Scientific Research Publishing

Journal of Information Security, 2018, 9, 265-298
http://www.scirp.org/journal/jis
ISSN Online: 2153-1242
ISSN Print: 2153-1234

Generating Rule-Based Signatures for Detecting Polymorphic Variants Using Data Mining and Sequence Alignment Approaches

Vijay Naidu*, Jacqueline Whalley, Ajit Narayanan

School of Engineering, Computer and Mathematical Sciences, Auckland University of Technology, Auckland, New Zealand
Email: *vijay.naidu@aut.ac.nz, jacqueline.whalley@aut.ac.nz, ajit.narayanan@aut.ac.nz

How to cite this paper: 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*, **9**, 265-298.
https://doi.org/10.4236/jis.2018.94019

Received: August 30, 2018
Accepted: October 8, 2018
Published: October 11, 2018

Abstract

Antiviral software systems (AVSs) have problems in detecting polymorphic variants of viruses without specific signatures for such variants. Previous alignment-based approaches for automatic signature extraction have shown how signatures can be generated from consensuses found in polymorphic variant code. Such sequence alignment approaches required variable length viral code to be extended through gap insertions into much longer equal length code for signature extraction through data mining of consensuses. Non-nested generalized exemplars (NNge) are used in this paper in an attempt to further improve the automatic detection of polymorphic variants. The important contribution of this paper is to compare a variable length data mining technique using viral source code to the previously used equal length data mining technique obtained through sequence alignment. This comparison was achieved by conducting three different experiments (*i.e.* Experiments I-III). Although Experiments I and II generated unique and effective syntactic signatures, Experiment III generated the most effective signatures with an average detection rate of over 93%. The implications are that future, syntactic-based smart AVSs may be able to generate effective signatures automatically from malware code by adopting data mining and alignment techniques to cover for both known and unknown polymorphic variants and without the need for semantic (run-time) analysis.

Keywords

NNge Classifier, Gap Penalties, JS.Cassandra Virus, Polymorphic Virus, Automatic Signature Generation, Sequence Alignment, Syntactic Exploration

1. Introduction

Computer worms and viruses continue to grow despite improved intrusion de-

tection, firewall and antivirus software systems (AVSs). For a malware detection system such as an AVS, the primary issue is to detect a worm or virus variant that is not stored in its signature database. Modern detection methods are frequently unable to detect new malware variants until they make an appearance even when a signature of one variant of that particular malware is known [1] [2] [3] [4], because of polymorphism. Such polymorphism usually leaves the payload instructions alone but changes the structure of the malware through modification in the encryption and decryption engines or through reordering of instructions [1] [5] [6] [7] [8] [9]. Polymorphism is typically built into the malware so that the same malware has both a structural (code sequence) and semantic (execution path) difference when propagating. Even if a signature is found for one variant of the malware through syntactic or semantic analysis, there is no guarantee that the same signature will work for other variants of the same malware.

Current signature extraction is by manual assessment using semantic information, by string-based syntactic approaches (see [10] [11] [12] [13] for more detail), or by a learning system that is, as yet, unknown. It has been recommended that learning sophisticated language classes, such as context-free or regular grammars, is not preferable from only positive inputs [14]. It is not known what an optimal negative class of virus should be (e.g. viral code with the payload taken out, non-viral programs, arbitrary code, etc.). AVSs have just about kept pace with new variants because of speedy and effective manual extraction of signatures from execution traces. But polymorphic variants, so far, have exhibited low levels of complexity, and growing sophistication of malware writers may soon make this semantic and post-event approach infeasible [8] [9]. In the worst possible case, a different signature may be required for every variant, leading to constant updating of AVS signature libraries and increased time required to scan incoming packets. For these reasons, a "smart" approach to automatic signature generation based on a purely syntactic approach to learning (*i.e.* an approach that does not require execution traces) is attractive.

A data mining algorithm (*i.e.* rule induction algorithm) is adopted in this paper to search and extract meaningful and smart information from malware source code in the form of rules which represent patterns (code sequence signatures) in malware data. In particular, a nearest neighbor rule induction algorithm such as NNge (details provided later) may work better in noisy domains such as malware code where there may be obfuscation and deliberate introduction of redundancy. If it is possible to generate a rule-based signature automatically from known polymorphic variants, it may also be possible to automatically create signatures that can detect entirely new variants that have not previously been encountered. If this is the case, future smart AVSs can be "pre-emptive" in that they already know, to some extent, what future variants of a virus may look like based on encountering known variants of that virus. The aim of this paper is to explore this possibility in more detail.

One of the issues in applying data mining algorithms to malware data directly is the problem of variable length strings [15], since most data mining algorithms assume equal length strings. There is surprisingly little work on the application of data mining algorithms to automatic malware signature generation, mainly due to the issue of dealing with variable length strings to detect the critical segments of the malware from which to obtain signatures [16] [17]. The little work that exists in data mining focuses on unusual behavior detection [18] [19] [20] based on semantics. The variable length of malware execution traces makes the application of most data mining algorithms difficult because of the default assumption that strings to be mined are of equal length. On the syntactic front, previous work [16] demonstrated how variable length malware source code could be converted into (much longer) equal length code through insertion and deletion of gaps by performing sequence alignment. However, no attempt was made to input these equal length sequences directly into a data mining algorithm.

The significance of this paper is to continue a purely syntactic exploration of the possibility of generating signatures automatically from malware source code without the need for semantic analysis. Syntactic techniques for signature extraction based on structural detection of malware are relatively unexplored in comparison to semantic techniques (*i.e.* techniques based on analyzing the execution behavior of malware). The primary benefit with a syntactic or structural technique is that new and previously unknown variants can be generated from the extracted syntactic or structural rules of existing variants (see [13] for more detail). For a semantic approach, an actual variant instance is required so that it can be run to create an execution trace. This execution trace can be compared with other execution traces from previous instances to determine whether a new signature is required and, if so, how effective that signature is in detecting the family of which this instance is a variant. For a syntactic approach, on the other hand, the set of actual instances so far found is a subset of possible instances of the language derivable using a grammar. Effectiveness of signatures can be determined by generating numerous possible instances even if they have not occurred.

Previous work used sequence alignment to extract consensuses (calculated order of the most frequent symbols found in each position) from malware code variants for the purpose of generating the minimum possible number of signatures for detecting those variants and previously unseen variants. But there was no attempt made to make the most of a by-product of the alignment for data mining purposes, which is the output of equal length malware code of variants. Our task in this paper is to compare signatures produced from the outcomes of data mining the variable length malware code before alignment with the outcomes of data mining the equal length malware code after alignment to determine which method produces better signatures automatically.

Malware is typically a script or program written first in a high-level language

(e.g. C, Java) and then compiled into hex code. The source code will contain instructions for the infector part (how to spread), the payload part (what action to take) and methods for encryption/decryption to hide the malware intent. The infector part also usually contains instructions on how to change the code so that new variants are produced on infection. This leads to many "variants" of the same family where the infector and payload are the same but differently coded. The run-time behavior of the variant is used by human experts to generate signatures (short strings of hex code) for storage in libraries of AVSs to scan incoming packets and the contents of memory to detect the variant and its family. One of the main problems for AVSs is that polymorphic techniques that change the order of the malware code can evade signatures that assume a constant left-to-right ordering in malware code variants. As will be seen below, some very old and well-known viruses still evade modern AVSs because their variants adopt simple code sequence changes that cannot be detected by the latest signatures.

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 1):

a) From the code of a set of seen variants P_s, automatically generate signatures to identify and detect 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 and detect 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 maximize true positive rates, and minimize false positive

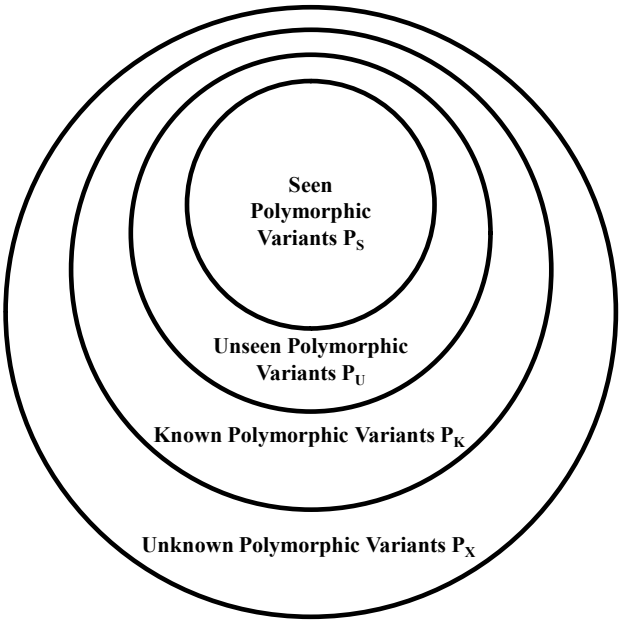

Figure 1. Our method comprising of eight steps.

and false negative rates in both cases above. As will be seen below, previous work has addressed a) through sequence alignment techniques that use insertion operations as well as substitution matrices for matching malware code. It is currently not known whether matching techniques that work well for a) will continue to work well for b), or whether data mining techniques that look for patterns in underlying structure are required to allow generalization to unknown variants.

Roadmap: Section 2 and Section 3 discuss the background and limitations of previous work. Section 4 discusses previous related work relevant to this paper. Section 5 and Section 6 discuss the data mining technique and sequence representations adopted in this paper. In Section 7, we describe our systems and methods. Section 8 summarizes the key features and steps by comparing the three different sets of experiments conducted in Section 7. Section 9 discusses the results. That is Section 9-1) compares the data mining results obtained from three different sets of experiments against other related work and Section 9-2) evaluates signatures generated through the three different sets of experiments against state of the art AVS products, and on the detection of JS.Cassandra polymorphic virus and its known and unknown variants. Section 10 and Section 11 contain the discussions and conclusions. The paper concludes with references and **Appendix** section. **Appendix** Sections A1-A3 explain the three different sets of experiments (Experiments I-III) that were individually performed with these methods.

2. Background

A key development in syntactic approaches has been adoption of string-based algorithms in bioinformatics for identifying structural matches in malware code. Such algorithms do not just look for the presence or absence of characters in specific positions but also manipulate the strings to allow for insertion of characters to expand the number of matching characters. Importantly, the results of such string manipulation are a set of equal length strings from an initial set of variable length strings. Earlier work [21] has demonstrated that string matching and sequence alignment algorithms taken from bioinformatics perform best with biologically represented strings (DNA or protein) rather than non-biological character sets, possibly due to being optimized for chemistry-based mutations between characters. We follow previous approaches in transforming malware code to an appropriate biological string representation before sequence alignment, with transformation of consensuses (*i.e.*, those parts of the malware strings that are common) back to hexadecimal (hex) code for signature generation (see [21] for more detail).

