instances and thus 0% correctly classified instances, the meta-signatures extracted in this experiment successfully identified the JS.Cassandra variants. Meta-signatures extracted in Experiment III were the most effective (~62%) of all followed by the meta-signatures extracted in Experiments I (~55%) and II (43%). The fact that the meta-signatures in DNA format performed better if the DNA sequences were aligned prior to rule mining (Experiment III vs. Experiment II) and extraction is reflected somewhat in the results of the work reported by Chen et al., 2012b where improved classification was observed if J48 classification was performed after a double alignment process.

Table 6.5: Comparison of the results of Experiments I-III with those reported previously for data mining approaches to malware detection reported in the literature.

Data Mining based Techniques		Correctly Classified Instances (%)	Incorrectly Classified Instances (%)	TP (True Positive) Rate	FP (False Positive) Rate	Precision	Recall	F1 Score
Experiment I		**100.00%**	**0.00%**	**1**	**0**	**1**	**1**	**1**
Experiment II		0.00%	100.00%	0	1	0	0	0
Experiment III		**100.00%**	**0.00%**	**1**	**0**	**1**	**1**	**1**
Chen, et al., 2012b – J48 before alignment	Training	85.00%	15.00%	-	-	-	-	-
	5-fold cross validation	60.00%	40.00%	-	-	-	-	-
	10-fold cross validation	63.33%	36.67%	-	-	-	-	-
	15-fold cross validation	68.33%	31.67%	-	-	-	-	-
	20-fold cross validation	60.00%	40.00%	-	-	-	-	-
Chen, et al., 2012b – J48 after double alignment	Training	96.67%	3.33%	-	-	-	-	-
	5-fold cross validation	78.33%	21.67%	-	-	-	-	-
	10-fold cross validation	66.67%	33.33%	-	-	-	-	-
	15-fold cross validation	70.00%	30.00%	-	-	-	-	-
	20-fold cross validation	63.33%	36.67%	-	-	-	-	-
(Kumar & Mishra, 2013)	Existing (known) dataset	95.9752%	4.0248%	0.96	0.094	0.962	0.96	0.959
	New (unknown) Dataset	86.6873%	13.3127%	0.867	0.275	0.872	0.87	0.858
(Prabha & Kavitha, 2012)	-	-	-	-	-	-	-	-
Statistical methods (Srakaew, Piyanuntcharatsr, & Adulkasem, 2015)	Reference Set	98.9167%	1.0833%	-	-	-	-	-
	Application Set	95.0477%	4.9523%	-	-	-	-	-
	10-fold cross validation	95.333%	4.667%	-	-	-	-	-
Abstract assembly method (Srakaew, Piyanuntcharatsr, & Adulkasem, 2015)	Reference Set	99.75%	0.25%	-	-	-	-	-
	Application Set	98.39%	1.661%	-	-	-	-	-
	10-fold cross validation	99.5%	0.5%	-	-	-	-	-

6.9.2 An evaluation of the state of the art AVSs and the meta-signatures on the detection of JS.Cassandra polymorphic malware and its variants

Table 6.6 presents the detection ratio obtained using the meta-signatures generated in Experiments I to II and five current state of the art AVSs. The malicious meta-signatures MS4 (1), MS9 (2), and MS26 (3) and the non-malicious meta-signatures MS35 (3) and MS37 (3) all successfully identified all 352 known (P_k) malicious polymorphic variants of the JS.Cassandra virus. None of the five state of the art AVSs fully identified all of these known (P_k) JS.Cassandra variants.

Table 6.6: Detection ratio using five state of the art AVSs and the 14 most effective malicious and 8 non-malicious meta-signatures from Experiments I to III with 'clamscan'.

Files Scanned	Metrics	Virus Identification Method				
		AVG	**AntiVir**	**ClamAV**	**ESET**	**F-Prot**
352 known (P_k) JS.Cassandra Malicious Variants	**Detection Ratio (Accuracy)**	312/352 (88.64%)	25/352 (7.10%)	340/352 (96.59%)	296/352 (84.09%)	4/352 (1.14%)
	Sensitivity/Recall	88.64%	7.10%	96.59%	84.09%	1.14%
	Specificity	0.00%	0.00%	0.00%	0.00%	0.00%
	Precision	100%	100%	100%	100%	100%
	F1 Score	93.97%	13.26%	98.26%	91.36%	2.25%
43 JS.Cassandra Non-Malicious (P_u) Variants	**Detection Ratio (Accuracy)**	0/43 (0.00%)	1/43 (2.32%)	0/43 (0.00%)	0/43 (0.00%)	0/43 (0.00%)
	Sensitivity/Recall	0.00%	0.00%	0.00%	0.00%	0.00%
	Specificity	100%	97.67%	100%	100%	100%
	Precision	0.00%	0.00%	0.00%	0.00%	0.00%
	F1 Score	0.00%	0.00%	0.00%	0.00%	0.00%
352 Random JavaScript Files	**Detection Ratio (Accuracy)**	0/352 (0.00%)	0/352 (0.00%)	0/352 (0.00%)	0/352 (0.00%)	0/352 (0.00%)
	Sensitivity/Recall	0.00%	0.00%	0.00%	0.00%	0.00%
	Specificity	100%	100%	100%	100%	100%
	Precision	0.00%	0.00%	0.00%	0.00%	0.00%
	F1 Score	0.00%	0.00%	0.00%	0.00%	0.00%
Files Scanned	**Metrics**	**Malicious MS1 (1) and MS3 (2) Non-Malicious MS41 (3) and MS43 (3)**	**Malicious MS3 (1) and MS6 (2)**	**Malicious MS7 (2)**	**Malicious MS4 (1) MS9 (2) Non-Malicious MS37 (3)**	**Malicious MS5 (3)**
352 known (P_k) JS.Cassandra Malicious Variants	**Detection Ratio (Accuracy)**	340/352 (96.59%)	85/352 (24.15%)	325/352 (92.33%)	**352/352 (100%)**	340/352 (96.59%)
	Sensitivity/Recall	96.59%	24.15%	92.33%	**100%**	96.59%
	Specificity	0.00%	0.00%	0.00%	**0.00%**	0.00%
	Precision	100%	100%	100%	**100%**	100%
	F1 Score	98.26%	38.90%	96.01%	**100%**	98.26%
43 JS.Cassandra Non-Malicious (P_u) Variants	**Detection Ratio (Accuracy)**	6/43 (13.95%)	1/43 (2.32%)	20/43 (46.51%)	43/43 (100%)	8/43 (18.60%)
	Sensitivity/Recall	0.00%	0.00%	0.00%	0.00%	0.00%
	Specificity	86.05%	97.67%	53.49%	0.00%	81.39%
	Precision	0.00%	0.00%	0.00%	0.00%	0.00%
	F1 Score	0.00%	0.00%	0.00%	0.00%	0.00%
352 Random JavaScript Files	**Detection Ratio (Accuracy)**	0/352 (0.00%)	0/352 (0.00%)	0/352 (0.00%)	0/352 (0.00%)	0/352 (0.00%)
	Sensitivity/Recall	0.00%	0.00%	0.00%	0.00%	0.00%
	Specificity	100%	100%	100%	100%	100%
	Precision	0.00%	0.00%	0.00%	0.00%	0.00%
	F1 Score	0.00%	0.00%	0.00%	0.00%	0.00%

Files Scanned	Metrics	Malicious MS9 (3)	Malicious MS15 (3)	Malicious MS20 (3)	Malicious MS24 (3)	Malicious MS26 (3)
352 known (P_k) JS.Cassandra Malicious Variants	**Detection Ratio (Accuracy)**	329/352 (93.46%)	344/352 (97.73%)	191/352 (54.26%)	202/352 (57.39%)	**352/352 (100%)**
	Sensitivity/Recall	93.46%	97.73%	54.26%	57.39%	**100%**
	Specificity	0.00%	0.00%	0.00%	0.00%	**0.00%**
	Precision	100%	100%	100%	100%	**100%**
	F1 Score	96.62%	98.85%	70.35%	72.93%	**100%**
43 JS.Cassandra Non-Malicious (P_u) Variants	**Detection Ratio (Accuracy)**	1/43 (2.32%)	29/43 (67.44%)	9/43 (20.93%)	14/43 (32.56%)	43/43 (100%)
	Sensitivity/Recall	0.00%	0.00%	0.00%	0.00%	0.00%
	Specificity	97.67%	32.56%	79.07%	67.44%	0.00%
	Precision	0.00%	0.00%	0.00%	0.00%	0.00%
	F1 Score	0.00%	0.00%	0.00%	0.00%	0.00%
352 Random JavaScript Files	**Detection Ratio (Accuracy)**	0/352 (0.00%)	0/352 (0.00%)	0/352 (0.00%)	0/352 (0.00%)	0/352 (0.00%)
	Sensitivity/Recall	0.00%	0.00%	0.00%	0.00%	0.00%
	Specificity	100%	100%	100%	100%	100%
	Precision	0.00%	0.00%	0.00%	0.00%	0.00%
	F1 Score	0.00%	0.00%	0.00%	0.00%	0.00%
Files Scanned	**Metrics**	**Malicious MS27 (3)**	**Non-Malicious MS7 (1) and MS11 (2)**	**Non-Malicious MS8 (1)**	**Non-Malicious MS12 (2)**	**Non-Malicious MS35 (3)**
352 known (P_k) JS.Cassandra Malicious Variants	**Detection Ratio (Accuracy)**	140/352 (39.77%)	339/352 (96.31%)	140/352 (39.77%)	325/352 (92.33%)	**352/352 (100%)**
	Sensitivity/Recall	39.77%	96.31%	39.77%	92.33%	**100%**
	Specificity	0.00%	0.00%	0.00%	0.00%	**0.00%**
	Precision	100%	100%	100%	100%	**100%**
	F1 Score	56.91%	98.12%	56.91%	96.01%	**100%**
43 JS.Cassandra Non-Malicious (P_u) Variants	**Detection Ratio (Accuracy)**	3/43 (6.98%)	37/43 (86.04%)	16/43 (37.21%)	20/43 (46.51%)	43/43 (100%)
	Sensitivity/Recall	0.00%	0.00%	0.00%	0.00%	0.00%
	Specificity	93.02%	13.95%	62.79%	53.49%	0.00%
	Precision	0.00%	0.00%	0.00%	0.00%	0.00%
	F1 Score	0.00%	0.00%	0.00%	0.00%	0.00%
352 Random JavaScript Files	**Detection Ratio (Accuracy)**	0/352 (0.00%)	0/352 (0.00%)	0/352 (0.00%)	0/352 (0.00%)	0/352 (0.00%)
	Sensitivity/Recall	0.00%	0.00%	0.00%	0.00%	0.00%
	Specificity	100%	100%	100%	100%	100%
	Precision	0.00%	0.00%	0.00%	0.00%	0.00%
	F1 Score	0.00%	0.00%	0.00%	0.00%	0.00%

Non-malicious MS7 (1) and non-malicious MS11 (2) identified 339 out of the 352 (with 96.31% accuracy) JS.Cassandra polymorphic malware files, whereas, non-malicious MS41 (3) and non-malicious MS43 (3) identified 340 out of the 352 (with 96.59% accuracy/detection rate) JS.Cassandra polymorphic malware (P_k) files, respectively. Malicious MS1 (1) and malicious MS3 (2) identified 340 out of the 352 (with 96.59% accuracy/detection rate) JS.Cassandra polymorphic malware (P_k) files, whereas, malicious MS15 (3) identified 344 out of the 352 (with 97.73% accuracy/detection rate) JS.Cassandra polymorphic malware (P_k) files, respectively.

Malicious MS4 (1), malicious MS9 (2), malicious MS26 (3), non-malicious MS35 (3) and non-malicious MS37 (3) were the only five meta-signatures that fully identified all 43 non-malicious JS.Cassandra polymorphic (P_u) files. These meta-signatures not only identified 352 malicious files successfully but also identified 43 non-malicious (P_u) files. As noted earlier, non-malicious (P_u) files still had some polymorphic functions intact inside. All the 43 non-malicious (P_u) files were still executable, but a few gave JavaScript runtime and compilation errors. These executable non-malicious (P_u) files might cause some serious potential threats, as the polymorphic functions inside these files might still make them polymorphic, in some cases. The results presented in Tables 6.1 and 6.6 shows that none of the existing AVSs fully identified these executable so-called non-malicious files (P_u) as malicious.

Malicious MS2 (1), non-malicious MS5 (1), non-malicious MS6 (1) and non-malicious MS9 (1) from Experiments 1 and malicious MS1 (2), malicious MS2 (2), malicious MS4 (2), malicious MS5 (2), malicious MS8 (2), non-malicious MS10 (2), non-malicious MS13 (2) and non-malicious MS14 (2) from Experiment II could not identify any of the 352 known (P_k) JS.Cassandra malicious variants, the 43 non-malicious (P_u) polymorphic variant files or the 352 randomly generated JavaScript files. Furthermore, malicious MS1 (3), malicious MS2 (3), malicious MS6 (3), malicious MS7 (3), malicious MS8 (3), malicious MS10 (3), malicious MS12 (3), malicious MS13 (3), malicious MS14 (3), malicious MS16 (3), malicious MS18 (3), malicious MS21 (3), malicious MS22 (3), malicious MS28 (3), malicious MS29 (3), malicious MS31 (3), non-malicious MS32 (3) and non-malicious MS36 (3) from Experiment III could not identify any of the JS.Cassandra 352 (known – P_k) malicious, the 43 non-malicious (P_u) polymorphic variant files or the 352 randomly generated JavaScript files. In total, 30 out of the 71 meta-signatures i.e. around 42.25% [30.98% malicious (22/71) and 11.27% non-malicious (8/71) meta-signatures] detected no variants from the three types of groups (i.e. malicious (P_k), non-malicious (P_u) and random). Specifically, four out of the nine meta-signatures i.e. 44.44% [11.11% malicious (1/9) and 33.33% non-malicious (3/9) meta-signatures] from first set of experiments and eight out of the 14 meta-signatures i.e. 57.14% [36% malicious (5/14) and 21.43% non-malicious (3/14) meta-signatures] from second set of experiments detected no variants from the three different types of groups. Moreover, 18 out of the 48 meta-signatures i.e. 37.50% [33.33% malicious (16/48) and 4.16% non-malicious (2/48) meta-signatures] from the third set of experiments detected no variants from the three different types of groups.

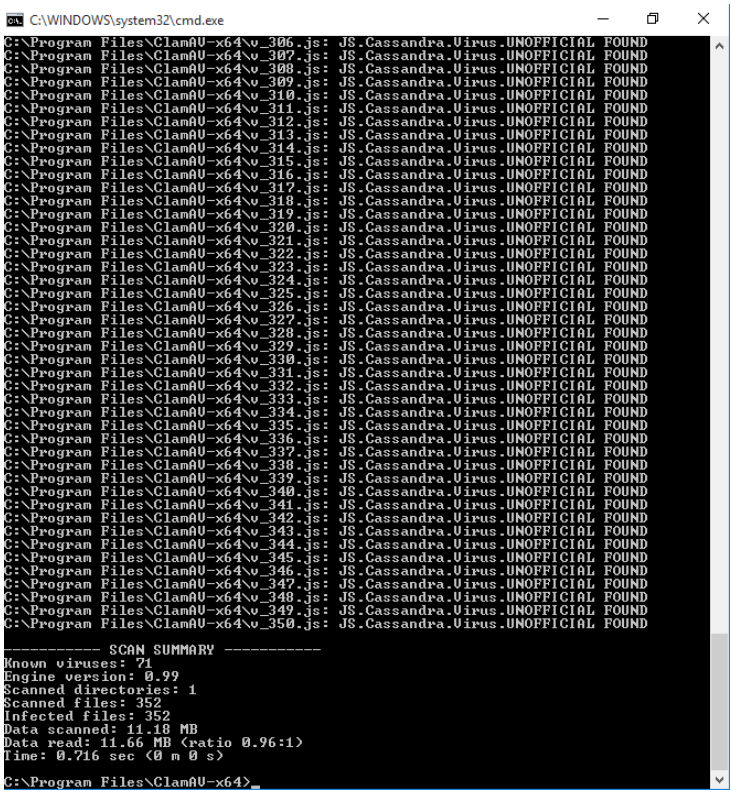

Figure 6.4: Screenshot of the scan result obtained from 'clamscan' antivirus scanner for 352 known (P_k) JS.Cassandra polymorphic malicious (P_k) variant files using the 71 meta-signatures.

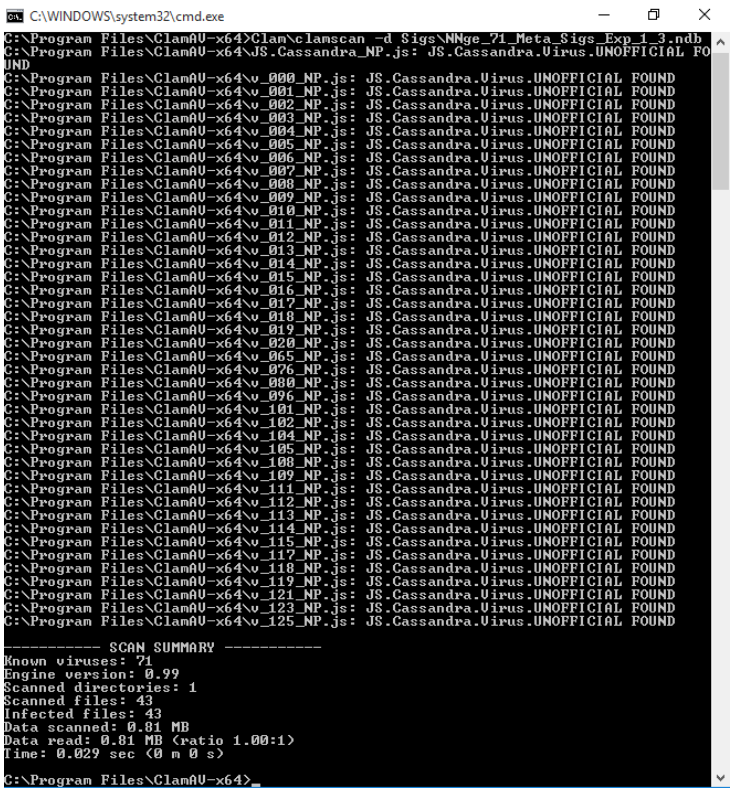

Figure 6.5: Screenshot of the scan result obtained from 'clamscan' antivirus scanner for 43 JS.Cassandra polymorphic non-malicious (P_u) variant files using the 71 meta-signatures.

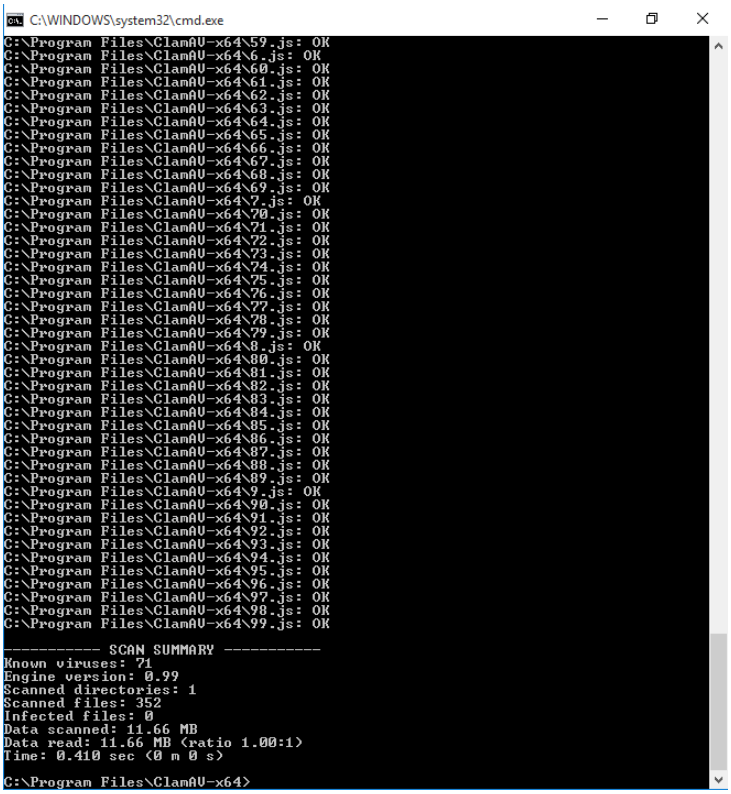

Figure 6.6: Screenshot of the scan result obtained from 'clamscan' antivirus scanner for 352 random JavaScript files using the 71 meta-signatures.

All the 71 meta-signatures (nine meta-signatures from the first set of experiments, 14 meta-signatures from the second set of experiments and 48 meta-signatures from the third set of experiments) were tested at once against the 352 known (P_k) JS.Cassandra polymorphic malicious files, 43 JS.Cassandra polymorphic non-malicious (P_u) variants and 352 random JavaScript files individually using 'clamscan' antivirus scanner.

Figures 6.4 to 6.6 are the screenshots of the scan results indicating that 352 out of the 352 malicious (P_k) files, 43 out of the 43 non-malicious (P_u) files and 0 out of the 352 randomly generated JavaScript files were successfully detected as infected by the 'clamscan' antivirus scanner using the 71 meta-signatures in 0.716 second, 0.029 second and 0.410 second, respectively.

The 71 meta-signatures were tested at once against the 100 unknown (P_x) JS.Cassandra malicious variants using Clamscan antivirus scanner by using the own generated (.ndb) database (see Appendix D – page no. 228 for more details). The uniqueness of these 100 new (P_x) malware variants was cross-checked by generating a CRC32b hash value for each variant, and no duplicates were found (see Appendix H – page no. 251 for more details). Table 6.7 gives the detection ratio obtained by testing the 71 meta-signatures generated in Experiments I to III and two current state of the art AVSs (ClamAV and

Bitdefender Total Security 2018) against the 100 new (P_x) JS.Cassandra variants. The state of the art ClamAV and Bitdefender Total Security 2018 AVSs had overall accuracies of 85% and 0%, respectively, and meta-signatures from Experiments I-III using Clamscan had overall accuracies of 100%, across all three experiments (see Table 6.7). Table 6.7 shows that all 100 (accuracy of 100%) JS.Cassandra unknown (P_x) variants were successfully detected by the Clamscan using the .ndb database.

Table 6.7: Detection ratio using two state of the art AVSs and the 71 meta-signatures obtained from Experiments I to III with Clamscan antivirus scanner.

Files Scanned	Metrics	Virus Identification Method				
		ClamAV	Bitdefender Total Security 2018	Nine Meta-Signatures (Experiment I)	14 Meta-Signatures (Experiment II)	48 Meta-Signatures (Experiment III)
100 unknown (P_x) JS.Cassandra Malicious Variants	Detection Ratio (Accuracy)	85/100 (85.00%)	0/100 (0.00%)	**100/100 (100.00%)**	**100/100 (100.00%)**	**100/100 (100.00%)**
	Sensitivity/Recall	85.00%	0.00%	**100.00%**	**100.00%**	**100.00%**
	Specificity	0.00%	0.00%	**0.00%**	**0.00%**	**0.00%**
	Precision	100.00%	100.00%	**100.00%**	**100.00%**	**100.00%**
	F1 Score	91.89%	0.00%	**100.00%**	**100.00%**	**100.00%**

45 out of the 71 meta-signatures were tested for false positives. Any duplicate meta-signatures along with meta-signatures that were six characters or below were removed. In total, 26 meta-signatures (i.e. 16 malicious and 10 non-malicious) were removed from the own generated (.ndb) database (see Appendix D – page no. 228 for more details). These meta-signatures were tested against the 352 known (P_k) variants, 43 non-malicious (P_u) variants, 100 new (P_x) variants and 18,127 clean files. The clean files contained a combination of 9000 PDF files, 500 Microsoft document files, 96 Linux files, 100 JAR files, 108 PDF files with embedded 3D videos, 200 RTF files and 8,123 Microsoft Windows files. These files were obtained from a BlogSpot called "contagio malware dump" (Parkour, 2013). Figure 6.7 shows the bar graphs of the detection scan results (accuracies) indicating that 352 of the 352 known (P_k) malicious variants, 43 of the 43 non-malicious (P_u) variants and 100 of the 100 unknown (P_x) malicious variants were successfully identified as infected by the Clamscan antivirus scanner using the 45 meta-signatures. Figure 6.7 shows that only 29 of the 18,127 clean files were detected as false positives (0.159% false positive rate) using the 45 meta-signatures, additionally, satisfying the false positive rate requisite of 0.1%.

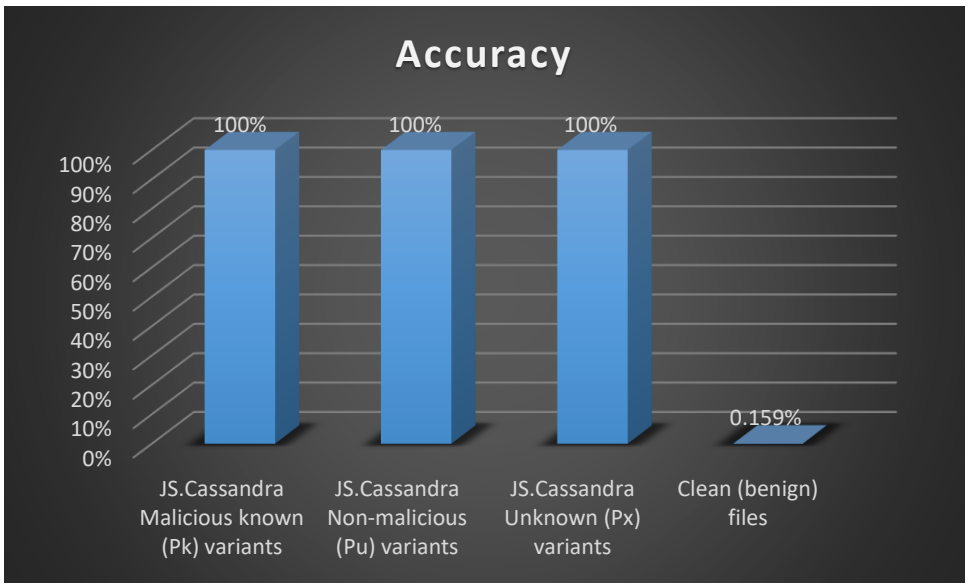

Figure 6.7: Bar graphs demonstrating the detection scan results (accuracies) of JS.Cassandra virus family and clean files using the 45 meta-signatures.

6.10. Summary

In this chapter, some of the limitations (discussed in Section 1.5.5 – see page no. 17) of previous work reported in this thesis (see Chapters 4 and 5) were addressed. A proposed syntactic structure approach was investigated and three sets of experiments conducted which involved various approaches to automatic signature extraction using the NNge classifier to generate rules that distinguish between malicious (P_k) and non-malicious (P_u) files.

Table 6.8 summarises the key features and steps involved in these three experiments. Experimental results from this testing process are provided in Table 6.6, Table 6.7, Figures 6.4 to 6.6 and Figure 6.7. The results show that this proposed string-based syntactic approach using an NNge rule generation and subsequent extraction and sequence alignment using SWA can successfully generate signatures which are capable of identifying the known (P_k) and unknown (P_x) polymorphic variants of the JS.Cassandra virus.

It was found from the experiments conducted in this chapter that Experiment III gave the highest number of successful meta-signatures than Experiments I and II. Experiment II gave the lowest number of successful meta-signatures. Not only did Experiment III gave the highest number of meta-signatures, but it also gave the highest number of effective meta-signatures. Moreover, Experiment III generated unseen meta-signatures that were not generated in Experiments I and II. The importance of multiple sequence alignment

prior to data mining significantly improved both the quality and quantity of meta-signatures in comparison to Experiments I and II. In comparison to previous work reported in this thesis (Chapters 4 and 5), the proposed syntactic structure approach to automatic signature extraction using NNge successfully addressed the limitations of previous work by generating signatures in the quickest, simplest and most accurate manner.

The next chapter investigates whether biosequence analysis techniques, such as the sequence alignment and phylogenetics, can lead to syntactic approach for the automatic extraction of syntactic viral signatures for a metamorphic virus family. Experiments are reported that are designed to address the final research question (Q4).

Table 6.8: The key features and steps involved in experiments conducted in this chapter.

Features/Steps	Experiment I	Experiment II	Experiment III
Hex to DNA conversion	For pairwise sequence alignment only.	For data mining and pairwise sequence alignment.	For multiple sequence alignment, data mining, and pairwise sequence alignment.
Multiple sequence alignment for data mining	No	No	Yes
Conversion of variable length sequences into fixed length sequences	By adding the letter 'x' towards the end of each sequence until all the variable length sequences were of equal lengths.	By adding the letter 'X' towards the end of each sequence until all the variable length sequences were of equal lengths.	By the process of multiple sequence alignment. All the gaps introduced by the process of alignment were substituted by 'X'.
Total attributes for data mining	24,565	49,129	93,438
Total number of labels for data mining	17 (hex labels: a-f, 0-9 and x)	Five (DNA labels: A,T,G,C and X)	Five (DNA labels: A,T,G,C and X)
File size of ARFF file	2.49 MB	3.87 MB	7.38 MB
Total time taken to generate NNge results by Weka	2 minutes and 32 seconds	6 minutes and 13 seconds	32 minutes and 28 seconds
Time taken to build model	0.62 second	0.73 second	1.23 seconds
Correctly classified instances (%) - Accuracy	22/22 (100.00%)	0/22 (0.00%)	22/22 (100.00%)
Incorrectly classified instances (%) - Inaccuracy	0/22 (0.00%)	22/22 (100.00%)	0/22 (0.00%)
Kappa statistic	1	-1	1
Mean absolute error	0	1	0
Root mean squared error	0	1	0
Relative absolute error (%)	0.00%	200.00%	0.00%
Root relative squared error (%)	0.00%	200.00%	0.00%
Total number of instances	22	22	22
Total number of rules generated	Two (one for malicious class and one for non-malicious class)	Two (one for malicious class and one for non-malicious class)	Three (one for malicious class and two for non-malicious class)
Sequence lengths of extracted hex/DNA data from generated rules	Malicious (hex): 246,676 Non-Malicious (hex): 74,498	Malicious (DNA): 132,103 Non-Malicious (DNA): 41,670	Malicious (DNA): 161,495 Non-Malicious 1 (DNA): 59,740 Non-Malicious 2 (DNA): 11,860
Total number of pairwise alignments performed	Six (three each for malicious and non-malicious classes)	Six (three each for malicious and non-malicious classes)	Nine (three each for malicious, non-malicious 1 and non-malicious 2 classes)
Total number of meta-signatures generated	Nine (Four for malicious class and five for non-malicious class)	14 (Nine for malicious class and five for non-malicious class)	48 (31 for malicious class, nine for non-malicious class 1 and eight for non-malicious class 2)

Chapter 7 Detection of Metamorphic Virus Variants and Classification of their Signatures adopting Biosequence Analysis Techniques

This chapter focuses on the detection of metamorphic virus and its variants using the biosequence analysis techniques. Sequence alignment and phylogenetic tree analysis are adopted in this chapter to extract syntactic viral signatures for the detection of metamorphic virus family and their classification.

7.1. Introduction

In previous work (Chapter 4 to 6), a JavaScript-based polymorphic virus known as JS.Cassandra was investigated syntactically through biosequence techniques for experimental and testing purposes. In this chapter, a JavaScript-based metamorphic virus known as Transcriptase (see page no. 225 for more details) is investigated, again for experimental and testing purposes. An established variant of Transcriptase is analysed, where dead code is applied into compromised files. This technique of dead code insertion modifies the statistical aspects of the code, and the resulting virus variants are harder to identify using statistical and structural methods. The same variant of Transcriptase (Transcriptase, 2013) along with two different datasets belonging to the Transcriptase family are considered in this research.

In this chapter, the aim is to extend a syntactic study of the likelihood of generating signatures automatically from malware code without the requirement for semantic analysis. Whereas previous syntactic work (Chapter 4 to 6) employed sequence alignments to obtain consensuses from malware code variants with the intent of generating the minimum attainable number of signatures for identifying those variants and for previously unseen (P_u) variants, no effort was made to make the most of a by-product of the alignment, which is the output of equal length malware code of variants. Additionally, previous work mainly focused on detecting the known (P_k) and unknown (P_x) variants of several polymorphic malware families. There was no attempt made to test the capability of signature-based syntactic detection method on detecting metamorphic malware.

Another aim of this chapter is to distinguish the parent variants from the child variants belonging to a Transcriptase metamorphic virus family using phylogenetic tree analysis. If both the parent and child variants have originated from an original Transcriptase root

and the parent variants are the first set of variants to have been generated, then the child variants are the second set of variants to have been generated after the generation of parent variants. Phylogenetic tree analysis is used in this research to infer the evolutionary relationships between possible parent and child variants belonging to the Transcriptase metamorphic virus family.

7.2. String-Based Syntactic Detection of Metamorphic Virus Variants Method: Systems and Methods

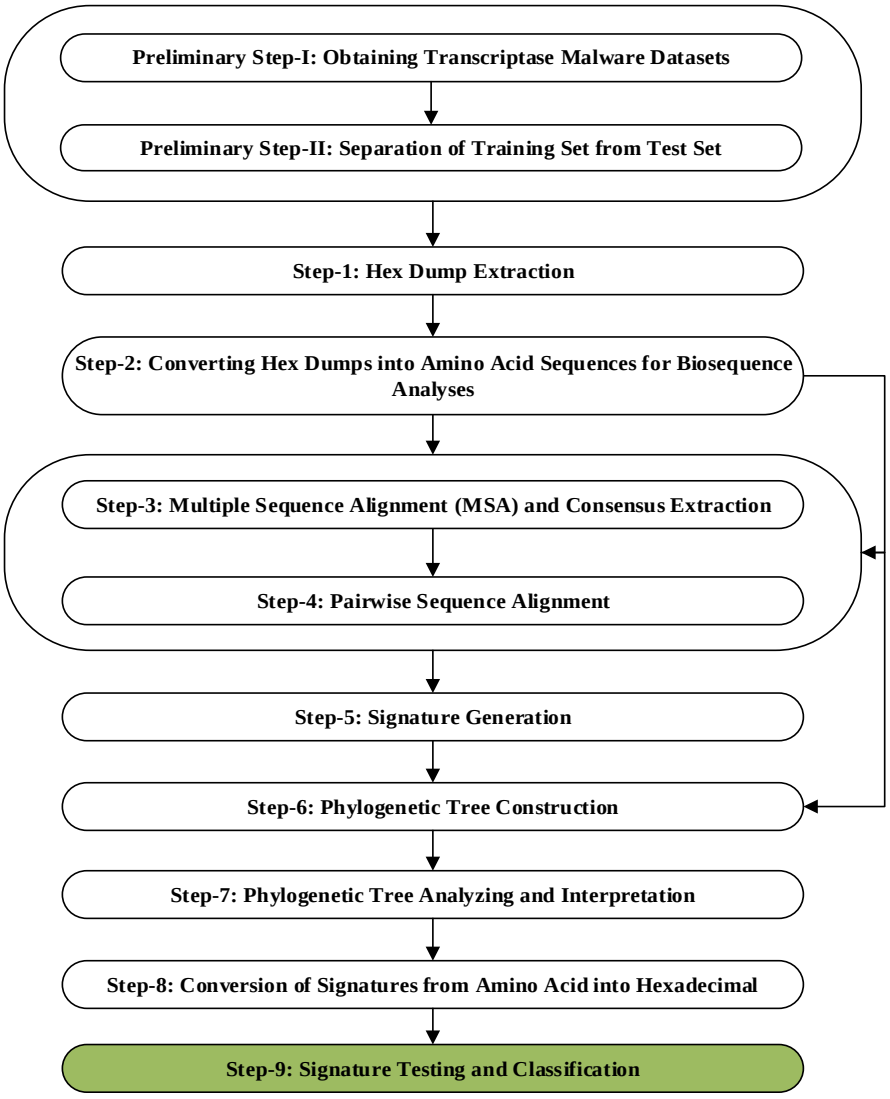

Figure 7.1: The nine steps in the String-Based Syntactic Detection of Metamorphic Malware Variants method.

The method adopted here consists of nine steps (see Figure 7.1). Downloading all the metamorphic malware and its variants, as well as hex (hexadecimal) dump extraction and testing were undertaken on a stand-alone system to prevent possible unintended infection of other systems. Network connectivity was used only at the testing stage as previous

experiments. More information with regards to the nine steps are detailed in Section 7.2.3 (see page no. 154).

