Detecting Zero-day Polymorphic Worms with Jaccard Similarity Algorithm

Malak Abdullah I. Almarshad[1], Mohssen M. Z. E. Mohammed[1] and Al-Sakib Khan Pathan[2]

[1]College of Computer and Information Sciences, Al-Imam Muhammad Ibn Saud Islamic University, Saudi Arabia
[2]Department of Computer Science and Engineering, Southeast University, Bangladesh
malakalmarshad@gmail.com, m_zin44@hotmail.com, and spathan@ieee.org

***Abstract*:** Zero-day polymorphic worms pose a serious threat to the security of Mobile systems and Internet infrastructure. In many cases, it is difficult to detect worm attacks at an early stage. There is typically little or no time to develop a well-constructed solution during such a worm outbreak. This is because the worms act only to spread from node to node and they bring security concerns to everyone using Internet via any static or mobile node. No system is safe from an aggressive worm crisis. However, many of the characteristics of a worm can be used to defeat it, including its predictable behavior and shared signatures. In this paper, we propose an efficient signature generation method based on string similarity algorithms to generate signatures for Zero-day polymorphic worms. Then, these signatures are practically applied to an Intrusion Detection System (IDS) to prevent the network from such attacks. The experimental results show the efficiency of the proposed approach compared to other existing mechanisms.

***Keywords*:** Algorithm, Attack, Internet, Matching, Mobile, Polymorphic, Signature, String, Worm, Zero-day**.**

1. Introduction

Computer worms are considered to be a major threat to any type of network. As nowadays, thousands of users use mobile devices like laptops, notebooks, tablet computers, smartphones and so on to browse Internet or even to do sensitive electronic transactions, the stakes are indeed high. Unlike ordinary viruses or other types of malicious programs [1], worms [2], [3], [4], [5], [6], [7] have the ability to replicate themselves without any human intervention. Worms that use the Internet as a propagation media, such as Code Red, Sapphire, and MS Blaster [8] spread very quickly and can cover a thousand of hosts in a matter of minutes or even seconds.

Polymorphic worms [9], [10] are special kind of worms that change their appearance dynamically by scrambling their payload. Each instance of this type of worm looks different from the previous ones, but works exactly the same way. This feature makes detection and prevention from such a worm hard to implement and maintain. Some of the polymorphic worms even use encryption to hide their real intent; which makes the situation even worse for the detection systems. Since the invention of Morris worm [11], which gained huge media attention during that time, fighting worms has become an open research area for experts to dig in. Malicious codes and worms cause huge harm not only for individuals, but also for organizations and government resources. In fact, worm attacks against critical government institutions, security services, military infrastructures, intelligence agencies could have enormous impact on a country's homeland security.

There are various approaches to tackling worm attacks. In the traditional techniques, the security administrator takes each worm instance, studies its signature, and stores it in the Intrusion Detection System (IDS) database manually, so the IDS can identify the worm later by filtering the network traffic. However, this traditional method is often ineffective and unreliable against the fast separation of modern Zero-day worms[1]. In fact, they do not fit well against obfuscated worms, they do not use program semantics, and they require human intervention to update signature database effectively. On the other hand, some modern security systems focus on automating the detection and prevention. This can overcome the delay and mistakes caused by human beings. Because of the significance of the issue, in the recent years, quite a good number of efforts have been made by researchers to gain in the race between worm's spreading and detection.

Nazario [8] classifies the detection and defense methods as follows:

Detection mechanisms include traffic analysis, honeypots and dark (black hole) network monitors and signature-based detection.

Defense mechanisms are classified as host-based defense, firewall and network, proxy-based and attaching the worm network.

Indeed, one of the most critical fields in computer science is network security. Even within the area of network security, polymorphic worm attack defense is considered as one of the most challenging fields. Following are the serious challenges in this area that have been discovered during our survey:

*Lack of general knowledge***:** Information about worms is often the monopoly of big anti-malware companies. They reveal what would serve their own interest and cover up what could be used against them. Crumbs are only left to the rest of the researchers.

*Attack is cheap, while defense is expensive***:** A teenager can launch a fast corrupting worm, while a complete team of professionals would be needed to defeat it.

*Lack of measurements***:** There is no universally accepted standard to assess the quality of the security solutions. The quality issue is often considered based on a particular context. While false positive[2] and false negative[3] ratios are used in most of the research works as common metrics, many other factors can have considerable effect on the final evaluation like for instance, the type of worm used, the size of the normal traffic, signature width, and so on.

*Fuzziness in the literature***:** Many publications (as research papers) in this area tend to describe their means in an

[1]Zero-day worm means a new kind of worm that has not passed through an IDS before - thus it is completely new to that IDS.

[2]Number of incorrectly identified unharmful code as malicious code. Detector generates alarm when there is no real attack.

[3]Number of incorrectly missed malicious code. Detector fails to detect real attack and no alarm is generated.

abstract or general way. Detailed or clear defense mechanism is a hard thing to find.