A sequence-based method to signature extraction was previously proposed and illustrated utilizing the Smith-Waterman algorithm (SWA) without gap penalties [10]. The method adopted in [10] was further fine-tuned [11] by selecting SWA with six different substitution matrices. Results demonstrated that it was

possible to extract signatures for pairs of malware strings and meta-signatures for a family of malware after implementing data mining rule-extraction methods (PRISM [22]) to the extracted signatures. That is, underlying patterns were mined after two rounds of matching: the first round dealt with pairwise matching of variable length malware variants to produce level 1 consensuses, where each consensus was of different length to other consensuses; the second round dealt with multisequence alignment of these variable length level 1 consensuses to produce level 2 equal length consensuses, or signatures. The two-stage alignment process (pairwise followed by multiple) was required because of the computational difficulty in running multiple sequence alignment directly on strings of vastly different lengths, as malware variants of a family tend to be. The initial pairwise alignment allowed pairwise recurring similarities to be first identified in consensuses before these consensuses were themselves multiply aligned to produce level 2 consensuses (signatures). These level 2 consensuses of equal length were then mined using PRISM to find underlying patterns, resulting in meta-signatures. Another relevant enhancement in syntactic methods was also recently published [12]. Two different dynamic programming techniques, namely, Needleman-Wunsch and SWA, were explored for matching purposes, and it was found that SWA gave the best results with 100% of unseen P_u variants in the test set P_k being detected. Recent work [13] adopted ten different combinations of gap open and gap extend penalties in conjunction with dynamic programming. It was found that changes in these parameters helped to generate effective signatures for detecting unseen P_u (test set P_k) polymorphic variants.

3. Limitations of Previous Work

Previous work using a sequence alignment approach [10] [11] [12] [13] had two limitations. First, the string matching search using the SWA found only the most optimally-conserved meta-signatures using left-to-right matching techniques. It was not known how successful these meta-signatures would be when used against unknown P_x variants where code has been moved and restructured (*i.e.* case b) above), thereby reducing the number of left-to-right matches. A rule-based or top-down approach that tries to find underlying patterns may overcome the limitation of signatures generated in left to right order, thereby reducing or nullifying the false positive and false negative rates [23] [24]. Rule-based signatures obtained in this way might potentially capture knowledge which makes the identification and detection of unknown P_x variants possible. Thus, the rule-based NNge approach (more details in Section 5) is explored in this research and detailed in this paper.

A second limitation, as noted above, was that the alignment using SWA was "pairwise" and only allowed alignment of two viral sequences at a time in the first round of alignment. Multiple sequence alignment was then used on all pairwise consensuses to generate equal length sequences for rule-based data mining using PRISM. However, in the first round, only those regions of similarity in the

pairwise alignment were extracted rather than regions of similarity across all viral sequences. It was not known how much "family" information was lost through pairwise comparison of variants. A rule-based data mining approach, on the other hand, allows all sequences to be used to extract signatures should take into account all the information in all the sequences at the same time, including both family generic and variant specific information. This should then lead to more effective signatures.

4. Related Work

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 1. Practically all previous approaches deal with only a restricted set of variants belonging to the same malware family and it is currently not known how generalizable these approaches are for detecting other variants of the same family, either unseen (P_u) or unknown (P_x). In our approach, new P_u and previously unknown P_x structural variants belonging to the JS.Cassandra polymorphic viral family are provided by one of the most respected grey hat hackers.

Some other related and selected previous work that primarily focuses on malware detection using data mining and bioinformatics approaches are shown in Table 2. Very little research has been undertaken using data mining and bioinformatics approaches for the detection of polymorphic virus and its unseen P_u

Table 1. Related research to the automatic signature generation in malware detection.

Researchers/Application	Type of Malware	Type of Approach	Description
Wespi *et al.* [25]	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 [26], Autograph [27] and Early Bird [28]	Worms	Syntactic	Generate signatures that constitute individual adjoining byte strings (tokens).
Polygraph [29]	Polymorphic worms	Syntactic	Generates an array of tokens, a subsequence of tokens and Bayes signatures based on probabilistic methods to detect polymorphic worms.
Nemean [30]	Worms	Semantic	Focus on generating signatures that defend against worms.
PAYL [31]	Worms	Semantic	Produces subsequence signature tokens that associate ingress/egress payload notifications to detect the initial replication of worms.
Hamsa [32]	Polymorphic worms	Semantic	Produces a set of signature tokens that can deal with polymorphic worms by investigating their invariant activity.
ShieldGen [33]	Worms	Semantic	Generates network signatures for unseen vulnerabilities (worms) that are based on protocol-aware for instance.
AutoRE [34]	Botnets	Semantic	Produces a spam signature creation architecture from spam emails that use botnets to detect them.
Coull and Szymanski [35]	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.

Continued

Scheirer *et al.* [36]	Polymorphic worms	Syntactic and Semantic	Detection of many polymorphic worms and uses intrusion detection techniques such as sliding window schemes and instruction semantics.
Wurzinger *et al.* [37]	Botnets	Semantic	Detects 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 [38]	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.* [39]	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 [40]	Botnets	Semantic	Generates signatures to detect 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 [41]	Botnets	Semantic	Detects C & C systems but does not produce signatures for those.
Ki *et al.* [42]	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 [43]	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.

Table 2. Some related and selected previous work in malware detection using data mining and bioinformatics approaches.

Researchers	Data Mining	Data Set	Type of Malware	Type of Approach
Chen *et al.* [16]	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 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.
Kumar *et al.* [44]	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 hex sequences from viral files and conversion of hex sequences into ASCII sequences. Multiple sequence alignment was applied on the converted ASCII sequences for data mining process.
Prabha *et al.* [45]	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 1056 malicious viral samples.	Extraction of hex dumps/Extraction of byte sequences in terms of *n*-grams of different sizes.

Continued

Srakaew *et al.* [18]	Data Mining Classifier Algorithm *i.e.* J48 by generating decision trees.	Reference Data Set: 1200 files in total out of which 900 are malicious and 300 are non-malicious. Application Data Set: 3251 files in total out of which 2951 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 hex 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.

variants, let alone its unknown P_x variants. The syntactic approach most closely related [44] adds nothing new to what was published by Chen *et al.* in 2012 [16], and replicates the structural sequence alignment and data mining approaches adopted in that paper and subsequently refined by [10] [11] [12] [13].

Previous use of sequence alignment and data mining has for the most part been semantic in nature, depending on system behavior patterns or using *n*-grams of bytes 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 versions. 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 below, 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.

5. Data Mining

Previous work [11] used PRISM on the consensuses derived after two rounds of alignment to generate rule-based signatures by performing several train/test (P_s/P_u) iterations with an overall accuracy of 62%. Although PRISM and NNge are both rule induction algorithms, the theoretical advantages of choosing NNge over PRISM are due to its potential for improved accuracy and production of extensive or verbose rules. Optimizing rules to produce minimal redundancy is counter-productive in malware signature generation, especially when trying to deal with P_x instances and to keep false positive and negative rates low. Moreover, in NNge, frequent removal of data instances and restoration of the training dataset are not required unlike in PRISM. These steps are overcome in NNge by joining the instances to its nearest neighbour (more details below).

As an instance of a polymorphic string-based technique, consider the structu-

rally-related set of sentences [11]:

The **cat saw the mouse** (Class 1)

The **mouse was seen by the cat** (Class 2)

We see that the **cat saw the mouse** (Class 1)

We see that the **mouse was seen by the cat** (Class 2)

PRISM and NNge were applied on the four structurally-related set of sequences by categorizing them into two classes, namely: Class 1—cat saw the mouse and Class 2—mouse was seen by the cat. The variable length strings were converted into equal length strings by expanding the shorter strings to have a length equal to the longest string by adding the letter "x" at the end of each short string.

PRISM gave the following rules with 75% accuracy after four iterations ("pos" = position):

*If pos*1 = *the*, *pos*2 = *cat*, *pos*3 = *saw*, *pos*4 = *the*, *pos*5 = *cat*, *pos*7 = *the*, *pos*8= *mouse*, *pos*9 = *x and pos*10 = *x then* ***Class*** **1**

*If pos*2 = *mouse*, *pos*3 = *was*, *pos*4 = *seen*, *pos*6 = *was*, *pos*7 = *seen*, *pos*8 = *by*, *pos*9 = *the*, *pos*9 = *x and pos*10 = *x then* ***Class*** **2**

NNge gave the following rules with 100% accuracy ("^" = conjunction; "{}" signifies disjunctive options):

Class **1** *IF*: *pos*1 *in* {*the*, *we*} ^ *pos*2 *in* {*cat*, *see*} ^ *pos*3 *in* {*saw*, *that*} ^ *pos*4 *in* {*the*} ^ *pos*5 *in* {*cat*, *mouse*} ^ *pos*6 *in* {*saw*, *x*} ^ *pos*7 *in* {*the*, *x*} ^ *pos*8 *in* {*mouse*, *x*} ^ *pos*9 *in* {*x*} ^ *pos*10 *in* {*x*}

Class **2** *IF*: *pos*1 *in* {*the*, *we*} ^ *pos*2 *in* {*mouse*, *see*} ^ *pos*3 *in* {*was*, *that*} ^ *pos*4 *in* {*the*, *seen*} ^ *pos*5 *in* {*mouse*, *by*} ^ *pos*6 *in* {*the*, *was*} ^ *pos*7 *in* {*cat*, *seen*} ^ *pos*8 *in* {*by*, *x*} ^ *pos*9 *in* {*the*, *x*} ^ *pos*10 *in* {*cat*, *x*}

The strings were extracted from the above-mentioned PRISM and NNge rules and are shown as follows in their corresponding classes:

PRISM:

Class 1: *the cat saw the cat the mouse*

Class 2: *mouse was seen was seen by the*

NNge:

Class 1: *the we cat see saw that the cat mouse saw the mouse*

Class 2: *the we mouse see was that the seen mouse by the was cat seen by the cat*

The results on this example string set show that NNge can generate rules with 100% accuracy over PRISM, which generated rules with 75% accuracy. One of the aims of this paper is to determine whether this result is generalizable to many more instances of strings (variants) belonging to different classes (families).