7.2.1 Datasets

Two individual datasets of Transcriptase are used in this research for experimental and testing purposes. Dataset 1 was generated during this research by adopting the information contained in documents supplied by the malware writer and dataset 2 was generated by the original writer of the Transcriptase malware. Further information regarding dataset 2 can be found in (Second Part To Hell, 2018). Dataset 1 was generated from the original Transcriptase files provided in (Transcriptase, 2013). Overall, dataset 1 contains 353 unique variants along with the original Transcriptase malware (making a total of 354 malicious files) and dataset 2 contains 1,397 unique variants. Both these datasets were scanned using 59 different commercial antivirus scanners for their maliciousness. All these scanners are widely used and the datasets were scanned using VirusTotal (VirusTotal, 2018). Only Microsoft Security Essentials could detect some of the variants successfully. In this research, dataset 1 is used as a training set and dataset 2 as a test set.

Table 7.1: Four different datasets used in this research along with their sources.

Datasets	Type of Dataset	Source	Total Number of Files
Dataset 1 (Training Set)	Malicious JavaScript Files	JS.Transcriptase (Transcriptase, 2013)	354
Dataset 2 (Test Set)	Malicious JavaScript Files	JS.Transcriptase (Second Part To Hell, 2018)	1,397
Dataset 3 (Test Set)	Benign JavaScript Files	Cassandra Project JavaScript Repository (Cassandra, 2018)	34
		DataTables JavaScript Repository (MIT license, 2017)	18
		jQuery JavaScript Repository (The jQuery Foundation, 2018)	134
		threejs JavaScript 3D Library (Mr.doob, 2018)	952
		AngularJS JavaScript HTML Enhanced Library (MIT License, 2018)	1,061
		HTML5 Charts JavaScript Library (MIT license, 2018)	93
		impress.js JavaScript Library (Szopka & Ingo, 2018)	26
		usejsdoc JavaScript Library (Mathews, 2017)	299
		Total files	2,617
Dataset 4 (Test Set)	Benign Miscellaneous Files	PDF files (Parkour, 2013)	9,110
		Microsoft files (Parkour, 2013)	500
		Linux files (Parkour, 2013)	96
		JAR files (Parkour, 2013)	50
		RTF files (Parkour, 2013)	200
		Windows system files (Naidu V. , 2018)	8,172
		Total files	18,128

In addition to datasets 1 and 2, two other datasets (i.e. datasets 3 and 4) are used in this research for verifying the efficacy of the classified signatures. These datasets will be used as the test sets. Dataset 3 will contain several benign JavaScript files obtained from different sources. These JavaScript files will help to check for false positives and will be an ideal dataset as the malware datasets considered in this research are simply JavaScript files. Dataset 4 comprises a combination of miscellaneous files. It contains several PDF, Word, Excel, PowerPoint, JAR, RTF, Linux and Windows system files. More detailed information about these datasets can be found in Table 7.1.

7.2.2 Sequence alignments and Phylogenetics

In this research work, multiple sequence alignment (MSA) is used to align the hex dump sequences (after converting them into amino acid sequences) of Transcriptase family. MSA is performed with the help of the online web application MAFFT (version 7) (Katoh, Misawa, Kuma, & Miyata, 2002; Katoh & Standley, 2013; Katoh, 2018) and BLOSUM62 (Henikoff & Henikoff, 1992; Eddy, 2004) is used as a substitution/scoring matrix with a gap opening penalty of 1.53. An automated strategy of either the progressive approach (FFT-NS-2) or the iterative refinement approach (FFT-NS-i) implemented by MAFFT is adopted here. The resulting alignment file is used to generate a consensus which is extracted using Jalview (Waterhouse, Procter, Martin, Clamp, & Barton, 2009; Jalview, 2018). The extracted consensus is pairwise-aligned with hex dump sequence of the original Transcriptase malware. Pairwise alignment is performed using EMBOSS Matcher (Huang & Miller, 1991; EMBOSS Matcher, 2018) and BLOSUM62 is used as a substitution matrix with gap open and gap extend penalties of 10 and 1. EMBOSS Matcher adopts a local alignment technique described by Waterman-Eggert (Waterman, Smith, & Beyer, 1976; Smith & Waterman, 1981; Waterman & Eggert, 1987). The resulting alignment file is used to extract signature/malware patterns (syntactic virus signatures).

The final stage in this research is to generate a phylogenetic tree. Phylogenetic tree is generated by employing the same set of hex dump sequences that is used for the process of MSA. Due to the large size of hex dump sequences, an online web application known as Phylo.io (Robinson, Dylus, & Dessimoz, 2016; Phylo.io, 2018) is adopted. Phylio.io is used to analyse and visualise the phylogenetic tree. Phylo.io is available within MAFFT web application. A rough tree option is selected to generate the tree using automatic distance measure and average linkage (UPGMA) clustering method. As stated earlier (in

page no. 25), the resulting phylogenetic tree is used to classify variants of the Transcriptase family with the help of which the extracted signature patterns are classified into two different groups and later used for testing against the malicious and benign datasets.

7.2.3 Proposed method comprising of nine steps

Preliminary Steps I and II: 14 training variants out of 354 instances were selected from dataset 1 (training set) for experimental purposes. The percentage of training to test ratio is 3.95% (14:354). A small percentage of training to test samples was adopted to approximate the real-world problem of detecting effective signatures from a small, formerly encountered batch of known variants for dealing with a possibly infinite batch of unknown variants. A CRC32b hash value was created for each of the 14 training variants and no duplicates were discovered, signifying that they were different. The proposed method comprises of nine steps (see Figure 7.1).

Step-1 (Hex dump extraction): Hex dumps were extracted from each of the 14 training variants using sigtool, which is freely available on the ClamAV (ClamAV, 2018) web page.

Step-2 (Hex dump to amino acid conversion): Hex dumps of the 14 training variants were converted into amino acid (protein) sequences adopting the encoding method proposed in previous chapters. Conversion of hexadecimal characters (i.e. 0-16, a-f) into amino acid sequences were accomplished adopting the following rules (hex → amino acid): '0' → 'G'; '1' → 'H'; '2' → 'I'; '3' → 'R'; '4' → 'K'; '5' → 'L'; '6' → 'M'; '7' → 'N'; '8' → 'Q'; '9' → 'P'; 'a' → 'A'; 'b' → 'B'; 'c' → 'C'; 'd' → 'D'; 'e' → 'E'; and 'f" → 'F'. For instance, the hex string '0123456789abcdef' becomes 'GHIRKLMNQPABCDEF' (amino acid sequence).

Step-3 (MSA and consensus generation): In this step, MSA is applied on only six of the 14 converted amino acid sequences using MAFFT. Only six training sequences were selected due to sequence size constraints placed by MAFFT. MSA is a process of matching three or more related amino acid or nucleotide (DNA) sequences. In bioinformatics, the main purpose of MSA is to infer the presence of ancestral/evolutionary relationships among the sequences. In this proposed method, the primary purpose of MSA was to extract annotations i.e. consensus sequence data from the resulting alignment using Jalview. Consensus sequence is a representation of most

common amino acids in a multiple protein alignment. Figure 7.2 shows the resulting MSA in Jalview. The sequence length of the extracted consensus was 909,393 amino acids long.

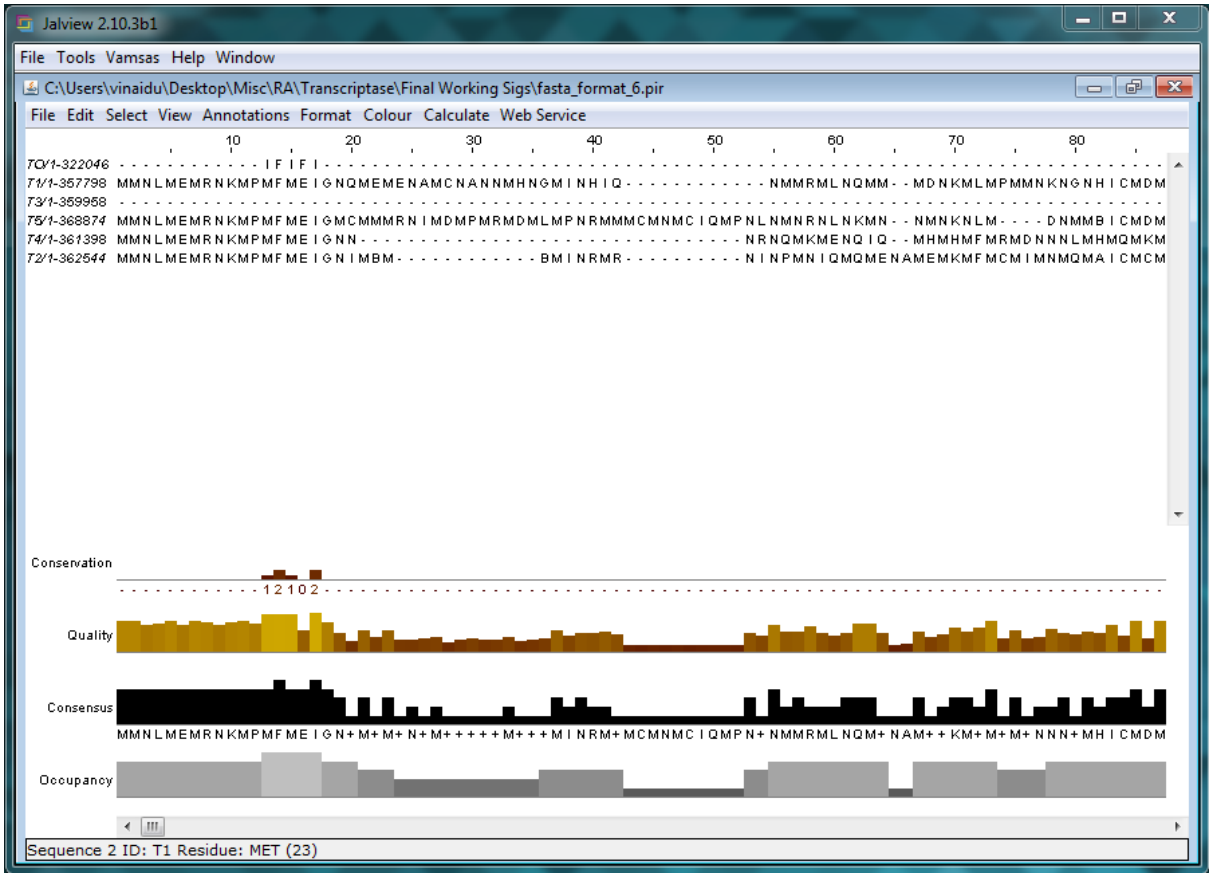

Figure 7.2: Jalview showing the MSA results generated in step-3.

Step-4 (Pairwise sequence alignment): In this step, the consensus sequence is pairwise-aligned with each of the 14 amino acid sequences from step-2 using EMBOSS Matcher. In bioinformatics, the purpose of pairwise alignment is to determine the regions of similarities (or conserved regions) that may signify structural, functional and/or evolutionary relationships among two amino acid or nucleotide sequences. In this proposed method, the main aim of pairwise alignment was to extract the regions of similarities from the resulting alignment (in the next step), which later becomes the signatures used to detect the variants of Transcriptase family. The resulting alignment from EMBOSS Matcher generated an alignment score of 695,878, identity percentage of 42.3%, similarity percentage of 48.4% and gaps percentage of 37.9%. Parts of the resulting alignment are shown below, where `CS` and `TO` signify the amino acid sequences of consensus and the original transcriptase malware. The numbers indicate the alignment positions/locations of the given sequences (i.e. `CS` and `TO`). '**.**' means low level of amino

acid similarity and ‘:’ means high level of amino acid similarity. ‘|’ means identical amino acid residues. ‘–’ represents gaps.

```
CS            138024 INIBINMPMEMNIEINIBINMLNIMEIGNAIKMRKNQIGMDBI------- 138066

                     |.|...:.:|:...:::..|.|||..||  ||....:|...|

TO               173 IGIGKALRIELKNIMHMENRMRNIMPNGN--KMHNRMLGDGAIFIFIGIG    220

CS            138067 -MNPMDHNIMIKRMFMKMLIGQBNRKRNRKPIMPIBINIBMMKNMLPRNM 138115

                      :||:... |:.||||:|.||.....|:     |.:.:.:.|.||

TO               221 MINPIGLR-MLMRMFMEMKIGLGMHNINK-----IGLKMFIGKQML----    260

CS            138116 MLMCDIMACLM--RINKPINEIMPMEM---NILNRQRKR----------- 138149

                     |.||...|..:  .|.|.:...|.|:|   ||:.|.|.|

TO               261 MCMCGDGAIFIFIGIGKKMLMRMLMDMIMLNIIGRIRGRHRIGDGAIFIF    310
```

Step-5 (Signature generation/extraction): In this step, signatures were extracted in the form of regions of similarities from the resulting pairwise alignment conducted in step-4. Only the regions of similarities that were sequentially longer than 10 amino acids were retained. In total, 373 signatures were extracted.

Step-6 (Phylogenetic tree construction): In this step, 14 amino acid sequences from step-2 were used to generate a phylogenetic tree. A rectangular lined up cladogram tree was generated in this step. A file in FASTA format was created which incorporated all the 14 sequences. The file was uploaded to Phylo.io website and the tree was generated.

Figure **7.3** shows the phylogenetic tree that was generated using Phylo.io.

Step-7 (Phylogenetic tree analysing and interpretation): In this step, the phylogenetic tree generated from the previous step was analysed and interpreted. The tree demonstrated two major clads/groups. First clad comprised of three sequences/variants (1_vic1, 2_vic2 and 3_vic3) and categorised them as parent sequences/variants. Second clad comprised of two sub-clads. First sub-clad consisted of only one sequence (4_o), which belonged to the sequence of the original Transcriptase malware. Second sub-clad consisted of 10 sequences/variants (5_var1, 6_var2, 7_var3, 8_var4, 9_var5, 10_var6, 11_var7, 12_var8, 13_var9 and 14_var10) and categorised them as child sequences/variants.

Step-8 (Conversion of signatures from amino acid into hexadecimal): In this step, the signatures generated in step-5 were converted back to hexadecimal strings using the encoding method demonstrated in step-2. The conversion was needed as the signatures were tested in the following step against the Transcriptase malware variants using clamscan, which is a part of ClamAV (ClamAV, 2018) and only accepts signatures in hexadecimal format.

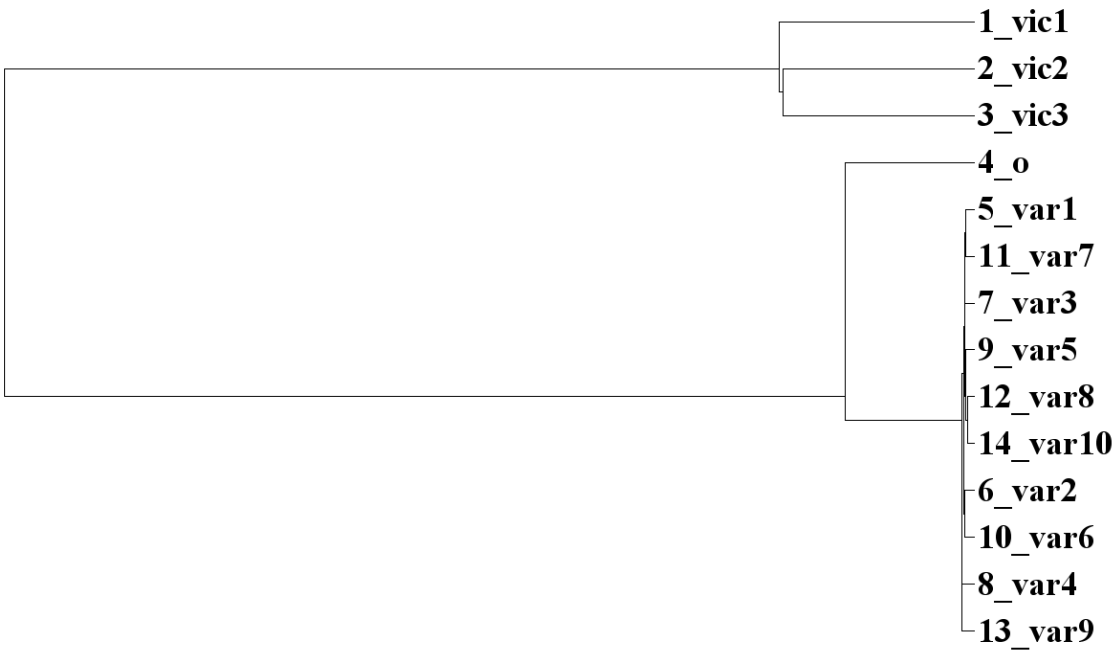

Figure 7.3: The phylogenetic tree of 14 Transcriptase sequences generated using Phylo.io.

Step-9 (Signature testing and classification): In this final step, individual signature testing and signature classification were accomplished. The signatures were classified after the first signature testing against dataset 1 (training set). The signatures were classified into two different categories, namely, parent and child signatures. A set of signatures that successfully identified all the variants of dataset 1 was labelled as parent signatures and a set of signatures that successfully identified all the variants except for parent variants was labelled as child signatures. There were only three parent variants in dataset 1 and they were all considered in the phylogenetic tree analysis. The original filenames of those three parent variants were victim1, victim2 and victim3 and were the files through which all the other child variants originated. This information was not fed into the phylogenetic tree analysis. Instead, the phylogenetics algorithm determined their relationship simply on the strength of their file ancestry/history (see

Figure **7.3**). All the remaining signatures including duplicates were discarded.

Second signature testing was conducted against datasets 2-4 (test sets) to verify the effectiveness of the signatures and their test statistics, such as true positive, true negative, false positive and false negative (as discussed in page no. 59). Additional test statistics, such as sensitivity/recall/true positive rate (TPR), specificity/true negative rate (TNR) and precision/positive predictive value (PPV) were also assessed. More details related to this can be found in the experimental results section.

7.3. Experimental Results

Out of 373 signatures generated in step-5, only 23 signatures were retained in step-9. After testing these signatures against the training and test sets, 22 signatures were labelled as child signatures and 1 signature was labelled as parent signature. Further information is supplied in Tables 7.2 to 7.5. This section is divided into two subsections, namely, training set and test set results. Seven individual test statistics were assessed in this section to validate the signatures generated in this research and they are described in Section 3.6 (see page no. 59).

7.3.1 Training set results

In this section, seven different commercial antivirus products were tested along with the parent and child signatures against the training set. Some of the antivirus products claim to adopt machine learning, deep learning and artificial intelligence techniques to detect the malicious files, which makes them the ideal state-of-the-art malware detection techniques with which to compare the signatures. The antivirus products were downloaded from their official websites and the databases of these products were all up-to-date. Table 7.2 shows the test results of seven antivirus products along with the parent and child signatures against the training set (dataset 1). The parent and child signatures were tested by placing these signatures inside our own created (.ndb) database (see Appendix D – page no. 228 for more details), which is adopted by Clamscan as a standard database file format for the purposes of signature testing.

The results in Table 7.2 show that the classified signatures can successfully identify the variants belonging to Transcriptase family in the training set (dataset 1). Parent signature identified 100% variants and child signatures identified 99.15% variants in the training set. Three variants were not identified by the child signatures as they were parent variants (see Figure 7.4 and step-7). This shows that the classified signatures can easily distinguish parent from child variants.

Table 7.2: Test results of seven commercial antivirus products together with parent and child signatures against training set (dataset 1).

Detection Product	Detection Techniques Adopted	Database Last Updated	Files Detected	True Positive	True Negative	False Positive	False Negative	Sensitivity/ Recall/TPR	Specificity /TNR	Precision/ PPV
Parent Signature – own Clamscan database	Biosequence Analysis Techniques	Feb 2018	**354/354 (100%)**	100%	0.00%	0.00%	0.00%	1	0	1
Child Signatures – own Clamscan database	Biosequence Analysis Techniques	Feb 2018	**351/354 (99.15%)**	99.15%	0.00%	0.00%	0.85%	0.9915	0	1
McAfee | Total Protection 2018	Machine Learning, Deep Learning, and Artificial Intelligence (Veeramachaneni & Arnaldo, 2016)	Feb 2018	0/354 (0.00%)	0.00%	0.00%	0.00%	100%	0	0	0
Kaspersky Anti-Virus 2018	Heuristic Analysis, Machine Learning (AO Kaspersky Lab, 2017; AO Kaspersky Lab, Machine Learning and Human Expertise, 2017; AO Kaspersky Lab, 2018)	Feb 2018	9/354 (2.54%)	2.54%	0.00%	0.00%	97.46%	0.0254	0	1
Norton Security Premium 2018	Advanced Machine Learning (Symantec Corporation, 2017)	Feb 2018	0/354 (0.00%)	0.00%	0.00%	0.00%	100%	0	0	0
Webroot SecureAnywhere AntiVirus 2018	Security Intelligence and Analytics Engine (Webroot Inc., 2013)	Feb 2018	0/354 (0.00%)	0.00%	0.00%	0.00%	100%	0	0	0
Microsoft Windows Defender Security	Windows Defender Advanced Threat Protection (jcaparas & Hall, 2018)	Feb 2018	104/354 (29.38%)	29.38%	0.00%	0.00%	70.62%	0.2938	0	1
Bitdefender Total Security 2018	Advanced Threat Defense, Machine-Learning (Bitdefender, 2016; Bitdefender, 2017)	Feb 2018	0/354 (0.00%)	0.00%	0.00%	0.00%	100%	0	0	0
ClamAV 2018	Open Source (GPL) Anti-Virus Engine (ClamAV, 2018)	Feb 2018	0/354 (0.00%)	0.00%	0.00%	0.00%	100%	0	0	0

None of the antivirus products could identify the transcriptase variants except for Kaspersky and Microsoft Windows Defender, which could only detect 2.54% and 29.38% of the variants, respectively.

7.3.2 Test set results

In this section, similar strategy as demonstrated in Section 7.3.1 of comparing the classified signatures with the seven antivirus products was applied. In this case, the comparison was carried out against the three different test sets. Tables 7.3 to 7.5 show the test results of seven antivirus products along with the parent and child signatures against the three individual test sets (datasets 2-4).

Table 7.3: Test results of seven commercial antivirus products together with parent and child signatures against test set 1 (dataset 2).

Detection Product	Files Detected	True Positive	True Negative	False Positive	False Negative	Sensitivity/ Recall/TPR	Specificity/ TNR	Precision /PPV
Parent Signature – own Clamscan database	**1,376/1,397 (98.5%)**	98.5%	0.00%	0.00%	1.5%	0.985	0	1
Child Signatures – own Clamscan database	0/1,397 (0.00%)	0.00%	0.00%	0.00%	100%	0	0	0
McAfee | Total Protection 2018	0/1,397 (0.00%)	0.00%	0.00%	0.00%	100%	0	0	0
Kaspersky Anti-Virus 2018	467/1,397 (33.43%)	33.43%	0.00%	0.00%	66.57%	0.3343	0	1
Norton Security Premium 2018	2/1,397 (0.14%)	0.14%	0.00%	0.00%	99.86%	0.0014	0	1
Webroot SecureAnywhere AntiVirus 2018	0/1,397 (0.00%)	0.00%	0.00%	0.00%	100%	0	0	0
Microsoft Windows Defender Security	968/1,397 (69.29%)	69.29%	0.00%	0.00%	30.71%	0.6929	0	1
Bitdefender Total Security 2018	0/1,397 (0.00%)	0.00%	0.00%	0.00%	100%	0	0	0
ClamAV 2018	0/1,397 (0.00%)	0.00%	0.00%	0.00%	100%	0	0	0

The results in Section 7.3.2 show that the parent signature can identify the variants belonging to Transcriptase family in the test set 1 (dataset 2). Parent signature identified 98.5% variants in the test set 1 (dataset 2) (see Table 7.3). Child signatures couldn't identify any variants in that test set. This shows that the variants considered in test set 1 (dataset 2) were all parent variants. To validate this, a phylogenetic tree was constructed and the same strategy as followed in step-6 was applied. In total, 34 variants of Transcriptase family along with the original malware (making a total of 35 malicious files) were considered for the process of phylogenetics. 14 variants belonged to dataset 1

(as considered in step-6) and the remaining 21 variants belonged to dataset 2. A rectangular lined up cladogram was constructed. Figure 7.4 shows the phylogenetic tree that was generated using Phylo.io.

Table 7.4: Test results of seven commercial antivirus products together with parent and child signatures against test set 2 (dataset 3).

Detection Product	Files Detected	True Positive	True Negative	False Positive	False Negative	Sensitivity/ Recall/TPR	Specificity/ TNR	Precision/ PPV
Parent Signature – own Clamscan database	0/2,617 (0.00%)	0.00%	100%	0.00%	0.00%	0	1	0
Child Signatures – own Clamscan database	0/2,617 (0.00%)	0.00%	100%	0.00%	0.00%	0	1	0
McAfee | Total Protection 2018	0/2,617 (0.00%)	0.00%	100%	0.00%	0.00%	0	1	0
Kaspersky Anti-Virus 2018	0/2,617 (0.00%)	0.00%	100%	0.00%	0.00%	0	1	0
Norton Security Premium 2018	0/2,617 (0.00%)	0.00%	100%	0.00%	0.00%	0	1	0
Webroot SecureAnywhere AntiVirus 2018	0/2,617 (0.00%)	0.00%	100%	0.00%	0.00%	0	1	0
Microsoft Windows Defender Security	0/2,617 (0.00%)	0.00%	100%	0.00%	0.00%	0	1	0
Bitdefender Total Security 2018	0/2,617 (0.00%)	0.00%	100%	0.00%	0.00%	0	1	0
ClamAV 2018	0/1,397 (0.00%)	0.00%	100%	0.00%	0.00%	0	1	0

Table 7.5: Test results of seven commercial antivirus products together with parent and child signatures against test set 3 (dataset 4).

Detection Product	Files Detected	True Positive	True Negative	False Positive	False Negative	Sensitivity/ Recall/TPR	Specificity/ TNR	Precision/ PPV
Parent Signature – own Clamscan database	0/18,128 (0.00%)	0.00%	100%	0.00%	0.00%	0	1	0
Child Signatures – own Clamscan database	2/18,128 (0.011%)	0.00%	99.989%	0.011%	0.00%	0	0.99989	0
McAfee | Total Protection 2018	6/18,128 (0.033%)	0.00%	99.967%	0.033%	0.00%	0	0.99967	0
Kaspersky Anti-Virus 2018	18/18,128 (0.099%)	0.00%	99.901%	0.099%	0.00%	0	0.99901	0
Norton Security Premium 2018	1/18,128 (0.005%)	0.00%	99.995%	0.005%	0.00%	0	0.99995	0
Webroot SecureAnywhere AntiVirus 2018	0/18,128 (0.00%)	0.00%	100%	0.00%	0.00%	0	1	0
Microsoft Windows Defender Security	1/18,128 (0.005%)	0.00%	99.995%	0.005%	0.00%	0	0.99995	0
Bitdefender Total Security 2018	4/18,128 (0.022%)	0.00%	99.978%	0.022%	0.00%	0	0.99978	0
ClamAV 2018	3/18,128 (0.016%)	0.00%	99.984%	0.016%	0.00%	0	0.99984	0

Figure 7.4 shows that all the variants from dataset 2 belonged to the same clad (i.e. third/last clad) distinguishing the parent variants from the child variants. Therefore, 1_TS_var_32 to 3_TS_var_24 (first clad) and 15_TS_var_11 to 35_TS_var_31 (third/last clad) were all classified as the parent variants and 5_TS_var_1 to 14_TS_var_10 (second sub-clad of second clad) were all classified as the child variants (see Figure 7.4). And 4_TS_O (first sub-clad of second clad) was the original Transcriptase malware (see Figure 7.4).

None of the antivirus products could identify the transcriptase variants from dataset 2 except for Kaspersky, Norton Security and Microsoft Windows Defender. And they could only detect 33.43%, 0.14% and 69.29% of the variants (see Table 7.3).

There were no false positives discovered against dataset 4 using the parent signature and 0.011% false positives using the child signatures (see Table 7.5), therefore satisfying the false positive rate requisite of 0.1%.

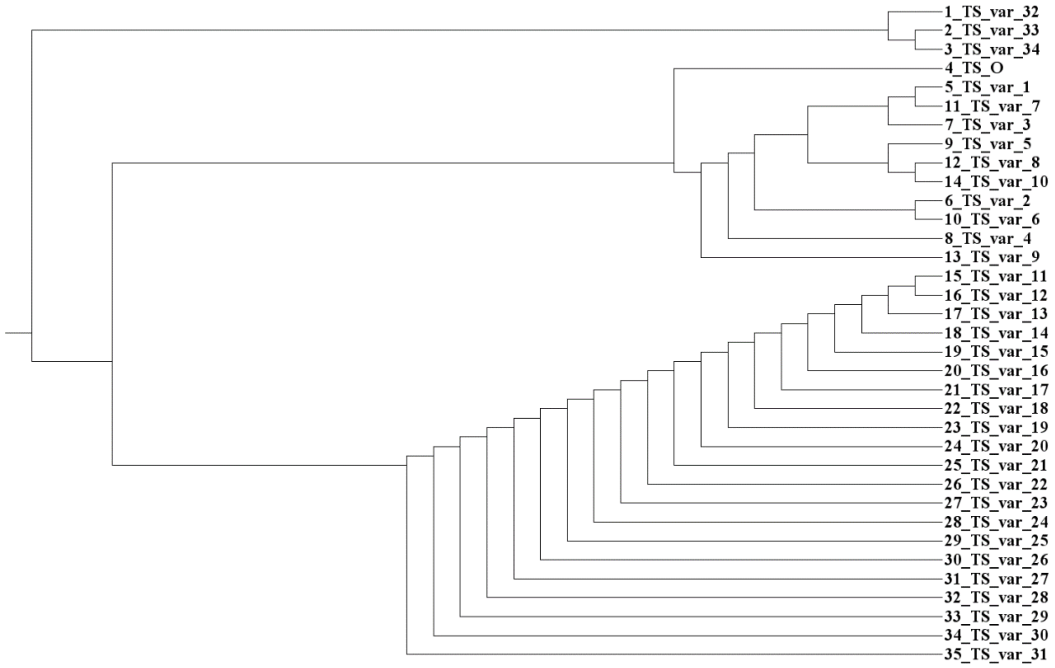

Figure 7.4: The phylogenetic tree of 35 Transcriptase sequences generated using Phylo.io.

7.4. Summary

In this chapter, a metamorphic JavaScript malware family of Transcriptase was examined. Two individual datasets belonging to Transcriptase family were analysed. Biosequence analysis techniques of sequence alignment and phylogenetics were adopted in this chapter. Signatures were generated and classified using these techniques. Two sets of signatures were classified for the detection of Transcriptase family, namely, parent and child signatures. Parent signature was able to identify 100% and 98.5% variants of datasets 1 and 2. Child signatures, on the other hand, could identify 99.15% variants of dataset 1.

Seven individual commercial antivirus products were compared with the parent and child signatures against the four different datasets. It was discovered that none of the antivirus products could identify at least 75% variants of the metamorphic malware. On the other hand, the classified signatures (i.e. both parent and child signatures) satisfied the false positive threshold of 0.1%.

Phylogenetic tree construction proved successful in distinguishing parent from child variants. Furthermore, this work demonstrates that it is possible to generate variable-length syntactic signatures and also classify them adopting biosequence analysis techniques.

The following chapter will discuss the conclusion and future work. More information will follow in the succeeding chapter.

Chapter 8 Conclusion and Future Work

8.1. Overview

In order to set up the foundation and fundamental range of this thesis, which presents a proposed string-based syntactic approach for detecting polymorphic malware (known – P_k and unknown – P_x) variants, background information concerning the syntactic approach in comparison to the semantic approach was discussed in Chapter 1. A comprehensive literature review on the different types of malware as well as the classification of viruses along with their masking strategies were discussed in Chapter 2. Classification of polymorphism together with the different levels of polymorphism were also discussed in Chapter 2. Also discussed were the mutation engine and the decryption routine used by any typical polymorphic malware. Lastly, Chapter 2 discussed different types of the state of the art detection approaches used by the current and modern AVS products together with their drawbacks. Chapter 3 presented the research methodology adopted in this thesis.

Chapter 4 examined the proposed efficient and effective syntactic approach of string matching algorithm such as the SWA for the automatic generation of signatures for the detection of some or all new polymorphic malware variants by extracting syntactic patterns from semantically rich (polymorphic) hex code. This proposed syntactic approach (with the aid of string matching dynamic programming SWA) to the automatic generation of viral meta-signatures (i.e. viral syntactic patterns) detected all the known (P_k) viral variants of JS.Cassandra polymorphic virus (see Table 4.1) and successfully addressed the first research question (Q1). The ESET antivirus cannot successfully detect all the known (P_k) variants of JS.Cassandra viral family. Moreover, ClamAV antivirus could hardly detect the unknown (P_x) variants of Win32.Kitti viral family but detected around 33% of unknown (P_x) variants belonging to the Win32.Cholera viral family. As stated earlier (i.e. in Subsection 1.6.1), the selected three polymorphic viral families were at least 5-11 years old. But as the experiments show (see Table 4.1), modern AVS products cannot successfully detect all the known (existing) (P_k) variants of the polymorphic malware family mentioned here, let alone the unknown (new) (P_x) variants.

Chapter 5 explored the advanced sequence alignment techniques in a string-based syntactic approach for detecting polymorphic malware variants by conducting further experiments on the previous chapter (i.e. Chapter 4). Chapter 5 was divided into three

parts. Part-I adopted two different dynamic programming algorithms i.e. the NWA and the SWA. Part-I (as well as Part-II and Part-III) conducted the process of multiple sequence alignment on meta-signatures to extract super-signatures. Previous chapter (i.e. Chapter 4) did not explore any of these advanced sequence alignment techniques. Part-II examined the effects of different combinations of gap open and gap extend penalties using the SWA. Previous chapter (i.e. Chapter 4) only explored one combination of gap open penalty (i.e. 10) and gap extend penalty (i.e. 1). Part-III examined the effects of using 71 different substitution matrices by conducting 71 different pairwise sequence alignments. Previous chapter (i.e. Chapter 4) conducted the processes of pairwise sequence alignments by employing one substitution matrix i.e. the ID matrix.

Part-I – The syntactic approach (with the help of string matching dynamic programming NWA and SWA) to the automatic generation of viral meta-signatures/super-signatures identified all the known (P_k) viral variants of JS.Cassandra polymorphic virus (see Table 5.4 and Figure 5.2) and successfully addressed the first part of the second research question (Q2a). Although, the three super-signatures obtained from the NWA and SWA detected only 96.59% of known (P_k) JS.Cassandra malware variants, the three meta-signatures obtained from the NWA and SWA detected 100% of known (P_k) JS.Cassandra malware variants. Furthermore, the proposed syntactic technique (with the help of string matching dynamic programming SWA) to the automatic generation of viral meta-signatures/super-signatures identified all the unknown (P_x) viral variants of W32.Kitti polymorphic virus (see Table 5.5 and Figure 5.3). In total, 29 out of the 37 overall meta-signatures (generated from both the NWA and SWA) i.e. around 78.38% were effective i.e. detected all/some known (P_k) variants of the JS.Cassandra polymorphic virus. Particularly, 12 (generated from the NWA) out of the 37 overall meta-signatures i.e. 32.43%, and 17 (generated from the SWA) out of the 37 overall meta-signatures i.e. 45.94% detected all/some known variants (P_k) of the JS.Cassandra polymorphic virus (see Table 5.4). To be more precise, eight (generated from the NWA) out of the 37 overall meta-signatures i.e. 21.62%, and 12 (generated from the SWA) out of the 37 overall meta-signatures i.e. 32.43% detected the known (P_k) variants of the JS.Cassandra virus with an accuracy of 96% and above (see Table 5.4). The implication of the findings is that syntactic approaches to automatic signature generation can complement current manual and semantic approaches, with the added advantage that polymorphic variants of a virus can also be identified once a sufficient number of variants are incorporated for string-based comparison. The ultimate goal for any syntactic approach will be to identify a

potential ‘grammar’ of a virus from a relatively small number of variants so that unknown (P_x) but reachable variants can be generated for robust automatic signature extraction. It is possible that hybrid variants (the combination of behaviours from more than one virus family) will need signatures based on structural aspects of the virus families as described by a grammar rather than string matching. AVSs currently are reactive rather than pro-active. Syntactic and string-based approaches represent the best chance of designing and developing proactive AVSs in the future to deal with the increasingly complex malware being confronted.