*Deployment difficulty***:** Most of the detection and defense systems are proven theoretically and not evaluated on the Internet with real-life scenarios. Deployment difficulty comes from the sensitivity of this field. Of course, we are not allowed to test a harmful worm in an open network!

Considering all these issues mentioned above, we are motivated to propose a practical mechanism to defend against Zero-day polymorphic worms. Our method primarily uses Jaccard similarity algorithm to extract signature from the worm instances without finding exact match. In the testing level, Jaccard algorithm can return similarity percentage between a worm signature and a suspicious packet. Due to the similarity algorithm's nature, since it is not based on the exact match, this would allow maintaining different levels of security. Network administrator can control the security level from *low* to *strict*. As the security level chosen by the administrator reflects the network policy, sensitive networks like bank or government LANs (Local Area Networks) can guarantee high level of security. Individuals' networks that do not have very sensitive information, can allow more network traffic to flow with relatively lower security level. The proposed mechanism is *network-based* and *host-based* at the same time (i.e., platform-independent) and it is a signature-based one. Our main focus is the efficiency and accuracy of the proposed string similarity algorithm in its ability to catch most of the Zero-day polymorphic worm instances.

The rest of the paper is organized as follows: Section 2 presents the related works in this area that motivated us to devise our mechanism, Section 3 gives some background information about the string similarity algorithms, Section 4 presents the design and implementation of our scheme. Experimental results and comparative analysis based on real-life cases are presented in Section 5 and finally, Section 6 concludes the paper with possible future works.

2. Related Works and Motivation

Polygraph [12] is a content-based automated signature generator for polymorphic worms. It is deployed at network level. Polygraph categorizes worms' signature into three classes: set of tokens, sequences of tokens, and weighted set of tokens. Three algorithms are proposed to generate signatures for each class. This gives the system the ability to deal with worms that use payload encryption. The results of the work show that it can generate high-quality signatures even under critical conditions like the presence of unclassified noise flows, or the presence of multiple kinds of worms at the same time. Polygraph can generate quality signature for noise flows under 80%. However, the negative side of this is that multiple fixed substrings should be found in all polymorphic worm instances, which is a difficult task.

Polygraph and another scheme, Hamsa [13] work on the network-level to detect Zero-day polymorphic worms [30], [31] and they generate multiple tokens as signatures, but Hamsa claims that it has a significant improvement in terms of speed and attack flexibility over Polygraph. Hamsa focuses on content-based signatures. Both of these schemes have the same token-based approach, but instead of relatively slow suffix tree method of token extraction, Hamsa uses a lightweight suffix array method. This improves the speed of token generation process up to 100 times. A greedy algorithm is used to extract multiple token signatures. Presence of noise changes the computational complexity of the problem, and affects the final quality of the produced signature. Average and maximum false positive rate have been shown as 0.095 and 0.75 respectively. However, the resulted signature in this scheme could be affected by the size of the suspicious traffic. In fact, for a working scheme, 100 samples of a worm would be needed as minimum.

LISABETH [14] is an automated content-based signatures generator for Zero-day polymorphic worms. It recognizes fixed bytes of traffic content, as originally proposed in Polygraph [12] and improved by Hamsa [13]. Signature generation takes 20% less time than that of Hamsa. It creates reliable signature with low false positive in presence of noise. It reduces computational overhead by avoiding redundant signature generation. Still, the drawbacks of this approach are the same as of Hamsa. Position-aware distribution signature (PADS) [15], [16] is designed to fill the gap between signature-based IDS and anomaly-based IDS. This system is based on double honeypot. It is able to automatically detect the presence of a new worm in the normal traffic. PADS inherits the positive aspects from both anomaly and signature based schemes. Its signature generation is based on counting frequency of occurrence of a specific byte in a specific position. Instead of using a fixed string like that is used in signature based approach, it uses flexible string to catch more worm variants. It is claimed that, it is much accurate than the position-unaware statistical signatures. However, the drawbacks are clear as it is not capable of detecting advanced worms, it cannot be merged with other IDSs like for instance: Snort [17], and it does have high computational overhead. In [18], the authors propose TaintCheck, which is an automated dynamic taint analysis to detect polymorphic attacks. The main functionality of TaintCheck is performed during the run time, so it does not need any code modification on the IDS. Any passing of traffic that is generated by unknown resource is marked as tainted. Tainted data are monitored during program execution. TaintCheck states that it can reliably detect most overwrite worms (those cause overwriting operation/attack). This scheme is vulnerability-based and host-based. It can be used to provide exhaustive information about the attack characteristics. No false positive was found while testing TaintCheck as it has been reported in the work. However, in this scheme, signature generation process cannot be achieved automatically and it is very much application-specific - a particular type of server must be used. COVERS (COntext-based, VulnERability-oriented Signature) [19] generates signatures to capture attacks that are targeted to the same previous vulnerability (as it might have had been exploited before). This makes the approach effective against polymorphic worms. Unlike network-based techniques, COVERS approach is able to produce signatures from single polymorphic worm instances, using correlation and input context identification. One of the advantages of COVERS that it introduces is low overhead. In addition to that, deployment of this approach does not require any modification to the IDS, or access to its source code. The experimental results for all evaluated attacks showed no false positive and low false negative. In spite of the noticeable advantages, the down sides of the approach are: it cannot be used by IDSs or firewalls; it is not for general purpose implementation as it is application specific; it requires manual involvement; and it depends on application's source

code.