NNge, first introduced by Martin (1995), is a nearest neighbor algorithm and an expansion of Nge [46], which generalizes by merging exemplars [47] and forming hyperrectangles in feature space that represent conjunction rules (if-then rules) with internal disjunction. The learning is incremental; each ex-

ample is first classified and then generalized by joining the example to its nearest neighbor, either a single instance or a hyperrectangle, in the same class. Each hyperrectangle is converted into a production rule. When a hyperrectangle covers just one instance it is regarded to be non-generalized exemplar [48]. An instance of a hyperrectangle is shown below [49]:

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

$$p2 = (22)\ AND$$

$$(p3 \geq 9\ AND\ p3 \leq 32)\ AND$$

$$p4 = (b\ or\ c)$$

This hyperrectangle covers strings "42210b" and "62231c" but not "3118b", for instance. Within the NNge algorithm [49] (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 generalization. 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 generalization stage extends G^b in sequence to cover I^n at most if the generalized instance does not overlap/cover an inconsistent instance/hyperrectangle [48].

***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 $(I^n) \neq$ *class* (G^b) *THEN Divide/Split* (G^b, I^n) */*Adjustment Stage*/*

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

IF G' overlaps with inconsistent hyperrectangles

THEN add I^n *as a non-generalized 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 Equation (1) (Classification Stage).

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

In Equation (1), $I_k^{\min}$ and $I_k^{\max}$ indicate the set of numerical values across the training collection which correspond to attribute k. For categorical (*i.e.* nominal) attributes, the length of this set is constantly 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 Equation (2) (Distance between the Corresponding Hyperrectangle).

$$d_{num}\left(I_k, G_k\right) = \begin{cases} 0, & I_k \text{ belongs to } G_k \\ 1, & \text{otherwise} \end{cases},$$

$$d_{num}\left(I_k, G_k\right) = \begin{cases} 0, & G_k^{\min} \le I_k \le G_k^{\max} \\ G_k^{\min} - I_k, & I_k < G_k^{\min} \\ I_k - G_k^{\max}, & I_k > G_k^{\max} \end{cases} \quad (2)$$

The constant w_k signifies weights corresponding to attributes and can be regulated throughout the training procedure [46] or can be assigned to mutual information [48] [50].

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 generalization stage comprises 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-generalized exemplar [48].

6. Sequence Representations

In the experiments that follow, two different types of code representation are tested for data mining using NNge. The first type uses the hex representation and the second uses a DNA version of the hex representation, using the conversion rules as follows:

Conversion of hexadecimal into binary code was accomplished employing the subsequent rules: "1" → "0001"; "2" → "0010"; "3" → "0011"; "4" → "0100"; "5" → "0101"; "6" → "0110"; "7" → "0111"; "8" → "1000"; "9" → "1001"; "0" → "0000"; "a" → "1010"; "b" → "1011"; "c" → "1100"; "d" → "1101"; "e" → "1110"; and "f" → "1111". Successive conversion of the binary code into DNA sequences was accomplished employing the subsequent rules: "00" → "A"; "11" → "T"; "10" → "G"; and "01" → "C".

So, for instance, the hex string "1234567890abcdef" becomes "0001001000110100010101100111100010010000101010111100110111101111" (binary code) and then becomes "ACAGATCACCCGCTGAGCAAGGGTTATCTGTT" (DNA sequence).

The experiments are intended to check whether data mining using DNA code produces better results than using hex code. Once viral code is converted to DNA code, sequence alignment using publicly validated and provably tested alignment software becomes possible.

Also, in the experiments below, "padding" was required to convert variable length viral strings into equal length strings for two of the experiments (Experiments I and II). For example, given hex strings "13ad3" and "245335623f", pad-

ding produces "13ad3xxxxxx" to make both strings of equal length.

7. Systems and Methods

Methods Overview (Experiments I-III): The method in Experiment I consists of six steps, summarized as follows. Step-1 deals with virus code variant generation P_k and separating the training set P_s from the test set P_u. Step-2 deals with the process of variable length data mining on a small percentage of the training P_s and test P_u sets using NNge classifier to generate rules for string extraction. Step-3 deals with the extraction of common training sequences (*i.e.* strings, or first-level rule-based consensuses) using the NNge rules. Step-4 deals with converting the hex code of the training P_s and test P_u sets (obtained from Step-1) as well as first-level consensuses (obtained from Step-3) into a form (in this case, DNA) acceptable for sequence alignment. Step-5a deals with the process of pairwise (local) sequence alignment between the first-level consensuses and some variants of the training set P_s (both obtained from Step-4) using the SWA to produce equal length sequences (*i.e.* second-level consensuses). Step-5b deals with the extraction of meta-signatures, or common substrings, from these second-level consensuses. Step-6 deals with the conversion of meta-signatures back into viral hex code for the purpose of signature testing against P_k and P_x viral sets. More details concerning each step are supplied in **Figure A1** in the **Appendix** section.

The method in Experiment II consists of six steps. The same procedure as Experiment I was used along with the same training P_s and test P_u sets, with the only difference being that some variants of the training set P_s were converted into DNA format prior to the process of variable length data mining. More details concerning each step are supplied in **Figure A2** in the **Appendix** section.

The method in Experiment III consists of seven steps. The same procedure as Experiments I and II was adopted and the same training P_s and test P_u sets were used, with the only difference being an additional step of multiple sequence alignment on the training set P_s to produce equal length sequences prior to the process of equal length data mining. More details concerning each step are supplied in **Figure A3** in the **Appendix** section.

8. Comparison of Three Sets of Experiments in Detail

Experiment I consist of taking 22 viral strings in hex (11 malicious (set M) and 11 non-malicious (set NM)), applying NNge to M_{HEX} and NM_{HEX}, and converting the NNge results into two variable length strings ($N1_{HEX}$, $N2_{HEX}$), as shown in the "cat mouse" examples previously. The hex strings are then converted to DNA for pairwise sequence alignment between $N1_{DNA}$ and P_s on the one hand and $N2_{DNA}$ and P_s on the other. This produces consensuses $C1_{DNA}$ (between $N1_{DNA}$ and P_s) and $C2_{DNA}$ ($N2_{DNA}$ and P_s), and these consensus $C1_{DNA}$ and $C2_{DNA}$ become the meta-signatures for use against P_k and P_x after converting back into hex (*i.e.* $C1_{HEX}$ and $C2_{HEX}$). Therefore, the viral code remains in hex format until

just before pairwise sequence alignment. Set NM in this paper is defined as malware that is generated by eliminating their key polymorphic functions and are partially functional with no payload properties.

Experiment II consists of taking 22 viral strings in hex (11 malicious (set M) and 11 non-malicious (set NM)). M_{HEX} and NM_{HEX} are then converted to DNA before applying NNge to M_{DNA} and NM_{DNA}, and converting the NNge results into two variable length strings ($N1_{DNA}$, $N2_{DNA}$). The $N1_{DNA}$ and $N2_{DNA}$ are then pairwise sequenced against P_s. This produces $C1_{DNA}$ (between $N1_{DNA}$ and P_s) and $C2_{DNA}$ ($N2_{DNA}$ and P_s), and $C1_{DNA}$ and $C2_{DNA}$ become the meta-signatures for use against P_k and P_x after converting back into hex (*i.e.* $C1_{HEX}$ and $C2_{HEX}$). The difference between Experiment I and Experiment II is that the viral strings are converted to DNA first before NNge is applied.

Experiment III consists of taking 22 viral strings in hex (11 malicious (set M) and 11 non-malicious (set NM)). M_{HEX} and NM_{HEX} are then converted to DNA. Multiple sequence alignment is then applied on M_{DNA} and NM_{DNA} to produce equal length sequences M_E and NM_E. Then NNge is applied to M_E and NM_E to produce variable length strings $N1_{DNA}$ and $N2_{DNA}$. $N1_{DNA}$ and $N2_{DNA}$ are then pairwise sequenced against P_s. This produces $C1_{DNA}$ (between $N1_{DNA}$ and P_s) and $C2_{DNA}$ ($N2_{DNA}$ and P_s), and $C1_{DNA}$ and $C2_{DNA}$ become the meta-signatures for use against P_k and P_x after converting back into hex (*i.e.* $C1_{HEX}$ and $C2_{HEX}$). The difference between Experiment II and Experiment III is that the viral strings are multiply aligned first to produce equal length strings before NNge is applied.

Table 3 summarizes the key features and steps by comparing the three sets of experiments (Experiments I-III) performed in this paper. Variable length data mining techniques produced ten unique and 13 common meta-signatures ($C1_{HEX}$ and $C2_{HEX}$). Experiment I generated five unique and four common meta-signatures ($C1_{HEX}$ and $C2_{HEX}$). Experiment II generated 5 unique and nine common meta-signatures ($C1_{HEX}$ and $C2_{HEX}$). Equal length data mining technique (Experiment III) produced 43 unique and five common meta-signatures ($C1_{HEX}$ and $C2_{HEX}$). As can be seen from Table 3, the length of sequences, and therefore the number of attributes where each position in a sequence represents an attribute value, varies from over 20,000 to over 90,000, making both sequence alignment and data mining heavy computational and memory-intensive tasks.

9. Results

1) Comparison of the Data Mining Results Obtained from Three Sets of Experiments as Well as from Other Related and Selected Previous Work

Table 4 presents the results of Experiments I-III and compares those results with the virus detection results presented in previously published works (see Table 2). In the case of the work by Chen *et al.* [16] only the percentages of correctly detected and incorrectly detected instances were reported (as for J48 method) and in the case of Prabha *et al.* [45] no performance metrics were reported. In the case of Srakaew *et al.* [18] other overall performance metrics such

Table 3. Comparison of three sets of experiments.

Features/Steps	Variable length data mining		Equal length data mining
	Experiment I	Experiment II	Experiment III
Hex to DNA conversion	For the process of pairwise sequence alignment only.	For the processes of data mining and pairwise sequence alignment.	For the processes of multiple sequence alignment, data mining and pairwise sequence alignment.
Multiple sequence alignment for the process of data mining	No	No	Yes
Conversion of variable length sequences into equal 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 number of attributes for the process of data mining	24,565	49,129	93,438
Total number of labels for the process of 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 the 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 (first-level consensuses) from NNge rules	Malicious (hex): 123,338 Non-Malicious (hex): 37,249	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 ($C1_{HEX}$, $C2_{HEX}$) 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)
Total number of unique meta-signatures ($C1_{HEX}$, $C2_{HEX}$)	Five	Five	43
Total number of common meta-signatures ($C1_{HEX}$, $C2_{HEX}$)	Four	Nine	Five

Table 4. Comparison of the results of Experiments I-III with those reported previously for data mining approaches to malware detection reported in the related work section (see Table 2).