Part-II – The syntactic approach (with the aid of string matching dynamic programming SWA and different combinations of gap open and gap extend penalties) to the automatic generation of viral super-signatures detected all known (P_k) and unknown (P_x) viral variants of JS.Cassandra, W32.CTX and W32.Kitti polymorphic virus (see Figures 5.6 to 5.8) and successfully addressed the second part of the second research question (Q2b). The findings show that increasing the gap open and gap extend penalties decreases the number of gaps (in some cases to the point where no gaps exist) in the final alignment (see Columns ‘Gap Open Penalty’, ‘Gap Extend Penalty’ and ‘Gaps Percentage’ in Table 5.9). Moreover, the meta-signatures obtained from the alignment with few or no gaps have proven to be more effective and successful in detecting known (P_k) and unknown (P_x) polymorphic variants than alignment with many gaps. From the experiment results provided in Table 5.10 (i.e. for polymorphic viruses JS.Cassandra and W32.CTX/W32.Cholera), it can be concluded that some of the final alignments, those with gap percentages of 0.5 or higher, have moderately effective meta-signatures (an accuracy of less than 100%). From the results presented in Table 5.10 (i.e. for W32.Kitti polymorphic virus), it can be concluded that the final alignments with no gap percentages (i.e. 0.00%) have effective meta-signatures (i.e. with an accuracy of 100%). Most importantly, the results from Table 5.9 indicate that the conversion of malware code into biological representations has served the task of identifying common code subsequences.

Part-III – The syntactic approach (with the aid of string matching dynamic programming SWA and using different substitution matrices) to the automatic generation of viral meta-signatures detected all the known (P_k) virus variants of JS.Cassandra polymorphic virus using the meta-signatures and super-signatures obtained in Step-4 (i.e. in Section 5.11.3) and Step-7 (i.e. in Section 5.11.6), respectively (see Tables 5.13 and 5.14) and successfully addressed the third part of the second research question (Q2c). Meta-

signatures: In total, 123 out of the 161 overall meta-signatures (generated from the six different substitution matrices) i.e. around 76.40% were effective i.e. detected all/some known (P_k) variants of the JS.Cassandra polymorphic virus. Particularly, 23 (generated from the BLOSUM40 substitution matrix) out of the 161 overall meta-signatures i.e. 14.28%, and 16 (generated from the DAYHOFF substitution matrix) out of the 161 overall meta-signatures i.e. 9.94% detected all/some known (P_k) variants of the JS.Cassandra polymorphic virus (see Table 5.13). ID substitution matrix was not considered here as only one meta-signature was extracted during the process and it was effective (see Table 5.13). Additionally, 21 (generated from the MATCH substitution matrix) out of the 161 overall meta-signatures i.e. 13.04%, 24 (generated from the PAM100 substitution matrix) out of the 161 overall meta-signatures i.e. 14.91%, and 38 (generated from the PAM350 substitution matrix) out of the 161 overall meta-signatures i.e. 23.60% detected all/some known (P_k) variants of the JS.Cassandra polymorphic virus (see Table 5.13). To be more precise, 22 (generated from the BLOSUM40 matrix) out of the 161 overall meta-signatures i.e. 13.66%, and 16 (generated from the DAYHOFF matrix) out of the 161 overall meta-signatures i.e. 9.94% detected the known (P_k) variants of the JS.Cassandra virus with an accuracy of 92% and above (see Table 5.13). Moreover, 14 (generated from the MATCH substitution matrix) out of the 161 overall meta-signatures i.e. 8.69%, 22 (generated from the PAM100 substitution matrix) out of the 161 overall meta-signatures i.e. 13.66%, and 38 (generated from the PAM350 substitution matrix) out of the 161 overall meta-signatures i.e. 23.60% detected the known (P_k) variants of the JS.Cassandra virus with an accuracy of 92% and above (see Table 5.13). Super-signatures: In total, 34 out of the 47 overall super-signatures (generated from the six different substitution matrices) i.e. around 72.34% were effective i.e. detected all/some known (P_k) variants of the JS.Cassandra polymorphic virus. Particularly, three (generated from the BLOSUM40 substitution matrix) out of the 47 overall super-signatures i.e. 6.38%, five (generated from the DAYHOFF substitution matrix) out of the 47 overall super-signatures i.e. 10.64%, and 11 (generated from the ID substitution matrix) out of the 47 overall super-signatures i.e. 23.40% detected all/some known (P_k) variants of the JS.Cassandra polymorphic virus (see Table 5.14). Additionally, nine (generated from the MATCH substitution matrix) out of the 47 overall super-signatures i.e. 19.15%, three (generated from the PAM100 substitution matrix) out of the 47 overall super-signatures i.e. 6.38%, and three (generated from the PAM350 substitution matrix) out of the 47 overall super-signatures i.e. 6.38% detected all/some known (P_k) variants of the JS.Cassandra polymorphic virus (see Table 5.14). To be more precise, three

(generated from the BLOSUM40 matrix) out of the 47 overall super-signatures i.e. 6.38%, five (generated from the DAYHOFF matrix) out of the 47 overall super-signatures i.e. 10.64%, and seven (generated from the ID substitution matrix) out of the 47 overall super-signatures i.e. 14.89% detected the known (P_k) variants of the JS.Cassandra virus with an accuracy of 96% and above (see Table 5.14). Moreover, seven (generated from the MATCH substitution matrix) out of the 47 overall super-signatures i.e. 14.89%, three (generated from the PAM100 substitution matrix) out of the 47 overall super-signatures i.e. 6.38%, and three (generated from the PAM350 substitution matrix) out of the 47 overall super-signatures i.e. 6.38% detected the known (P_k) variants of the JS.Cassandra virus with an accuracy of 96% and above (see Table 5.14).

Chapter 6 examined the rule-based approach (obtained and inspired from the NNge classifier data mining algorithm supplementing the string matching dynamic programming SWA) to the automatic generation of viral meta-signatures with the help of three different sets of experiments, which identified all the 352 known (P_k) malware variants of JS.Cassandra polymorphic virus as well as the 43 non-malware (P_u) variants of JS.Cassandra polymorphic virus (see Table 6.6 and Figures 6.4 to 6.6 – page nos. 141, 144-145). And this lead to addressing the third research question (Q3) in this thesis successfully. In total, 41 out of the 71 overall meta-signatures i.e. around 57.75% [30.98% malicious (22/71) and 26.76% non-malicious (19/71) meta-signatures] were effective i.e. detected all/some variants from the two different types of groups (i.e. malicious (P_k) and non-malicious (P_u)). Particularly, five out of the nine meta-signatures i.e. 55.55% [33.33% malicious (3/9) and 22.22% non-malicious (2/9) meta-signatures] from first set of experiments and six out of the 14 meta-signatures i.e. 43% [28.6% malicious (4/14) and 14.3% non-malicious (2/14) meta-signatures] from second set of experiments detected all/some variants from the two different types of groups (see Table 6.6). Additionally, 30 out of the 48 meta-signatures i.e. 62.50% [31.25% malicious (15/48) and 31.25% non-malicious (15/48) meta-signatures] from the third set of experiments detected all/some variants from the two different types of groups. Only 11 out of the 30 effective meta-signatures obtained from the third set of experiments are shown in Table 6.6. As first and second sets of experiments were performed using two different representational approaches (i.e. hex/DNA) along with the third set of experiments containing aligned DNA sequences, all with the same (unchanged) instances each time, some of the meta-signatures obtained from the three sets were identical to each other (see Tables 6.2 to 6.4 – page nos. 128, 132 and 137). Malicious MS1 (1), malicious MS3 (2),

non-malicious MS41 (3) and non-malicious MS43 (3) share identical meta-signature (see Tables 6.2 to 6.4, 6.6). On the other hand, malicious MS4 (1), malicious MS9 (2) and non-malicious MS37 (3) share identical meta-signature (see Tables 6.2 to 6.4, 6.6). Although, the second set of experiments generated rules with 100% inaccuracy, the overall combined percentage of effective meta-signatures generated from all the three sets of experiments is 57.75% (see Tables 6.2 to 6.4, 6.6). On the other hand, the overall combined percentage of non-effective meta-signatures generated from all the three sets of experiments is 42.25% (see Tables 6.2 to 6.4, 6.6).

In Chapter 6, the successful identical meta-signature with 100% accuracy [i.e. MS4 (1), MS9 (2) and MS37 (3) – see Table 6.6] obtained from the three sets of experiments for JS.Cassandra polymorphic malware and its known (P_k) variants was decoded into '`String.fromCharCode(`' which is a JavaScript function. Second best meta-signature with 97.73% accuracy [i.e. MS15 (3) – see Table 6.6] obtained from the third set of experiments for JS.Cassandra polymorphic malware and its known (P_k) variants was decoded into '`19-`' which is a part of JavaScript function. Third best identical meta-signature with 96.59% accuracy [i.e. MS1 (1), MS3 (2), MS41 (3) and MS43 (3) – see Table 6.6] obtained from the three sets of experiments for JS.Cassandra polymorphic malware and its known (P_k) variants was decoded into '`+'='+Math.round(Math.random()*`' which is a JavaScript function. These functions will be a common function inside the source code for the original (P_s) JS.Cassandra polymorphic virus and its known (P_k) variants. The main aim in Chapter 6 was not only to achieve identification of polymorphic malware (and its known (P_k) variants) with (NNge) rule-based meta-signatures automatically, but also enable them with syntactic meaning that is understandable by malware analysts, and which could help us in detecting the future unknown (P_x) polymorphic malware variants of the same family. The initial approaches (i.e. Chapters 4 and 5) (Naidu & Narayanan, 2016a; Naidu & Narayanan, 2016b; Naidu & Narayanan, 2016c) are a long and slow process of extracting the effective common substrings/meta-signatures by performing several pairwise alignments. The current rule-based approach reduces these tasks and generates lengthy rules by assembling the deeply conserved regions from all the input (big data) polymorphic malware (and non-malware) instances at once.

Chapter 7 examined a metamorphic JavaScript malware family of Transcriptase. Two individual datasets belonging to Transcriptase family were analysed. Biosequence

analysis techniques of sequence alignment and phylogenetics were adopted in this chapter. Signatures were generated and classified using these techniques. Results are presented in Tables 7.2 to 7.5 and Figures 7.3 and 7.4 (see page nos. 159-161, 157 and 162). Two sets of signatures were classified for the detection of Transcriptase family, namely, parent and child signatures. Parent signature was able to identify 100% and 98.5% variants of datasets 1 and 2. Child signatures, on the other hand, could identify 99.15% variants of dataset 1. Seven individual commercial antivirus products were compared with the parent and child signatures against the four different datasets. It was discovered that none of the antivirus products could identify at least 75% variants of the metamorphic malware. On the other hand, the classified signatures (i.e. both parent and child signatures) satisfied the false positive threshold of 0.1%.

At present, serious concerns exist as to whether modern AVS technologies will identify new (future/unknown) variants of polymorphic malware. The ultimate goal would be to detect all new (future (P_x)) polymorphic variants (see Figure 2.1 – page no. 29) using a syntactic approach to identify variants both within a virus family as well as across virus families. The research demonstrates that there is a need for a good identification software system that can effectively and efficiently identify potentially old (pre-existing), current (known/existing) and future (new/unknown) malware variants (see Tables 6.1 and 6.6 – page nos. 125 and 141). String-based syntactic and rule-based approaches together look very promising in designing and developing proactive AVSs in the future to deal with the increasingly complex malware being encountered.

8.2. Contribution of this Thesis

In this thesis, the proposed string-based syntactic techniques were developed based on sequence alignments techniques. Moreover, advanced sequence alignment techniques were applied and combined the data mining algorithm with sequence alignment techniques in a novel way to refine the algorithm.

Bearing in mind the above statement, the major difference between the string-based syntactic technique and other AVSs (e.g. Bitdefender, ESET and Symantec) is that string-based syntactic technique attempts to identify future variants by taking into account already existing malware belonging to the same family using conserved syntactic patterns (i.e. conserved signatures), whereas in other AVSs the existing signatures does not capture the future variants and instead creates new signatures for those variants each and

every time when encountered. The major contribution will be to add the above stated to the existing knowledge for the future string-based syntactic AVSs.

Figure 2.1 presents the distribution of polymorphic malware variants. The research demonstrated in this thesis shows that current manual driven techniques deal only with P_S (see Figure 2.1 – page no. 29). The summarised and integrated research question investigated was 'Do advanced sequence alignment techniques and data mining algorithm (e.g. NNge) produce consensuses (and rules) that not only 'fit' the known (P_k) variants (training set) but also generalise well to unknown (P_x) variants (test set)?' As stated earlier, the study in this research used three families of polymorphic malware and their variants as input. JS.Cassandra virus was chosen along with its existing or known (P_k) malware variants to check whether exploring advanced sequence alignment techniques and data mining algorithm, such as by introducing relatively sophisticated gap open and extend facilities, etc. still capture known (P_k) variants that are known to be captured without such facilities. W32.CTX/W32.Cholera and W32.Kitti viruses were used on the other hand to generate new or unknown (P_x) malware variants for exploring advanced sequence alignment techniques and data mining algorithm, such as by testing the effects of gap open and gap extend facilities, by using different substitution matrices, by generating rules, etc. Well-established viruses were chosen because their structure and behaviour are sufficiently well understood by commercial AVS developers and therefore any new variants reported here will not pose a serious threat to the latest AVS versions. An outlined contribution of this thesis chapter wise (i.e. starting from Chapter 4) is discussed below:

1. **Chapter 4:** This chapter (Naidu & Narayanan, 2016a) investigated efficient and effective approach of string matching algorithm such as the SWA for the automatic generation of signatures for the detection of all the known (existing) (P_k) variants of JS.Cassandra polymorphic virus (see Table 4.1). That is, this chapter successfully extracted syntactic patterns (i.e. meta-signatures) and these patterns were in turn employed for the complete identification of all known (351) malware variants (P_k) belonging to the JS.Cassandra virus. Also, the syntactic patterns of JS.Cassandra virus successfully and completely detected some unknown/new (12) malware variants (P_x) of JS.Cassandra virus (see Appendix D – page no. 235). Further, the syntactic patterns belonging to the W32.Kitti and W32.CTX/Cholera polymorphic viruses successfully and completely detected some unknown (1105 and 198) malware variants (P_x) of

W32.Kitti and W32.CTX viruses, respectively (see Table 4.1). The experiments from this chapter also demonstrated to some extent that current and modern state-of-the-art AVS products cannot completely and successfully detect the known (existing) (P_k) polymorphic malware variants, let alone future (new/unknown) variants (P_x) developed in the laboratory (see Table 4.1). Hence, the main contribution of this chapter was to add to the existing knowledge that it was possible to identify syntactic structures that helped to determine whether a piece of code contains a virus type and its variants.

2. **Chapter 5 (Part-I):** This chapter (Naidu & Narayanan, 2016c) explored the advanced sequence alignment techniques by examining the effects of two different dynamic programming (string-based) approaches (i.e. the NWA and SWA) for the automatic generation of signatures for the detection of all the known (P_k) variants of JS.Cassandra polymorphic virus and some unknown (future/new) variants (P_x) of W32.Kitti polymorphic virus, respectively (see Tables 5.4 and 5.5 and Figures 5.2 to 5.4). That is, this chapter successfully extracted syntactic patterns (i.e. meta-signatures) both globally and locally by conducting the processes of pairwise alignments using the NWA and SWA and these patterns were in turn employed for the complete identification of all known (351) malware variants (P_k) belonging to the JS.Cassandra virus as well as for the complete identification of some unknown (1105) malware variants (P_x) belonging to the W32.Kitti virus (see Tables 5.4 and 5.5). Also, this chapter successfully extracted super syntactic patterns (i.e. super-signatures) by conducting the processes of multiple and pairwise sequence alignments on meta-signatures obtained from the NWA and SWA based pairwise alignments and these super patterns were in turn employed for the complete identification of all known (P_k) and some unknown (P_x) malware variants belonging to the JS.Cassandra and W32.Kitti polymorphic viruses, respectively (see Figures 5.2 to 5.4). Thus, the main contribution of this chapter was to add to the existing knowledge that signatures can be identified both globally (i.e. by using the NWA) and locally (i.e. by using the SWA) for known (P_k) as well as unknown (P_x) polymorphic variants.
3. **Chapter 5 (Part-II):** This chapter explored the advanced sequence alignment techniques by examining the effects of using ten different combinations of gap open and gap extend penalties in a string-based approach for the automatic generation of signatures for the detection of all the known (P_k) variants of JS.Cassandra polymorphic virus and some unknown (P_x) variants of W32.CTX/Cholera polymorphic virus and W32.Kitti virus, respectively (see Table 5.10 and Figures 5.6

to 5.8). That is, this chapter successfully extracted syntactic patterns (i.e. meta-signatures) using the ten different combinations of gap penalties and these patterns were in turn employed for the complete identification of all known (351) malware variants (P_k) belonging to the JS.Cassandra virus as well as for the complete identification of some unknown (198 and 1105) malware variants (P_x) belonging to the W32.CTX virus and W32.Kitti virus, respectively (see Table 5.10). Also, this chapter successfully extracted super syntactic patterns (i.e. super-signatures) by conducting the processes of multiple and pairwise sequence alignments on meta-signatures obtained from the processes of first pairwise alignments (using ten different combinations of gap penalties) and these super patterns were in turn employed for the complete identification of all known (P_k) and some unknown (P_x) malware variants belonging to the JS.Cassandra, W32.CTX, and W32.Kitti polymorphic viruses, respectively (see Figures 5.6 to 5.8). Therefore, the main contribution of this chapter was to add to the existing knowledge that introducing different combinations of gap penalties helped to identify signatures for known (P_k) as well as unknown (P_x) polymorphic variants.

4. **Chapter 5 (Part-III):** This chapter (Naidu & Narayanan, 2016b) explored the advanced sequence alignment techniques by examining the effects of using different substitution matrices in a string-based approach for the automatic generation of signatures for the detection of all the known (P_k) variants of JS.Cassandra polymorphic virus (see Tables 5.13 and 5.14). That is, this chapter successfully extracted syntactic patterns (i.e. meta-signatures) using different substitution matrices and these patterns were in turn employed for the complete identification of all known (351) malware variants (P_k) belonging to the JS.Cassandra virus (see Table 5.13). Also, this chapter successfully extracted super syntactic patterns (i.e. super-signatures) by conducting the processes of multiple and pairwise sequence alignments as well as data mining on meta-signatures obtained from the processes of first pairwise alignments (using different substitution matrices) and these super patterns were in turn employed for the complete identification of all known (P_k) malware variants belonging to the JS.Cassandra polymorphic virus (see Table 5.14). So, the main contribution of this chapter was to add to the existing knowledge that using different substitution matrices (e.g. BLOSUM, PAM) helped to capture new (never seen before) signatures for known (P_k) polymorphic variants.
5. **Chapter 6:** This chapter addressed some of the limitations of the initial work (i.e. Chapters 4 and 5) (Naidu & Narayanan, 2016a; Naidu & Narayanan, 2016b; Naidu &

Narayanan, 2016c) and investigated a syntactic structural approach using data mining algorithm NNge to the automatic generation of signatures for the detection of all the known (P_k) variants of JS.Cassandra polymorphic virus (see Table 6.6 and Figures 6.4 to 6.6). That is, this chapter successfully extracted syntactic patterns (i.e. meta-signatures) by conducting three different sets of experiments using NNge and these patterns were in turn employed for the complete identification of all known (351) malware variants (P_k) belonging to the JS.Cassandra virus (see Figure 6.4). Therefore, the main contribution of this chapter was to add to the existing knowledge that using a combination of data mining and sequence alignment techniques helped to identify one or more master syntactic rules each containing several different syntactic patterns (i.e. signatures) within for known (P_k) polymorphic variants. The contributions of this chapter are as follows:

- Adopting a data mining algorithm, NNge, to generate rule-based signatures automatically from real malware data.
- Comparing variable length data mining algorithm to equal length data mining algorithm using NNge on malware source code by conducting three different experiments (Experiments I-III).
- Distinguishing malicious variants from non-malicious with the help of rules generated using the data mining algorithm, NNge.
- Testing the derived rule-based signatures against real malware data and comparing the results to other commercial AVSs.
- Comparing the overall performance metrics such as true positive rate, false positive rate, precision, recall, etc. with other related work on malware detection using data mining algorithms.

6. **Chapter 7:** The contributions of this chapter are as follows:
    - Classifying viral signatures acquired from the metamorphic Transcriptase malware family adopting biosequence analysis techniques.
    - Distinguishing Transcriptase malware variants adopting phylogenetics.
    - Generating syntactic variable-length viral signatures from Transcriptase malware family adopting sequence alignment techniques.
    - Testing the classified viral signatures against two different Transcriptase malware datasets and comparing the test results against seven individual commercial antivirus products.
    - Testing the classified viral signatures against benign datasets for false positives.

8.3. Limitations of the Study

The major limitations of the proposed approach demonstrated in this thesis are as follows:

1. The study did not examine a multiple sequence alignment as opposed to pairwise sequence alignment for semantic signature generation employing the bioinformatics MAFFT (Multiple Alignment using Fast Fourier Transform) (Katoh, Misawa, Kuma, & Miyata, 2002; Katoh & Standley, 2013; Katoh, 2018) algorithm, which will help in detecting complex polymorphic malware as well as metamorphic malware (and their variants). In this thesis, the focus was on syntactic signature extraction.
2. The study did not employ the proposed approaches in extracting syntactic patterns from complex polymorphic malware, such as the TPE i.e. Trident Polymorphic Engine, as well as from a combination or hybrid of two or more polymorphic malware.
3. The study did not generate a single consensus (i.e. a super-signature) that could completely identify all the possible (known – P_k and unknown – P_x) variants of a particular polymorphic family.
4. The current string matching techniques do not work well with non-biological representations, such as hexadecimal, etc.

Some additional limitations of the study presented in this thesis are discussed in this section. The focus on well-known and old viruses does not take into account the rapid evolution of other forms of malware, such as ransomware and DoS attacks that involve external manipulation. Furthermore, the study does not take into account the unknown (new) variants (P_x) generated from the polymorphic *virus* construction kits. Building such a library of unknown (P_x) polymorphic variants will allow us to investigate the impact of a new polymorphic malware detection system in relation to old and existing malware variants. However, nearly all malware has a self-replicating component irrespective of its function. On the assumption that the meta-signatures and super-signatures are capturing essential aspects of malware replication, the results described here may be applicable to other malware types (not just viruses or worms) that also involve a replication step.

8.4. Future Work

Future Work on Chapters 4 to 6: As stated earlier, polymorphic malware remains difficult to identify since such malware is able to reform into functionally similar but syntactically different variants to bypass signature detection by conventional AVSs.

Investigations as part of this thesis have identified a proposed approach to polymorphic virus variant detection through the use of pairwise/multiple sequence alignment techniques as well as data mining algorithm to identify consensuses (subsequences conserved across different and variable length virus code) that lead to signatures for virus detection in next-generation antivirus systems. The question arises as to how this proposed approach to automatic virus detection can cope with the most complex viruses yet discovered: metamorphic malware. Whereas polymorphism changes the order of the viral code but not typically the code itself to avoid signature detection, metamorphic malware changes the code so that a functionally identical but syntactically non-identical variant is generated. To date, it is not known how to generate signatures for metamorphic viruses.

The aim of the future proposal will be to examine a multiple sequence as opposed to pairwise sequence alignment for semantic signature generation employing the bioinformatics MAFFT (Multiple Alignment using Fast Fourier Transform) algorithm. Changes in code across a set of different metamorphic variants of the same virus will be identified through a substitution matrix that identifies the probabilities of changes in hex code between variants (all variants must be aligned together, hence multiple as opposed to pairwise alignment). Also, the degree of such changes will be used to generate a phylogenetic tree tracing the predicted evolution of metamorphic variants from the original to all descendant variants using standard phylogeny distance techniques. FFT will identify unique frequency signatures from the substitution matrix and evolutionary tree to distinguish one family of metamorphic variants from another. The approach is speculative and, if successful, will transform our understanding of signature generation.

Future Work on Chapter 7: Future work could include more work on implementing the biosequence analysis techniques on other types of malware, such as Ransomware, Trojan horse and Rootkit. On the inference that the signatures are trapping essential attributes of malware morphing, the results illustrated here may be suitable to other types of malware that also involve a morphing stage. Further work can also include the implementation of phylogenetics for the purpose of malware classification on a larger scale.

Overall: The proposed topic of this thesis will be of great interest to the malware analysts and security experts because of the increasingly growing threats of malware to the normal computing activities. The approach adopted in this thesis is unique and, as far as, the only active research in the world currently analysing complex polymorphic viral code using

state of the art bioinformatics tools and techniques for the automatic extraction of signatures for future anti-viral software development.

Malware detection adopting deep learning techniques will be considered as a future work for classification and labelling of malware data as well as feature extraction (Kolosnjaji, Zarras, Webster, & Eckert, 2016; Kalash, et al., 2018). These techniques are well-known for their accurate detection, better performance in comparison to conventional machine learning techniques, real-time malware detection and prevention, low false positives, minimal/no manual/reverse engineering, etc. (Saxe, Harang, Wild, & Sanders, 2018; Yan, Qi, & Rao, 2018).

A database of all the generated meta-signatures obtained from this thesis will be incorporated in the future automatic software systems (as a future work) thereby successfully detecting some or all new (unknown) polymorphic malware variants. The major contribution will be to add to the existing knowledge on how to detect some or all new (unseen) polymorphic malware variants with the help of this proposed approach.

Future prospects of this research are to implement an anti-viral software product that will successfully detect and predict the unseen new variants not only belonging to a polymorphic malware family but also belonging to a metamorphic malware family. There is a likelihood of attracting funding and possibly a collaboration with well-established companies like Microsoft, Bitdefender, Symantec and Malware Research Labs.

8.5. Further Work

Further work has been done on the 100 new (never seen before) variants (P_x) of JS.Cassandra virus as an extension of the work presented in this thesis. This new dataset was obtained from the original virus writer (SPTH, 2015; Second Part To Hell, 2018) of JS.Cassandra virus. The uniqueness of these 100 new malware variants was cross-checked by generating a CRC32b hash value for each variant, and no duplicates were found (see Appendix H – page no. 251). The results of that further work are demonstrated in Table 8.1. In total, 282 meta-signatures and 54 super-signatures were tested against the 100 new variants of JS.Cassandra virus. All the meta-signatures and super-signatures tested here were obtained from Chapters 4 to 6 to detect the JS.Cassandra virus and its variants. One meta-signature from Chapter 4. 37 meta-signatures and four super-signatures from Chapter 5 (Part-I). Twelve meta-signatures and three super-signatures

from Chapter 5 (Part-II). 161 meta-signatures and 47 super-signatures from Chapter 5 (Part-III). And 71 meta-signatures from Chapter 6.

From Table 8.1, it could be said that almost all of the meta-signatures and super-signatures were completely effective in detecting the new variants of the JS.Cassandra virus with an overall accuracy of 100%. Except for some, such as the two super-signatures obtained from the NWA (Chapter 5 – Part-I) had an overall accuracy of 85%, the two super-signatures obtained from the SWA (Chapter 5 – Part-I) had an overall accuracy of 85%, the twelve meta-signatures obtained from Chapter 5 – Part-II had an overall accuracy of 97%, the one meta-signature obtained from the ID substitution matrix (Chapter 5 – Part-III) had an overall accuracy of 12%, and the 24 meta-signatures obtained from the MATCH substitution matrix (Chapter 5 – Part-III) had an overall accuracy of 85%. The state of the art ClamAV and Bitdefender Total Security 2018 AVSs had overall accuracies of 85% and 0%, respectively. So, the essential contribution of this section was to add to the existing knowledge that syntactic virus signatures captured from the string-based syntactic techniques helped to identify a dataset of unknown (P_x) variants.

Table 8.1: Detection ratio using two state of the art AVSs and the 282 meta-signatures and 54 super-signatures from Chapters 5 to 7 with 'clamscan'.

Type of Files Scanned	Detection Ratio (with Accuracy) and Statistical Measures	ClamAV	One Meta-Signature (Chapter 5)	16 Meta-Signatures - NWA (Chapter 6 – Part-I)	21 Meta-Signatures - SWA (Chapter 6 – Part-I)
100 New JS.Cassandra Polymorphic Malicious Malware Variants	Detection Ratio (Accuracy)	85/100 (85.00%)	100/100 (100.00%)	100/100 (100.00%)	100/100 (100.00%)
	Sensitivity/Recall	85.00%	100.00%	100.00%	100.00%
	Specificity	0.00%	0.00%	0.00%	0.00%
	Precision	100.00%	100.00%	100.00%	100.00%
	F1 Score	91.89%	100.00%	100.00%	100.00%
Type of Files Scanned	**Detection Ratio (with Accuracy) and Statistical Measures**	**Bitdefender Total Security 2018**	**Two Super-Signatures - NWA (Chapter 6 – Part-I)**	**Two Super-Signatures - SWA (Chapter 6 – Part-I)**	**Twelve Meta-Signatures (Chapter 6 – Part-II)**
100 New JS.Cassandra Polymorphic Malicious Malware Variants	Detection Ratio (Accuracy)	0/100 (0.00%)	85/100 (85.00%)	85/100 (85.00%)	97/100 (97.00%)
	Sensitivity/Recall	0.00%	85.00%	85.00%	97.00%
	Specificity	0.00%	0.00%	0.00%	0.00%
	Precision	100.00%	100.00%	100.00%	100.00%
	F1 Score	0.00%	91.89%	91.89%	98.48%
Type of Files Scanned	**Detection Ratio (with Accuracy) and Statistical Measures**	**Three Super-Signatures (Chapter 6 – Part-II)**	**34 Meta-Signatures – BLOSUM40 (Chapter 6 – Part-III)**	**24 Meta-Signatures – DAYHOFF (Chapter 6 – Part-III)**	**One Meta-Signature – IDENTITY (Chapter 6 – Part-III)**
100 New JS.Cassandra Polymorphic Malicious Malware Variants	Detection Ratio (Accuracy)	100/100 (100.00%)	100/100 (100.00%)	100/100 (100.00%)	12/100 (12.00%)
	Sensitivity/Recall	100.00%	100.00%	100.00%	12.00%
	Specificity	0.00%	0.00%	0.00%	0.00%
	Precision	100.00%	100.00%	100.00%	100.00%
	F1 Score	100.00%	100.00%	100.00%	21.43%
Type of Files Scanned	**Detection Ratio (with Accuracy) and Statistical Measures**	**24 Meta-Signatures – MATCH (Chapter 6 – Part-III)**	**31 Meta-Signatures – PAM100 (Chapter 6 – Part-III)**	**47 Meta-Signatures – PAM350 (Chapter 6 – Part-III)**	**Seven Super-Signatures – BLOSUM40 (Chapter 6 – Part-III)**
100 New JS.Cassandra Polymorphic Malicious Malware Variants	Detection Ratio (Accuracy)	85/100 (85.00%)	100/100 (100.00%)	100/100 (100.00%)	100/100 (100.00%)
	Sensitivity/Recall	85.00%	100.00%	100.00%	100.00%
	Specificity	0.00%	0.00%	0.00%	0.00%
	Precision	100.00%	100.00%	100.00%	100.00%
	F1 Score	91.89%	100.00%	100.00%	100.00%
Type of Files Scanned	**Detection Ratio (with Accuracy) and Statistical Measures**	**Six Super-Signatures – DAYHOFF (Chapter 6 – Part-III)**	**Eleven Super-Signatures – IDENTITY (Chapter 6 – Part-III)**	**Ten Super-Signatures – MATCH (Chapter 6 – Part-III)**	**Seven Super-Signatures – PAM100 (Chapter 6 – Part-III)**
100 New JS.Cassandra Polymorphic Malicious Malware Variants	Detection Ratio (Accuracy)	100/100 (100.00%)	100/100 (100.00%)	100/100 (100.00%)	100/100 (100.00%)
	Sensitivity/Recall	100.00%	100.00%	100.00%	100.00%
	Specificity	0.00%	0.00%	0.00%	0.00%
	Precision	100.00%	100.00%	100.00%	100.00%
	F1 Score	100.00%	100.00%	100.00%	100.00%

Type of Files Scanned	Detection Ratio (with Accuracy) and Statistical Measures	Six Super-Signatures – PAM350 (Chapter 6 – Part-III)	Nine Meta-Signatures – Experiment I (Chapter 7)	14 Meta-Signatures – Experiment II (Chapter 7)	48 Meta-Signatures – Experiment III (Chapter 7)
100 New JS.Cassandra Polymorphic Malicious Malware Variants	**Detection Ratio (Accuracy)**	100/100 (100.00%)	100/100 (100.00%)	100/100 (100.00%)	100/100 (100.00%)
	Sensitivity/Recall	100.00%	100.00%	100.00%	100.00%
	Specificity	0.00%	0.00%	0.00%	0.00%
	Precision	100.00%	100.00%	100.00%	100.00%
	F1 Score	100.00%	100.00%	100.00%	100.00%

References

Abou-Assaleh, T., Cercone, N., & Sweidan, R. (2004). Detection of new malicious code using N-grams signatures. *Proceedings Second Annual Conference on Privacy, Security and Trust PST* (pp. 193-196). New Brunswick: University of New Brunswick UNB Libraries.

Abuzaid, A. M., Saudi, M. M., Taib, B. M., & Abdullah, Z. H. (2013). An Efficient Trojan Horse Classification (ETC). *International Journal of Computer Science Issues (IJCSI)*, 96-104: IJCSI.

Adleman, L. M. (1988). An abstract theory of computer viruses. *Advances in Cryptology - CRYPTO '88, Proceedings Eighth Annual International Cryptology Conference* (pp. 354–374). Santa Barbara, California: Springer International Publishing.

Agrawal, H. (2012). *United States Patent No. US20120072988A1*.

Ali, Z., & Soomro, T. (2018). An Efficient Mining Based Approach using PSO Selection Technique for Analysis and Detection of Obfuscated Malware. *Journal of Information Assurance & Cyber security*, 1-13: IBIMA Publishing.

Altschul, S. (1991). Amino acid substitution matrices from an information theoretic perspective. *Journal of Molecular Biology*, 555-565: Elsevier.

Aniello, L. (2016, March 4). *Research Center for Cyber Intelligence and Information Security.* Retrieved from Sapienza - University of Rome - CIS: http://www.dis.uniroma1.it/~aniello/ssd2016/machine_learning_for_malware_detection.pdf: CIS SAPIENZA, Access Date: October, 2016.

AO Kaspersky Lab. (2017). *Machine Learning and Human Expertise.* Retrieved from Kaspersky Lab: https://media.kaspersky.com/en/business-security/machine-learning-human-expertise.pdf: Kaspersky Lab, Access Date: March 2018.

AO Kaspersky Lab. (2017, March 30). *Machine learning in Kaspersky Endpoint Security 10 for Windows*. Retrieved from Kaspersky Lab: https://support.kaspersky.com/13263: Kaspersky Lab, Access Date: March 2018.

AO Kaspersky Lab. (2018, February 5). *Heuristic analysis in Kaspersky Endpoint Security 10 for Windows*. Retrieved from Kaspersky Lab: https://support.kaspersky.com/12370: Kaspersky Lab, Access Date: March 2018.

Armitage, A. (2007). Mutual Research Designs: Redefining Mixed Methods Research Design. *British Educational Research Association Annual Conference* (pp. 1-10). London: Institute of Education, University of London.

Armoun, S. E., & Hashemi, S. (2013). A General Paradigm for Normalizing Metamorphic Malwares. *Frontiers of Information Technology (FIT), 2012 10th International Conference on Frontiers of Information Technology*. Islamabad: IEEE.

Attaluri, S., McGhee, S., & Stamp, M. (2009). Profile hidden Markov models and metamorphic virus detection. *Journal in Computer Virology*, 151–169: Springer.

Audi, R. (2002). The sources of knowledge. In P. Moser, *The Oxford Handbook of Epistemology* (pp. 71-94). Oxford, United Kingdom: Oxford University Press.

Aycock, J. (2006). *Computer Viruses and Malware.* New York City: Springer US.

Baichoo, S., & Ouzounis, C. (2017). Computational complexity of algorithms for sequence comparison, short-read assembly and genome alignment. *Biosystems*, 72-85: Elsevier.

Bailey, M., Oberheide, J., Andersen, J., Mao, Z., Jahanian, F., & Nazario, J. (2007). Automated classification and analysis of internet malware. *RAID'07 Proceedings Tenth International Conference on Recent Advances in Intrusion Detection* (pp. 178-197). Gold Goast: Springer Berlin Heidelberg.

Baldangombo, U., Jambaljav, N., & Horng, S.-J. (2013, August 13). *[1308.2831] A static malware detection system using data mining methods*. Retrieved from arXiv: https://arxiv.org/abs/1308.2831: Cornell University Library, Access Date: March 2016.

Ball, J. R. (2014, March). *Detection and Prevention of Android Malware Attempting to Root the Device.* Retrieved from Defense Technical Information Center: http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA600990: Defense Technical Information Center, Access Date: September 2016.

Baratloo, A., Hosseini, M., Negida, A., & El Ashal, G. (2015). Part 1: Simple Definition and Calculation of Accuracy, Sensitivity and Specificity. *Emergency*, 48-49: BMJ Publishing Group Ltd and the College of Emergency Medicine.

Bearden, R. L.-T. (2017). Automated Microsoft Office Macro Malware Detection Using Machine Learning. *International Conference on Big Data (BIGDATA)* (pp. 4448-4452). Boston, MA: IEEE.

Beaucamps, P. (2007). Advanced Polymorphic Techniques. *International Journal of Computer Science*, 194-205: IAENG.

Belcebu, B. (2014). *Tutorials – Win32 Polymorphism*. Retrieved from VX Heavens: http://vxheaven.org/lib/static/vdat/tuwin32p.htm: VX Heaven, Access Date: September 2015.

Belcebu, B. (n.d.). *VX Heaven*. Retrieved from VX Heaven: VX Heaven, Access Date: September 2015.

Berg, P. E. (2011). *Behavior-based Classification of Botnet Malware.* Gjøvik, Norway: Department of Computer Science and Media Technology, Gjøvik University College, Access Date: September 2016.

Berthold, M. (2004, 12 20). *Random Data File Creator (RDFC)*. Retrieved from Bertel: http://www.bertel.de/software/rdfc/index-en.html: Bertel, Access Date: September 2015.

Beveridge, D., Karnik, A., Beets, K., Heppner, T., & Raman, K. (2017). *United States Patent No. US20170061125A1*.

Bitdefender. (2016, December 12). *Machine-learning powers Bitdefender's intellectual property program*. Retrieved from Bitdefender: https://www.bitdefender.com/news/machine-learning-powers-bitdefenders-intellectual-property-program-3226.html: Bitdefender, Access Date: April 2018.

Bitdefender. (2017). *What is Bitdefender Advanced Threat Defense*. Retrieved from Bitdefender: https://www.bitdefender.com/consumer/support/answer/2024/: Bitdefender, Access Date: April 2018.

Borello, J.-M. F. (2010). From the design of a generic metamorphic engine to a black-box classification of antivirus detection techniques. *Journal in Computer Virology*, 277–287: Springer.

Borello, J.-M., & Me, L. (2008). Code obfuscation techniques for metamorphic viruses. *Journal of Computer Virology*, 211-220: Springer.

Brown University. (2010, October 21). *Malicious Software - Ch04-Malware.* Retrieved from Brown University - Department of Computer Science: http://cs.brown.edu/cgc/net.secbook/se01/handouts/Ch04-Malware.pdf: Brown University, Access Date: October 2016.

Carlin, D., O'Kane, P., & Sezer, S. (2017). Dynamic Analysis of Malware Using Run-Time Opcodes. In I. P. Carrascosa, H. K. Kalutarage, & Y. Huang, *Data Analytics and Decision Support for Cybersecurity* (pp. 99-125). Berlin, Germany: Springer.

Cassandra. (2018, February 10). *Index of /cassandra*. Retrieved from Apache Software Foundation Distribution Directory: http://apache.mirror.serversaustralia.com.au/cassandra/: Apache Software Foundation Distribution Directory, Access Date: April 2018.

Cendrowska, J. (1987). PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 349-370: Elsevier.

Cesare, S. (2010, May). *Fast Automated Unpacking and Classification of Malware.* Retrieved from covert.io: http://www.covert.io/research-papers/security/Fast%20Automated%20Unpacking%20and%20Classification%20of%20Malware.pdf: covert.io, Access Date: March 2016.

Cesare, S., & Xiang, Y. (2010). Classification of malware using structured control flow. *Proceedings Eight Australasian Symposium on Parallel and Distributed Computing AusPDC* (pp. 61-70). Brisbane: Australian Computer Society, Inc.

Chakraborty, A., & Bandyopadhyay, S. (2013). FOGSAA: Fast Optimal Global Sequence Alignment Algorithm. *Nature Scientific Reports*, 1-9: Nature.

Charras, C., & Lecroq, T. (1997, January 14). *Exact String Matching Algorithms*. Retrieved from Rouen Computing Laboratory, University of Rouen: http://www-igm.univ-mlv.fr/~lecroq/string/index.html: University of Rouen, Access Date: March 2016.

Chaumette, S., Ly, O., & Tabary, R. (2011). Automated extraction of polymorphic signatures using abstract interpretation. *Proceedings Fifth International Conference on Network and Systems Security NSS* (pp. 41-48). Milan: IEEE.

Chen, S.-Y., Jeng, T.-H., Huang, C.-C., Chen, C.-C., & Chou, K.-S. (2017). Doctrina: Annotated Bipartite Graph Mining for Malware-Control Domain Detection. *International Conference on Communication and Network Security* (pp. 67-75). Tokyo, Japan: ACM.

Chen, Y., Narayanan, A., Pang, S., & Ban, T. (2012). Malicious Software Detection Using Multiple Sequence Alignment and Data Mining. *The Twenty-Sixth IEEE International Conference on Advanced Information Networking and Applications AINA* (pp. 8–14). Fukuoka: IEEE.

Chen, Y., Narayanan, A., Pang, S., & Ban, T. (2012). Multiple sequence alignment and artificial neural networks for malicious software detection. *Proceedings of 2012 Eighth International Conference on Natural Computation ICNC* (pp. 261-265). Chongqing, Sichuan: IEEE.

Chen, Y., Wan, A., & Liu, W. (2006). A fast parallel algorithm for finding the longest common sequence of multiple biosequences. *BMC Bioinformatics*, 1-12: Springer Nature.

Choo, K.-K. R. (2007, March). *Zombies and botnets.* Retrieved from Australian Government - Australian Institute of Criminology: http://aic.gov.au/media_library/publications/tandi_pdf/tandi333.pdf: Australian Institute of Criminology, Access Date: September 2015.

Choudhary, S. P., & Vidyarthi, D. (2015). A Simple Method for Detection of Metamorphic Malware using Dynamic Analysis and Text Mining. *International Conference on Communication Networks* (pp. 265-270). Bangalore: Elsevier.

Chowdhury, A., Kumar, A., Dubey, H., & Shekokar, N. (2017). *United States Patent No. US20170193229A1*.

Christodorescu, M., Jha, S., Seshia, S., Song, D., & Bryant, R. (2005). Semantics-Aware Malware Detection. *IEEE Symposium on Security and Privacy* (pp. 32-46). Oakland, California: IEEE Computer Society.

Christodorescu, M., Kinder, J., Jha, S., Katzenbeisser, S., & Veith, H. (2005, November 22). *Malware Normalization*. Retrieved from Department of Computer Sciences, University of Wisconsin: http://pages.cs.wisc.edu/~mihai/publications/Malware%20Normalization/: University of Wisconsin, Access Date: September 2016.

Chua, M., & Balachandran, V. (2018). Effectiveness of Android Obfuscation on Evading Anti-malware. *ACM Conference on Data and Application Security and Privacy* (pp. 143-145). Tempe, AZ: ACM.

ClamAV. (2016, May 3). *ClamavNet*. Retrieved from ClamAV: http://www.clamav.net/download.html: ClamAV, Access Date: September 2016.

ClamAV. (2018). *ClamavNet*. Retrieved from ClamAV: https://www.clamav.net/: ClamAV, Access Date: April 2018.

CLC bio. (2007). *Bioinformatics explained: Smith-Waterman.* Aarhus, Denmark: CLC bio.

Clustal. (2012, August 31). *Clustal: Multiple Sequence Alignment*. Retrieved from Clustal: http://www.clustal.org/download/clustalw_help.txt: Clustal, Access Date: September 2015.

Codreanu, D., Neagu, M., & Chiriac, M. (2014). *United States Patent No. US8813222B1*.

Cohen, F. B. (1987). Computer viruses: Theory and experiments. *Computers & Security*, 22–35.

Cohen, F. B. (1989). Computational aspects of computer viruses. *Computers & Security*, 325–344: Elsevier.

Copy, M. (. (2016). *The Ten Most Common Spyware Threats*. Retrieved from Street Directory: http://www.streetdirectory.com/travel_guide/158768/security/the_ten_most_common_spyware_threats.html: Streetdirectory Pte Ltd, Access Date: September 2016.

Coull, S., & Szymanski, B. (2008). Sequence alignment for masquerade detection. *Computational Statistics & Data Analysis*, 4116-4131.

Cowie, N. A., Muttik, I. G., & Wolff, D. J. (2008). *United States Patent No. US7346781B2.*

Creswell, J. (2014). *Research Design: Qualitative, Quantitative and Mixed Methods Approaches.* California, United States: SAGE Publications.

Crotty, M. (1998). *The foundations of social research: Meaning and Perspective in the Research.* California, United States: SAGE Publications Ltd.

Cui, W., Peinado, M., Wang, H., & Locasto, M. (2007). Shieldgen: Automatic data patch generation for unknown vulnerabilities with informed probing. *Symposium on Security and Privacy* (pp. 252-266). Oakland, CA: IEEE.

Datta, R., & Saha, S. (2011, August). *An Empirical comparison of rule based classification techniques in medical databases.* Retrieved from IDEAS: https://ideas.repec.org/p/ift/wpaper/1107.html: IDEAS, Access Date: September 2016.

Dayhoff, M. O., Schwartz, R. M., & Orcutt, B. C. (1978). A model of evolutionary change in proteins (1978). *Atlas of Protein Sequence and Structure*, 345-351: National Biomedical Research Foundation.

Debray, S. K., Coogan, K., & Townsend, G. M. (2008). *On the Semantics of Self-Unpacking. Malware Code.* Tucson: Dept of Computer Science, University of Arizona.

Devesa, J., Santos, I., Cantero, X., Penya, Y., & Bringas, P. G. (2010). Automatic Behaviour-based Analysis and Classification System for Malware Detection. *International Conference on Enterprise Information Systems* (pp. 395–399). Funchal, Madeira - Portugal: SciTePress.

Diao, L., Chan, V., & Lu, P. (2015). *United States Patent No. US8935788B1*.

Dohi, K., Benkrid, K., Ling, C., Hamada, T., & Shibata, Y. (2010). Highly efficient mapping of the Smith-Waterman algorithm on CUDA-compatible GPUs. *International Conference on Application-specific Systems Architectures and Processors* (pp. 29-36). Rennes, France: IEEE.

Dumitraş, T., & Neamtiu, I. (2011). Experimental Challenges in Cyber Security: A Story of Provenance and Lineage for Malware. *Conference on Cyber security experimentation and test* (pp. 1-9). San Francisco, CA: USENIX Association Berkeley.

Eddy, S. (2004). Where did the BLOSUM62 alignment score matrix come from? *Nature Biotechnology*, 1035–1036: Nature.

Elhadi, A. A., Maarof, M. A., & Osman, A. H. (2012). Malware detection based on hybrid signature behaviour application programming interface call graph. *American Journal of Applied Sciences*, 283-288: Science Publications.

Ellis, D., Aiken, J., Attwood, K., & Tenaglia, S. (2004). A Behavioral Approach to Worm Detection. *WORM '04 Proceedings of the ACM Workshop on Rapid Malcode* (pp. 43-53). Washington, DC: ACM.

EMBL-EBI. (2016). *EMBL European Bioinformatics Institute*. Retrieved from The European Bioinformatics Institute: http://www.ebi.ac.uk/: The European Bioinformatics Institute, Access Date: March 2016.

EMBOSS Matcher. (2018). *EMBOSS Matcher*. Retrieved from EMBL-EBI: https://www.ebi.ac.uk/Tools/psa/emboss_matcher/: The European Bioinformatics Institute, Access Date: March 2016.

EmEditor. (2018). *EmEditor (Text Editor) – Text Editor for Windows supporting large files and Unicode!:*. Retrieved from EmEditor: https://www.emeditor.com/: EmEditor, Access Date: March 2018.

Eskandari, M., & Hashemi, S. (2012). A graph mining approach for detecting unknown malware. *Journal of Visual Languages and Computing*, 154-162: Elsevier.

Fazlali, M., & Khodamoradi, P. (2018). Metamorphic Malware Detection Using Minimal Opcode Statistical Patterns. In Y. Maleh, *Security and Privacy Management, Techniques, and Protocols* (pp. 337-359). Hershey, Pennsylvania: IGI Global.

Feng, Y., Bastani, O., Martins, R., Dillig, I., & Anand, S. (2017). Automated Synthesis of Semantic Malware Signatures using Maximum Satisfiability. *NDSS Symposium 2017* (pp. 1-15). San Diego, California: Internet Society.

Ferrie, P. (2013, May 3). *Virus Bulletin :: Read the Transcript.* Retrieved from Virus Bulletin: https://www.virusbulletin.com/virusbulletin/2013/05/read-transcript: Virus Bulletin, Access Date: March 2015.

Ferris, T. (2006, August 2). *The Art of Stealthy Viruses.txt.* Retrieved from The Hackademy: http://repo.thehackademy.net/depot_madchat/vxdevl/library/The%20Art%20of%20Stealthy%20Viruses.txt: Hackademy, Access Date: March 2015.

Filiol, E. (2002). *Applied Cryptanalysis of Cryptosystems and Computer Attacks Through Hidden Ciphertexts Computer Viruses.* France: INRIA.

Filiol, E. (2007). Metamorphism, formal grammars and undecidable code mutation. *International Journal of Computer Science*, 70-75: IAENG.

Fisher, S. (2016, November 21). *Free Online Virus Scanners*. Retrieved from the balance: https://www.thebalance.com/free-online-virus-scanners-1356651: The Balance, Access Date: March 2016.

Flake, H. (2004). Structural Comparison of Executable Objects. *Proceedings of IEEE Conference on Detection of Intrusions and Malware and Vulnerability Assessment* (pp. 161–173). Dortmund: IEEE.

Fosnock, C. (2005). *Computer Worms: Past, Present, and Future.* East Carolina: East Carolina University.

Frank, E., Hall, M. A., & Witten, I. H. (2016). *Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques" (Fourth Edition).* Burlington: Morgan Kaufmann. Retrieved from Department of Computer Science, The University of Waikato - WEKA: http://www.cs.waikato.ac.nz/ml/weka/: Morgan Kaufmann, Access Date: January 2017.

Fukushima, Y., Sakai, A., Hori, Y., & Sakurai, K. (2010). A behaviour based malware detection scheme for avoiding false positive. *Proceedings Sixth IEEE Workshop on Secure Network Protocols NPSec* (pp. 79-84). Kyoto: IEEE.

Gao, D., Reiter, M. K., & Song, D. (2005). Behavioral distance for intrusion detection. *8th International Symposium, RAID* (pp. 63–81). Seattle, WA: Springer Berlin Heidelberg.

Garhwal, A. S. (2018). *Bioinformatics-inspired analysis for watermarked images with multiple print and scan.* Auckland: Auckland University of Technology, Access Date: April 2018.

Gartside, P. (2005). *United States Patent No. US6851058B1*.

Geers, M., Çağlayan, F., & Heij, R. (2013). *Low-Cost Smith-Waterman Acceleration.* Delft, Netherlands: Delft University of Technology.

Ghayyur, O., Aleem, M., & Islam, M. (2018). Time Efficient Novel Parallel Technique for Needleman Wunsch and Smith Waterman Algorithms. *International Conference on Computing, Mathematics and Engineering Technologies* (pp. 1-6). Sukkur, Pakistan: IEEE.

Gold, E. M. (1967). Language identification in the limit. *Information and Control*, 447-474: Elsevier.

Goldhaber, A. S., & Nieto, M. M. (2010). Photon and Graviton Mass Limits. *Reviews of Modern Physics*, 939-979: American Physical Society.

Gonnet, G. (1998, September 15). *Dayhoff Matrices*. Retrieved from Department of Computer Science, ETH Zurich: https://www.inf.ethz.ch/personal/gonnet/DarwinManual/node149.html: ETH Zurich, Access Date: April 2016.

Gragido, W., & Elisan, C. (2012). *Polymorphic and Metaphoric Threats and Your Cyber Future.* Retrieved from Black Hat: https://www.blackhat.com/docs/webcast/polymorphis-and-metaphoric-threats-and-your-cyber-future.pdf: Black Hat, Access Date: April 2015.

Green, J. P., Chandnani, A. D., & Christensen, S. D. (2011). *United States Patent No. US20120266244A1*.

Green, J. P., Chandnani, A. D., & Christensen, S. D. (2018). *United States Patent No. US9858414B2*.

Griffin, K., Schneider, S., Hu, X., & Chiueh, T.-c. . (2009). Automatic generation of string signatures for malware detection. *International Workshop on Recent Advances in Intrusion Detection* (pp. 101-120). Saint-Malo: Springer Berlin Heidelberg.

Gualtieri, M. (2002). *Securing Software: The Methods, the Problems, the Solutions?* Pittsburgh: Department of Computer Science, University of Pittsburgh, Access Date: April 2017.

Gurnani, B. (2017). *Malware Detection using the Index of Coincidence.* San Jose, California: San Jose State University, Access Date: April 2017.

Hajmasan, G. F., Lukacs, S., & Fulop, B. (2016). *United States Patent No. US9460284B1*.

Hamm, J., & Johnson, S. (2002, September 11). *Polymorphic Viruses A brief survey.* Retrieved from University of Verona: http://profs.sci.univr.it/~giaco/download/Watermarking-Obfuscation/Polymorph_final.ppt: University of Verona, Access Date: April 2016.

Hassen, M. C. (2017). Malware classification using static analysis based features. *Symposium Series on Computational Intelligence (SSCI)* (pp. 1-7). Honolulu, HI: IEEE.

Heikki, M. (1997). Methods and Problems in Data Mining. In M. Heikki, *Database Theory — ICDT '97* (pp. 41-55). Heidelberg: Springer Berlin Heidelberg.

Heller, S. (1996). Symantec Norton Antivirus 2.0 for Windows 95. *Journal of Chemical Information and Modeling*, 1229–1229: American Chemical Society.

Henikoff, S., & Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences (PNAS)*, 10915–10919: United States National Academy of Sciences.

Hexacorn Limited. (2012, September 2). *PESectionExtractor – Extracting PE sections and their strings*. Retrieved from Hexacorn Limited: http://www.hexacorn.com/blog/2012/09/02/pesectionextractor-extracting-pe-sections-and-their-strings/: Hexacorn Limited, Access Date: April 2016.

Hood, G. (2016). *Online Virus Scanner*. Retrieved from Gary's Hood: http://www.garyshood.com/virus/: Gary's Hood, Access Date: October 2016.

Hosmer, C. (2008). *Polymorphic & Metamorphic Malware.* Retrieved from Black Hat: https://www.blackhat.com/presentations/bh-usa-08/Hosmer/BH_US_08_Hosmer_Polymorphic_Malware.pdf: Black Hat, Access Date: April 2016.

HSIAO, H.-C., DENG, S., Salamat, B., Gupta, R., & Das, S. M. (2017). *United States Patent No. US9832211B2.*

Huang, Q., CAO, H., & Yu, K. (2017). *United States Patent No. US9817974B1*.

Huang, X., & Miller, W. (1991). A time-efficient, linear-space local similarity algorithm. *Advances in Applied Mathematics*, 337–357: Elsevier.

HubPages. (2014, October 19). *The History of the Computer Viruses*. Retrieved from HubPages: http://hubpages.com/technology/The-History-of-the-Computer-Viruses: HubPages, Access Date: August 2016.

Huda, S. A. (2017). A fast malware feature selection approach using a hybrid of multi-linear and stepwise binary logistic regression. *Concurrency and Computation: Practice and Experience*, 1-18: John Wiley & Sons Ltd.

Huda, S., Miah, S., Hassan, M. M., Islam, R., Yearwooda, J., Alrubaiand, M., & Almogren, A. (2017). Defending unknown attacks on cyber-physical systems by semi-supervised approach and available unlabeled data. *Information Sciences*, 211-228: Elsevier.

Idika, N., & Mathur, A. P. (2007). *A survey of malware detection techniques, Technical Report 286.* Retrieved from SERC Security and Software Engineering Research Center: http://www.serc.net/system/files/SERC-TR-286.pdf: SERC Security and Software Engineering Research Center, Access Date: September 2016.

Infoplease. (2014, August). *Computer Virus Timeline*. Retrieved from Infoplease: http://www.infoplease.com/ipa/A0872842.html: Infoplease, Access Date: September 2016.

Irshad, M., al-Khateeb, H., Mansour, A., Ashawa, M., & Hamisu, M. (2018). Effective methods to detect metamorphic malware: a systematic review. *International Journal of Electronic Security and Digital Forensics*, 138-154: Inderscience Publishers.

Jalview. (2018). *Jalview*. Retrieved from Jalview: http://www.jalview.org/: Jalview, Access Date: September 2017.

jcaparas, & Hall, J. (2018, April 24). *Windows Defender Advanced Threat Protection.* Retrieved from Microsoft: https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-atp/windows-defender-advanced-threat-protection: Microsoft, Access Date: March 2018.

Jidigam, R., Austin, T., & Stamp, M. (2015). Singular value decomposition and metamorphic detection. *Journal of Computer Virology and Hacking Techniques*, 203–216: Springer.

Johanson, G., & Williamson, K. (2013). *Research Methods: Information, Systems and Contexts.* Prahran, Victoria: Tilde Publishing and Distribution.

Johnston, A. (2014, May 23). *Computer Virus Timeline*. Retrieved from Prezi: https://prezi.com/e-go-dl_ynji/computer-virus-timeline/: Prezi Inc., Access Date: March 2016.

Kalash, M., Rochan, M., Mohammed, N., Bruce, N., Wang, Y., & Iqbal, F. (2018). Malware Classification with Deep Convolutional Neural Networks. *International Conference on New Technologies, Mobility and Security* (pp. 1-5). Paris, France: IEEE.

Kaspersky. (2016). *About Us | Kaspersky Lab*. Retrieved from Kaspersky Lab: http://www.kaspersky.com/about: Kaspersky Lab, Access Date: April 2016.

Katoh, K. (2018, July). *MAFFT alignment and NJ / UPGMA phylogeny*. Retrieved from MAFFT Computational Biology Research Consortium: http://mafft.cbrc.jp/alignment/server/index.html: Computational Biology Research Consortium, Access Date: April 2016.

Katoh, K., & Standley, D. (2013). MAFFT Multiple Sequence Alignment Software Version 7: Improvements in Performance and Usability. *Molecular Biology and Evolution*, 772–780.

Katoh, K., Misawa, K., Kuma, K., & Miyata, T. (2002). MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, 3059–3066: Oxford Academic.

Kaur, R., & Singh, M. (2014). Efficient hybrid technique for detecting zero-day polymorphic worms. *IEEE International Advance Computing Conference* (pp. 95-100). Gurgaon, India: IEEE.

Kaushal, K., Swadas, P., & Prajapati, N. (2012). Metamorphic Malware Detection Using Statistical Analysis. *International Journal of Soft Computing and Engineering*, 49-53: Blue Eyes Intelligence Engineering & Sciences Publication Pvt. Ltd.

Kearse, M., Moir, R., Wilson, A., Stones-Havas, S., Cheung, M., Sturrock, S., . . . Drummond, A. (2012). Geneious Basic: an integrated and extendable desktop software platform for the organization and analysis of sequence data. *Bioinformatics*, 1647–1649: Oxford Academic.

Kephart, J., & Arnold, W. (1994). Automatic Extraction of Computer Virus Signatures. *Fourth Virus Bulletin International Conference* (pp. 178–184). England: Virus Bulletin Ltd.

Khan, M., Siddiqui, S., & Ferens, K. (2017). Cognitive modeling of polymorphic malware using fractal based semantic characterization. *International Symposium on Technologies for Homeland Security* (pp. 1-7). Waltham, MA: IEEE.

Ki, Y., Kim, E., & Kim, H. (2015). A Novel Approach to Detect Malware Based on API Call Sequence Analysis. *International Journal of Distributed Sensor Networks*, 1-9: SAGE Publications.

Kim, H.-A., & Karp, B. (2004). Autograph: Toward automated, distributed worm signature detection. *USENIX Security Symposium* (pp. 19-19). San Diego, CA: USENIX Association Berkeley.

Kim, J., & Pramanik, S. (1994). An efficient method for multiple sequence alignment. *International Conference on Intelligent Systems for Molecular Biology* (pp. 212-218). Stanford, California: AAAI Press.

Kinder, J., Katzenbeisser, S., Schallhart, C., & Veith, H. (2005). Detecting malicious code by model checking. *Second International Conference, DIMVA* (pp. 174-187). Vienna: Springer Berlin Heidelberg.

Kirat, D., & Vigna, G. (2015). MalGene: Automatic Extraction of Malware Analysis Evasion Signature. *ACM SIGSAC Conference on Computer and Communications Security* (pp. 769-780). Denver, Colorado: ACM.

Koklu, M., Kahramanli, H., & Allahverdi, N. (2015). Applications of Rule Based Classification Techniques for Thoracic Surgery. *Management, Knowledge and Learning - Joint International Conference 2015 - Technology, Innovation and Industrial Management TIIM* (pp. 1991–1998). Bari: ToKnowPress, Access Date: April 2016.

Kolesnikov, O., & Lee, W. (2004). Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic. *Usenix Security Symposium* (pp. 1-22). San Diego, CA: Georgia Tech Library, Access Date: August 2017.

Kolosnjaji, B., Zarras, A., Webster, G., & Eckert, C. (2016). Deep Learning for Classification of Malware System Call Sequences. *Australasian Joint Conference on Artificial Intelligence* (pp. 137-149). Hobart, Australia: Springer.

Kolter, J., & Maloof, M. (2006). Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 2721-2744: JMLR, Inc. and Microtome Publishing.

Komashinskiy, D., & Kotenko, I. (2010). Malware detection by data mining techniques based on positionally dependent features. *Proceedings Eighteenth Euromicro Conferences on Parallel, Distributed and Network-based Processing PDP* (pp. 617-623). Pisa: IEEE Computer Society.

Konstantinou, E., & Wolthusen, S. (2008). *Metamorphic Virus: Analysis and Detection.* Egham: Royal Holloway, University of London.

Korczynski, D., & Yin, H. (2017). Capturing Malware Propagations with Code Injections and Code-Reuse Attacks. *Conference on Computer and Communications Security* (pp. 1691-1708). Dallas, Texas: ACM.

Kothari, C. (2004). *Research Methodology: Methods and Techniques.* New Delhi, Delhi: New Age International.

Koyutürk, M. G. (2005). Pairwise Local Alignment of Protein Interaction Networks Guided by Models of Evolution. *Annual International Conference on Research in Computational Molecular Biology* (pp. 48-65). Cambridge, Massachusetts: Springer.

Kreibich, C., & Crowcroft, J. (2004). Honeycomb: creating intrusion detection signatures using honeypots. *ACM SIGCOMM computer communication review*, 51-56: ACM.

Kruegel, C., Kirda, E., Mutz, D., Robertson, W., & Vigna, G. (2005). Polymorphic worm detection using structural information of executables. *RAID'05 Proceedings of the 8th international conference on Recent Advances in Intrusion Detection* (pp. 207-226). Heidelberg: Springer-Verlag Berlin.

Kubovič, O. (2017, June 20). *Machine learning by ESET: The road to Augur*. Retrieved from welivesecurity by ESET: https://www.welivesecurity.com/2017/06/20/machine-learning-eset-road-augur/: WeLiveSecurity, Access Date: October 2017.

Kumar, A. (2016, June 10). *What is a Polymorphic Virus and how do you deal with it*. Retrieved from The Windows Club: http://www.thewindowsclub.com/polymorphic-virus: The Windows Club, Access Date: April 2016.

Kumar, S., & Filipski, A. (2007). Multiple sequence alignment: In pursuit of homologous DNA positions. *Genome Research*, 127-135: Cold Spring Harbor Laboratory Press.

Kumar, V., & Mishra, S. K. (2013). Detection of Malware by using Sequence Alignment Strategy and Data Mining Techniques. *International Journal of Computer Applications*, 16-19: Foundation of Computer Science.

Lal, D., & Verma, M. (2017). Large-Scale Sequence Comparison. In J. Keith, *Bioinformatics. Methods in Molecular Biology* (pp. 191-224). New York, NY: Humana Press.

LAVASOFT. (2013, November 5). *History of Malware*. Retrieved from LAVASOFT: http://www.lavasoft.com/mylavasoft/company/blog/history-of-malware: LAVASOFT, Access Date: June 2016.

Leder, F., Steinbock, B., & Martini, P. (2009). Classification and detection of metamorphic malware using value set analysis. *International Conference on Malicious and Unwanted Software* (pp. 39-46). Montreal: IEEE.

Leder, F., Steinbock, B., & Martini, P. (2009). Classification and Detection of Metamorphic Malware using Value Set Analysis. *International Conference on Malicious and Unwanted Software* (pp. 39-46). Montreal, QC: IEEE.

Li, X., Loh, P., & Tan, F. (2011). Mechanisms of Polymorphic and Metamorphic Viruses. *European Intelligence and Security Informatics Conference* (pp. 149-154). Athens, Greece: IEEE.

Li, Z., Sanghi, M., Chen, Y., Kao, M.-Y., & Chavez, B. (2006). Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience. *Symposium on Security and Privacy* (pp. 32-47). Berkeley/Oakland, CA: IEEE.

Lindorfer, M., Kolbitsch, C., & Comparetti, P. M. (2011). Detecting Environment-Sensitive Malware. *International Workshop on Recent Advances in Intrusion Detection* (pp. 338-357). Menlo Park, CA: Springer.

Linux Mint. (2016, October 28). *Linux Mint – from freedom came elegance*. Retrieved from Linux Mint: https://www.linuxmint.com/: Linux Mint, Access Date: November 2016.

Liu, F. (2014). *Intelligent collision detection and avoidance techniques for autonomous agents.* Auckland: Auckland University of Technology, Access Date: April 2018.

Lowe, T. (1992, December 10). *gapmis/EDNAFULL.h* . Retrieved from Github: https://github.com/xflouris/gapmis/blob/master/EDNAFULL.h: Github, Access Date: September 2016.

Lukacs, S., & LUTAS, A. V. (2016). *United States Patent No. US9531735B1*.

Marczyk, G., DeMatteo, D., & Festing, D. (2005). *Essentials of Research Design and Methodology.* Hoboken, New Jersey: John Wiley & Sons.

Marinescu, A. (2003, March 4). *An Analysis of Simile*. Retrieved from Symantec: https://www.symantec.com/connect/articles/analysis-simile: Symantec, Access Date: April 2015.

Martin, B. (1995, May). *Instance-based learning: nearest neighbour with generalisation.* Retrieved from Research Commons, University of Waikato Research: http://researchcommons.waikato.ac.nz/handle/10289/1095: University of Waikato, Department of Computer Science, Access Date: May 2016.

Mathews, M. (2017, September 14). *jsdoc*. Retrieved from GitHub Inc.: https://github.com/jsdoc3/jsdoc: Github Inc., Access Date: October 2017.

McAfee. (1999, December 30). *Virus Profile: Brain*. Retrieved from McAfee Inc.: https://home.mcafee.com/virusinfo/virusprofile.aspx?key=221: McAfee Inc., Access Date: September 2016.

McLaughlin, N., Rincon, J. M., Kang, B., Yerima, S., Miller, P., Sezer, S., . . . Ahn, G. J. (2017). Deep Android Malware Detection. *Conference on Data and Application Security and Privacy* (pp. 301-308). Scottsdale, Arizona: ACM.

MIT license. (2017, December 19). *DataTables*. Retrieved from GitHub Inc.: https://github.com/DataTables/DataTables: Github Inc., Access Date: October 2017.

MIT License. (2018, February 10). *angular.js*. Retrieved from GitHub Inc.: https://github.com/angular/angular.js: Github Inc., Access Date: October 2017.

MIT license. (2018, February 10). *Chart.js*. Retrieved from GitHub Inc.: https://github.com/chartjs/Chart.js: Github Inc., Access Date: October 2017.

Mitchell, T. M. (1997). *Machine learning.* Maidenhead: McGraw-Hill.

Mohamed, G., & Ithnin, N. (2017). SBRT: API Signature Behaviour Based Representation Technique for Improving Metamorphic Malware Detection. *International Conference of Reliable Information and Communication Technology* (pp. 767-777). Johor Bahru, Malaysia: Springer.

Moreland, K. (2006). *Basic Local Alignment Search Tool (BLAST).* Dallas: University of Texas, Access Date: October 2017.

Moser, A., Kruegel, C., & Kirda, E. (2007). Limits of static analysis for malware detection. *Proceedings IEEE Twenty-Third Annual Computer Security Applications Conference ACSAC* (pp. 421-430). Florida: IEEE.

Moskovitch, R., Elovici, Y., & Rokach, L. (2008). Detection of unknown computer worms based on behavioral classification of the host. *Journal of Computational Statistics & Data Analysis*, 4544-4566: Elsevier.

Moustafa, A. (2010, March 31). *JAligner: Java Implementation of the Smith-Waterman algorithm for biological sequence alignment*. Retrieved from SourceForge: http://jaligner.sourceforge.net/: DHI Group, Inc., Access Date: October 2015.

Mr.doob. (2018, February 10). *three.js*. Retrieved from GitHub Inc.: https://github.com/mrdoob/three.js: Github Inc., Access Date: October 2017.

Musale, M., Austin, T., & Stamp, M. (2015). Hunting for metamorphic JavaScript malware. *Journal of Computer Virology and Hacking Techniques*, 89–102: Springer.

Musil, S. (2014, November 23). *Stealthy Regin malware is a 'top-tier espionage tool'*. Retrieved from CNET: https://www.cnet.com/news/stealth-malware-found-spying-on-telecoms-energy-sectors/: CNET, Access Date: October 2016.

Nachenberg, C. (1996). *Understanding and Managing Polymorphic Viruses.* California: The Symantec Enterprise Papers, Access Date: October 2017.

Nachenberg, C. (1998). *United States Patent No. US5854916A.*

Nachenberg, C. (2006). *United States Patent No. US7130981B1.*

Nachenberg, C. S. (1998). *United States Patent No. US5826013A.*

Nachenberg, C. S. (2002). *United States Patent No. US6357008B1.*

Nachenberg, C., & Szor, P. (2008). *United States Patent No. US7337471B2.*

Naidu, V. (2018, February 23). *Windows_Systems_Files_2018*. Retrieved from MediaFire: http://www.mediafire.com/file/024064jz6ce3jge/Windows_Systems_Files_2018.rar: MediaFire, Access Date: March 2018.

Naidu, V., & Narayanan, A. (2014). Further experiments in biocomputational structural analysis of malware. *Proceedings of the Tenth IEEE International Conference on Natural Computation ICNC 2014* (pp. 605-610). Xiamen: IEEE.

Naidu, V., & Narayanan, A. (2016). A syntactic approach for detecting viral polymorphic malware variants. In *Intelligence and Security Informatics* (pp. 146-165). Switzerland: Springer International Publishing.

Naidu, V., & Narayanan, A. (2016). Needleman-Wunsch and Smith-Waterman Algorithms for Identifying Viral Polymorphic Malware Variants. *Proceedings of the Fourteenth IEEE International Conference on Dependable, Autonomic and Secure Computing DASC 2016* (pp. 326-333). Auckland: IEEE.

Naidu, V., & Narayanan, A. (2016). Using different substitution matrices in a string-matching technique for identifying viral polymorphic malware variants. *IEEE Congress on Evolutionary Computation (CEC)* (pp. 2903 - 2910). Vancouver: IEEE.

Naidu, V., Whalley, J., & Narayanan, A. (2017). Exploring the Effects of Gap-Penalties in Sequence-Alignment Approach to Polymorphic Virus Detection. *Journal of Information Security (JIS)*, 296-327: Scientific Research Publishing Inc.

Naidu, V., Whalley, J., & Narayanan, A. (2018). Generating Rule-Based Signatures for Detecting Polymorphic Variants Using Data Mining and Sequence Alignment Approaches. *Journal of Information Security (JIS)*, 265-298: Scientific Research Publishing Inc.

Narayanan, A. (2013). Society under threat… but not from AI. *AI & SOCIETY*, 87–94: Springer.

Narayanan, A. C. (2013). Bio-inspired data mining: Treating malware signatures as biosequences. *Computing Research Repository (CoRR)*, 1-33: Cornell University Library, Access Date: February 2016.