Data Mining based Techniques		Correctly Classified Instances (%)	Incorrectly Classified Instances (%)	TP (True Positive) Rate	FP (False Positive) Rate	Precision	Recall	F1 Score
Experiment I (Variable length)		100.00%	0.00%	1	0	1	1	1
Experiment II (Variable length)		0.00%	100.00%	0	1	0	0	0
Experiment III (Equal length)		100.00%	0.00%	1	0	1	1	1
Chen *et al.* [16]—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.* [16]—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 *et al.* [44]	Existing (known)dataset (Average)	95.9752%	4.0248%	0.96	0.094	0.962	0.96	0.959
	New (unknown)dataset (Average)	86.6873%	13.3127%	0.867	0.275	0.872	0.867	0.858
Prabha *et al.* [45]	-	-	-	-	-	-	-	-
Statistical method by Srakaew *et al.* [18]	Reference Set	98.9167%	1.0833%	-	-	-	-	-
	Application Set	95.0477%	4.9523%	-	-	-	-	-
	10-fold cross validation	95.333%	4.667%	-	-	-	-	-
Abstract assembly method by Srakaew *et al.* [18]	Reference Set	99.75%	0.25%	-	-	-	-	-
	Application Set	98.39%	1.661%	-	-	-	-	-
	10-fold cross validation	99.5%	0.5%	-	-	-	-	-

as true positive rate, false positive rate, precision, recall and F1 score were not reported. 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 (see Table 4). Although Experiment II achieved 100% incorrectly classified instances and thus 0% correctly classified instances, the meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) extracted in this experiment successfully detected the JS.Cassandra variants (known P_k and unknown P_x). Meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) extracted in Experiment III were the most effective (~62%) of all followed by the meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) extracted in Experiments I (~55%) and II (43%) (see Section 9-2). The fact that the meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) in DNA format performed better if the DNA sequences were aligned prior to rule mining (Experiment III vs. Experiment II) and extraction is also re-

flected in the results of the work reported by Chen *et al.* [16], where improved classification was observed if J48 classification was performed after a double alignment process.

2) An Evaluation of the State of the Art AVS Products and Meta-Signatures ($C1_{HEX}$ and $C2_{HEX}$) on the Detection of JS.Cassandra Virus and Its Known P_k and Unknown P_x Variants

Table 5 presents the detection ratio obtained using the meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) generated in Experiments I to III and five current state of the art AVSs. The malicious meta-signatures $C1_{HEX4}$ (I), $C1_{HEX9}$ (II), and $C1_{HEX26}$ (III)

Table 5. Detection ratio using five state of the art AVSs and the 14 most effective malicious and 8 non-malicious meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) from Experiments I to III with Clamscan scanner.

Files Scanned	Metrics	Virus Detection 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 $C1_{HEX1}$ (I), $C1_{HEX3}$ (II), non-malicious $C2_{HEX41}$ (III) and $C2_{HEX43}$ (III)	Malicious $C1_{HEX3}$ (I) and $C1_{HEX6}$ (II)	Malicious $C1_{HEX7}$ (II)	Malicious $C1_{HEX4}$ (I), $C1_{HEX9}$ (II), non-malicious $C2_{HEX37}$ (III)	Malicious $C1_{HEX5}$ (III)
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%

Continued

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 $C1_{HEX9}$ (III)**	**Malicious $C1_{HEX15}$ (III)**	**Malicious $C1_{HEX20}$ (III)**	**Malicious $C1_{HEX24}$ (III)**	**Malicious $C1_{HEX26}$ (III)**
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 $C1_{HEX27}$ (III)**	**Non-malicious $C2_{HEX7}$ (I), $C2_{HEX11}$ (II)**	**Non-malicious $C2_{HEX8}$ (I)**	**Non-malicious $C2_{HEX12}$ (II)**	**Non-malicious $C2_{HEX35}$ (III)**
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%

Continued

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%

and the non-malicious meta-signatures $C2_{HEX35}$ (III) and $C2_{HEX37}$ (III) successfully detected all 352 known (P_k) malicious variants of the JS.Cassandra polymorphic virus, where (I), (II) and (III) represent the meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) generated from Experiments I, II and III. None of the five state of the art AVSs fully detected all known (P_k) JS.Cassandra variants. Scan results for AVG, AntiVir and F-Prot AVS products were obtained from an open source online website known as "Gary's Hood" [51]. We used "Gary's Hood" online tool [51] as it allows multiple files to be scanned at the same time adopting the four existing AVS products/scanners (*i.e.* AVG, AntiVir, ClamAV and F-Prot). ESET AVS product was installed on a private machine with Windows based operating system and Clamscan antivirus scanner was installed on a private machine with Linux based (Linux Mint) [52] operating system using their own ClamAV database and using the own generated (.ndb) databases [10] containing the corresponding malicious or non-malicious meta-signature ($C1_{HEX}$ and $C2_{HEX}$). The databases of all the AVS products were up-to-date with the latest updates.

In total, 71 meta-signatures (9 meta-signatures from Experiment I, 14 meta-signatures from Experiment II and 48 meta-signatures from Experiment III) were generated from malicious and non-malicious sequences. All the 71 meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) were scanned/tested against the 352 known (P_k) JS.Cassandra malicious variants, 43 JS.Cassandra non-malicious (P_u) variants and 352 random JavaScript files individually by placing these meta-signatures inside their own generated (.ndb) database [10]. The testing process was conducted using Clamscan antivirus scanner. None of the scans took longer than a second.

Table 5 shows the scan results of some of the effective meta-signatures tested against the malicious, non-malicious and random datasets. Non-malicious $C2_{HEX7}$ (I) and $C2_{HEX11}$ (II) detected 339 out of the 352 (with 96.31% accuracy) JS.Cassandra malicious (P_k) variants, whereas non-malicious $C2_{HEX41}$ (III) and $C2_{HEX43}$ (III) detected 340 out of the 352 (with 96.59% accuracy) JS.Cassandra

malicious (P_k) variants. Malicious $C1_{HEX1}$ (I) and $C1_{HEX3}$ (II) (*i.e.* meta-signature number 1 and meta-signature number 3 of the 71 meta-signatures for the malicious class C1) detected 340 out of the 352 (with 96.59% accuracy) JS.Cassandra malicious (P_k) variants, whereas malicious $C1_{HEX15}$ (III) detected 344 out of the 352 (with 97.73% accuracy) JS.Cassandra malicious (P_k) variants. The reason non-malicious meta-signatures ($C2_{HEX}$) detect malicious variants and malicious meta-signatures ($C1_{HEX}$) detect non-malicious variants is that the malicious and non-malicious variants both contain common functions which lead to the generation of common meta-signatures between the two classes $C1_{HEX}$ and $C2_{HEX}$ (see Table 3 for total number of common meta-signatures).

Malicious $C1_{HEX4}$ (I), $C1_{HEX9}$ (II), $C1_{HEX26}$ (III) along with non-malicious $C2_{HEX35}$ (III) and $C2_{HEX37}$ (III) were the only five meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) that fully detected all 43 non-malicious (P_u) JS.Cassandra variants. These meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) not only detected 352 malicious (P_k) variants successfully but also detected 43 non-malicious (P_u) variants. As noted in Figure A1, non-malicious (P_u) variants still had some polymorphic functions intact inside. All 43 non-malicious (P_u) variants were still executable. The results presented in Table 5 shows that none of the existing AVSs fully detected these non-malicious (P_u) variants as malicious.

The same batch of 71 meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) was once again tested against the 100 unknown (P_x) JS.Cassandra malicious variants by using the own generated (.ndb) database [10]. The testing process was conducted using Clamscan antivirus scanner. 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. Table 6 gives the detection ratio obtained by testing the 71 meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) generated in Experiments I to III and two current state of the art AVSs (ClamAV and Bitdefender Total Security 2017) against the 100 new (P_x) JS.Cassandra variants. ClamAV and Bitdefender Total Security 2017 AVSs had overall accuracies of 85% and 0%, respectively, and meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) from Experiments I-III using

Table 6. Detection ratio using two state of the art AVSs and the 71 meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) obtained from Experiments I to III with Clamscan antivirus scanner.

Files Scanned	Metrics	Virus Detection Method				
		ClamAV	Bitdefender Total Security 2017	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%

Clamscan had overall accuracies of 100%, across all three experiments (see Table 6). Table 6 shows that all 100 (accuracy of 100%) JS.Cassandra unknown (P_x) variants were successfully detected by the Clamscan using the .ndb database augmented with our meta-signatures.

The 71 meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) were tested for false positives. First, any duplicate meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) along with meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) that were six characters or below were removed. In total, 26 meta-signatures (*i.e.* 16 malicious $C1_{HEX}$ and 10 non-malicious $C2_{HEX}$) were removed from the generated (.ndb) database [10]. The remaining 45 meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) were tested against the 352 known (P_k) variants, 43 non-malicious (P_u) variants, 100 new (P_x) variants and 18,123 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 8119 Microsoft Windows files. These files were obtained from a BlogSpot called "contagio" [53].

Figures 2(a)-(c) are the screenshots of the scan results 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 ($C1_{HEX}$ and $C2_{HEX}$). Figure 2(d) shows that only 29 of the 18,123 clean files were

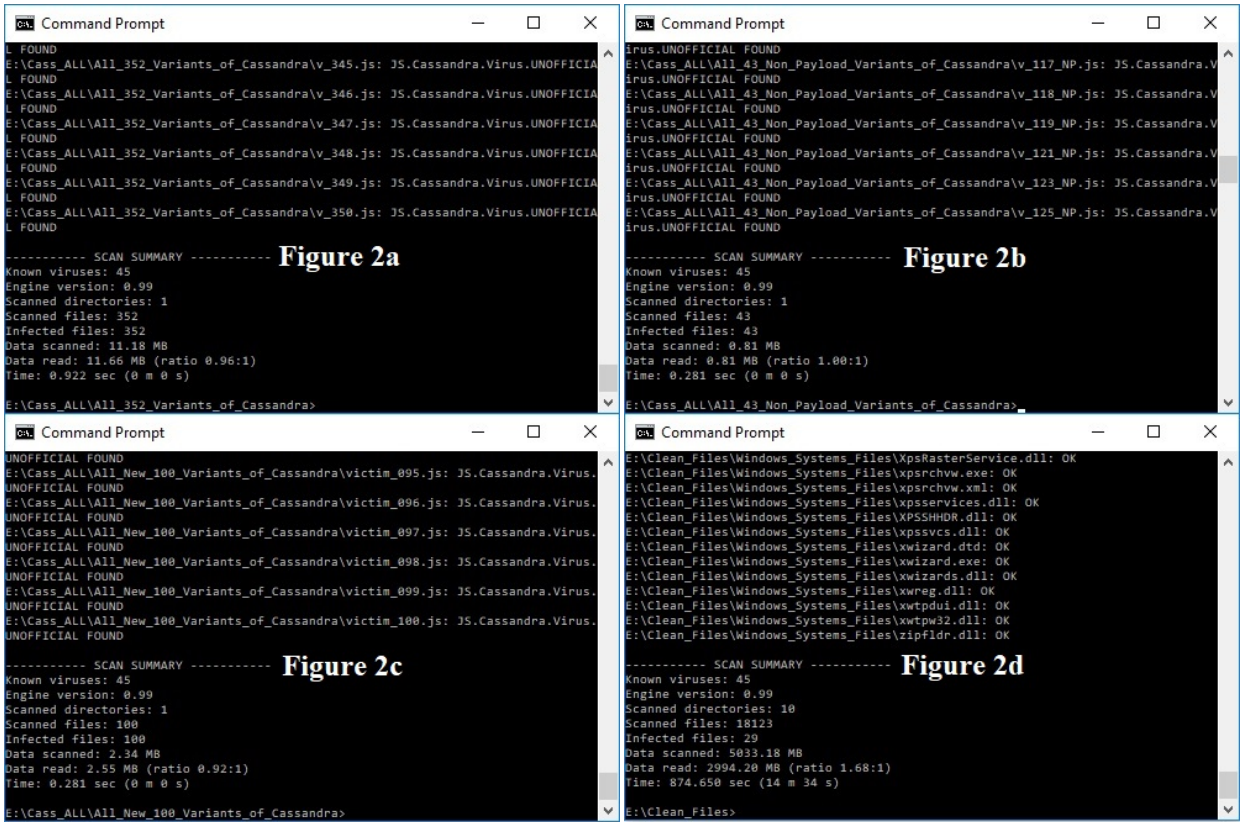

Figure 2. Screenshot of the scan results obtained from Clamscan antivirus scanner for JS.Cassandra variants and clean files using the 45 meta-signatures ($C1_{HEX}$ and $C2_{HEX}$).

detected as false positives (0.159% false positive rate) using the 45 meta-signatures ($C1_{HEX}$ and $C2_{HEX}$), thereby satisfying the false positive rate requisite of 0.1%.

10. Discussions

It was found from the experiments conducted in this paper that Experiment III (equal length data mining technique) gave the highest number of successful meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) in comparison to Experiments I and II (variable length data mining technique). Experiment II gave the lowest number of successful meta-signatures ($C1_{HEX}$ and $C2_{HEX}$). Not only did Experiment III gave the highest number of meta-signatures ($C1_{HEX}$ and $C2_{HEX}$), but it also gave the highest number of effective meta-signatures ($C1_{HEX}$ and $C2_{HEX}$). Moreover, Experiment III generated meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) 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 ($C1_{HEX}$ and $C2_{HEX}$) in comparison to Experiments I and II. In comparison to previous reported work (see Section 4 and Section 5), the syntactic approach to automatic signature generation using NNge successfully has addressed the limitations of previous work by generating signatures in the quickest, simplest and most accurate manner.

In total, 45 out of the 71 overall meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) *i.e.* around 63.38% (33.80% malicious (24/71) and 29.58% non-malicious (21/71)) were effective *i.e.* detected seen (P_s) and unseen (P_u) variants from the two different types of groups (*i.e.* malicious and non-malicious). Specifically, six out of the nine meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) generated from Experiment I (*i.e.* around 66.66% meta-signatures—44.44% malicious (4/9) and 22.22% non-malicious (2/9)) detected seen (P_s) and unseen (P_u) variants belonging to malicious and non-malicious groups (see Table 5). And seven out of the 14 meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) generated from Experiment II (*i.e.* 50% meta-signatures—28.57% malicious (4/14) and 21.43% non-malicious (3/14)) detected seen (P_s) and unseen (P_u) variants belonging to malicious and non-malicious groups (see Table 5). Additionally, 32 out of the 48 meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) generated from Experiment III (*i.e.* 66.66% meta-signatures—35.41% malicious (17/48) and 31.25% non-malicious (15/48)) detected seen (P_s) and unseen (P_u) variants belonging to malicious and non-malicious groups (see Table 5). Only 11 out of the 30 effective meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) obtained from Experiment III are shown in Table 5.

As Experiments I and II were performed using two different representational approaches (*i.e.* hex/DNA) along with Experiment III containing aligned DNA sequences, all with the same (unchanged) instances each time, some of the meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) obtained from the three sets were identical to each other. Malicious $C1_{HEX1}$ (I), $C1_{HEX3}$ (II), non-malicious $C2_{HEX41}$ (III) and $C2_{HEX43}$ (III) share identical meta-signature. On the other hand, malicious $C1_{HEX4}$ (I), $C1_{HEX9}$ (II) and non-malicious $C2_{HEX37}$ (III) share identical meta-signature.

Although Experiment II generated rules with 100% inaccuracy, the overall combined percentage of effective meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) generated from all three sets of experiments was 57.75%. On the other hand, the overall combined percentage of non-effective meta-signatures ($C1_{HEX}$ and $C2_{HEX}$) generated from all three sets of experiments was 42.25%.

The key differences between previous related work [10] [11] [12] [13] and the work presented here are as follows:

1) Previous work adopted left-to-right string matching techniques to find the most optimally-conserved meta-signatures. The work presented in this paper adopts a rule-based or top-down approach that attempts to find underlying patterns.

2) Previous work generated equal length consensuses using sequence alignment techniques, whereas the current work generates variable length consensuses adopting a variable length data mining technique (NNge).

3) 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.

11. Conclusions

In this paper, some of the limitations (discussed in Section 3) of previous work [10] [11] [12] [13] were addressed. The learning task of maximizing true positive rates and minimizing false positive and false negative rates was satisfied. A syntactic approach was investigated and three sets of experiments were conducted which involved various approaches to automatic signature generation using the NNge classifier to generate rules that distinguish between malicious and non-malicious files. The results show that this string-based syntactic approach using an NNge rule generation and subsequent extraction and sequence alignment using SWA can successfully generate signatures ($C1_{HEX}$ and $C2_{HEX}$) which are capable of detecting the known (P_k) (*i.e.* seen and unseen) as well as unknown (P_x) polymorphic variants of the JS.Cassandra virus (see **Table 5**, **Table 6** and **Figure 2**). Remarkably, this research demonstrated that it is possible to detect seen (P_s) (training set), unseen (P_u) (test set) as well as unknown (P_x) variants using the training signatures obtained from a very small proportion (typically 3% and below) of training variants of that test family. A minimal number of training variants was deliberately chosen because the need to detect large numbers of test variants from a minimal number of training variants accurately represents the syntactic malware signature generation approach in the real world.

The use of newly generated novel (P_x) variants differentiates our approach from all previous research that adopts existing malware samples from an online repository. In comparison to the semantic-based approaches as shown in **Table**

1, we explored a purely syntactic approach which produces variable length syntactic viral signatures ($C1_{HEX}$ and $C2_{HEX}$) that detect known (P_k) and unknown (P_x) variants belonging to a polymorphic viral family (JS.Cassandra virus), independently of execution traces, and without needing numerous infections.

In conclusion, the contributions of this paper are listed as follows:

1) Adopting a data mining algorithm, NNge, to generate rule-based signatures automatically from real malware data.

2) 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).

3) Distinguishing malicious variants from non-malicious with the help of rules generated using the data mining algorithm, NNge.

4) Testing the derived rule-based signatures against real malware data and comparing the results to other commercial AVSs.

5) 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) Detecting known P_k (*i.e.* P_s and P_u) and unknown P_x variants of a polymorphic malware family using rule-based signatures (see **Figure 1** for the distribution of polymorphic malware variants).

More work is required to apply the current rule-based approach to more intricate polymorphic as well as metamorphic viruses.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Thompson, G.R. and Flynn, L.A. (2007) Polymorphic Malware Detection and Identification via Context-Free Grammar Homomorphism. *Bell Labs Technical Journal*, **12**, 139-147. https://doi.org/10.1002/bltj.20256

[2] Harley, D. and Lee, A. (2007) The Root of All Evil?—Rootkits Revealed. http://eset.version-2.sg/softdown/manual/Whitepaper-Rootkit_Root_Of_All_Evil.pdf

[3] Mishra, U. (2010) An Introduction to Virus Scanners. https://ssrn.com/abstract=1916673

[4] Rad, B.B., Masrom, M. and Ibrahim, S. (2011) Evolution of Computer Virus Concealment and Anti-Virus Techniques: A Short Survey. *International Journal of Computer Science Issues*, **8**, 113-121.

[5] Vinod, P., Laxmi, V. and Gaur, M.S. (2009) Survey on Malware Detection Methods. *Proceedings of the* 3rd *Hackers' Workshop on Computer and Internet Security*, Kanpur, 17-19 March 2009, 74-79.

[6] Cesare, S. and Xiang, Y. (2010) A Fast Flowgraph Based Classification System for Packed and Polymorphic Malware on the Endhost. *Proceedings of the* 24th *IEEE International Conference on Advanced Information Networking and Applications*,

Perth, 20-23 April 2010, 721-728.

[7] Cesare, S., Xiang, Y. and Zhou, W. (2013) Malwise—An Effective and Efficient Classification System for Packed and Polymorphic Malware. *IEEE Transactions on Computers*, **62**, 1193-1206. https://doi.org/10.1109/TC.2012.65

[8] Rad, B.B., Masrom, M. and Ibrahim, S. (2012) Camouflage in Malware: From Encryption to Metamorphism. *International Journal of Computer Science and Network Security*, **12**, 74-83.

[9] Cabrera, A. and Calix, R.A. (2016) On the Anatomy of the Dynamic Behavior of Polymorphic Viruses. *Proceedings of the* 2016 *International Conference on Collaboration Technologies and Systems*, Orlando, 31 October-4 November 2016, 424-429. https://doi.org/10.1109/CTS.2016.0081

[10] Naidu, V. and Narayanan, A. (2016) A Syntactic Approach for Detecting Viral Polymorphic Malware Variants. In: Chau, M., *et al.*, Eds., *Pacific Asia Workshop on Intelligence and Security Informatics* (*PAISI*), LNCS 9650, Springer, Berlin, 146-165.

[11] Naidu, V. and Narayanan, A. (2016) The Effects of Using Different Substitution Matrices in a String-Matching Technique for Identifying Viral Polymorphic Malware Variants. *Proceedings of the IEEE Congress on Evolutionary Computation*, Vancouver, 24-29 July 2016, 2903-2910.