Narayanan, A., Chen, Y., Pang, S., & Ban, T. (2012). The Effects of Different Representations on Malware Motif Identification. *Proceedings of Eighth International Conference on Computational Intelligence and Security CIS* (pp. 86-90). Guangzhou: IEEE.

Narayanan, A., Chen, Y., Pang, S., & Ban, T. (2013). The Effects of Different Representations on Static Structure Analysis of Computer Malware Signatures. *The Scientific World Journal*, 8: Hindawi Publishing Corporation.

Narudin, F. A., Feizollah, A., Anuar, N. B., & Gani, A. (2016). Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing*, 343–357: Springer.

Nataraj, L., Yegneswaran, V., Porras, P., & Zhang, J. (2011). A comparative assessment of malware classification using binary texture analysis and dynamic analysis. *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence* (pp. 21-30). Chicago: ACM.

Natarajan, S., Paul, N., Sobrier, J., THAMILARASU, K., BAYAR, B., & Sutton, M. A. (2017). *United States Patent No. US9609015B2.*

Newsome, J., Karp, B., & Song, D. (2005). Polygraph: automatically generating signatures for polymorphic worms. *Symposium on Security and Privacy* (pp. 226-241). Oakland, CA: IEEE.

Newsome, J., Karp, B., & Song, D. (2005). Polygraph: Automatically Generating Signatures for Polymorphic Worms. *Proceedings of IEEE Symposium on Security and Privacy* (pp. 226–241). Oakland, California: IEEE.

Nguyen, M. N. (2018). Auto-detection of sophisticated malware using lazy-binding control flow graph and deep learning. *Computers & Security*, 128-155: Elsevier.

Nhuong, N. Y. (2014). Semantic Set Analysis for Malware Detection. *International Conference on Computer Information Systems and Industrial Management* (pp. 688-700). Ho Chi Minh City, Vietnam: Springer.

NOD21. (2004). *NOD21*. Retrieved from NOD21: http://www.nod21.com: NOD21, Access Date: May 2016.

Norton. (2010). *The Stuxnet Worm*. Retrieved from Norton by Symantec: https://us.norton.com/stuxnet: Symantec, Access Date: August 2016.

Notredame, C., Higgins, D. G., & Heringa, J. (2000). T-Coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 205-217, 302(1): ScienceDirect, Access Date: May 2016.

Oberheide, J., Bailey, M., & Jahanian, F. (2009). PolyPack: An Automated Online Packing Service for Optimal Antivirus Evasion. *Proceeding WOOT'09 Proceedings of the 3rd USENIX conference on Offensive technologies* (pp. 9-9). Montreal: USENIX Association.

OpNux. (2015, April 9). *Top 5 Free Open Source Antivirus List*. Retrieved from OpNux – Open Source and Linux: http://opnux.com/top-5-free-open-source-antivirus-list: OpNux, Access Date: March 2016.

Laxmi, V., P, V., & Gaur, M. (2009). Survey on malware detection methods. *Hackers' Workshop on Computer and Internet Security* (pp. 74-79). Kanpur, India: Prabhu Goel Research Centre for Computer & Internet Security.

Paganini, P. (2014, December 24). *Virlock, the first self-reproducing polymorphic Ransomware*. Retrieved from Security Affairs: http://securityaffairs.co/wordpress/31442/malware/virlock-polymorphic-ransomware.html: Security Affairs, Access Date: November 2016.

Panda, M., & Patra, M. R. (2009). Semi-Naïve Bayesian Method for Network Intrusion Detection System. In M. Panda, & M. R. Patra, *Neural Information Processing* (pp. 614-621). Heidelberg: Springer Berlin Heidelberg.

Parkour, M. (2013, March 24). *16,800 clean and 11,960 malicious files for signature testing and research.* Retrieved from Contagio Malware Dump: http://contagiodump.blogspot.co.nz/2013/03/16800-clean-and-11960-malicious-files.html: Contagio Malware Dump, Access Date: November 2017.

Patel, V., Gandhi, K., & Bhatti, D. (2017). A Shared Memory Based Implementation of Needleman-wunsch Algorithm using Skewing Transformation. *International Journal of Advanced Research in Computer Science*, 202-208: Institute of Advanced Scientific Research.

Pearson, W. R. (2013). Selecting the Right Similarity-Scoring Matrix. *Current Protocols in Bioinformatics*, 3.5.1–3.5.9: Wiley Online Library.

Pham, P. D. (2011). Applying GPUs for Smith-Waterman Sequence Alignment Acceleration. *International Journal on Computing*, 174-179: Computing Online.

Phylo.io. (2018). *Phylo.io*. Retrieved from Phylo.io: http://phylo.io/: Phylo.io, Access Date: November 2017.

Prabha, A. P., & Kavitha, P. (2012). Malware Classification through HEX. *International Journal of Computer Applications*, 6-12: Foundation of Computer Science.

Prasad, D., & Jaganathan, S. (2018). Improving the performance of Smith–Waterman sequence algorithm on GPU using shared memory for biological protein sequences. *Cluster Computing*, 1–10: Springer.

Preda, M. D., Christodorescu, M., Jha, S., & Debray, S. (2007). A semantics-based approach to malware detection. *POPL '07 Proceedings of the Thirty-Fourth annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (pp. 377-388). Nice: ACM.

Preda, M., & Maggi, F. (2017). Testing android malware detectors against code obfuscation: a systematization of knowledge and unified methodology. *Journal of Computer Virology and Hacking Techniques*, 209–232: Springer.

Privitera, G., & Ahlgrim-Delzell, L. (2018). *Research Methods for Education.* Thousand Oaks, Ventura County, California: SAGE Publications.

Prlić, A., Domingues, F. S., & Sippl, M. J. (2000, June 5). Structure-derived substitution matrices for alignment of distantly related sequences. *Protein Engineering, Design and Selection*, 545-550, 13(8): Oxford Academic.

Quinlan, J. R. (1979). Discovering rules by induction from large collections of examples. In D. Michie, *Expert Systems in the Micro-Electronic Age* (pp. 168-201). Edinburgh: Edinburgh University Press.

Rad, B. B., Masrom, M., & Ibrahim, S. (2012). Camouflage in Malware: from Encryption to Metamorphism. *International Journal of Computer Science and Network Security*, 74-83: IJCSNS.

Rafique, M., & Caballero, J. (2013). Firma: Malware clustering and network signature generation with mixed network behaviors. *International Symposium on*

Research in Attacks, Intrusions, and Defenses (pp. 144-163). Rodney Bay, St. Lucia: Springer.

Raphel, J., & P., V. (2015). Pruned Feature Space for Metamorphic Malware Detection using Markov Blanket. *International Conference on Contemporary Computing* (pp. 377-382). Noida, India: IEEE.

Rastogi, V., Chen, Y., & Jiang, X. (2014). Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks. *IEEE Transactions on Information Forensics and Security*, 99-108: IEEE.

Rice, P., Longden, I., & Bleasby, A. (2000). EMBOSS: the European Molecular Biology Open Software Suite. *Trends in Genetics*, 276-277: Cell Press.

Rieck, K., Schwenk, G., Limmer, T., Holz, T., & Laskov, P. (2010). Botzilla: Detecting the "Phoning Home" of Malicious Software. *ACM Symposium on Applied Computing* (pp. 1978-1984). Sierre, Switzerland: ACM.

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 120–126 : ACM.

Robiah, Y., Rahayu, S., Zaki, M., Shahrin, S., Faizal, M., & Marliza, R. (2009). A new generic taxonomy on hybrid malware detection technique. *International Journal of Computer Science and Information Security*, 56-60: IJCSIS.

Robinson, O., Dylus, D., & Dessimoz, C. (2016). Phylo.io: Interactive Viewing and Comparison of Large Phylogenetic Trees on the Web. *Molecular Biology and Evolution*, 2163–2166: Oxford Academic.

Rossow, C., & Dietrich, C. (2013). ProVeX: Detecting Botnets with Encrypted Command and Control Channels. *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (pp. 21-40). Berlin, Germany: Springer.

Rothbauer, P. M. (2008). Triangulation. In L. M. Given, *The SAGE Encyclopedia of Qualitative Research Methods* (pp. 892-894). Melbourne, Victoria: Sage Publications.

Rouse, M. (2005, September). *What is blended threat? - Definition from WhatIs.com*. Retrieved from TechTarget: http://searchsecurity.techtarget.com/definition/blended-threat: TechTarget, Access Date: November 2016.

Sabin, T. (2004, April 7). *Comparing Binaries with Graph Isomorphisms*. Retrieved from SecuriTeam: http://www.securiteam.com/securityreviews/5EP0320CKC.html: SecuriTeam, Access Date: October 2016.

Salkind, N. (2010). *Encyclopedia of Research Design.* Thousand Oaks, Ventura County, California: SAGE Publications, Inc.

Salzberg, S. (1991). A Nearest Hyperrectangle Learning Method. *Machine Learning*, 277–309: Springer.

Sandro, A. (2016, June). *Backdoor | Malware Wiki | Fandom powered by Wikia*. Retrieved from The Malware Wiki: http://malware.wikia.com/wiki/Backdoor: The Malware Wiki, Access Date: October 2016.

SANS Institute. (2003). *Viral Polymorphism - viral-polymorphism-105483.* Retrieved from SANS Cyber Defense: https://cyber-defense.sans.org/resources/papers/gsec/viral-polymorphism-105483: SANS Cyber Defense, Access Date: May 2016.

Sathyanarayan, V., Kohli, P., & Bruhadeshwar, B. (2008). Signature generation and detection of malware families. *ACISP '08 Proceedings of the 13th Australasian conference on Information Security and Privacy* (pp. 336–349). Wollongong: Springer Berlin Heidelberg.

Satish, S., & Ramzan, Z. (2013). *United States Patent No. US8401982B1*.

Saxe, J., Harang, R., Wild, C., & Sanders, H. (2018). A Deep Learning Approach to Fast, Format-Agnostic Detection of Malicious Web Content. *Cryptography and Security*, 1-7: Springer.

Scaife, N., Carter, H., Traynor, P., & Butler, K. R. (2016). CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. *International Conference on Distributed Computing Systems* (pp. 303-312). Nara: IEEE.

Scheirer, W., & Chuah, M. C. (2008). Syntax vs. semantics: competing approaches to dynamic network intrusion detection. *International Journal of Security and Networks*, 24-35: Inderscience Enterprises Ltd.

Schiffman, M. (2010, February 22). *A Brief History of Malware Obfuscation: Part 2 of 2*. Retrieved from Cisco Blogs: https://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_2_of_2: Cisco Blogs, Access Date: March 2016.

Schipka, M. (2009). *United States Patent No. US20090013405A1*.

Schultz, M., Eskin, E., Zadoc, E., & Stolfo, S. (2001). Data mining methods for detection of new malicious executables. *Proceedings IEEE Symposium on Security & Privacy* (pp. 38-49). Oakland, California: IEEE.

Schwartz, M. J. (2013, January 22). *Virut Malware Botnet Torpedoed By Security Researchers*. Retrieved from Dark Reading: http://www.darkreading.com/attacks-and-breaches/virut-malware-botnet-torpedoed-by-security-researchers/d/d-id/1108302?: Dark Reading, Access Date: June 2016.

Scotland, J. (2012). Exploring the philosophical underpinnings of research: Relating ontology and epistemology to the methodology and methods of the scientific, interpretive, and critical research paradigms. *English Language Teaching*, 9–16: Oxford Academic.

Sebastián, M., Rivera, R., Kotzias, P., & Caballero, J. (2016). AVclass: A Tool for Massive Malware Labeling. *International Symposium on Research in Attacks, Intrusions, and Defenses* (pp. 230-253). Berlin, Germany: Springer.

Second Part To Hell. (2018, February 14). *Second Part To Hell*. Retrieved from Twitter: https://twitter.com/SPTHvx/status/963923106262904833: Twitter, Access Date: February 2018.

Seshardi, V., Ramzan, Z., Satish, S., & Kalle, C. (2012). *United States Patent No. US8266698B1*.

Shafiq, M., Tabish, S., & Farooq, M. (2008). Embedded malware detection using Markov n-grams. *Proceedings Fifth International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment DIMVA* (pp. 88-107). Paris: Springer Berlin Heidelberg.

Sharma, A., & Sahay, S. K. (2014). Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey. *International Journal of Computer Applications*, 7-11: Foundation of Computer Science.

Sharman, R., Krishna, K., Rao, H., & Upadhyaya, S. (2006). Malware and Antivirus Deployment for Enterprise Security. In M. Warkentin, & R. Vaughn, *Enterprise Information Systems Assurance and Systems Security: Managerial and Technical Issues* (pp. 42-61). Hershey, Pennsylvania: Idea Group Publishing.

Shen, Z.-X., Hsu, C.-W., & Shieh, S. W. (2017). Security Semantics Modeling with Progressive Distillation. *IEEE Transactions on Mobile Computing*, 3196 - 3208: IEEE.

Shim, Y., Kim, T., & Im, E. (2015). A Study on Similarity Calculation Method for API Invocation Sequences. *International Conference on Rough Sets and Knowledge Technology* (pp. 492-501). Tianjin, China: Springer.

Sikora, L., & Zelinka, I. (2017). Swarm Virus, Evolution, Behavior and Networking. In I. Zelinka, & G. Chen, *Evolutionary Algorithms, Swarm Dynamics and Complex Networks* (pp. 213-239). Springer, Berlin: Springer.

Singh, S., Estan, C., Varghese, G., & Savage, S. (2004). Automated worm fingerprinting. *Symposium on Opearting Systems Design & Implementation* (pp. 4-4). San Francisco, CA: USENIX Association Berkeley.

Singh, T., Troia, F., Corrado, V. A., Austin, T., & Stamp, M. (2016). Support vector machines and malware detection. *Journal of Computer Virology and Hacking Techniques*, 203–212: Springer.

Skoudis, E., & Zeltser, L. (2004). *Malware: Fighting Malicious Code.* Upper Saddle River, New Jersey: Prentice Hall Professional.

Smith, C., & Matrawy, A. (2009). Computer Worms: Architectures, Evasion Strategies, and Detection Mechanisms. *Journal of Information Assurance and Security*, 69-83: MIR Labs.

Smith, N., Gutierrez, E., Woodruff, A., & Kapoor, A. (2017). *United States Patent No. US9679140B2*.

Smith, T. F., & Waterman, M. S. (1981). Comparison of biosequences. *Advances in Applied Mathematics*, 482-489: Elsevier.

Smith, T. F., & Waterman, M. S. (1981). Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 195–197: Elsevier.

SOPHOS. (2015). *SOPHOS*. Retrieved from SOPHOS: https://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/W32~CTX-A/detailed-analysis.aspx: SOPHOS, Access Date: February 2016.

Šošić, M., & Šikić, M. (2017). Edlib: a C/C ++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, 1394–1395: Oxford Academic.

SPTH. (2004). *Second Part To Hell's Artworks – VIRUSES*. Retrieved from Second Part To Hell (SPTH): http://spth.virii.lu/Cassandra-testset.rar: SPTH, Access Date: February 2015.

SPTH. (2013). *Second Part To Hell's Artworks*. Retrieved from Second Part To Hell SPTH: http://spth.virii.lu/w32.kitti.rar: SPTH, Access Date: February 2015.

SPTH. (2015). *JS.Cassandra by Second Part To Hell*. Retrieved from Second Part To Hell (SPTH): http://spth.virii.lu/rrlf4/rRlf.13.html: SPTH, Access Date: March 2015.

Spurlock, J. R., Schmugar, C. D., & Howard, F. P. (2011). *United States Patent No. US7917955B1*.

Srakaew, S., Piyanuntcharatsr, W., & Adulkasem, S. (2015). On the Comparison of Malware Detection Methods Using Data Mining . *International Journal of Security and Its Applications*, 293-318: SERSC.

Sridhara, S., & Stamp, M. (2013). Metamorphic worm that carries its own morphing engine. *Journal of Computer Virology and Hacking Techniques*, 49-58: Springer.

Stoesser, G., Baker, W., van den Broek, A., Camon, E., Garcia-Pastor, M., Kanz, C., . . . Vaughan, R. (2002). The EMBL Nucleotide Sequence Database. *Nucleic Acids Research*, 21–26, 28(1): Oxford Academic. Retrieved from EMBL-EBI: http://www.ebi.ac.uk/Tools/sss/ncbiblast/

Storlie, C. A. (2014). Stochastic Identification of Malware with Dynamic Traces. *The Annals of Applied Statistics*, 1-18: Project Euclid.

Sun, B. L. (2017). Malware family classification method based on static feature extraction. *International Conference on Computer and Communications (ICCC)* (pp. 507-513). Chengdu, China: IEEE.

Sun, N., Winkler, P., Chu, C., Jia, H., Geffner, J., Lee, T., . . . Swiderski, F. (2006). *United States Patent No. US11523199.*

Symantec. (1997, September). *Understanding Heuristics: Symantec's Bloodhound Technology*. Retrieved from Symantec: https://www.symantec.com/avcenter/reference/heuristc.pdf: Symantec, Access Date: February 2016.

Symantec. (2006, July 21). *W32.Blaster.Worm Removal Tool*. Retrieved from Symantec:

https://www.symantec.com/security_response/writeup.jsp?docid=2003-081119-5051-99: Symantec, Access Date: May 2015.

Symantec. (2014, April). *Symantec Internet Security Threat Report.* Retrieved from Symantec: http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf: Symantec, Access Date: February 2015.

Symantec Corporation. (2017, December 18). *About Advanced Machine Learning in Endpoint Protection 14*. Retrieved from Symantec: https://support.symantec.com/en_US/article.TECH236704.html: Symantec, Access Date: February 2018.

Szopka, B., & Ingo, H. (2018, February 10). *impress.js*. Retrieved from GitHub Inc.: https://github.com/impress/impress.js: GitHub Inc., Access Date: February 2018.

Szor, P. (2005). Advanced Code Evolution Techniques and Computer Virus Generator Kits. In P. Szor, *The Art of Computer Virus Research and Defense* (pp. 251-294). Boston, Massachusetts: Addison-Wesley Professional.

Szor, P. (2005). *The Art of Computer Virus Research and Defense.* Utah: Addison-Wesley Professional.

Szor, P. (2008). *United States Patent No. US7418729B2.*

Szor, P. (2011). *United States Patent No. US7937764B2.*

Tabish, S., Shafiq, M., & Farooq, M. (2009). Malware detection using statistical analysis of byte-level file content. *Proceedings ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics CSI-KDD* (pp. 23-31). Paris: ACM.

Tang, H., Zhu, B., & Ren, K. (2009). A new approach to malware detection. In H. Tang, B. Zhu, & K. Ren, *Advances in Information Security and Assurance* (pp. 229-238). Berlin: Springer Berlin Heidelberg.

Teblyashkin, I. A., Peternev, V. N., & Gryaznov, D. O. (2007). *United States Patent No. US7234167B2.*

The American Heritage Dictionary. (2018). *American Heritage Dictionary Entry: transcription*. Retrieved from The American Heritage® Dictionary of the English Language: https://ahdictionary.com/word/search.html?q=transcription: The American Heritage® Dictionary of the English Language, Access Date: March 2018.

The jQuery Foundation. (2018, February 10). *jquery*. Retrieved from GitHub Inc.: https://github.com/jquery/jquery: GitHub Inc., Access Date: February 2018.

Thompson, G. R., & Flynn, L. A. (2007). Polymorphic malware detection and identification via con-text-free grammar homomorphism. *Bell Labs Technical Journal - Information Technology/Network Security*, 139-147: Nokia Bell Labs.

Transcriptase. (2013, June 8). *Second Part To Hell's Artworks*. Retrieved from SPTH: http://spth.virii.lu/Transcriptase.rar: SPTH, Access Date: February 2017.

Troia, F., Visaggio, C. A., Austin, T., & Stamp, M. (2016). Advanced transcriptase for JavaScript malware. *International Conference on Malicious and Unwanted Software (MALWARE)* (pp. 121-128). Fajardo, Puerto Rico: IEEE.

Troy, C., MacHugh, D., Bailey, J., Magee, D., Loftus, R., Cunningham, P., . . . Bradley, D. (2003). Principles and Methods of Sequence Analysis. In E. Koonin, & M. Galperin, *Sequence - Evolution - Function: Computational Approaches in Comparative Genomics* (pp. 111-192). Boston: Kluwer Academic.

Tucci, L., O'Brien, K., Blott, M., & Santambrogio, M. (2017). Architectural optimizations for high performance and energy efficient Smith-Waterman implementation on FPGAs using OpenCL. *Design, Automation & Test in Europe Conference & Exhibition* (pp. 716-721). Lausanne, Switzerland: IEEE.

Upchurch, J., & Zhou, X. (2013). First Byte: Force-Based Clustering of Filtered Block N-Grams to Detect Code Reuse in Malicious Software. *Conference on Malicious and Unwanted Software: "The Americas"* (pp. 68-76). Fajardo, PR: IEEE.

Uto, N. (2013). A Methodology for Retrieving Information from Malware Encrypted Output Files: Brazilian Case Studies. *Future Internet*, 140-167: MDPI.

VATAMANU, C., COSOVAN, D., Gavrilut, D., & LUCHIAN, H. (2016). *United States Patent No. US20160335432A1*.

Veeramachaneni, K., & Arnaldo, I. (2016). *From Machine Learning to Artificial Intelligence*. Retrieved from McAfee: https://www.mcafee.com/us/solutions/machine-learning.aspx: McAfee, Access Date: November 2016.

Vijan, S., & Mehra, R. (2011). Biological Sequence Alignment for Bioinformatics Applications Using MATLAB. *International Journal of Computer Science and Emerging Technologies*, 310-315: IJCSET.

VirtualBox. (2018). *Oracle VM VirtualBox*. Retrieved from VirtualBox: https://www.virtualbox.org/: VirtualBox, Access Date: August 2017.

VirusTotal. (2018). *VirusTotal - Free Online Virus, Malware and URL Scanner*. Retrieved from VirusTotal: https://www.virustotal.com/: VirusTotal, Access Date: October 2017.

Vu, T. N., Nguyen, T. T., Trung, H. P., Duy, T. D., Van, K. H., & Le, T. D. (2017). Metamorphic Malware Detection by PE Analysis with the Longest Common Sequence. *International Conference on Future Data and Security Engineering* (pp. 262-272). Ho Chi Minh City, Vietnam: Springer.

VX Heavens. (2006, July). *Ready Rangers Liberation Front*. Retrieved from VX Heaven: http://vxheaven.org/vx.php?id=zr06&lang=en&fid=1225: VX Heaven, Access Date: February 2016.

VX Heavens. (2009). *Virus collection (VX heaven)*. Retrieved from VX Heaven: http://vxheaven.org/vl.php?dir=Virus.Win32.CTX: VX Heaven, Access Date: March 2015.

VX Heavens. (2016). *VX Heavens Library*. Retrieved from VX Heaven: http://vxheaven.org/: VX Heaven, Access Date: April 2015.

Wagner, M., Rind, A., Thür, N., & Aigner, W. (2017). A knowledge-assisted visual malware analysis system: Design, validation, and reflection of KAMAS. *Computers & Security*, 1-15: Elsevier.

Wang, C. D. (2017). A Malware Detection Method Based on Sandbox, Binary Instrumentation and Multidimensional Feature Extraction. *International Conference on Broadband and Wireless Computing, Communication and Applications* (pp. 427-438). Palau Macaya, Barcelona: Springer.

Wang, K., Cretu, G., & Stolfo, S. (2005). Anomalous Payload-Based Worm Detection and Signature Generation. *International Workshop on Recent Advances in Intrusion Detection* (pp. 227-246). Seattle, WA: Springer.

Waterhouse, A., Procter, J., Martin, D., Clamp, M., & Barton, G. (2009). Jalview Version 2--a multiple sequence alignment editor and analysis workbench. *Bioinformatics*, 1189-1191: Oxford Academic.

Waterman, M., & Eggert, M. (1987). A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. *Journal of Molecular Biology*, 723-728: Elsevier.

Waterman, M., Smith, T., & Beyer, W. (1976). Some biological sequence metrics. *Advances in Mathematics*, 367-387: Elsevier.

WayBackMachine. (2010). *Virus.Win32.CTX.10853*. Retrieved from WayBackMachine (Internet Archive): https://web.archive.org/web/20100519150727/http://www.securelist.com/en/descriptions/old20455: Kaspersky Lab ZAO, Access Date: June 2015.

Webroot Inc. (2013). *Webroot Security Intelligence.* Retrieved from Webroot: https://www.webroot.com/shared/pdf/Network_Security_Services_FINAL.pdf: Webroot, Access Date: December 2017.

WEF. (2012). *Global Risks 2012: Insight Report.* Retrieved from World Economic Forum: http://reports.weforum.org/global-risks-2012/: World Economic Forum, Access Date: June 2016.

Wespi, A., Dacier, M., & Debar, H. (1999). An Intrusion-Detection System Based on the Teiresias Pattern-Discovery Algorithm. *EICAR Proceedings* (pp. 1-15). Aalborg, Denmark: EICAR.

Wettschereck, D., & Dietterich, T. G. (1995). An Experimental Comparison of the Nearest-Neighbor and Nearest-Hyperrectangle Algorithms. *Machine Learning*, 5–27: Springer.

Whigham, N. (2016, July 11). *Stuxnet virus: 'First atomic bomb of cyber warfare' already dropped*. Retrieved from news.com.au: http://www.news.com.au/technology/online/security/alex-gibney-film-gives-chilling-insight-into-the-world-of-state-sponsored-cyber-warfare-unleashed-by-

stuxnet/news-story/a7063ae03dcb5cd6ed2a576d6a8ea9dc: News Limited Copyright, Access Date: October 2016.

Witten, I. (2014, May 12). *More Data Mining with Weka. Class 3 – Lesson 1, Decision trees and rules.* Retrieved from Department of Computer Science, University of Waikato: http://www.cs.waikato.ac.nz/ml/weka/mooc/moredataminingwithweka/slides/Class3-MoreDataMiningWithWeka-2014.pdf: University of Waikato, Access Date: September 2016.

Wüchner, T., Ochoa, M., & Pretschner, A. (2015). Robust and Effective Malware Detection Through Quantitative Data Flow Graph Metrics. *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (pp. 98-118). Milano, Italy: Springer.

Wurzinger, P., Bilge, L., Holz, T., Goebel, J., Kruegel, C., & Kirda, E. (2009). Automatically Generating Models for Botnet Detection. *European Symposium on Research in Computer Security* (pp. 232-249). Saint-Malo, France: Springer.

Xia, X. (2007). *Bioinformatics and the Cell: Modern Computational Approaches in Genomics, Proteomics and Transcriptomics.* Berlin, Germany: Springer.

Xie, Y., Yu, F., Achan, K., Panigrah, R., Hulten, G., & Osipkov, I. (2008). Spamming botnets: signatures and characteristics. *ACM SIGCOMM Computer Communication Review*, 171-182: ACM.

Xinguang, T., Miyi, D., Chunlai, S., & Xin, L. (2009). Detecting network intrusions by data mining and variable-length sequence pattern matching. *Journal of Systems Engineering and Electronics*, 405–411: Beijing Institute of Aerospace Information (BIAI).

Xu, B., Li, C., Zhuang, H., Wang, J., Wang, Q., & Zhou, X. (2017). Efficient Distributed Smith-Waterman Algorithm Based on Apache Spark. *International Conference on Cloud Computing* (pp. 608-615). Honolulu, CA: IEEE.

Xu, D., & Zhang, Y. (2012). Ab initio protein structure assembly using continuous structure fragments and optimized knowledge-based force field. *Proteins*, 1715-1735: John Wiley & Sons.

Yan, J., Qi, Y., & Rao, Q. (2018). Detecting Malware with an Ensemble Method Based on Deep Neural Network. *Security and Communication Networks*, 1-17: Hindawi Publishing Corporation.

Yanow, D., & Schwartz-Shea, P. (2015). *Interpretation and Method: Empirical Research Methods and the Interpretive Turn.* Abingdon, United Kingdom: Routledge.

Ye, Y., Li, T., Adjeroh, D., & Iyengar, S. (2017). A Survey on Malware Detection Using Data Mining Techniques. *ACM Computing Surveys*, 1-40: ACM.

Ye, Y., Li, T., Jiang, Q., & Wang, Y. (2010). CIMDS: Adapting postprocessing techniques of associative classification for malware detection. *IEEE*

Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 298-307: IEEE.

Yegneswaran, V., Giffin, J., Barford, P., & Jha, S. (2005). An architecture for generating semantic aware signatures. *USENIX Security Symposium* (pp. 34-43). Baltimore, MD: USENIX Association Berkeley.

Yu, Y.-K., Bundschuh, R., & Hwa, T. (2002). Statistical significance and extremal ensemble of gapped local hybrid alignment. In M. Lässig, & A. Valleriani, *Biological Evolution and Statistical Physics* (pp. 3-21). Berlin, Germany: Springer-Verlag Berlin Heidelberg.

Zaharie, D., Perian, L., & Negru, V. (2011). A View Inside The Classification With Non-nested Generalized Exemplars. *IADIS European Conference Data Mining 2011* (pp. 19-26). Rome: IADIS.

Zahid, S., Hasan, L., Khan, A., & Ullah, S. (2015). A Novel Structure of the Smith-Waterman Algorithm for Efficient Sequence Alignment. *International Conference on Digital Information, Networking, and Wireless Communications* (pp. 6-9). Moscow, Russia: IEEE.

Zahn, D. (2014). *United States Patent No. US8775333B1*.

Zarka, R., Cordier, A., Egyed-Zsigmond, E., Lamontagne, L., & Mille, A. (2013). Similarity Measures to Compare Episodes in Modeled Traces. *International Conference on Case-Based Reasoning* (pp. 358-372). Saratoga Springs, NY: Springer.

Zeltser, L. (2011, October). *How antivirus software works: Virus detection techniques*. Retrieved from TechTarget: http://searchsecurity.techtarget.com/tip/How-antivirus-software-works-Virus-detection-techniques: TechTarget, Access Date: August 2016.

Zhang, Q. (2009, May 16). *Polymorphic and Metamorphic Malware Detection.* Retrieved from NCSU LIBRARIES: https://repository.lib.ncsu.edu/handle/1840.16/5484: NCSU LIBRARIES, Access Date: September 2016.

Zhang, Q., & Reeves, D. (2007). MetaAware: Identifying metamorphic malware. *Computer Security Applications Conference* (pp. 411-420). Florida: IEEE.

Zhang, Q., Reeves, D., Ning, P., & Iyer, S. (2007). Analyzing network traffic to detect self-decrypting exploit code. *ACM symposium on Information, computer and communications security* (pp. 4-12). Singapore: ACM.

Zhao, Y., Tang, Y., Wang, Y., & Chen, S. (2013). Generating malware signature using transcoding from sequential data to amino acid sequence. *International Conference on High Performance Computing and Simulation* (pp. 266-272). Helsinki, Finland: IEEE.

Zhu, H.-J., You, Z.-H., Zhu, Z.-X., Shi, W.-L., Chen, X., & Cheng, L. (2018). DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing*, 638-646: Elsevier.

Zimmer, V., Spurlock, J., Venugopal, R., Smith, N., Muttik, I., & Poornachandran, R. (2016). *United States Patent No. US15283238.*

Zuo, Z., & Zhou, M. (2004). Some further theoretical results about computer viruses. *The Computer Journal*, 627–633: Oxford Academic.

Appendix A

A brief review of the other masking strategies employed by viruses is discussed in this section.

A.1 No Masking

No masking at all is one masking strategy which is exceptionally easy to apply in a computer virus. Although it is obvious, however, that it is inadequate – once the existence of a virus is established, it is meaningless to identify and inspect (Aycock, 2006).

A.2 Stealth

A stealth virus is a virus that not only conceals the virus body but also effectively takes actions to hide the infection itself. Moreover, the stealth virus attempts to mask from everything, not just from the AVSs. Some instances of stealth procedures are given below (Aycock, 2006):

1. A file has been modified and to make it look like it was not modified recently, an infected file's authentic time logs can be renewed after infection (Aycock, 2006).

2. The virus can store all pre-compromised information about a file, such as the time log, size of a file, and the contents of a file. Then, calls from I/O system can be interrupted, and the virus reverts the authentic information in return to any I/O procedures on the compromised file, assembling it as the file is uninfected. This approach can be applied to boot block I/O operations as well (Aycock, 2006).

3. Some machines store the secondary boot loader as continuous disk blocks, to be easier on primary boot loader's function. On these machines, there are two viewpoints of the secondary boot loader, one as a string of blocks, and second as a file in the file directory. A virus can penetrate itself into the blocks of the secondary boot loader, thereby transferring the primary blocks somewhere else in the file directory. The outcome is that of a normal routine, the file directory view shows no evident modification, thus the virus is masked. The virus is executed because of the original boot loader (Aycock, 2006).

One alternative is a reverse stealth virus, which shows that everything is infected – the destruction is conducted by the AVS uncontrollably (and mistakenly) attempting to

disinfect (Aycock, 2006). The best example of a stealth virus is the 'Regin' malware which is not only complex but also hard to detect and secure against (Musil, 2014).

A.3 Encryption

In an encrypted virus, the virus body structure is encrypted making it difficult to detect. The virus body usually contains a payload, infection, and a trigger. When the virus body is in the stage of encryption, it is not executable unless decrypted. First, something is executed in the virus then a decryptor loop decodes the virus body and transmits authority to the virus body. The basic goal of this kind of virus is to have a decryptor loop that is smaller than the virus body and therefore has a lower profile making it harder for an AVS to identify. This method of 'encryption' is not the same approach as what cryptographers consider to be encryption; virus encoding is better understood as an obfuscation. A decoding loop can decrypt the virus structure in one location, or decrypt to a different location; this decision may be determined by an external factor, such as the ability of virus writer in the compromised file's source code. This kind of approach is called in-place decryption (Aycock, 2006). Pseudocode for an encoded virus is shown below (Aycock, 2006).

Before Decryption

```
for i in 0
length(body) :
   decrypt code;
goto decrypted_body
```

```
???
```

After Decryption

```
for i in 0 length(body) :
   decrypt code;
goto decrypted_body
```

```
decrypted_body :
   infect ()
if trigger () is true :
      payload ()
```

Virus encryption can be performed in five different ways:

1. **Simple encryption:** No kind of key is utilised for the process of simple encryption, just general no parameter operations, such as decrementing and incrementing, logical NOT, bitwise operators, and negation based on arithmetic operators (Aycock, 2006).

2. **Static encryption key:** A static, fixed key is utilised for the process of encryption and which remains the same from one infection to another. Arithmetic operators, such as

logical operators, that are, XOR and addition operators are used for the process of generating a static encryption key (Aycock, 2006).

3. **Variable encryption key:** In this process of encryption technique, the key during the beginning of the process remains fixed, but modifies as the decryption takes place (Aycock, 2006).

4. **Substitution cipher:** A more basic encryption process could use lookup tables which plot byte value among the encoded and decoded code. The substitution cipher can utilise 1:1 plotting or 1:n plotting depending on the complexity of the virus body. A homophonic substitution cipher permits 1:n plotting, thereby increasing the level of complexity by allowing several encoded values to relate to one decoded value (Aycock, 2006).

5. **Strong encryption:** There is no excuse why any *viruses* cannot utilise the process of strong encryption. Earlier, the size of the source code was a major issue, as the virus had to store several decryption codes in it, but this is not an issue anymore: most set-ups now include libraries of strong encryption keys which can be utilised by several viruses (Aycock, 2006).

The major drawback with the above-discussed encryption processes is that the encoded virus body remains unchanged from one infection to another. That is the encrypted virus body remains constant, and this constancy can easily be used by the AVSs to detect the virus, as there is no hidden strategy whatsoever. But with the help of randomly generated keys, this issue can be avoided. Every time a virus code is decrypted, a random key is generated by the decryption routine to encrypt the virus code again, after the infection. This method of randomly generating keys can be implemented to any one of the encryption techniques explained above (Aycock, 2006).

'CryptoLocker' is malware, specifically a ransomware Trojan that mainly attacks Microsoft Windows-based computer systems.

A.4 Strong Encryption

Usually, a virus stores their decryption keys in them, and this might be a weakness, which can be easily detected and analysed by the AVSs. This is one way of storing the decryption keys for the process of encryption. But there are two other approaches that can be used for the process of encryption (Aycock, 2006):

1. **The decryption key comes from the external source of an infected machine:**

a. A virus can obtain the decryption key from a website, but that implies that the virus would have to transport the address of that website at all times, which could be blacklisted as a precautionary measure. This issue can be avoided if the virus can obtain the decryption key utilising a website search engine. Normally, any electronic information torrent that a virus can control would be employable for the generation of the decryption key, particularly the one with large amounts of traffic that are unexpected to be blacklisted: such as instant chat messaging, file-distributing networks, email messages, IRC, etc. (Aycock, 2006).

b. A binary virus is a virus that comes in two components and does not get destructive unless both the components are located on the same machine. Very few known binary viruses came into existence such as 'RMNS' (spreads via p2p – peer to peer services, IRC, file distributing networks, internet file downloads, etc.) and 'Dichotomy'. One demonstration of binary viruses would be where V1 and V2 are two components of a binary virus. V1 virus would have a strongly encoded source code, and V2 virus would have the decryption key. Both V1 and V2 viruses need to work together and should be in the same location (at the time of decryption) for the binary virus process to execute successfully. If they transmit together, then both will exhibit the same chance of getting detected and inspected, overcoming the main goal of detaching the decryption key. And if V1 and V2 viruses transmit separately then their propagation would be independent (Aycock, 2006).

2. **The decryption key comes from internal source of an infected machine:** With the help of environmental key creation, the decrypted key is developed based on the factors situated in the target's surrounding, such as, domain computer name; system date or time; some information in the machine (example, important document file name, file contents, etc.); the recent computer user name; operating system's language setting; IP address.

This makes it very simple to target viruses to specific groups or individuals. A target has no idea that they hold the decryption key (Aycock, 2006). 'CryptoLocker', a ransomware Trojan malware is the best example that uses the techniques of strong encryption (Scaife, et al., 2016).

A.5 Oligomorphism

Suspecting the key of an encoded virus is randomly modified with every new infection, the only untouched area of the virus is the code inside the decrypted routine. AVS will utilise this aspect for identification, so the next ideal improvement is to modify the code inside the decryptor routine with every infection (Aycock, 2006).

An oligomorphic virus is an encoded virus which has a tiny, fixed amount of several decryptor routines at its disposition. This virus can also be known as, semi-polymorphic virus. For every new infection, the virus chooses a new decryptor routine from this small finite amount. For instance, 'Whale' had 30 variants each with dissimilar decryptor routines, and 'Memorial' had 96 variants each with different decryptor routines (Aycock, 2006).

With regards to identification, viruses based on oligomorphism are marginally difficult to detect. AVS will not simply look for just one decryptor routine but instead will look for all routines, by simply having all the possible decryptor routines of a virus listed in their database (Aycock, 2006).

Appendix B

A brief review of the obfuscation techniques employed by polymorphic viruses is discussed in this section. This review of the polymorphic obfuscation techniques is divided into three sections. Section B.1 covers 'Polymorphic obfuscation based on self-identification'; Section B.2 details 'Polymorphic obfuscation based on Syntactic reconstruction', and Section B.3 discusses 'Polymorphic obfuscation based on Semantic reconstruction'.

B.1 Polymorphic Obfuscation based on Self-Identification

There are two challenging questions that emerge with regards to polymorphic malware. Firstly, how can a malware identify that it has formerly compromised a file if its existence is concealed adequately well? Secondly, how does a malware modify its decryptor routine from one infection to other (Aycock, 2006)? The first challenging question will be covered in this section and the second challenging question will be covered in Section B.2.

1. **Self-Identification:** At an initial glimpse, it might seem simple for a polymorphic malware to identify if it has formerly compromised some code – when the malware mutates for a new file infection, it can also modify any form of itself that it encounters. This will not work, nonetheless, since a malware must be able to identify infection by either of its virtually unlimited aspects. This shows that the infection identification approach must be self-reliant of the correct code utilised by the malware (Aycock, 2006).

2. **File Time Logs:** A malware could modify the time logs of a compromised file, as long as for every infection, the combination of its date and time is a certain value *K* which is constant. Many applications only display the last two units of the year, so the year of a compromised file could be incremented by 50 without seeking attention (Aycock, 2006).

3. **File Size:** The size of a compromised file could be modified out to a certain significant size, like a multiple of 4848 (Aycock, 2006).

4. **Data Concealment:** In file formats, such as ELF, which is a complicated executable, not all sections of the file's contents may be utilised by a computer system. Malware can conceal a flag in new sections, or look for an unused association of features that

it has placed in the file. For instance, on Windows in a file header which is executable, the malware ‘Zperm’ observes for the alphabet ‘Z’ as the small linker variant (Aycock, 2006).

5. **File System Attributes:** Some file systems permit files to be labelled with random features, whose presence is not in every instance made recognisable. Malware can utilise these to stock source code, flags, or information which suggests that a file has been compromised. Figure B.1 (shown below) displays the ‘surrogate data streams’ being utilised in a file system based on NTFS to append a flag to a program file; the existence of this flag will not come up in directory index, in the file system browser, or in the file size (Aycock, 2006).

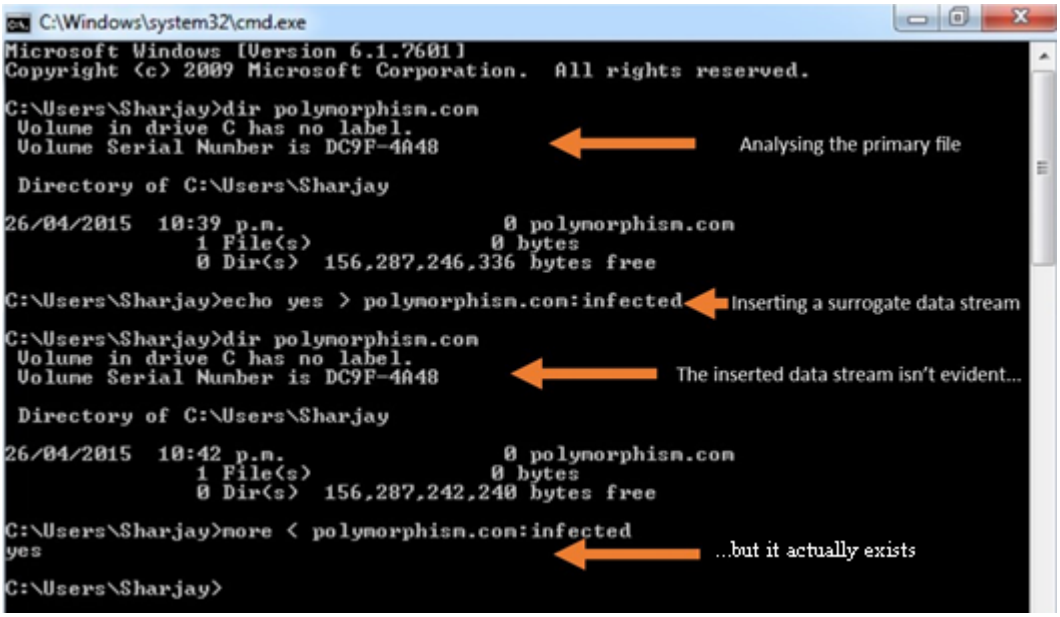

Figure B.1: The concept of NTFS file system based surrogate data streams (Aycock, 2006, p.39).

6. **External Storage:** The evidence that a file is compromised need not be exactly related to the file itself. For instance, malware could employ a hash function to plot the name of a compromised file into a hidden string, and under the Windows registry employ that string to generate a key. Malware can then utilise the presence of that key as an infection signal. However, if the registry key was determined it will not be sufficient to successfully disclose the name of the compromised file (Aycock, 2006).

As all these infection-identification approaches function for polymorphic malware, they also function for the specific types of non-polymorphic malware as well, such as worm, rabbit. It was once indicated that computer systems could be protected against particular malware by bypassing the self-identification signal for malicious activity on an

uncompromised system. Sadly, there are lots of malware now to make this possible (Aycock, 2006; Zhang, 2009; Kumar, 2016).

B.2 Polymorphic Obfuscation based on Syntactic Reconstruction

This is the technique used by many polymorphic viruses to modify its syntactic structure. Although, the polymorphic changes its syntactic structure, the semantics of those polymorphic viruses remains unchanged. This approach is used by the polymorphic viruses to bypass identification and classification based on byte level. Byte-level identification is employed by many AVSs (Zeltser, 2011). Many of this approaches used by polymorphic viruses are adopted from the area of file obfuscation (Cesare, 2010).

Polymorphism is explained as having a similar definition to that of metamorphism, in cases where it is used to illustrate the automated syntactic modification of viral instructions and viral code. Under such phraseology, polymorphism is used to explain syntactic changes of fixed sections of the viral instruction data. The remaining viral code sections are encrypted at the byte stage without modifying the semantics or the syntax of the instructions. In this section, polymorphism and metamorphism are considered to be similar to one another (Cesare, 2010).

1. **Dead Code Insertion:** Semantically, dead code insertion is similar to a null operation. Dead code is also called a semantic NOP (i.e. No OPeration) or as junk code. Insertion of this kind of junk code has no semantic effect on the malware. However, the size of the malware increases when dead junk code is inserted. Also, dead code insertion changes the instructions and byte level data of the malware. For instance, '`push %ebx`' and '`pop %ebx`', best represent the idea behind dead code insertion (Cesare, 2010).

2. **Instruction Replacement:** The process of instruction replacement exchanges a particular set of instructions or an individual instruction with semantically identical, but altering instruction sets or instructions. The size of the malware using this kind of approach may increase or decrease, based on some instructions substituted. For example, '`mov $0, %eax`' is substituted to '`xor %eax, %eax`' (Cesare, 2010).

3. **Variable Renaming:** The process of variable renaming and the corresponding approach of register displacement modifies the purpose of registers and variables in a piece of code. It is done in such a way that the process of variable renaming uses

distinctive registers and variables keeping the instructions semantically identical when compared with the original piece of code. For instance, '`mov $0, %eax`' gets renamed to '`mov $0, %ebx`'; '`mov $1, %ebx`' gets renamed to '`mov $1, %ecx`'; '`add %eax, %ebx`' gets renamed to '`add % ebx, %ecx`'; '`push %ebx`' gets renamed to '`push %ecx`' (Cesare, 2010).

4. **Code Rearranging:** The process of code rearranging modifies the syntactic rearrangement of a malware code. This process does not modify the genuine or semantic implementation location of the file. Nevertheless, the syntactic arrangement as seen in the original malware code is modified. The process of code rearranging comprises the approaches of branch inverting, branch obfuscation, the use of opaque base insertion, and branch transposition (Cesare, 2010).

5. **Branch Obfuscation:** The process of branch obfuscation tries to conceal the object of a branch instruction. For instance, the employment of SEH i.e. Structured Exception Handling on the Microsoft Windows based operating system, best explains the process of branch obscure. In a malware, the employment of SEH to hide control flow is very common. This exact technique of SEH incorporates an indirect branch. The process in indirect branching utilises the content of data as the object of a branch. This decodes the detection of control flow into a complicated data flow inspection issue. The employment of a branch function expands this technique and executes many branches via an individual routine. The main idea behind the process of branch obfuscation is to make a static inspection of the malware by a malware analyst as well as the procedure of automation more complicated. For instance, '`mov $0x8048200, %eax`' then '`jmp *%eax`', involves an indirect branch function (Cesare, 2010).

6. **Branch Transposition:** The process of branch transposition in conditional branches, transposes the condition of the branch. Considering primarily when the condition is true the branch may move the control, the process of branch transposition modifies the branch condition when false. To keep the primary semantics of the malware file, the process of branch transposition inverts the branch instruction as well. For instance, the statement of a branch condition true would be modified to a false statement. Moreover, the condition in the process would also be transposed. Branch transposition is a kind of instruction replacement based on control flow statements. For example,

the code '`jc $0x80482000`' can be transposed to '`cmc # complement carry flag jnc $0x80482000`' via branch transposition (Cesare, 2010).

7. **Branch Inverting:** Branch inverting is a homogenous approach to branch transposition in which the branch instruction is redrafted by replacing it with semantically similar code with distinct control flow features. For instance, if the primary code has a statement with a branch condition of true then the modified code has a statement with a branch condition of false to the primary failed instruction. The modified failed instruction then thoroughly branches to the primary conditional branch object. For example, the code '`jz $0x80482000`' inverts to '`jnz L jmp $0x80482000 L:`' (Cesare, 2010).

8. **Opaque Base Insertion:** The process of opaque base insertion always calculates to the same outcome. It is challenging for a malware analyst or automated system analysis to recognise the base insertion result because of the complex construction of opaque base insertion. The process of opaque base insertion can be utilised to embed expendable branching in the control flow of malware. They can also be utilised to allocate variable values which are challenging to diagnose statically. The main purpose of opaque base insertion is for code concealment, and also to avoid getting detected by a malware analyst or automated static system evaluation. For instance, '`mov $1, %eax jz $0x80482000`', is a rationalised opaque base insertion (Cesare, 2010).

9. **Code Packing:** Code packing is a powerful approach utilised to conceal malware and obstruct the understanding of a malware analyst towards malware's objective. In a particular month in 2007, around 79% of detected malware from a well-known AVS was known to be packed. Moreover, in 2006, nearly 50% of new malware were found to be repacked variants of existing malware (Cesare, 2010).

   Code packing is not only utilised to obstruct the understanding of malware by a malware analyst but also utilised by malware to avoid getting identified by the AVSs. Polypack (Oberheide, Bailey, & Jahanian, 2009) – an automated online packing utility for bypassing excellent antivirus systems, examined the usefulness of code packing towards antivirus identification by supplying a utility to pack malware, utilising a bunch of tools based on code packing. AVSs often have the potential of revealing tools with known packing code, but also had a commercial concern revealing tools

with unknown packing code. Nonetheless, polypack showed that packing could be a powerful tool to bypass an AVS with several commercial malware identification packages, but lacking to detect the packed variants of existing malware (Cesare, 2010).

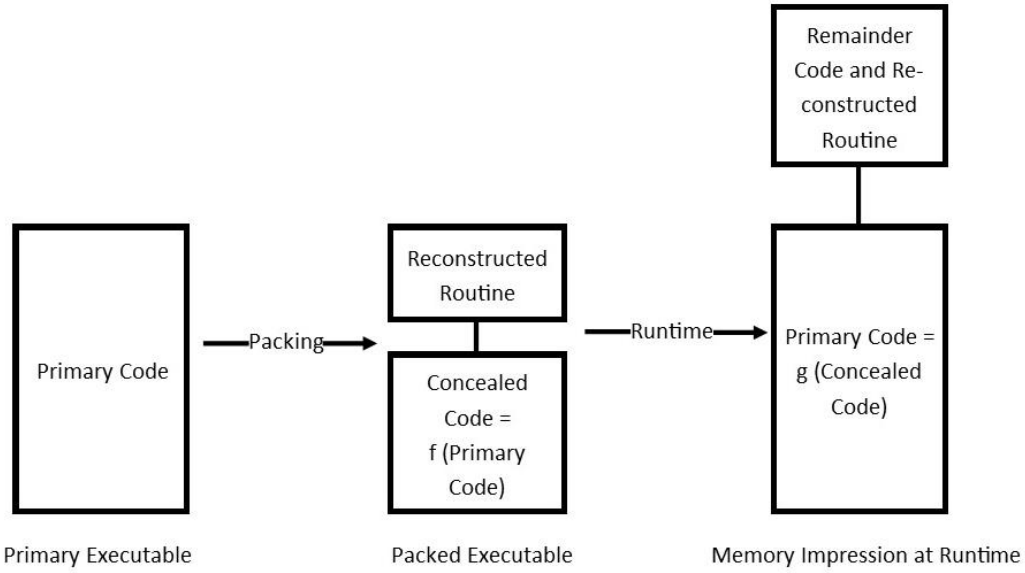

Figure B.2: The traditional code packing modification (Cesare, 2010, p.33).

It is a known fact that code packing is utilised in many malware, but code packing can also assist in supplying software security and restrictions for the intellectual property incorporated in a program. It is not certainly preferable to blacklist all instances of code packing as being indicators of malicious activity. Code packing software products are an open source which is available freely and commercially sold to everyone as a genuine tool. In consequence, revealing packed programs supplies an advantage. It is preferable to confirm if the packed data are malicious, instead of detecting only the case where unknown data are packed (Cesare, 2010). Figure B.2 shows the traditional code packing modification.

10. **Instruction Correspondence:** Particularly on CISC (Complex Instruction Set Computing) CPU design such as the Intelx86, there are often several individual instructions which have the similar outcome. For all these instruction the register would be set from r1 to zero (Aycock, 2006):

```
clear r1

xor r1,r1

and 0, r1

move 0, r1
```

11. **Instruction Series Correspondence:** Instruction correspondence can be postulated to series of instructions. While individual instruction correspondence is at the leniency of the CPU's instruction sequence, instruction series correspondence is more convenient, and can be implemented in both low-level and high-level languages (Aycock, 2006):

```
x = 1   ⇔   y = 22

    x = y – 15
```

12. **Register Renaming:** A slight, but considerable, the modification can be initiated just by modifying the registers that instructions utilise. While this creates no deviation from a high-level aspect, like a human interpreting the register code, renaming modifies the bit formats that encrypts the instructions; this makes the job harder for the AVSs focusing on the malware instructions. For instance (Aycock, 2006):

```
r1 = 18          r3 = 18

r2 = 54           ⇔      r1 = 54

r3 = r1 + r2       r2 = r3 + r1
```

The idea of register renaming normally broadens to variable renaming (discussed above) in complex-level languages like a macro malware might implement (Aycock, 2006).

13. **Runtime Code Formation:** One method to change the code is to conceal some code until execution. Either new code can be created, or existing code can be altered (Aycock, 2006).

```
r1 = 18       r1 = 18

r2 = 54    => r2 = 54

r3 = r1 + r2    create r3 = r1 + r2

          call created_code
```

14. **Consistency:** The primary code can be divided into numerous strings of execution, which not only modifies the code, but also can considerably obscure automatic inspection. For example (Aycock, 2006):

```
r1 = 18       begin string S
r2 = 54    => r1 = 18
r3 = r1 + r2    wait for signal
          r3 = r1 + r2
               ...
          S:
             r2 = 54
             send signal
             exit string S
```

15. **Inlining and Outlining:** Code inlining is an approach generally used to evade subroutine call overhead, that substitutes a subroutine call with the code of the subroutine. For instance (Aycock, 2006):

```
    ...                ...
call S1          r1 = 18
call S2          r2 = r3 + r2
    ...         =>     r4 = r1 + r2
S1:
r1 = 18       r1 = 18
 r2 = r3 + r2     r2 = 54
 r4 = r1 + r2     r3 = r1 + r2
 return         ...
S2:
 r1 = 18
 r2 = 54
 r3 = r1 + r2
 return
```

Code outlining is the inverse approach, it does not need to maintain any logical code arrangement, nevertheless (Aycock, 2006):

```
   ...          ...
r1 = 18      r1 = 18
 r2 = r3 + r2     r2 = r3 + r2
 r4 = r1 + r2     call S12
=>  r3 = r1 + r2
 r1 = 18       ...
 r2 = 54     S12:
 r3 = r1 + r2       r4 = r1 + r2
   ...            r1 = 18
                  r2 = 54
                  return
```

Another way is to transform the code in question into the threaded code, which has no relation to threads utilised for the concept of consistent programming, regardless of the term. Threaded code is utilised as a replacement approach to executing interpreters based on programming language. Subroutines involved in the threaded code do not restore to their location from which they were implemented, but rather instantly jump to the following subroutine; the threaded code by itself is just a range of code addresses. For instance (Aycock, 2006):

```
   ...          ...
r1 = 18      next = &code
 r2 = r3 + r2     goto [next]
 r4 = r1 + r2     CODE:
=>    &I1
 r1 = 18       &I2
 r2 = 54       &X
 r3 = r1 + r2     X:
   ...            r1 = 18
                  r2 = 54
                  r3 = r1 + r2
```

```
    ...
I1:
    R1 = 18
    inc next
    goto [next]
I2:
    r2 = r3 + r2
    r4 = r1 + r2
    inc next
    goto [next]
```

16. **Subroutine Interposing:** Inlining and outlining variations preserve the primary code, but rebundle it in several methods. Code can also be modified by merging self-determining subroutines together, as shown in the instance below (Aycock, 2006):

```
  ...              ...
call S1          call S12
call S2            ...
  ...         => S12:
S1:                r5 = 18
r1 = 18            r1 = 18
    r2 = r3 + r2       r6 = r3 + r2
    r4 = r1 + r2       r2 = 54
return           r4 = r5 + r6
         S2:           r3 = r1 + r2
    r1 = 18        return
    r2 = 54
    r3 = r1 + r2
    return
```

To ignore variance with registers utilised by S2, some registers had to be renamed from the code of S1. The total execution in the interposed subroutine is similar to the primary code with regards to the values calculated (Aycock, 2006).

Several of these code modification techniques discussed here are also utilised in the area of code obfuscation; investigation in the area of code obfuscation is utilised to attempt and obstruct reverse engineering. There are also numerous amount of complex code obfuscations carried out by advanced compilers. Not all compiler approaches and code transformation approaches have yet been utilised by malware writers (Aycock, 2006).

B.3 Polymorphic Obfuscation based on Semantic Reconstruction

Semantic transformation based polymorphic malware are the extended version of the syntactic transformation based polymorphic malware, where the new variation is an obtained creation of the primary malware. Semantic modification of a malware happens due to the malware writers changing the primary source code or the malware functionality. This can happen to a genuine progress of the malware through its life cycle of the program development. Moreover, it can happen when a malware writer recycles the existing malware code to write a new malware variant (Cesare, 2010).

1. **Code Insertion:** The process of code insertion happens when fresh functionality is implemented in the existing malware code (Cesare, 2010).

2. **Code Deletion:** The process of code deletion happens when an existing functionality is eliminated from the malware code (Cesare, 2010).

3. **Code Replacement:** The process of code replacement happens when an existing functionality in the malware code is substituted by a different code or algorithm (Cesare, 2010).

4. **Code Transposition:** The process of code transposition happens when in an existing malware a particular functionality and code is eliminated from its original location and semantically placed into a separate location in the malware code (Cesare, 2010).

Appendix C

C.1 Materials and Tools

This section presents the materials and tools used in this thesis for experimental purposes.

C.1.1 W32.CTX/W32.Cholera Virus

The Win32.Cholera/W32.Cholera/W32.CTX is a polymorphic virus which attacks executable files of the format PE (Portable Executable). This virus is programmed in assembly language, and it employs an EPO (Entry Point Obfuscation) approach, which makes its identification difficult (NOD21, 2004; SOPHOS, 2015; WayBackMachine, 2010). EPO is a method employed by virus writers to prevent AVS scanners from examining the (malicious) files that have been captured (Schiffman, 2010). Win32.Cholera/W32.Cholera/W32.CTX was therefore chosen to fully evaluate and challenge the signatures generated via proposed methods in this research. The original source files were downloaded from 'VX Heavens' (VX Heavens, 2009) website. All the 198 unknown (new) polymorphic variants (P_x) of W32.Cholera virus were generated manually by executing one of the original (P_s) virus files (in this case, a file named 'Virus.Win32.CTX.10853').

C.1.2 JS.Cassandra Virus

Unlike any other JavaScript virus, JS.Cassandra is comprised of four distinct polymorphic engines: polymorphic engine I, which includes Garbage or Junk codes; polymorphic engine II, which modifies its Body (Body Changing); polymorphic engine III, which modifies its Variables (Variable Changing); and polymorphic engine IV, which modifies its Numbers (Number Changing). The likelihood of the virus to decode using the polymorphic engine I is either 1:3 or every $1:4^{th}$ line (inside its viral code) and engine I has a polymorphism level of 3 (see page no. 35). The likelihood of the virus to decode using polymorphic engine II and III are 1:3 and engine II has a polymorphism level of 6, whereas, engine III has a polymorphism level of 2. The likelihood of the virus decoding using polymorphic engine IV is either 1:1 or every $1:10^{th}$ number (found inside its viral code). Engine IV has a polymorphism level of 2 (SPTH, 2015; Belcebu, n.d.). JS.Cassandra virus is selected for this research because unlike the majority of similar malware its source code and the source code of its known (P_k) variants are readily available. The original (P_s) JS.Cassandra virus with its original source code was downloaded from its author's (Second Part to Hell) website (SPTH, 2015). All the 351

known (P_k) polymorphic variants of JS.Cassandra virus were also retrieved from the virus author's website (SPTH, 2004).

C.1.3 W32.Kitti Virus

The W32.Kitti virus (SPTH, 2013) was fully written in assembly language. The mutation engine in W32.Kitti virus alters instruction to overlapped code. This virus worms/sneaks over shared network disks and portable disks. In total, 1105 unknown (P_x) polymorphic malware variants were generated manually and were obtained by executing the original virus (P_s) file (in this case, a filename 'oc.exe').

C.1.4 Transcriptase Virus

In biological sciences, a transcriptase is defined as an enzyme that initiates the creation of ribonucleic acid (RNA) from a template of deoxyribonucleic acid (DNA) during the process of transcription (The American Heritage Dictionary, 2018). The term Transcriptase was adopted by the creator of the metamorphic JavaScript malware in (Transcriptase, 2013) and is a proof-of-concept malware (Ferrie, 2013), which is used in this thesis. The parallelism amidst the biological procedure of transcriptase and such a metamorphic JavaScript generator is somewhat flimsy (Musale, Austin, & Stamp, 2015; Troia, Visaggio, Austin, & Stamp, 2016; Ferrie, 2013).

Once executed inside a folder, Transcriptase infects every JavaScript files within that folder. Every infection emanates in a mutated instance of the malware that is attached to the victim code. The aim of the mutation is to avoid detection through signature-based techniques (Musale, Austin, & Stamp, 2015; Troia, Visaggio, Austin, & Stamp, 2016; Ferrie, 2013).

The generator of Transcriptase adopts a customized meta-language to implement its metamorphosis. The meta-language data is executed adopting an executable program that is executed in JavaScript, and the executable program per se is a component of the malware program. The benefit of acquiring a customized meta-language is to in fact add information, needed to generate extremely mutated instances by the malware writer, simultaneously avoiding the code to expand uncontrollably over time (Musale, Austin, & Stamp, 2015; Troia, Visaggio, Austin, & Stamp, 2016; Ferrie, 2013).

C.1.5 JAligner

'JAligner' (Moustafa, 2010) is an open source Java application that implements the SWA and is used in this thesis for the process of pairwise alignment of biological sequences. 'JAligner' provides the options of gap open and gap extend penalties along with the wide selection of the 71 different substitution matrices. Chapters 5 to 7 will use 'JAligner' and the results of the alignments will be processed for the identification and extraction of common substrings (meta-signatures). The open source master version of 'JAligner' can perform pairwise sequence alignment using the NWA and is employed in Chapter 6 – Part-I. This tool can be retrieved from the following website: http://jaligner.sourceforge.net/.

C.1.6 Weka

Weka (Frank, Hall, & Witten, 2016) is an open source machine learning tool for conducting data mining jobs and consists of tools for clustering, association rules, data preprocessing, visualisation, classification, and regression. Weka is employed in Chapters 4, 6 and 7 for rule extraction and classification. PRISM and NNge classifiers are used in this thesis, which are inbuilt Weka classifiers.

C.1.7 ClamAV and 'sigtool'

ClamAV (ClamAV, 2018; ClamAV, 2016) is an open source antivirus tool that is used in Chapters 4 to 7 for the purpose of testing any signatures or generic signatures generated during this research study against variants in the signature's malware family. The testing is performed with the help of 'clamscan' virus scanner by creating a .ndb database, which is a part of the ClamAV tool. One other part of ClamAV is the 'sigtool' which is used in this thesis for the purpose of extracting hexadecimal dumps of the malicious files. The tools and source code of ClamAV can be retrieved through the following website: https://www.clamav.net/downloads.

C.1.8 VirusTotal

'VirusTotal' (VirusTotal, 2018) is an online tool that scans and examines files for malicious activities. This tool constitutes of 56 well-known AVSs with their up-to-date databases. In this thesis, 'VirusTotal' provides confidence that the manual code alterations for non-malicious (P_u) files are effective and also confirms their uniqueness. This tool creates a unique SHA256 signature for every suspicious file that is uploaded to its online web tool and can be accessed through the following website:

https://www.virustotal.com/en/. In this thesis, 'VirusTotal' (VirusTotal, 2018) is used to check malware authenticity. Furthermore, there are several other freely available online virus scanners, such as 'Metascan Online', 'VirSCAN', 'CA Online Malware Scanner', and 'Gary's Hood' (Hood, 2016). In this thesis, 'Gary's Hood' (Hood, 2016) is used to scan multiple malicious files for experimental detection purposes (Chapter 6).

C.1.9 MAFFT

MAFFT (Katoh & Standley, 2013; Katoh, et al., 2002) is an online tool that performs multiple sequence alignment of larger sequences (big data) and is employed in Chapter 7 for converting variable-length sequence into fixed length sequences for the process of data mining. This tool can be accessed via the following website: http://mafft.cbrc.jp/alignment/software/.

C.1.10 Random Data File Creator (RDFC)

RDFC (Berthold, 2004) is a console application which is used to generate random binary files of any sizes by filling with random binary numbers and is a Microsoft Windows application. This tool is employed in this thesis to generate random files for signature testing purposes for its effectiveness and to check for the false positive and false negative rates. This tool can be downloaded from the following website: http://www.bertel.de/software/rdfc/index-en.html.

Appendix D

D.1 Clamscan Database File

The content inside a typical .ndb clamscan database file for a virus family and its use in the experiments conducted in Chapters 4 to 7 is shown below, for instance:

Virus:0::537472696e672e66726f6d43686172436f646528*

Where, 'Virus' is the virus filename.

'0' is the 'TargetType' (i.e. the type of target file, in this case, it is a JavaScript file) and several options are available within which are, '0' is for any file, '1' is for portable executable file, '2' is OLE2 component i.e. a vb script file, '3' is for normalised HTML, '4' is for mail type files, and '5' is for graphics files. Option '0' was chosen in this case (Naidu & Narayanan, 2016).

'' is the 'Offset' type to tell the scanner about where the signature applies inside the file (similar to {n} wildcard) & three options are available within, which are, '*' is for anywhere inside the file, 'n' is for n bytes from beginning of file, & 'EOF – n' is for End Of File minus the n bytes. Option '*' was chosen in this case* (Naidu & Narayanan, 2016).

'537472696e672e66726f6d43686172436f646528' is the meta-signature (in hexadecimal format) for JS.Cassandra polymorphic viral family obtained from the seven step approach in Chapter 4.

D.2 Clamscan Scan Results for 43 Malicious (P_k), 43 Non-Malicious (P_u) and 43 Random Files

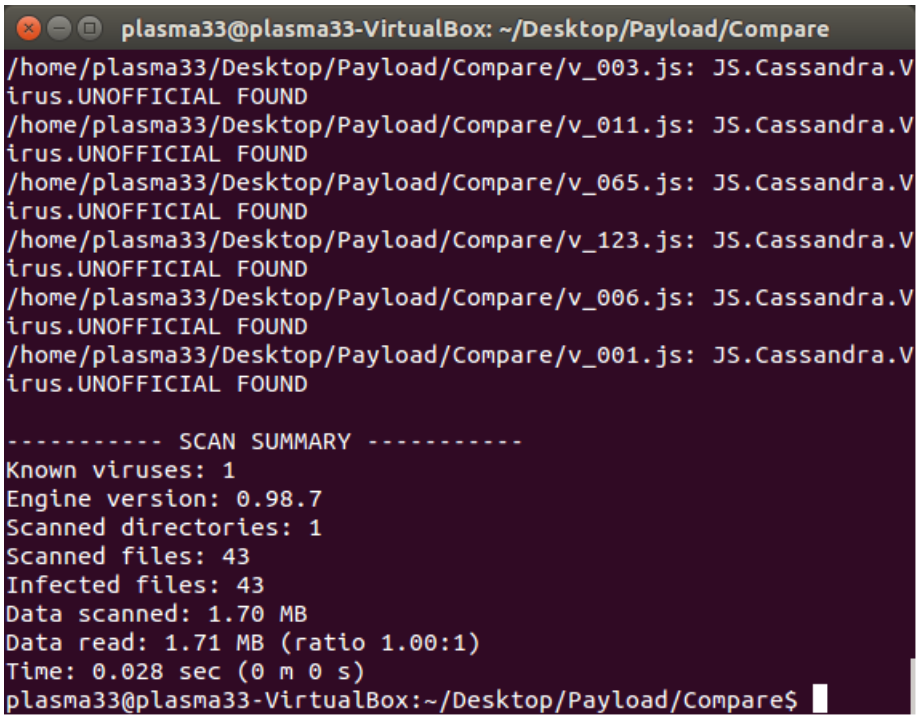

Figure D.1: Screenshot of the scan result obtained from 'clamscan' antivirus scanner for the 43 malicious (P_k) files using the meta-signature.

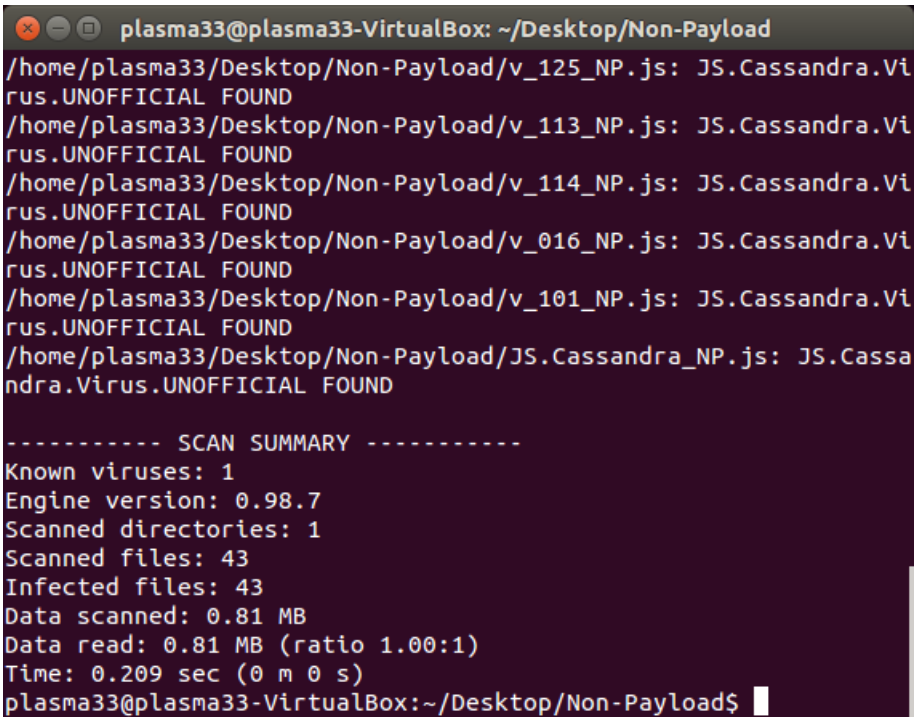

Figure D.2: Screenshot of the scan result obtained from 'clamscan' antivirus scanner for the 43 non-malicious (P_u) files using the meta-signature.

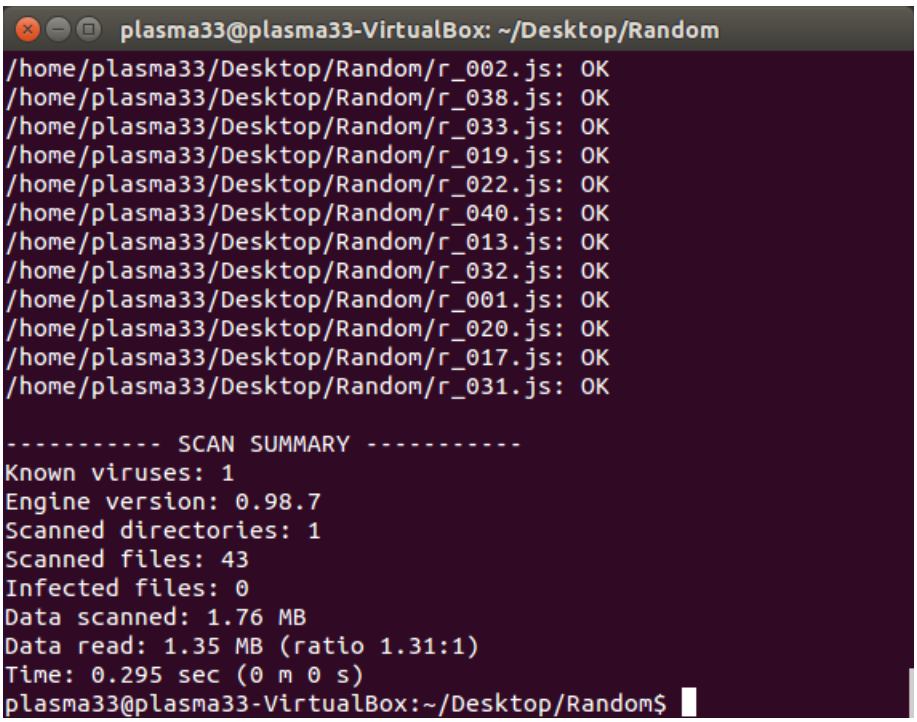

Figure D.3: Screenshot of the scan result obtained from ‘clamscan’ antivirus scanner for the 43 random files using the meta-signature.