[12] Naidu, V. and Narayanan, A. (2016) Needleman-Wunsch and Smith-Waterman Algorithms for Identifying Viral Polymorphic Malware Variants. *Proceedings of the* 14th *IEEE International Conference on Dependable, Autonomic and Secure Computing*, Auckland, 8-12 August 2016, 326-333.

[13] 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*, **8**, 296-327. https://doi.org/10.4236/jis.2017.84020

[14] Gold, E. (1967) Language Identification in the Limit. *Information and Control*, **5**, 447-474. https://doi.org/10.1016/S0019-9958(67)91165-5

[15] Xinguang, T., Miyi, D., Chunlai, S. and Xin, L. (2009) Detecting Network Intrusions by Data Mining and Variable-Length Sequence Pattern Matching. *Journal of Systems Engineering and Electronics*, **20**, 405-411.

[16] Chen, Y., Narayanan, A., Pang, S. and Tao, B. (2012) Malicious Software Detection Using Multiple Alignment and Data Mining. *Proceedings of* 26*th International Conference on Advanced Information Networking and Applications*, Fukuoka, 26-29 March 2012, 8-14.

[17] Narayanan, A. and Chen, Y. (2013) Bio-Inspired Data Mining: Treating Malware Signatures as Biosequences. Computing Research Repository (CoRR), 1-33.

[18] Srakaew, S., Piyanuntcharatsr, W. and Adulkasem, S. (2015) On the Comparison of Malware Detection Methods Using Data Mining with Two Feature Sets. *International Journal of Security and Its Applications*, **9**, 293-318. https://doi.org/10.14257/ijsia.2015.9.3.23

[19] Rieck, K., Holz, T., Willems, C., Düssel, P. and Laskov, P. (2008) Learning and Classification of Malware Behavior. In: Zamboni, D., Ed., *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, LNCS 5137, Springer, Berlin, 108-125.

[20] Singhal, P. and Raul, N. (2012) Malware Detection Module Using Machine Learning Algorithms to Assist with Centralized Security in Enterprise Networks. *International Journal of Network Security & Its Applications*, **4**, 61-67. https://doi.org/10.5121/ijnsa.2012.4106

[21] Naidu, V. and Narayanan, A. (2014) Further Experiments in Biocomputational Structural Analysis of Malware. 10*th International Conference on Natural Computation*, Xiamen, 19-21 August 2014, 605-610.

[22] Cendrowska, J. (1988) PRISM: An Algorithm for Inducing Modular Rules. *International Journal of Man-Machine Studies*, **27**, 349-370. https://doi.org/10.1016/S0020-7373(87)80003-2

[23] Witten, I.H. More Data Mining with Weka. Class 3, Lesson 1, Decision Trees and Rules. University of Waikato, Hillcrest. http://www.cs.waikato.ac.nz/

[24] Koklu, M., Kahramanli, H. and 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*, Bari, 27-29 May 2015, 1991-1998.

[25] Wespi, A., Dacier, M. and Debar, H. (1999) An Intrusion-Detection System Based on the Teiresias Pattern-Discovery Algorithm. IBM Thomas J. Watson Research Division.

[26] Kreibich, C. and Crowcroft, J. (2004) Honeycomb: Creating Intrusion Detection Signatures Using Honeypots. *ACM SIGCOMM Computer Communication Review*, **34**, 51-56. https://doi.org/10.1145/972374.972384

[27] Kim, H.-A. and Karp, B. (2004) Autograph: Toward Automated, Distributed Worm Signature Detection. *USENIX Security Symposium*, San Diego, 286.

[28] Singh, S., Estan, C., Varghese, G. and Savage, S. (2004) Automated Worm Fingerprinting. OSDI, 4.

[29] Newsome, J., Karp, B. and Song, D. (2005) Polygraph: Automatically Generating Signatures for Polymorphic Worms. *IEEE Symposium on Security and Privacy*, Oakland, 8-11 May 2005, 226-241.

[30] Yegneswaran, V., Giffin, J.T., Barford, P. and Jha, S. (2005) An Architecture for Generating Semantic-Aware Signatures. 14*th USENIX Security Symposium*, Baltimore, 1-5 August 2005, 97-112.

[31] Wang, K., Cretu, G. and Stolfo, S.J. (2005) Anomalous Payload-Based Worm Detection and Signature Generation. In: *International Workshop on Recent Advances in Intrusion Detection*, Springer, Berlin, 227-246.

[32] Li, Z., Sanghi, M., Chen, Y., Kao, M.-Y. and Chavez, B. (2006) Hamsa: Fast Signature Generation for Zero-Day Polymorphic Worms with Provable Attack Resilience. *IEEE Symposium on Security and Privacy*, Berkeley, 21-24 May 2006, 15.

[33] Cui, W., Peinado, M., Wang, H.J. and Locasto, M.E. (2007) Shieldgen: Automatic Data Patch Generation for Unknown Vulnerabilities with Informed Probing. *IEEE Symposium on Security and Privacy*, Oakland, 20-23 May 2007, 252-266.

[34] Xie, Y., Yu, F., Achan, K., Panigrahy, R., Hulten, G. and Osipkov, I. (2008) Spamming Botnets: Signatures and Characteristics. *ACM SIGCOMM Computer Communication Review*, **38**, 171-182. https://doi.org/10.1145/1402946.1402979

[35] Coull, S.E. and Szymanski, B.K. (2008) Sequence Alignment for Masquerade Detection. *Computational Statistics & Data Analysis*, **52**, 4116-4131. https://doi.org/10.1016/j.csda.2008.01.022

[36] Scheirer, W. and Chuah, M.C. (2008) Syntax vs. Semantics: Competing Approaches to Dynamic Network Intrusion Detection. *International Journal of Security and Networks*, **3**, 24-35. https://doi.org/10.1504/IJSN.2008.016199

[37] Wurzinger, P., Bilge, L., Holz, T., Goebel, J., Kruegel, C. and Kirda, E. (2009) Automatically Generating Models for Botnet Detection. In: *European Symposium on*

Research in Computer Security, Springer, Berlin, 232-249.

[38] Rieck, K., Schwenk, G., Limmer, T., Holz, T. and Laskov, P. (2010) Botzilla: Detecting the Phoning Home of Malicious Software. *Proceedings of the ACM Symposium on Applied Computing*, Sierre, 22-26 March 2010, 1978-1984.

[39] Zhao, Y., Tang, Y., Wang, Y. and Chen, S. (2013) Generating Malware Signature Using Transcoding from Sequential Data to Amino Acid Sequence. *International Conference on High Performance Computing and Simulation*, Helsinki, 1-5 July 2013, 266-272.

[40] Rossow, C. and Dietrich, C.J. (2013) Provex: Detecting Botnets with Encrypted Command and Control Channels. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, Berlin, 21-40. https://doi.org/10.1007/978-3-642-39235-1_2

[41] Rafique, M.Z. and Caballero, J. (2013) Firma: Malware Clustering and Network Signature Generation with Mixed Network Behaviors. In: *International Workshop on Recent Advances in Intrusion Detection*, Springer, Berlin, 144-163.

[42] Ki, Y., Kim, E. and Kim, H.K. (2015) A Novel Approach to Detect Malware Based on API Call Sequence Analysis. *International Journal of Distributed Sensor Networks*, **2015**, Article No. 4. https://doi.org/10.1155/2015/659101

[43] Kirat, D. and Vigna, G. (2015) Malgene: Automatic Extraction of Malware Analysis Evasion Signature. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Denver, 12-16 October 2015, 769-780.

[44] Kumar, V. and Mishra, S.K. (2013) Detection of Malware by Using Sequence Alignment Strategy and Data Mining Techniques. *International Journal of Computer Applications*, **61**, 16-19.

[45] Prabha, A.P.M. and Kavitha, P. (2012) Malware Classification through HEX Conversion and Mining. *Proceedings of International Conference on E-Governance & Cloud Computing Services*, December 2012, 6-12.

[46] Salzberg, S. (1991) A Nearest Hyperrectangle Learning Method. *Machine Learning*, **6**, 277-309. https://doi.org/10.1007/BF00114779

[47] Panda, M. and Patra, M.R. (2009) Semi-Naïve Bayesian Method for Network Intrusion Detection System. In: Leung, C.S., *et al.*, Eds., 16*th International Conference on Neural Information Processing*, Part I, LNCS 5863, Springer, Berlin, 614-621.

[48] Zaharie, D., Perian, L. and Negru, V. (2011) A View inside the Classification with Non-Nested Generalized Exemplars. IADIS European Conference Data Mining.

[49] Martin, B. (1995) Instance-Based Learning: Nearest Neighbour with Generalisation. Working Paper Series 95/18 Computer Science, Hamilton, University of Waikato, Hillcrest, 90.

[50] Wettschereck, D. and Dietterich, T.G. (1995) An Experimental Comparison of the Nearest-Neighbor and Nearest-Hyperrectangle Algorithms. *Machine Learning*, **19**, 1-25. https://doi.org/10.1007/BF00994658

[51] Gary's Hood (2016) Online Virus Scanner. http://www.garyshood.com/virus/

[52] Linux Mint (2016) From Freedom Came Elegance. https://www.linuxmint.com/

[53] Contagio (2013) 16,800 Clean and 11,960 Malicious Files for Signature Testing and Research. http://contagiodump.blogspot.co.nz/2013/03/16800-clean-and-11960-malicious-files.html

[54] Oracle VM VirtualBox (2016) VirtualBox. https://www.virtualbox.org/

[55] JS. Cassandra by Second Part to Hell (2015) rRlF#4 (Redemption). http://spth.virii.lu/rrlf4/rRlf.13.html

[56] Viruses (2004) Second Part to Hell's Artworks VIRUSES. http://spth.virii.lu/Cassandra-testset.rar

[57] ClamavNet (2015) ClamAV Is an Open Source Antivirus Engine for Detecting Trojans, Viruses, Malware & Other Malicious Threats. https://www.clamav.net/

[58] Weka 3: Data Mining Software in Java (2016) Weka 3—Data Mining with Open Source Machine Learning Software in Java. http://www.cs.waikato.ac.nz/ml/weka/

[59] JAligner (2010) JAligner: Java Implementation of the Smith-Waterman Algorithm for Biological Sequence Alignment—SourceForge. http://jaligner.sourceforge.net/

[60] VirusTotal (2016) Free Online Virus, Malware and URL Scanner. https://www.virustotal.com/

[61] Katoh, K., Misawa, K., Kuma, K. and Miyata, T. (2002) MAFFT: A Novel Method for Rapid Multiple Sequence Alignment Based on Fast Fourier Transform. *Nucleic Acids Research*, **30**, 3059-3066. https://doi.org/10.1093/nar/gkf436

[62] Katoh, K. and Standley, D.M. (2013) MAFFT Multiple Sequence Alignment Software Version 7: Improvements in Performance and Usability. *Molecular Biology and Evolution*, **30**, 772-780. https://doi.org/10.1093/molbev/mst010

[63] MAFFT Alignment and NJ/UPGMA Phylogeny (2016) MAFFT Version 7. http://mafft.cbrc.jp/alignment/server/index.html

Appendix

A1. Experiment I

More details referring to the steps involved in this experiment can be found in the previous work [13] and **Figure A1**. Our method for Experiment I consists of six steps (see **Figure A1**).