Table D.1: Generated CRC32b Hash Value and File Size in Bytes for the 43 Malicious (P_k) Files, 43 Non-Malicious (P_u) Files, and 43 Random Files.

Malicious (P_k) variants		**Non-malicious (P_u) variants**		**Random variants**	
CRC32b Hash Value	**File Size (bytes)**	**CRC32b Hash Value**	**File Size (bytes)**	**CRC32b Hash Value**	**File Size (bytes)**
26489347	7,767	ab657f45	1,823	bfa1e3d9	9,216
848562f1	8,324	3d94f85f	2,662	8956b4ef	4,096
fab48c8c	54,183	634041fe	28	07baad1b	5,120
7c4ea313	9,938	90dd470d	3,697	0ca68128	7,168
bd3b9fdc	8,759	0631e490	2,981	819ec3a5	10,240
9904ef9c	8,392	5273cd32	2,137	048d638c	11,264
511621c7	9,400	32b7909a	3,748	f6425fdb	13,312
a7bc9795	10,059	d1f95eae	2,125	9fda09d6	15,360
a878abc3	10,763	cd486121	2,868	8b2ff426	17,408
ec3797e7	12,282	e2220b79	3,874	e3702e86	20,480
a2e5c540	10,799	d4cffb98	2,965	ae57c29a	19,456
9c8432d2	10,873	91f3e71f	1,688	dbfae5e5	133,120
2b40aa76	8,639	03fe3ba9	1,439	0d4cd9da	52,224
92c87b26	11,334	a6308749	3,021	8a8d3664	35,840
52653b1d	9,507	6eaa1574	1,885	fd11933b	24,576
851c41b7	9,740	e7cb7513	2,872	df99e215	27,648
f006361e	14,900	0b52da18	3,876	fe998b7c	22,528
8ead30b2	9,945	31838f54	2,887	c39cd570	33,792
1ea87480	46,691	505c6653	27	8b461802	45,056
ecd82d26	10,677	db5a61cc	1,840	79b2dfb3	26,624
59e9feb3	139,909	634041fe	28	d3951707	37,888
70763a3b	14,767	c7b5d591	6,461	68cdd062	103,424
f4289eb6	52,595	26fae117	35,637	12307f18	59,392
c4290e04	29,603	687ff9af	15,230	c0cd6499	52,224
3797337e	25,828	8be6dc4c	9,394	6ec3f157	74,752
4736f8b5	45,659	bb0289a8	22,372	3d5fa9ad	54,272
c2b04d58	45,551	2bcc0d72	21,713	e2f21971	44,032
a181f255	92,807	afcab12f	69,603	384a6e54	49,152
f09e878a	52,166	91a706d4	29,312	210033d2	34,816
1508c8c9	92,418	c08fd2bc	63,980	edecfc3f	13,312
516fb310	45,161	4beeda26	20,518	bad13ac3	32,768
1fb5398a	48,795	fab8097c	22,407	df86781e	43,008
1ce21c33	63,703	1f89ec68	37,841	7f6a41dc	41,984
477f3b8b	73,644	a55be2cf	44,847	4e4e73b6	21,504
e13deddd	101,869	6090639e	69,861	2969861d	23,552
652475dc	108,964	394b9964	76,520	5e26f76b	25,600
6c2eb137	104,588	b9c78d0d	74,971	cafef1e2	28,672
8efd2988	72,866	abc0e54d	41,369	1da5bf28	29,696
0c380868	52,306	7ef94488	25,542	0c200f97	30,720
2fb372eb	88,438	f041406f	55,714	91ea7ac7	31,744
bf934d5b	60,612	2c940492	29,215	2ac33710	36,864
200270d4	79,344	00b715f9	44,847	afc9301b	38,912
70266dfb	103,509	d2d13cbc	70,014	cb672fef	39,936
Total File Size →	1,878,074	**Total File Size →**	935,839	**Total File Size →**	1,482,752

Table D.2: Test statistics of some of the AVSs after testing against the 43 malicious (P_k) and 43 non-malicious (P_u) variants of JS.Cassandra and the 43 random files.

Top AVSs		AVG	Avast	Avira	Bitdefender	ClamAV	ESET-NOD32	Kaspersky	McAfee	Microsoft	Panda	Symantec	Trend Micro
43 Malicious (P_k) Files	Detection Rate (Accuracy)	17/43 (39%)	35/43 (81%)	12/43 (28%)	1/43 (2.3%)	40/43 (93%)	22/43 (51%)	1/43 (2.3%)	22/43 (51%)	43/43 (100%)	1/43 (2.3%)	1/43 (2.3%)	1/43 (2.3%)
	Sensitivity/Recall	39%	81%	28%	2.3%	93%	51%	2.3%	51%	100%	2.3%	2.3%	2.3%
	Specificity	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
	Precision	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
	F1 Score	57%	90%	44%	4.5%	96%	68%	4.5%	68%	100%	4.5%	4.5%	4.5%
43 Non-Malicious (P_u) Files	Detection Rate (Accuracy)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)
	Sensitivity/Recall	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
	Specificity	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Precision	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
	F1 Score	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
43 Random Files	Detection Rate (Accuracy)	0/43 (0.0%)	0/43 (0.0%)	1/43 (2.3%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)	0/43 (0.0%)
	Sensitivity/Recall	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
	Specificity	100%	100%	98%	100%	100%	100%	100%	100%	100%	100%	100%	100%
	Precision	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
	F1 Score	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

D.3 Further experiments

The meta-signature of JS.Cassandra virus family not only detected 43 malicious files successfully but also detected 43 non-malicious (P_u) files. These non-malicious (P_u) files still had some polymorphic functions intact within them. All of the 43 non-malicious (P_u) files were still executable, but a few gave JavaScript runtime and compilation errors. These executable non-malicious (P_u) files might cause some serious potential threats, as the polymorphic functions inside these files in some cases might still make them polymorphic. Table D.2 shows that none of the 12 AVSs detected these executable non-malicious (P_u) files as malicious. One instance of such a file is explained here.

Table D.3: Generated CRC32b Hash Value and File Size in Bytes for Original Variant File (V_O), Modified Variant File (V_M) and 12 Variants (V_{M1}-V_{M12}).

Malicious File	CRC32b Hash Value	File Size in bytes
Original Variant File V_O	848562f1	8,324
Modified Variant File V_M	fc79adfe	5,695
Variant 1 V_{M1}	557562ad	6,289
Variant 2 V_{M2}	150e0d7a	6,855
Variant 3 V_{M3}	5645b651	7,457
Variant 4 V_{M4}	42f590e7	7,465
Variant 5 V_{M5}	fcb28864	8,049
Variant 6 V_{M6}	dd679959	8,055
Variant 7 V_{M7}	204f3304	9,019
Variant 8 V_{M8}	3e2ef86f	9,649
Variant 9 V_{M9}	3d987ac4	10,713
Variant 10 V_{M10}	0ecba96f	11,255
Variant 11V_{M11}	bbe2b767	12,825
Variant 12 V_{M12}	55a47fbf	14,031
-	**Total File Size →**	125,681

A variant of JS.Cassandra was chosen and around four polymorphic functions were removed from it. Still, more than ten functions were intact inside that variant file. The modified variant file was executed, and 12 new unique polymorphic variants were generated from it for the purpose of distinguishing between malicious (P_k) and non-malicious (P_u) files experimentally. An infinite number of new unique polymorphic variants could be generated from the same file. Table D.3 provides the CRC32b hash value and file size in bytes for the original variant file (V_O), modified variant file (V_M)

and 12 new variants (V_{M1}-V_{M12}). No two files have the same CRC32b hash value and file size.

All 12 new variant files (V_{M1}-V_{M12}), the original variant file (V_O) and the modified original file (V_M) were scanned against the 12 AVSs. Scan results of the 12 AVSs are shown in Table D.4. Only 'Microsoft' antivirus was able to detect all the 14 malicious files successfully. 'Bitdefender' and 'Kaspersky', could not detect any of the 14 malicious files. 'McAfee', could only detect one of the 14 malicious files – the original variant file V_O. Original variant file V_O before modification could be detected by five AVSs, but after the modification, V_M could only be detected by the 'Microsoft' antivirus tool.

Table D.4: Detection Capabilities of Top Well-Known AVSs for Original Variant File (V_O), Modified Variant File (V_M) and 12 New Variants (V_{M1}-V_{M12}).

Top AVs	**AVG**	**Avast**	**Avira**	**Bitdefender**	**ClamAV**	**ESET-NOD32**
Original Variant File (V_O)	**Yes**	No	No	No	**Yes**	**Yes**
Modified Variant File (V_M)	No	No	No	No	No	No
12 Variant Files (V_{M1}-V_{M12})	No	No	No	No	No	No
Top AVs	**Kaspersky**	**McAfee**	**Microsoft**	**Panda**	**Symantec**	**Trend Micro**
Original Variant File (V_O)	No	**Yes**	**Yes**	No	No	No
Modified Variant File (V_M)	No	No	**Yes**	No	No	No
12 Variant Files (V_{M1}-V_{M12})	No	No	**Yes**	No	No	No

The meta-signature of JS.Cassandra virus family was tested to detect the same 14 malicious files (V_O, V_M, and V_1-V_{12}). All 14 files were successfully detected as infected by the 'clamscan' antivirus scanner using the meta-signature in 0.008 sec (Figure D.4).

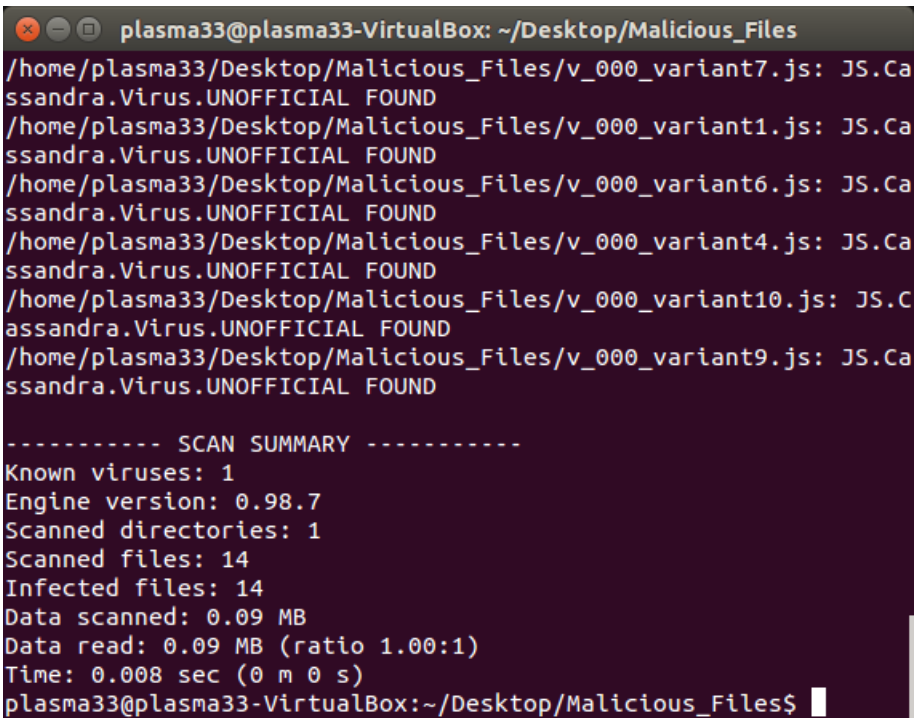

Figure D.4: Screenshot of the scan result obtained from 'clamscan' antivirus scanner for 12 variant files (V_{M1}-V_{M12}), original variant file (V_O) and modified variant file (V_M).

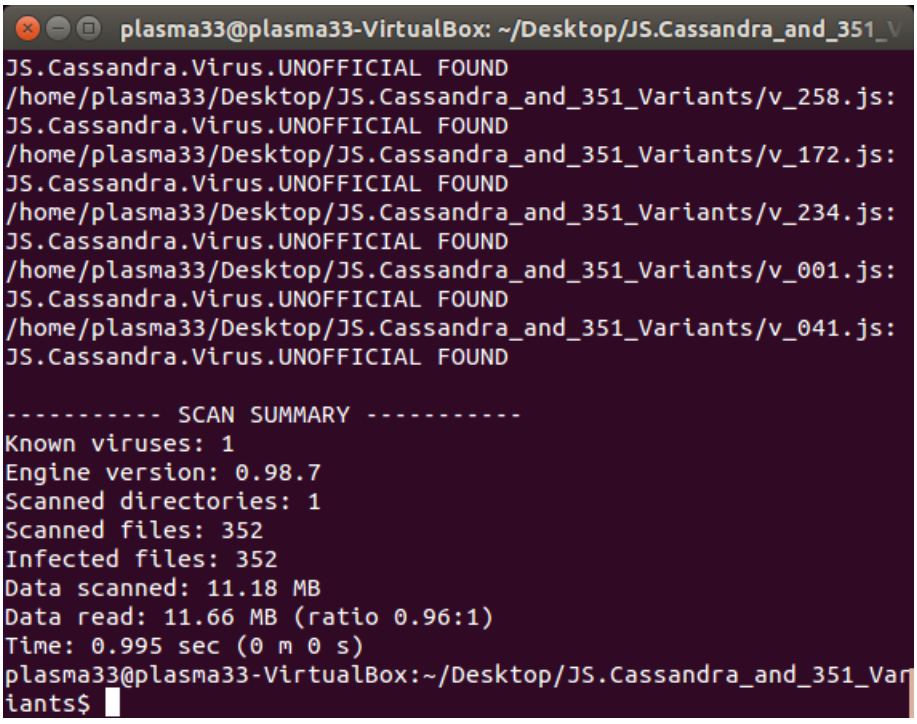

Figure D.5: Screenshot of the scan result obtained from 'clamscan' antivirus scanner for 352 malicious files (P_k) of JS.Cassandra.

Appendix E

Table E.1: Full results of the 71 pairwise local alignments performed in Step-3. In bold are the matrices selected in Step-3 in Section 5.11.3 for further analysis in Part-III.

Substitution Matrix	Identity (%)	Similarity (%)	Gaps (%)	Length	Score
BLOSUM30	51.72%	56.47%	29.43%	41714	176113.00
BLOSUM35	52.43%	52.43%	27.48%	41242	157336.00
BLOSUM40	**52.83%**	**57.81%**	**27.22%**	**41181**	**177285.00**
BLOSUM45	53.46%	53.46%	23.75%	40371	135108.00
BLOSUM50	53.60%	53.60%	24.32%	40502	149366.00
BLOSUM55	53.90%	53.90%	23.58%	40330	155055.00
BLOSUM60	52.48%	52.48%	20.74%	39667	102094.00
BLOSUM62	52.76%	52.76%	21.32%	39796	104663.00
BLOSUM65	52.76%	52.76%	21.32%	39796	104663.00
BLOSUM70	52.84%	52.84%	21.93%	39931	102124.00
BLOSUM75	52.84%	52.84%	21.93%	39931	102124.00
BLOSUM80	54.79%	54.79%	26.63%	41041	170844.00
BLOSUM85	53.26%	53.26%	23.29%	40239	104206.00
BLOSUM90	53.58%	53.58%	24.22%	40478	105990.00
BLOSUM100	54.71%	54.71%	31.69%	42094	178888.00
BLOSUMN	53.85%	53.85%	26.60%	41034	104333.00
DAYHOFF	**49.41%**	**62.32%**	**21.70%**	**39868**	**112237.00**
EDNAFULL	52.07%	52.07%	36.22%	38989	51413.00
GONNET	50.68%	56.88%	19.48%	39377	123052.00
IDENTITY	**100.00%**	**100.00%**	**0.00%**	**397**	**397.00**
MATCH	**63.57%**	**63.57%**	**18.81%**	**6083**	**1192.00**
PAM10	51.19%	51.19%	44.41%	43576	108131.00
PAM20	52.48%	52.48%	39.63%	42275	104507.00
PAM30	52.74%	52.74%	37.26%	41660	101659.00
PAM40	52.94%	52.94%	35.49%	41213	97752.00
PAM50	53.25%	53.25%	32.89%	40571	92401.00
PAM60	52.09%	56.14%	31.09%	41943	97310.00
PAM70	52.18%	56.22%	30.26%	41740	98984.00
PAM80	51.72%	56.21%	28.56%	41325	93804.00
PAM90	51.70%	56.25%	27.18%	40996	89616.00
PAM100	**51.60%**	**62.28%**	**25.74%**	**40656**	**91799.00**
PAM110	51.10%	62.45%	25.35%	40565	88963.00
PAM120	50.71%	62.69%	22.81%	40117	88503.00
PAM130	50.71%	62.69%	22.81%	40117	88503.00
PAM140	50.85%	62.60%	21.92%	39917	90323.00
PAM150	50.30%	62.45%	21.77%	39884	84555.00
PAM160	49.12%	62.65%	20.66%	39636	80017.00
PAM170	50.47%	62.38%	25.71%	40818	131637.00
PAM180	49.94%	62.28%	25.68%	40812	128560.00
PAM190	50.04%	62.42%	25.32%	40694	129164.00
PAM200	50.39%	61.52%	24.92%	40634	120995.00
PAM210	49.16%	61.84%	22.65%	40081	110664.00

PAM220	49.16%	61.84%	22.65%	40081	110664.00
PAM230	49.19%	61.96%	22.37%	40019	111471.00
PAM240	49.19%	61.96%	22.37%	40019	111471.00
PAM250	49.41%	62.32%	21.70%	39868	112237.00
PAM260	48.74%	62.27%	21.37%	39794	109500.00
PAM270	48.54%	62.33%	24.39%	40479	152705.00
PAM280	48.73%	62.49%	23.52%	40278	144159.00
PAM290	48.05%	62.37%	23.34%	40238	139598.00
PAM300	47.75%	62.38%	23.22%	40211	136884.00
PAM310	48.59%	62.05%	23.01%	40162	134398.00
PAM320	47.56%	61.93%	23.47%	40267	131468.00
PAM330	47.56%	61.93%	23.47%	40267	131468.00
PAM340	46.85%	61.74%	23.19%	40203	126252.00
PAM350	**47.05%**	**69.15%**	**24.62%**	**40532**	**165839.00**
PAM360	45.98%	69.21%	24.63%	40534	163200.00
PAM370	46.06%	69.45%	24.11%	40413	163778.00
PAM380	46.13%	69.65%	23.72%	40325	155074.00
PAM390	45.89%	69.04%	24.22%	40440	149355.00
PAM400	45.31%	69.07%	24.36%	40472	144366.00
PAM410	45.40%	67.78%	26.84%	41081	180922.00
PAM420	44.59%	68.30%	26.06%	40866	179081.00
PAM430	44.59%	68.30%	26.06%	40866	179081.00
PAM440	43.67%	68.01%	26.56%	40985	174344.00
PAM450	44.69%	67.43%	27.20%	41165	170906.00
PAM460	45.04%	67.98%	26.21%	40903	161955.00
PAM470	44.17%	67.24%	27.70%	41256	197122.00
PAM480	43.56%	67.29%	27.76%	41270	192432.00
PAM490	43.56%	67.29%	27.76%	41270	192432.00
PAM500	44.43%	66.82%	28.19%	41402	188980.00

Appendix F

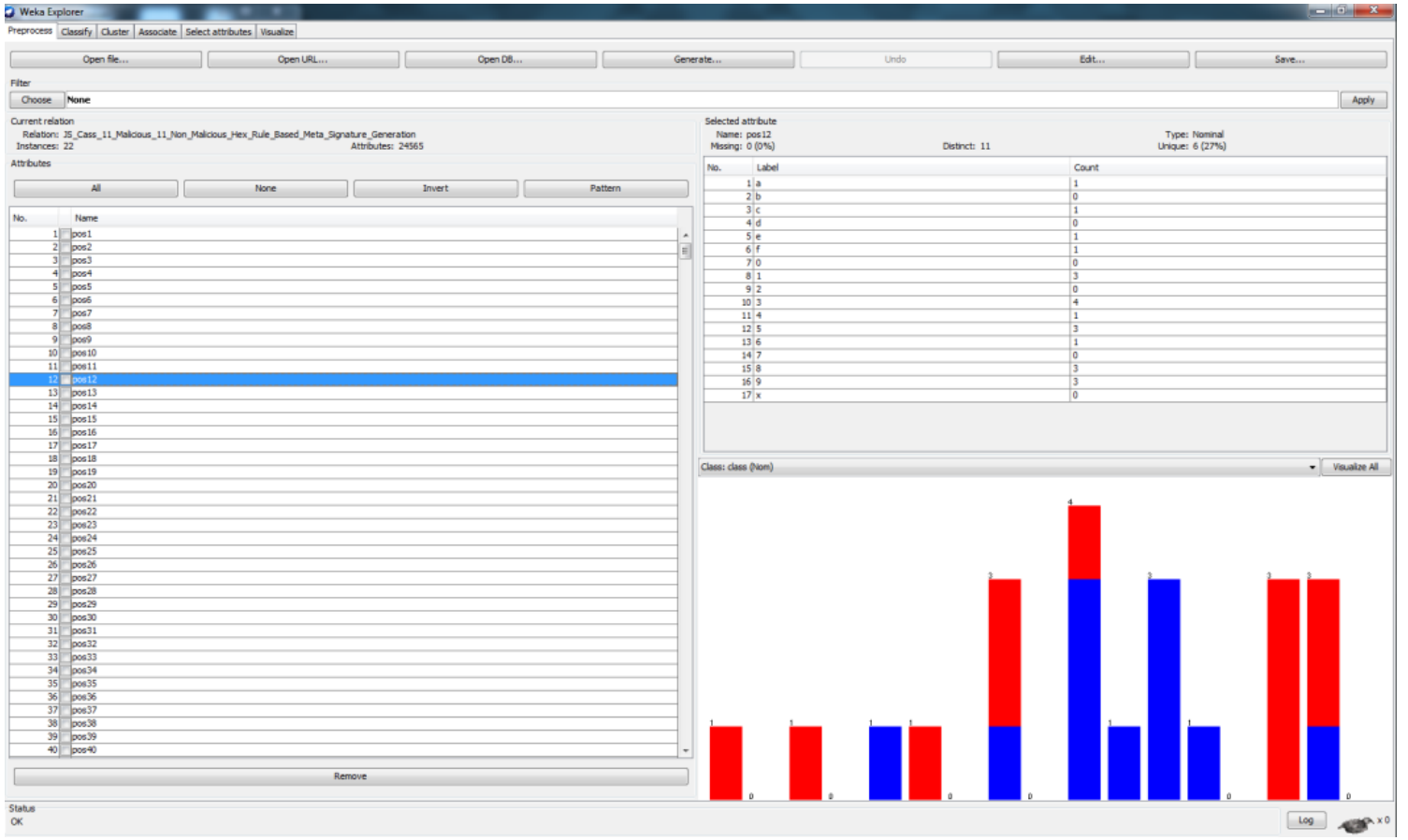

Figure F.1: Screenshot of the preprocess panel obtained from Weka during the generation of NNge rules in Step-2 (Experiment I).

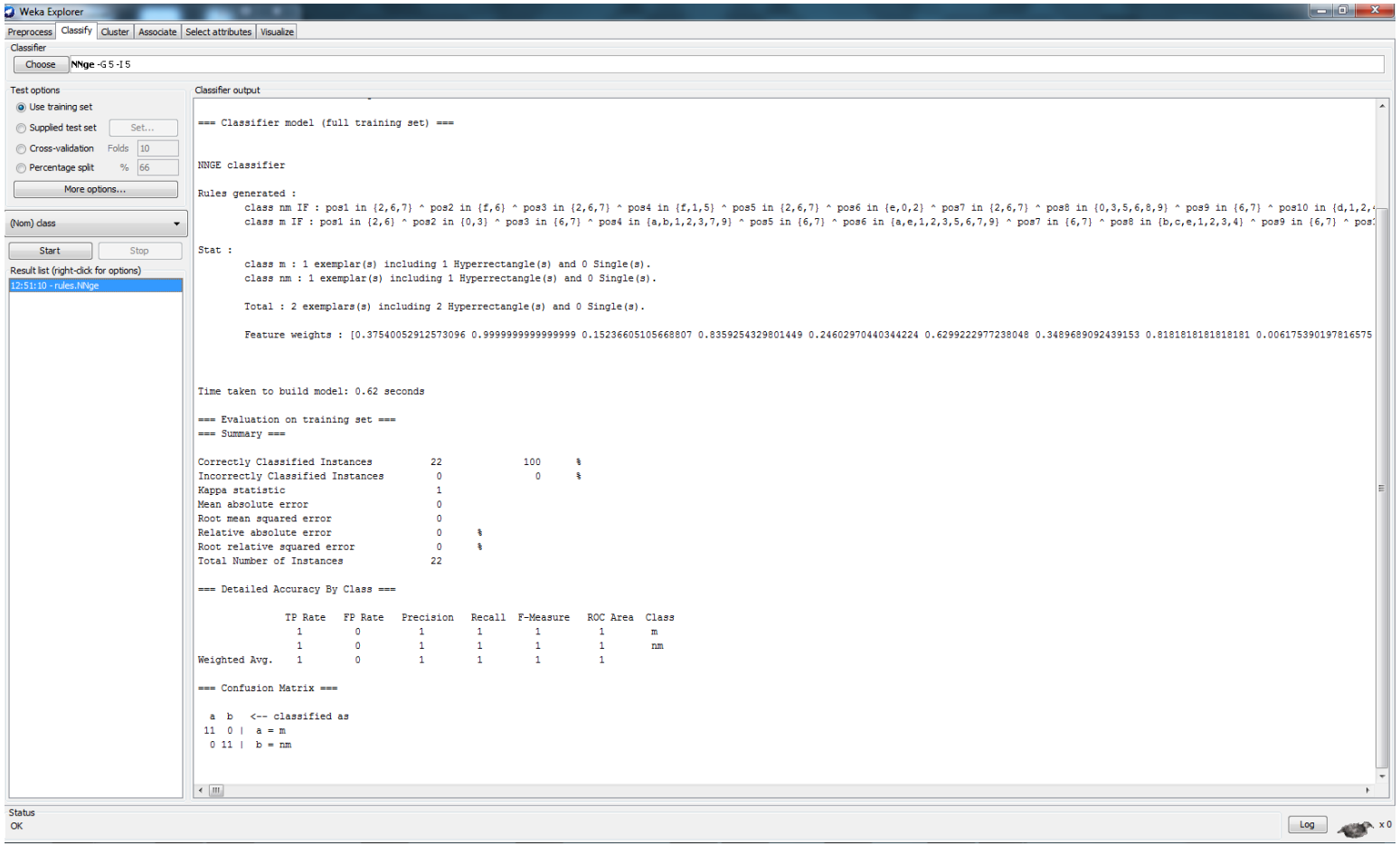

Figure F.2: Screenshot of the classifier model and evaluation information inside the classifier panel obtained from Weka during the generation of NNge rules in Step-2 (Experiment I).

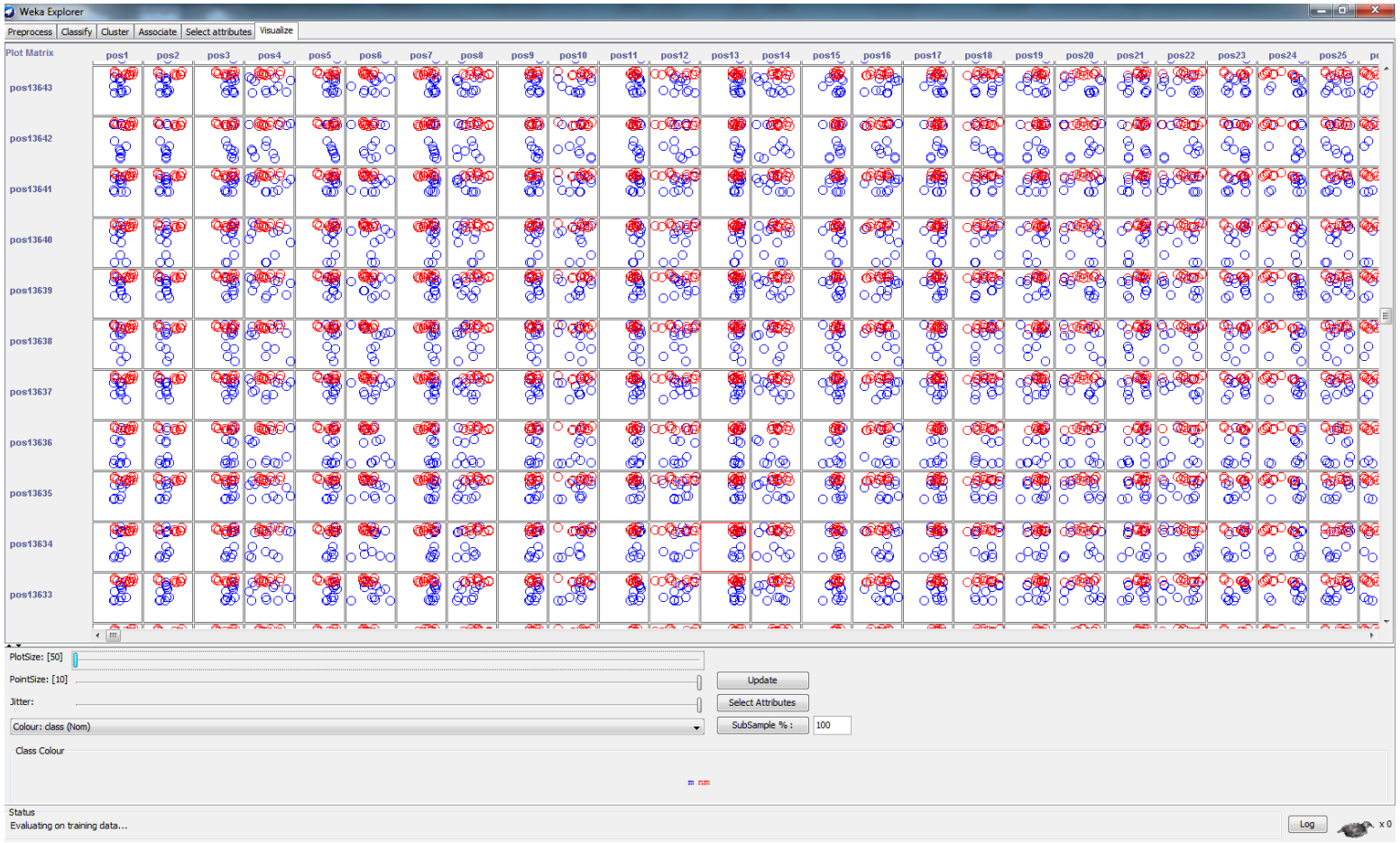

Figure F.3: Screenshot of the visualize panel showing 275 individual plot matrices between pos1-pos25 and pos13633-pos13643 obtained from Weka during the generation of NNge rules in Step-2 (Experiment I).

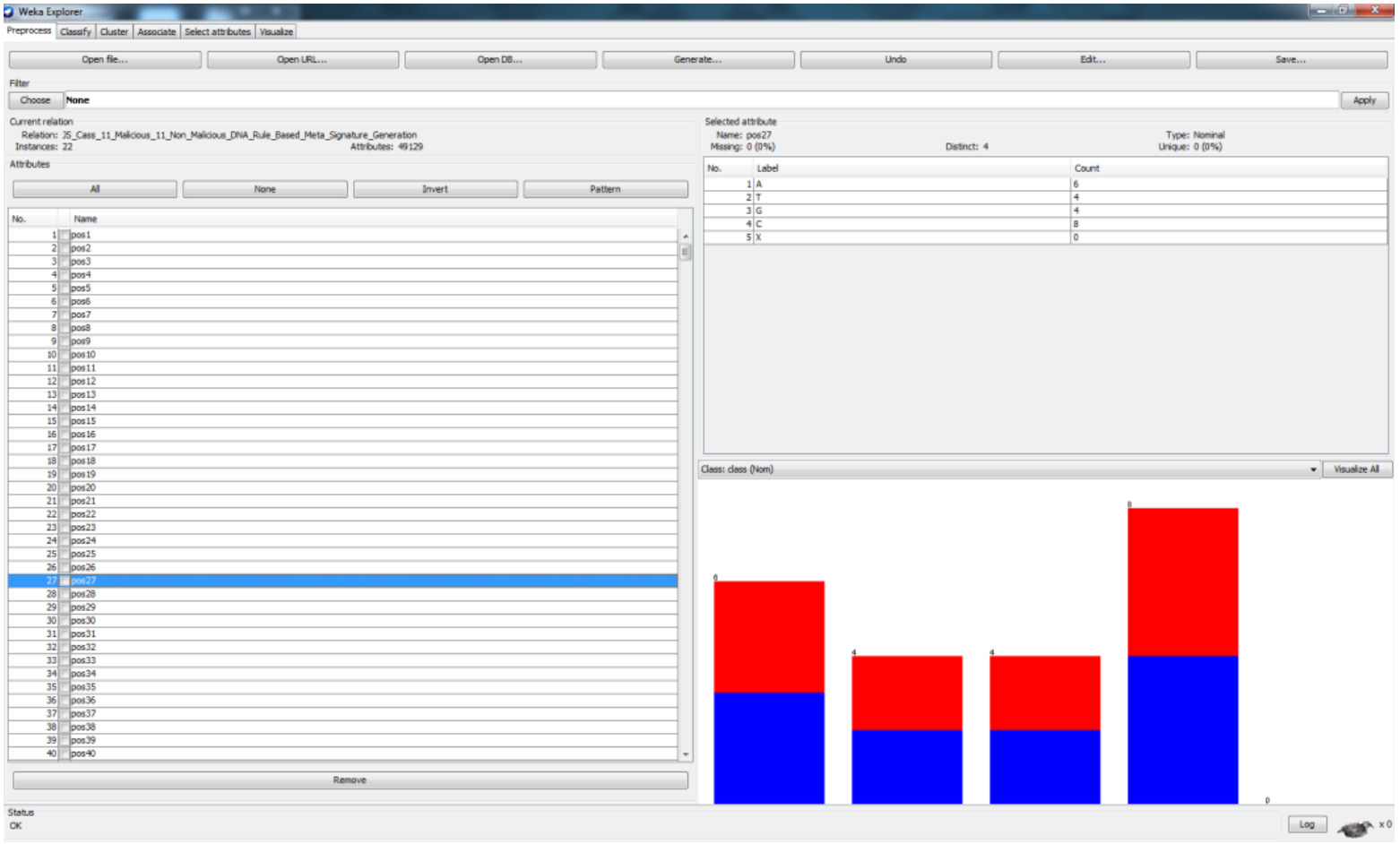

Figure F.4: Screenshot of the preprocess panel obtained from Weka during the generation of NNge rules in Step-3 (Experiment II).

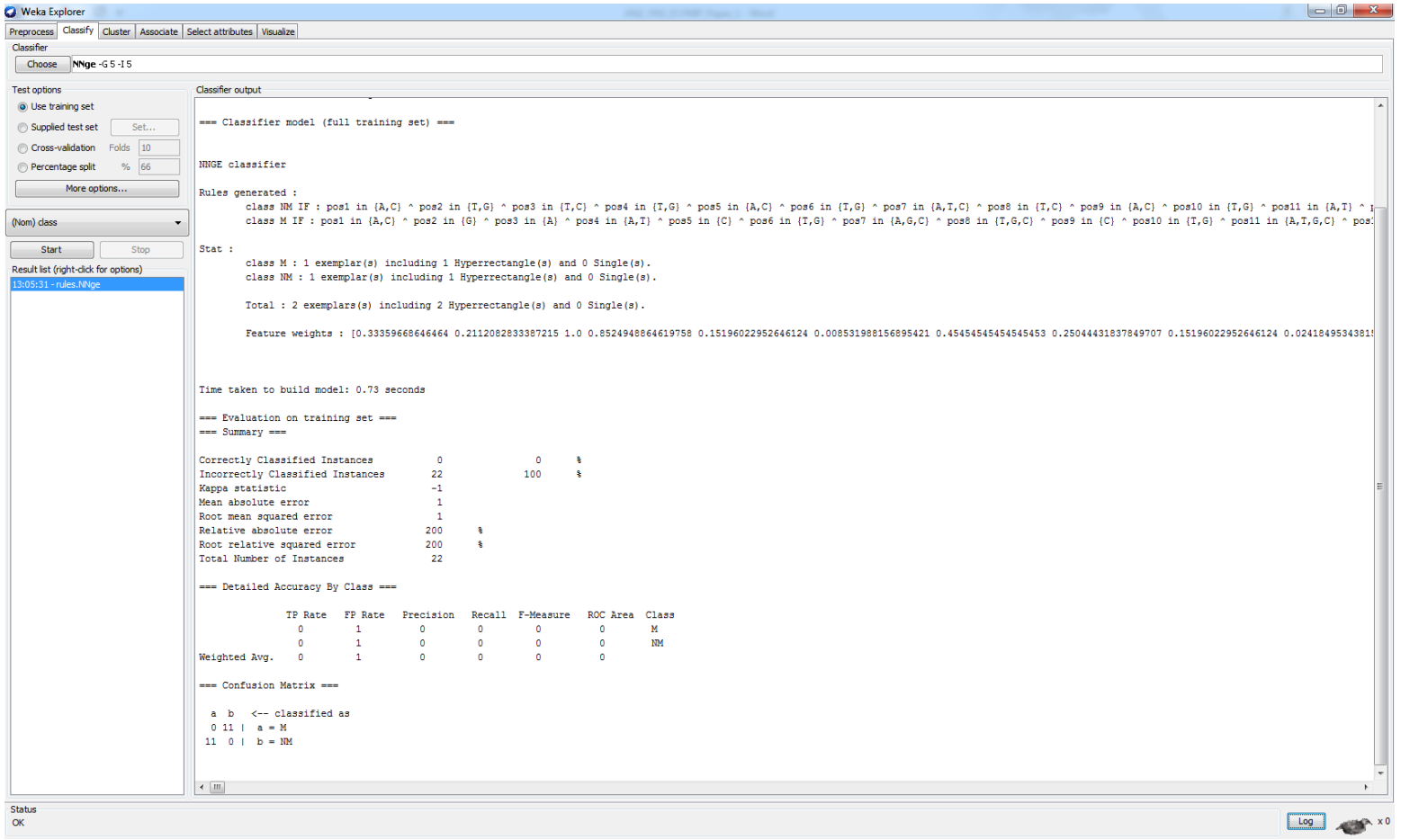

Figure F.5: Screenshot of the classifier model and evaluation information inside the classifier panel obtained from Weka during the generation of NNge rules in Step-3 (Experiment II).

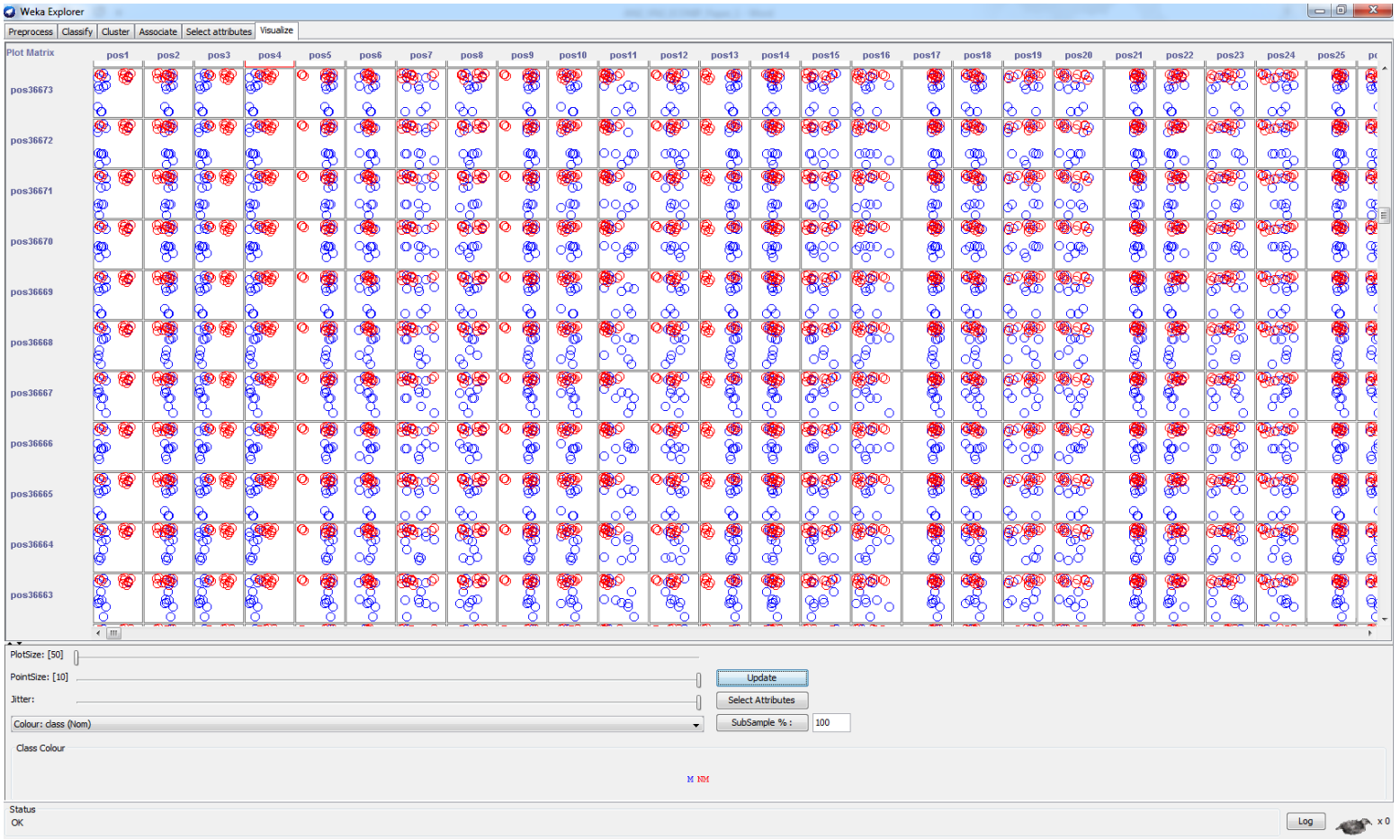

Figure F.6: Screenshot of the visualize panel showing 275 individual plot matrices between pos1-pos25 and pos36663-pos36673 obtained from Weka during the generation of NNge rules in Step-3 (Experiment II).

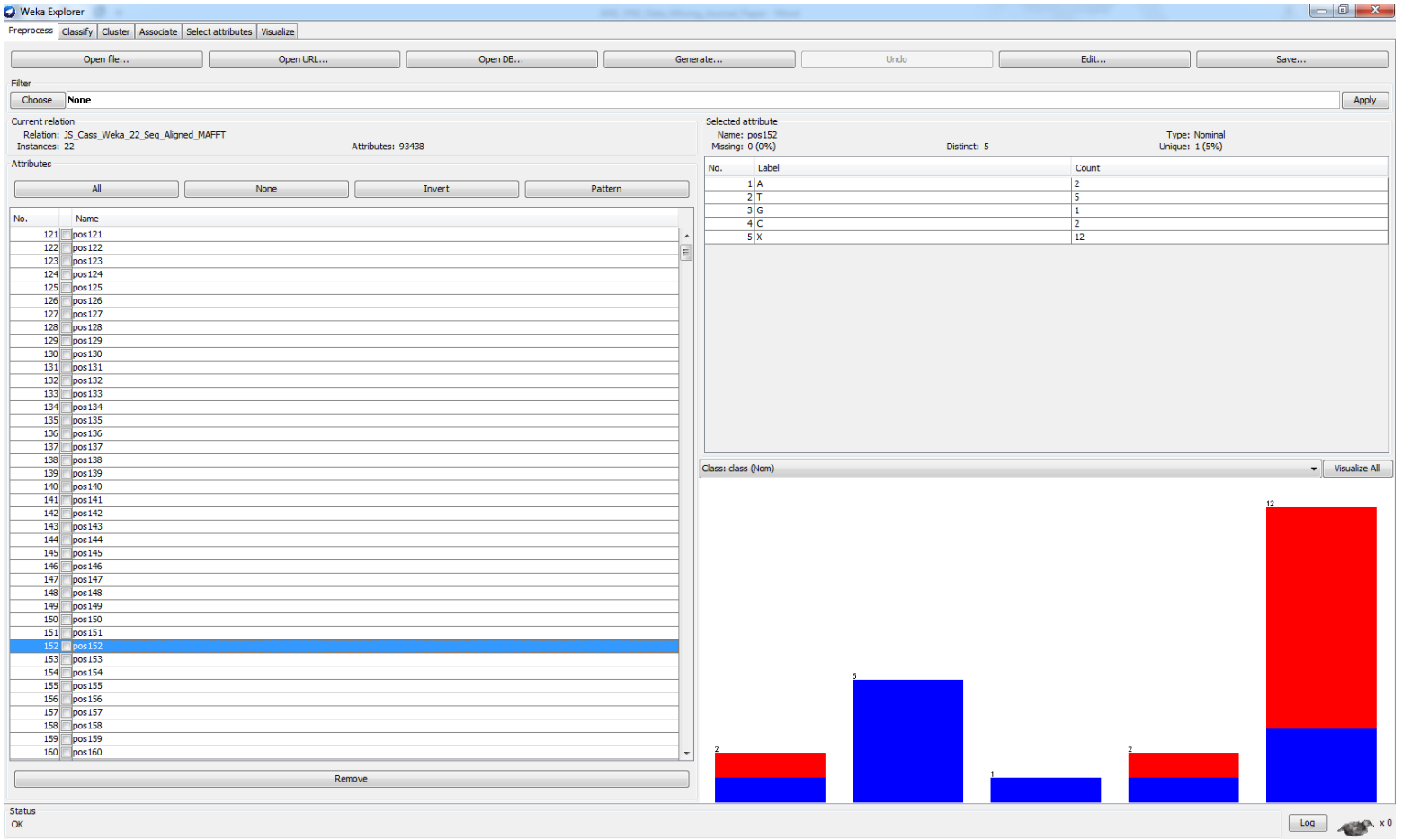

Figure F.7: Screenshot of the preprocess panel obtained from Weka during the generation of NNge rules in Step-4 (Experiment III).

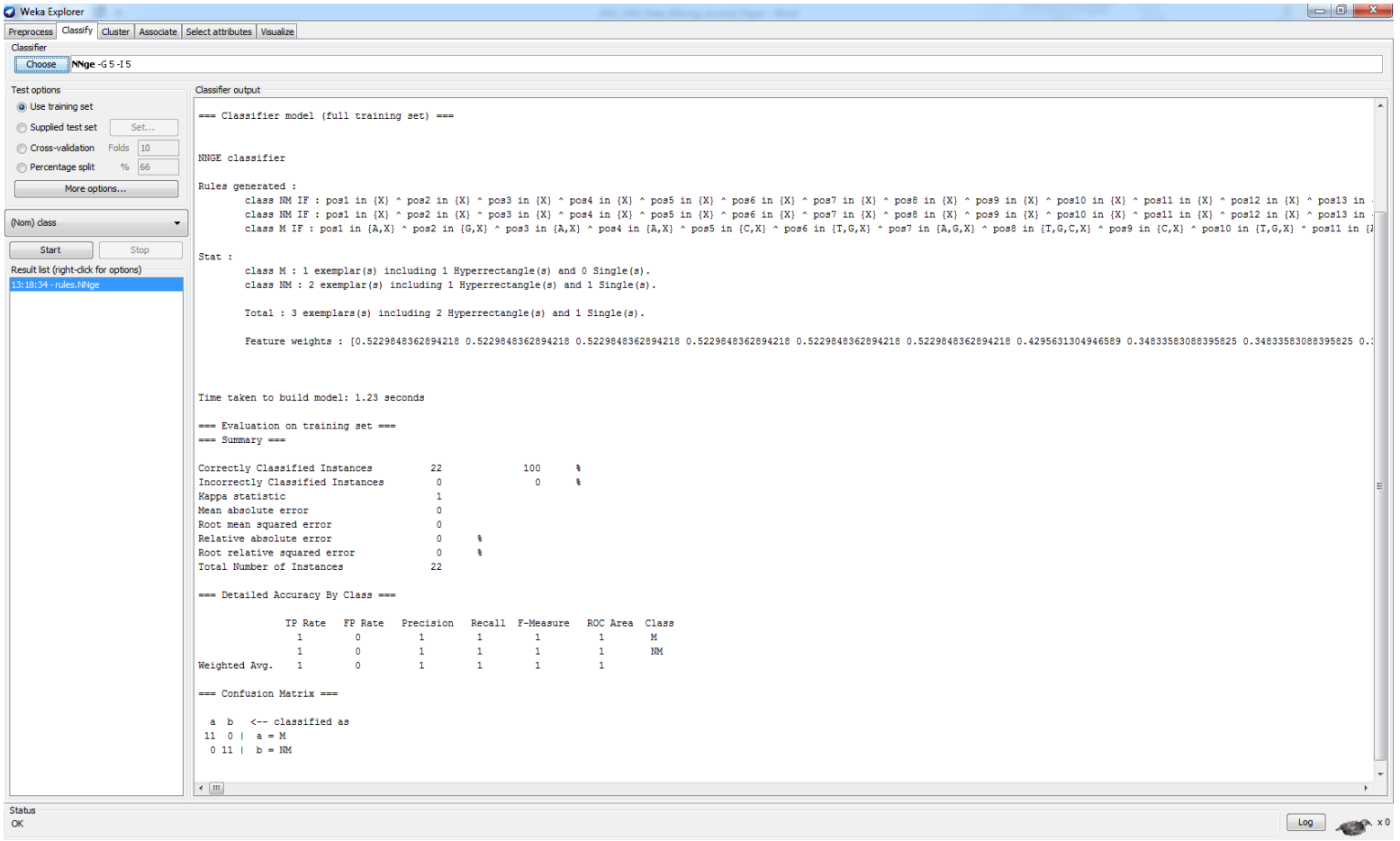

Figure F.8: Screenshot of the classifier model and evaluation information inside the classifier panel obtained from Weka during the generation of NNge rules in Step-4 (Experiment III).

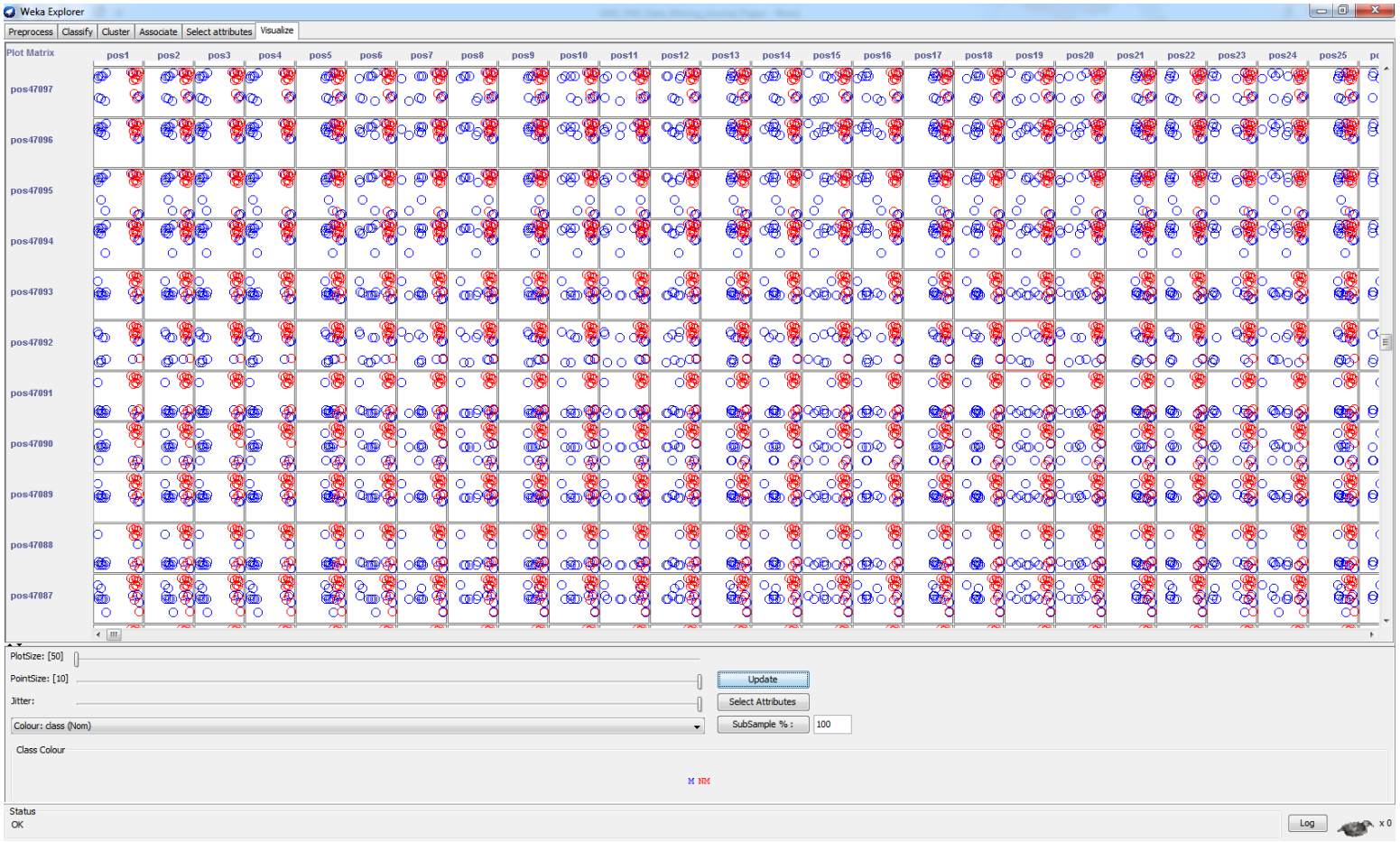

Figure F.9: Screenshot of the visualize panel showing 275 individual plot matrices between pos1-pos25 and pos47087-pos47097 obtained from Weka during the generation of NNge rules in Step-4 (Experiment III).

Appendix G

Table G.1: Full results of the pairwise local alignments that were performed in Step-3 in Chapter 5: Part-II.

Polymorphic malware 1	Pairwise Alignment	Gap Open Penalty	Gap Extend Penalty	Identity Percentage	Similarity Percentage	Gaps Percentage	Alignment Length	Alignment Score
JS.Cassandra Virus	Original JS.Cassandra virus and Variant 1	5	0.5	74.71%	74.71%	25.29%	767	368.00
		5	1	79.64%	79.64%	20.36%	658	306.00
		10	0.5	69.34%	69.34%	30.66%	698	301.00
		10	1	96.18%	96.18%	3.82%	314	263.00
		15	0.5	65.55%	65.55%	34.45%	714	272.50
		15	1	97.27%	97.27%	2.73%	293	249.00
		20	0.5	65.08%	65.08%	34.92%	693	252.00
		20	1	98.51%	98.51%	1.49%	269	242.00
		25	0.5	98.51%	98.51%	1.49%	269	238.50
		25	1	98.51%	98.51%	1.49%	269	237.00
	Variant 1 and Variant 2	5	0.5	67.06%	67.06%	32.94%	2714	887.00
		5	1	70.97%	70.97%	29.03%	2652	540.00
		10	0.5	58.42%	58.42%	41.58%	2862	507.00
		10	1	80.16%	80.16%	19.84%	766	345.00
		15	0.5	66.45%	66.45%	33.55%	1079	391.00
		15	1	89.30%	89.30%	10.70%	430	310.00
		20	0.5	66.45%	66.45%	33.55%	1079	341.00
		20	1	89.30%	89.30%	10.70%	430	300.00
		25	0.5	89.30%	89.30%	10.70%	430	312.00
		25	1	89.30%	89.30%	10.70%	430	290.00
	Variant 2 and Variant 3	5	0.5	57.07%	57.07%	42.93%	3634	574.00
		5	1	77.85%	77.85%	22.15%	957	385.00
		10	0.5	64.53%	64.53%	35.47%	1139	352.50
		10	1	85.02%	85.02%	14.98%	601	295.00
		15	0.5	68.69%	68.69%	31.31%	741	306.00
		15	1	85.33%	85.33%	14.67%	450	262.00
		20	0.5	68.69%	68.69%	31.31%	741	276.00
		20	1	85.33%	85.33%	14.67%	450	242.00
		25	0.5	85.33%	85.33%	14.67%	450	253.00
		25	1	85.33%	85.33%	14.67%	450	222.00
	Variant 3 and Variant 4	5	0.5	65.42%	65.42%	34.58%	2325	642.00
		5	1	83.56%	83.56%	16.44%	669	373.00
		10	0.5	77.58%	77.58%	22.42%	678	374.00
		10	1	95.22%	95.22%	4.78%	418	360.00
		15	0.5	95.22%	95.22%	4.78%	418	359.00
		15	1	95.22%	95.22%	4.78%	418	350.00
		20	0.5	95.22%	95.22%	4.78%	418	349.00
		20	1	95.22%	95.22%	4.78%	418	340.00
		25	0.5	95.22%	95.22%	4.78%	418	339.00
		25	1	95.22%	95.22%	4.78%	418	330.00
	Variant 4 and Variant 5	5	0.5	65.18%	65.18%	34.82%	2533	715.00
		5	1	71.79%	71.79%	28.21%	1843	467.00
		10	0.5	85.64%	85.64%	14.36%	613	433.50
		10	1	100.00%	100.00%	0.00%	397	397.00
		15	0.5	85.64%	85.64%	14.36%	613	408.50
		15	1	100.00%	100.00%	0.00%	397	397.00
		20	0.5	79.21%	79.21%	20.79%	635	398.00
		20	1	100.00%	100.00%	0.00%	397	397.00
		25	0.5	100.00%	100.00%	0.00%	397	397.00
		25	1	100.00%	100.00%	0.00%	397	397.00

Polymorphic malware 2	Pairwise Alignment	Gap Open Penalty	Gap Extend Penalty	Identity Percentage	Similarity Percentage	Gaps Percentage	Alignment Length	Alignment Score
W32.CTX Virus	Original W32.CTX virus and Variant 1	5	0.5	63.57%	63.57%	36.43%	8222	2595.5
		5	1	94.57%	94.57%	5.43%	2304	1922
		10	0.5	73.72%	73.72%	26.28%	4083	1903.5
		10	1	93.48%	93.48%	6.52%	2317	1772
		15	0.5	91.73%	91.73%	8.27%	2335	1726.5
		15	1	92.72%	92.72%	7.28%	2323	1649
		20	0.5	90.51%	90.51%	9.49%	2350	1625.5
		20	1	91.73%	91.73%	8.27%	2335	1531
		25	0.5	90.51%	90.51%	9.49%	2350	1525.5
		25	1	99.29%	99.29%	0.71%	1553	1507
	Variant 1 and Variant 2	5	0.5	55.62%	55.62%	44.38%	12492	2785.50
		5	1	96.15%	96.15%	3.85%	2309	2015.00
		10	0.5	66.14%	66.14%	33.86%	5333	2073.00
		10	1	94.99%	94.99%	5.01%	2334	1893.00
		15	0.5	74.98%	74.98%	25.02%	3565	1893.50
		15	1	94.12%	94.12%	5.88%	2346	1790.00
		20	0.5	73.62%	73.62%	26.38%	3593	1781.00
		20	1	93.13%	93.13%	6.87%	2358	1692.00
		25	0.5	89.92%	89.92%	10.08%	2411	1703.50
		25	1	93.13%	93.13%	6.87%	2358	1602.00
	Variant 2 and Variant 3	5	0.5	71.99%	71.99%	28.01%	4349	2238.50
		5	1	89.59%	89.59%	10.41%	2614	1930.00
		10	0.5	76.46%	76.46%	23.54%	3645	2054.00
		10	1	96.41%	96.41%	3.59%	2060	1804.00
		15	0.5	74.50%	74.50%	25.50%	3686	1913.50
		15	1	94.74%	94.74%	5.26%	2090	1758.00
		20	0.5	73.37%	73.37%	26.63%	3710	1799.00
		20	1	94.74%	94.74%	5.26%	2090	1718.00
		25	0.5	93.62%	93.62%	6.38%	2115	1741.00
		25	1	94.74%	94.74%	5.26%	2090	1678.00
	Variant 3 and Variant 4	5	0.5	64.57%	64.57%	35.43%	4849	1988.50
		5	1	94.40%	94.40%	5.60%	2017	1707.00
		10	0.5	77.16%	77.16%	22.84%	3008	1863.50
		10	1	94.22%	94.22%	5.78%	1956	1613.00
		15	0.5	76.93%	76.93%	23.07%	3008	1807.50
		15	1	93.07%	93.07%	6.93%	1891	1573.00
		20	0.5	75.99%	75.99%	24.01%	3024	1759.50
		20	1	94.15%	94.15%	5.85%	1829	1558.00
		25	0.5	76.97%	76.97%	23.03%	2861	1725.50
		25	1	94.15%	94.15%	5.85%	1829	1543.00
	Variant 4 and Variant 5	5	0.5	70.24%	70.24%	29.76%	2860	1174
		5	1	74.76%	74.76%	25.24%	2639	859
		10	0.5	60.45%	60.45%	39.55%	3009	863
		10	1	100.00%	100.00%	0.00%	736	736
		15	0.5	73.83%	73.83%	26.17%	1479	753.5
		15	1	100.00%	100.00%	0.00%	736	736
		20	0.5	100.00%	100.00%	0.00%	736	736
		20	1	100.00%	100.00%	0.00%	736	736
		25	0.5	100.00%	100.00%	0.00%	736	736
		25	1	100.00%	100.00%	0.00%	736	736

Polymorphic malware 3	Pairwise Alignment	Gap Open Penalty	Gap Extend Penalty	Identity Percentage	Similarity Percentage	Gaps Percentage	Alignment Length	Alignment Score
W32.Kitti Virus	Original W32.Kitti virus and Variant 1	5	0.5	81.20%	81.20%	18.80%	3490	2290.00
		5	1	86.35%	86.35%	13.65%	3297	2061.00
		10	0.5	76.25%	76.25%	23.75%	3486	2225.00
		10	1	100.00%	100.00%	0.00%	1868	1868.00
		15	0.5	76.25%	76.25%	23.75%	3486	2215.00
		15	1	100.00%	100.00%	0.00%	1868	1868.00
		20	0.5	76.25%	76.25%	23.75%	3486	2205.00
		20	1	100.00%	100.00%	0.00%	1868	1868.00
		25	0.5	76.25%	76.25%	23.75%	3486	2195.00
		25	1	100.00%	100.00%	0.00%	1868	1868.00
	Variant 1 and Variant 2	5	0.5	73.04%	73.04%	26.96%	53027	13149.50
		5	1	79.07%	79.07%	20.93%	34109	9585.00
		10	0.5	76.45%	76.45%	23.55%	3482	2233.00
		10	1	100.00%	100.00%	0.00%	1868	1868.00
		15	0.5	76.45%	76.45%	23.55%	3482	2223.00
		15	1	100.00%	100.00%	0.00%	1868	1868.00
		20	0.5	76.45%	76.45%	23.55%	3482	2213.00
		20	1	100.00%	100.00%	0.00%	1868	1868.00
		25	0.5	76.45%	76.45%	23.55%	3482	2203.00
		25	1	100.00%	100.00%	0.00%	1868	1868.00
	Variant 2 and Variant 3	5	0.5	83.79%	83.79%	16.21%	3343	2309.50
		5	1	88.12%	88.12%	11.88%	3266	2130.00
		10	0.5	76.35%	76.35%	23.65%	3484	2229.00
		10	1	100.00%	100.00%	0.00%	1868	1868.00
		15	0.5	76.35%	76.35%	23.65%	3484	2219.00
		15	1	100.00%	100.00%	0.00%	1868	1868.00
		20	0.5	76.35%	76.35%	23.65%	3484	2209.00
		20	1	100.00%	100.00%	0.00%	1868	1868.00
		25	0.5	76.35%	76.35%	23.65%	3484	2199.00
		25	1	100.00%	100.00%	0.00%	1868	1868.00
	Variant 3 and Variant 4	5	0.5	84.84%	84.84%	15.16%	3324	2316.00
		5	1	88.18%	88.18%	11.82%	3265	2129.00
		10	0.5	76.35%	76.35%	23.65%	3484	2229.00
		10	1	100.00%	100.00%	0.00%	1868	1868.00
		15	0.5	76.35%	76.35%	23.65%	3484	2219.00
		15	1	100.00%	100.00%	0.00%	1868	1868.00
		20	0.5	76.35%	76.35%	23.65%	3484	2209.00
		20	1	100.00%	100.00%	0.00%	1868	1868.00
		25	0.5	76.35%	76.35%	23.65%	3484	2199.00
		25	1	100.00%	100.00%	0.00%	1868	1868.00
	Variant 4 and Variant 5	5	0.5	87.03%	87.03%	12.97%	3285	2349.00
		5	1	90.51%	90.51%	9.49%	3225	2217.00
		10	0.5	76.45%	76.45%	23.55%	3482	2233.00
		10	1	100.00%	100.00%	0.00%	1868	1868.00
		15	0.5	76.45%	76.45%	23.55%	3482	2223.00
		15	1	100.00%	100.00%	0.00%	1868	1868.00
		20	0.5	76.45%	76.45%	23.55%	3482	2213.00
		20	1	100.00%	100.00%	0.00%	1868	1868.00
		25	0.5	76.45%	76.45%	23.55%	3482	2203.00
		25	1	100.00%	100.00%	0.00%	1868	1868.00

Appendix H

Table H.1: Generated CRC32b Hash Value and File Size in bytes of the JS.Cassandra variants.

Malicious (P_k) Filename	**CRC32b Hash Value**	**File Size in bytes**	**Non-Malicious (P_u) Filename**	**CRC32b Hash Value**	**File Size in bytes**
JS.Cassandra.js (Original Malicious Virus – P_s)	26489347	7,767	JS.Cassandra_NP.js (Non-Malicious Virus – P_u)	ab657f45	1,823
v_000.js (Malicious Variant 1 – P_k)	848562f1	8,324	v_000_NP.js (Non-Malicious Variant 1 – P_u)	3d94f85f	2,662
v_001.js (Malicious Variant 2 – P_k) – for use in **Step-4** and **Step-5**	fab48c8c	54,183	v_002_NP.js (Non-Malicious Variant 2 – P_u)	90dd470d	3,697
v_002.js (Malicious Variant 3 – P_k)	7c4ea313	9,938	v_003_NP.js (Non-Malicious Variant 3 – P_u)	0631e490	2,981
v_003.js (Malicious Variant 4 – P_k)	bd3b9fdc	8,759	v_004_NP.js (Non-Malicious Variant 4 – P_u)	5273cd32	2,137
v_004.js (Malicious Variant 5 – P_k)	9904ef9c	8,392	v_005_NP.js (Non-Malicious Variant 5 – P_u)	32b7909a	3,748
v_005.js (Malicious Variant 6 – P_k)	511621c7	9,400	v_006_NP.js (Non-Malicious Variant 6 – P_u)	d1f95eae	2,125
v_006.js (Malicious Variant 7 – P_k)	a7bc9795	10,059	v_007_NP.js (Non-Malicious Variant 7 – P_u)	cd486121	2,868
v_007.js (Malicious Variant 8 – P_k)	a878abc3	10,763	v_008_NP.js (Non-Malicious Variant 8 – P_u)	e2220b79	3,874
v_008.js (Malicious Variant 9 – P_k)	ec3797e7	12,282	v_009_NP.js (Non-Malicious Variant 9 – P_u)	d4cffb98	2,965
v_009.js (Malicious Variant 10 – P_k)	a2e5c540	10,799	v_010_NP.js (Non-Malicious Variant 10 – P_u)	91f3e71f	1,688
v_010.js (Malicious Variant 11 – P_k)	9c8432d2	10,873	–	**Total File Size →**	30,568
–	**Total File Size →**	161,539			

Table H.2: Generated CRC32b Hash Values and File Sizes in Bytes for 100 New (P_x) Malware Variants of JS.Cassandra Virus.

First 50 Malicious (P_x) Variants			Second 50 Malicious (P_x) Variants		
Filename	**CRC32b Hash Value**	**File Size (bytes)**	**Filename**	**CRC32b Hash Value**	**File Size (bytes)**
victim_001.js	fb8fccaf	33,050	victim_051.js	93301735	64,390
victim_002.js	7457878a	37,732	victim_052.js	2def8c3f	15,489
victim_003.js	7af43c5d	9,186	victim_053.js	e02076b6	16,660
victim_004.js	687ee66f	23,256	victim_054.js	dffb0823	10,720
victim_005.js	52745426	10,838	victim_055.js	3bae0183	10,379
victim_006.js	972aa98b	10,167	victim_056.js	92acbdc3	11,121
victim_007.js	5390b420	12,531	victim_057.js	87ae11d3	134,038
victim_008.js	5ae0f541	14,130	victim_058.js	6aff7b5d	10,096
victim_009.js	0fa3a55b	29,349	victim_059.js	01523ceb	192,495
victim_010.js	0d102b71	293,347	victim_060.js	fdd45783	11,104
victim_011.js	51036d4a	11,232	victim_061.js	63e98a11	21,071
victim_012.js	445a3eae	101,860	victim_062.js	fc0b761d	12,719
victim_013.js	98fd097d	24,124	victim_063.js	4adf09fd	14,198
victim_014.js	361a20de	13,150	victim_064.js	7876bb84	30,715
victim_015.js	75b47971	96,449	victim_065.js	fa7aed76	20,160
victim_016.js	c9b01595	10,696	victim_066.js	4b65271b	23,247
victim_017.js	0fba7f20	15,711	victim_067.js	acea45f5	13,058
victim_018.js	f55fa526	53,456	victim_068.js	eda7e7f2	9,987
victim_019.js	7d181fee	29,323	victim_069.js	dc5c64c5	36,977
victim_020.js	1ebe0331	21,787	victim_070.js	3495e532	9,377
victim_021.js	4e47fbdc	14,305	victim_071.js	fbc0e685	38,944
victim_022.js	0c72ac9f	10,904	victim_072.js	83c410a0	12,479
victim_023.js	4305732f	14,509	victim_073.js	7860fe50	21,141
victim_024.js	56937158	91,550	victim_074.js	f79001d5	75,464
victim_025.js	78eb8e39	57,497	victim_075.js	1dd75579	35,219
victim_026.js	9a404150	19,345	victim_076.js	3305b4ac	7,417
victim_027.js	ea2de1e7	15,904	victim_077.js	8582f3ba	10,143
victim_028.js	5fa264f6	13,193	victim_078.js	addf3ab8	20,599
victim_029.js	39821c49	16,600	victim_079.js	5506de3c	21,378
victim_030.js	594b7bc0	9,885	victim_080.js	4e3b05d0	15,560
victim_031.js	f13dec5b	10,287	victim_081.js	b577bf12	7,139
victim_032.js	db92f605	8,805	victim_082.js	f90c0024	22,550
victim_033.js	c6731067	11,921	victim_083.js	88d49884	22,391
victim_034.js	cc7aa8b6	7,438	victim_084.js	b0eb1828	16,933
victim_035.js	a5beb6d4	11,570	victim_085.js	c02f9cff	24,380
victim_036.js	7401ef34	22,952	victim_086.js	657863c6	14,102
victim_037.js	1d3d8f93	20,730	victim_087.js	f7d5a081	21,132
victim_038.js	22fe4afb	15,469	victim_088.js	8154da61	14,085
victim_039.js	f27b1448	36,242	victim_089.js	9ec6e8b7	64,953
victim_040.js	116c03d6	12,482	victim_090.js	252a9b16	9,646
victim_041.js	2729c948	12,797	victim_091.js	8606d169	10,771
victim_042.js	ba7cd8d2	17,026	victim_092.js	10f3faa2	8,000
victim_043.js	eae5d235	16,058	victim_093.js	78a2a83e	13,504
victim_044.js	22595278	22,864	victim_094.js	2b7e56aa	17,997
victim_045.js	9783eb42	11,231	victim_095.js	be0a93c7	20,269
victim_046.js	83160322	11,997	victim_096.js	fafae3ec	9,712
victim_047.js	ffdf36b7	12,274	victim_097.js	9444733d	11,403
victim_048.js	fadac221	83,147	victim_098.js	e01c5570	9,636
victim_049.js	cb2fa95b	57,495	victim_099.js	ad2f1653	11,832
victim_050.js	3150d643	17,753	victim_100.js	943a4a77	83,787
Total File Size →		1,535,604	**Total File Size →**		1,340,567