Hex dump extraction (Step-1) and testing (Step-6) were undertaken on a stand-alone system to prevent possible unintended infection of other systems. Downloading of polymorphic malware (and seen P_s as well as unseen P_u variants) was performed using "Oracle VM VirtualBox" [54] (an x86 software package with virtualization capability) with a pre-installed Linux-based (Linux Mint) [52] operating system image. Due to possible security sensitivity, some of the methods below (Step-1 and Step-6) are not described in detail, especially details concerning generating hex dumps from polymorphic malware. Interested readers are requested to contact the corresponding author, using their academic email addresses, for further information.

23 withheld variants (P_s and P_u) were selected for Experiment I. A CRC32b hash value was generated for each of these 23 withheld variants and no duplicates were found, indicating that they were unique. The percentage of training to

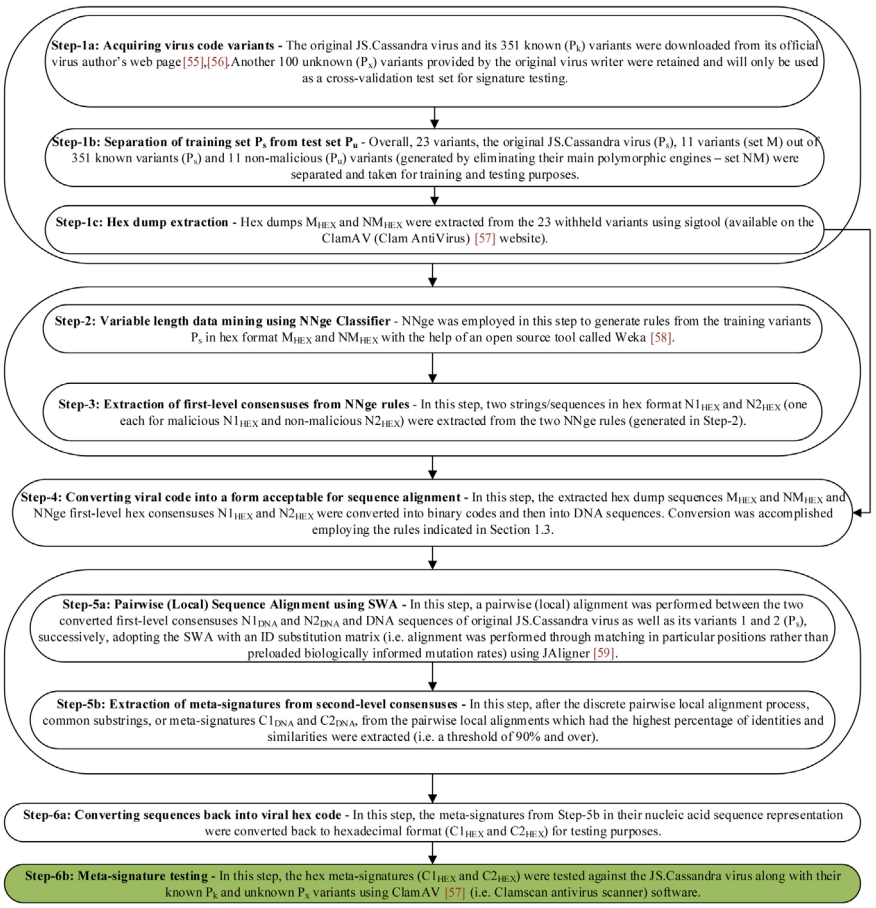

Figure A1. Our method for Experiment I comprising of six steps.

test ratio (P_s to P_u) in P_k for malicious JS.Cassandra virus is 3.125% (11:352). A severely reduced proportion of training to test samples was used to reflect the current difficulty in detecting signatures that generalize from a small, previously encountered set of known (P_k) variants to a potentially infinite set of new (P_x) variants.

All 23 withheld variants (P_s and P_u) were checked using the "VirusTotal" [60] (a free online scanner for malware) website to confirm and validate that malicious functionality was maintained in the 11 malicious variants *i.e.* set M (along with the original JS.Cassandra virus) and eliminated in the 11 non-malicious/non-payload variants (set NM). "VirusTotal" [60] employs 56 well-known AVSs and so provides good assurance that our manual code alterations for non-malicious variants were effective. The scan results of the 23 variants obtained from "VirusTotal" website indicated that on average 35.06% and 71.43% of the 56 AVS products successfully detected the 11 malicious variants (set M) and original JS.Cassandra virus, respectively. On average, 0.00% and 0.714% of the 56 AVS products successfully detected the 11 non-malicious variants (set NM). Only four out of the 56 AVSs detected a few of the non-malicious variant files as malicious, as some of the non-malicious variant files still had their polymorphic functions in place.

For the process of data mining using NNge (Step-2), the variable length hex sequences were converted into equal length sequences by constraining the shorter sequences to have a length equal to the longest sequence by adding the letter "x" at the end of each short sequence. Lower case "x" was added as the hex sequences were represented in lower cases. An ARFF (Attribute-Relation File Format) file was created which contained the hex dump sequences (M_{HEX} and NM_{HEX}) for the 22 JS.Cassandra variants. The 23rd variant was not included in the ARFF file since it will only be used in Step-4 and Step-5 for the process of pairwise sequence alignment.

In total, the ARFF file consisted of 24,565 attributes (one attribute per position) and two classes (malicious and non-malicious). The NNge classifier was trained on the full dataset. Two NNge rules (one for each class) were generated with a data fitting 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*: *pos*1 *in* {2, 6} ^ *pos*2 *in* {0, 3} ^ *pos*3 *in* {6, 7} ^ *pos*4 *in* {*a*, *b*, 1, 2, 3, 7, 9} ^ *pos*5 *in* {6, 7} ^ *pos*6 *in* {*a*, *e*, 1, 2, 3, 5, 6, 7, 9} ^ *pos*7 *in* {6, 7} ... *and so on.*

***Non-Malicious* (*nm*)**—*class nm IF*: *pos*1 *in* {2, 6, 7} ^ *pos*2 *in* {*f*, 6} ^ *pos*3 *in* {2, 6, 7} ^ *pos*4 *in* {*f*, 1, 5} ^ *pos*5 *in* {2, 6, 7} ^ *pos*6 *in* {*e*, 0, 2} ^ *pos*7 *in* {2, 6, 7} ... *and so on.*

The best instance to represent the process of rule extraction (Step-3) using the above-mentioned rule is, for (m) the first substring at pos 1 becomes the first substring in the new NNge rule extracted string, and so on:

"260367*ab*1237967*ae*123567967...". The length of the malicious string ($N1_{HEX}$) was 123,338 hex characters, whereas, the length of the non-malicious string ($N2_{HEX}$) was 37,249 hex characters. Only hex data (by excluding the letter "x") from the two NNge rules were extracted.

After conversion (Step-4), six discrete pairwise alignments (Step-5a) were first conducted (sequence 1 with sequence 2, sequence 3 with sequence 4, etc.). The equal combination of gap open (*i.e.* 10) and gap extend (*i.e.* 1) penalty (as used in [10] [11] [12] [13]) was used during the processes of pairwise sequence alignment. Step-5b resulted in nine common substrings ($C1_{DNA}$ and $C2_{DNA}$) from the six pairwise local alignments. One of the nine meta-signatures, with a sequence length 50, generated from one of the six pairwise alignments, is shown below in nucleic acid representation:

CAATCAAGGCGCGCTCCCGTGCGATCTCACGGCCGTTCGTGAGAAC GATC

In Step-6a and Step-6b, the nine DNA meta-signatures were first converted into hex ($C1_{HEX}$ and $C2_{HEX}$) and then later tested against the JS.Cassandra viral variants (P_k and P_x) using clamscan scanner. One of the nine hex meta-signatures, with a sequence length 25, is shown below in hex representation:

4342999*d*5*b*98*dd*1*a*5*bdb*8818*d*

A2. Experiment II

The same procedure as Experiment I was used along with the same JS.Cassandra (training) variants, with the only difference being that the variants (M_{HEX} and NM_{HEX}) were converted into DNA format (M_{DNA} and NM_{DNA}) prior to NNge rule generation. The conversion to DNA format was undertaken as normal using the DNA representational method as detailed in Section 6. Our method for Experiment II consists of six steps (see **Figure A2**).

In this step (Step-3), as for Experiment I, equal length sequences were created by adding the letter "X" at the end of each sequence to the length of the longest variant. Upper case "X" was added as the DNA sequences were represented in upper cases. In total, the resultant 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) were generated in under seven minutes. Partial segments of the two NNge (DNA) rules are shown below:

***Malicious* (*M*)**—*class M IF*: *pos*1 *in* {*A*, *C*} ^ *pos*2 *in* {*G*} ^ *pos*3 *in* {*A*} ^ *pos*4 *in* {*A*, *T*} ^ *pos*5 *in* {*C*} ^ *pos*6 *in* {*T*, *G*} ^ *pos*7 *in* {*A*, *G*, *C*} ^ *pos*8 *in* {*T*, *G*, *C*} ... *and so on.*

***Non-Malicious* (*NM*)**—*class NM IF*: *pos*1 *in* {*A*, *C*} ^ *pos*2 *in* {*T*, *G*} ^ *pos*3 *in* {*T*, *C*} ^ *pos*4 *in* {*T*, *G*} ^ *pos*5 *in* {*A*, *C*} ^ *pos*6 *in* {*T*, *G*} ^ *pos*7 *in* {*A*, *T*, *C*} ^ *pos*8 *in* {*T*, *C*} ... *and so on.*

In this step (Step-4), two strings (first-level consensuses—$N1_{DNA}$ and $N2_{DNA}$)

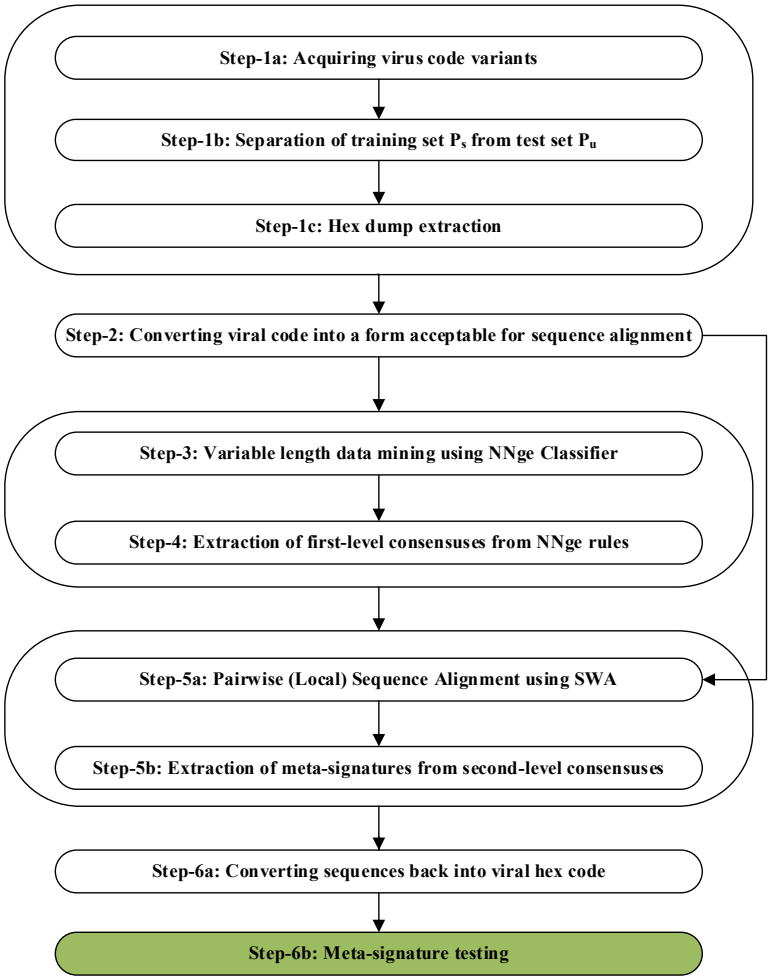

Figure A2. Our method for Experiment II comprising of six steps.

in DNA format were extracted in the same way as for Experiment I from these two NNge rules and the substrings in each position were concatenated as illustrated here for the Malicious class: "*ACGAATCTGAGCTGC...*".

The sequence length of the malicious NNge DNA string ($N1_{DNA}$) was 132,103 bases, whereas the sequence length of non-malicious NNge DNA string ($N2_{DNA}$) was 41,670 bases. In this step (Step-5a), pairwise local alignment was then performed using SWA and the ID matrix with same gap penalties in a process similar to that described for Experiment I (Step-5a). In total, as in Experiment I, six pairwise alignments were performed in this step (Step-5a).

Overall, 14 common substrings (*i.e.* meta-signatures—$C1_{DNA}$ and $C2_{DNA}$) were obtained in this step (Step-5b) from the six pairwise local alignments. One of the 14 meta-signatures, with a sequence length 59, generated from one of the six pairwise alignments, is shown below in nucleic acid representation:

ACAGGAAGGCCTTCAATCAAGGCGCGCTCCCGTGCGATCTCACGGC CGTTCGTGAGAAC

In Step-6a and Step-6b, the 14 DNA meta-signatures were first converted into hex ($C1_{HEX}$ and $C2_{HEX}$) and then later tested against the JS.Cassandra viral va-

riants (P_k and P_x) using clamscan scanner. One of the 14 hex meta-signatures, with a sequence length 28, is shown below in hex representation:

28297*d*0*d*0*a*66756*e*6374696*f*6*e*20

A3. Experiment III

This experiment takes a different approach from Experiments I and II to dealing with the need for equal length sequences in order to generate rules using an equal length data mining approach. Multiple sequence alignment is undertaken prior to NNge rule generation to convert the variable length sequences (M_{DNA} and NM_{DNA}) into equal length sequences (M_E and NM_E) by inserting gaps (**Figure A3**). In Step-3, a multiple sequence alignment using MAFFT [61] [62] [63] was conducted on the 22 variable length DNA sequences (M_{DNA} and NM_{DNA}). The final alignment file had overall sequence identity and similarity percentages of 38.35% and 65.13%, respectively. All the gaps introduced at this stage were substituted by the letter "X". Upper case "X" was added as the DNA sequences were represented in upper cases.

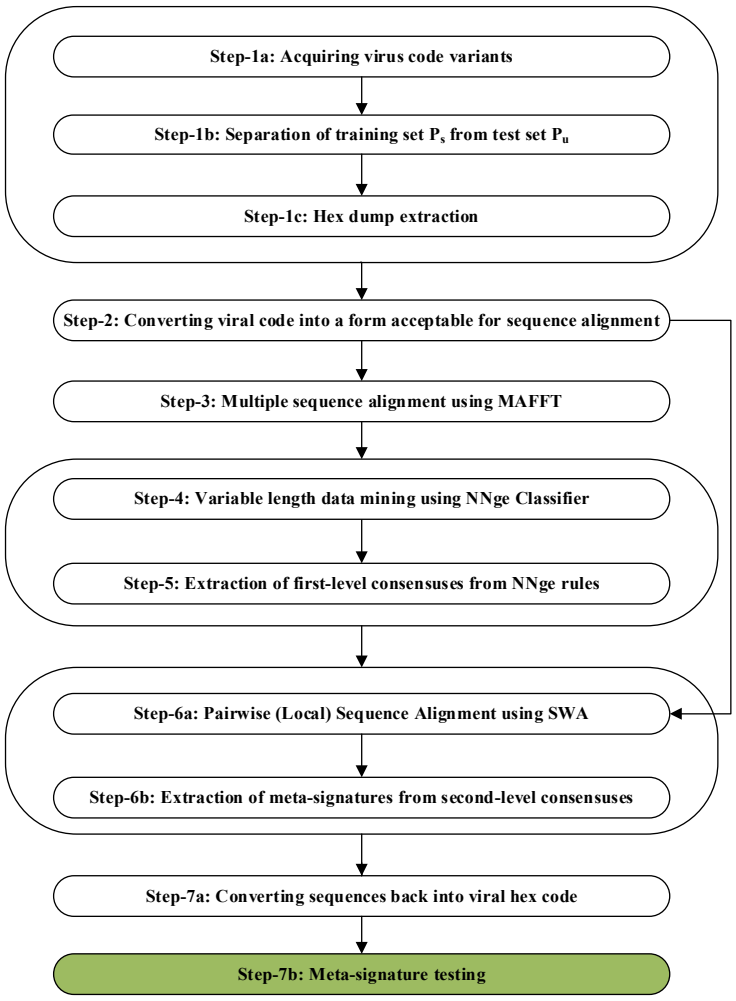

Figure A3. Our method for Experiment III comprising of seven steps.

In Step-4, the same NNge classification was undertaken using Weka. The data was converted into Weka's ARFF file format and consisted of 93,438 attributes and two classes malicious and non-malicious. 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*: *pos*1 *in* {*A*, *X*} ^ *pos*2 *in* {*G*, *X*} ^ *pos*3 *in* {*A*, *X*} ^ *pos*4 *in* {*A*, *X*} ^ *pos*5 *in* {*C*, *X*} ^ *pos*6 *in* {*T*, *G*, *X*} ^ *pos*7 *in* {*A*, *G*, *X*} ^ *pos*8 *in* {*T*, *G*, *C*, *X*} ^ *pos*9 *in* {*C*, *X*} ^ *pos*10 *in* {*T*, *G*, *X*} ... *and so on.*

***Non-Malicious* 1 (*NM*1)**—*class NM IF*: *pos*1 *in* {*X*} ^ *pos*2 *in* {*X*}… ^ *pos*96 *in* {*T*, *X*} ^ *pos*97 *in* {*A*, *X*} ^ *pos*98 *in* {*G*, *X*} ^ *pos*99 *in* {*A*, *X*} ^ *pos*100 *in* {*A*, *X*} ^ *pos*101 *in* {*C*, *X*} ^ *pos*102 *in* {*T*, *G*, *X*} ^ *pos*103 *in* {*G*, *C*, *X*} ... *and so on.*

***Non-Malicious* 2 (*NM*2)**—*class NM IF*: *pos*1 *in* {*X*} ^ *pos*2 *in* {*X*} … ^ *pos*1294 *in* {*X*} ^ *pos*1295 *in* {*C*} ^ *pos*1296 *in* {*A*} ^ *pos*1297 *in* {*G*} ^ *pos*1298 *in* {*T*} ^ *pos*1299 *in* {*C*} ^ *pos*1300 *in* {*A*} ^ *pos*1301 *in* {*T*} ... *and so on.*

In Step-5, three strings (first-level consensuses—$N1_{DNA}$ and $N2_{DNA}$) 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 ($N1_{DNA}$) was 161,495 bases, whereas, the sequence lengths of the non-malicious DNA strings ($N2_{DNA}$) were 59,740 bases (*NM*1) and 11,860 bases (*NM*2).

Next, in Step-6a, local pairwise sequence alignment between these DNA sequences (first-level consensuses—$N1_{DNA}$ and $N2_{DNA}$) extracted from each of the NNge rules and the three malicious JS.Cassandra variants (P_s) in DNA format was performed one by one using SWA and the ID matrix, as per Experiments I and II. In this step (Step-6b), common substrings that are the meta-signatures ($C1_{DNA}$ and $C2_{DNA}$) for JS.Cassandra were extracted from the nine second-level consensuses generated from the process of nine pairwise local alignments. In total, 48 meta-signatures ($C1_{DNA}$ and $C2_{DNA}$) were obtained. The meta-signature of sequence length 88 obtained from one of the nine pairwise alignments is shown below in its nucleic acid representation:

GGAAGTGCTAGCGTTCTCCCGTGCGCAAGGACATCCGACCTCACGG
AAGTGCTAGCGACCGTGCGCACGTTCGTCAGGAAGGCAGGGA

In Step-7a and Step-7b, the 48 DNA meta-signatures were first converted into hex ($C1_{HEX}$ and $C2_{HEX}$) and then later tested against the JS.Cassandra viral variants (P_k and P_x) using clamscan scanner. One of the 48 hex meta-signatures, with a sequence length 22, is shown below in hex representation:

292*c*283538322*f*36292*c*28