ARBOR [20] implements a new approach that can identify the characteristics of a particular worm, and then filter future traffic from instances of the same worm or its variants. By using this method, the availability of main network server will be significantly increased. ARBOR claims that it is completely automatic and does not need any human intervention. There is no need for IDS source code, and it has low runtime overhead. During experiments, it was effective against most worms and showed no false positive. Our analysis shows that this scheme still cannot evade attacks that are fragmented through multiple packets.

Vigilante [21] introduces an end-to-end architecture to automate worm detection. Vigilante analyzes dynamic data-flow instead of analyzing worm behavior or worm body. It simply detects infection attempts with broad coverage. This can evade the three most common infection techniques used by worms, including: edge injection, code injection, and data injection. It does not require access to source code as well. Under some conditions, it may produce false positive. Also another drawback for Vigilante is that its signature may lead to false negative. The success of this scheme depends on the threads scheduling order. Another work Double-honeynet as reported in [22], produces accurate worm signatures. The method works both as network-based and host-based. However, long deployment time for the honeynets is a real drawback of this work.

Considering all the pros and cons of various existing schemes and understanding the significance of the research issue, we have come up with our proposal that would solve almost all the negative issues as noted in this section. The subsequent sections would describe our approach in detail.

3. String Similarity Algorithms

String approximate matching algorithms play a rising role in string related research and applications. An unlimited number of applications depend on this type of processing, such as search engines, post classification, document clustering, topic detection, topic tracking, question generation, question answering, post scoring, short answer scoring, automated translation, text summarization, movies and music classification, finding plagiarized documents and so on. Similarity finding between two words is a fundamental part of string similarity which is then used as a primary stage for sentence, paragraph, and document similarity tests.

String similarity is classified into two main categories: character based and semantic based. Character based is used to measure the similarity between two strings which depends on their character structures, while semantic based similarity is based on the meaning of the two words. One of our objectives in this work is about character based methods, to determine the degree of similarity between two instances of polymorphic worms. A similarity function outputs how similar two strings are and returns a value within [0,1]. Typically, the smaller the value, the more differences are between the two strings. If the similarity value between the two strings is zero, it means that they do not have any common substring [23]. Many similarity algorithms have been developed among which, Dice, Jaccard, and Cosine [24], [25] are the most commonly used. For our work, we have chosen the Jaccard, since it is more suitable for long paragraphs or whole document.

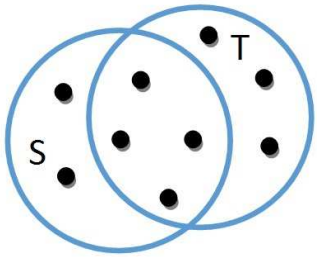


Figure 1. Two sets with Jaccard similarity, 4/9.

3.1 Jaccard Similarity

The Jaccard similarity value of two sets S and T is:

$$SIM(S,T) = |S \cap T| / |S \cup T|$$

that is, the ratio of the size of the intersection of S and T to the size of their union [24]. And, for string similarity, Jaccard is computed as the number of shared terms over the number of all unique terms in both strings. For instance, in Figure 1, we see two sets: S and T. There are four elements in their intersection and a total of nine elements that appear in S and T both. Thus, SIM(S,T) = 4/9.

3.2 Worm Instances to Sets

The critical question is: How to apply this set machinery to polymorphic worms' samples? Originally, Jaccard similarity algorithm was used to find the likeness ratio between two sets. But, we would apply it to find the perfect signature among different polymorphic instances.

A commonly used approach is to shingle the document, and we will use it to process the worm sample. This method takes a group of characters and considers them as a single object. A k-shingle is basically a consecutive set of k words. The following example demonstrates the concept of shingles division using a normal document (not a worm sample).

D_1: I am Norah.
D_2: Norah I am.
D_3: I do not like beef and red fish.
D_4: I do not like them, Norah I am.

The (k = 1)-shingles of D_1∪ D_2∪ D_3∪ D_4 are: {[i], [am], [Norah], [do], [not], [like], [beef], [and], [red], [fish], [them]}.

The (k = 2)- shingles of D_1∪ D_2∪ D_3∪ D_4 are: {[I am], [am Norah], [Norah Norah], [Norah I], [am I], [I do], [do not], [not like], [like beef], [beef and], [and red], [red fish], [like them], [them Norah]}.

k-shingles can also be created at the character level. The (k = 3)-character shingles of D_1∪ D_2 are: {[iam], [amn], [mno], [nor], [ora], [rah], [ahn], [hno], [ahi], [hia]}.

k is a constant and can be picked as desired from the positive numbers set. However, if we choose k to be too small, then most sequences of k characters will appear in most of the documents. In that case, Jaccard similarity will be high, and this is not logically sound. This leads to an important question that is: How large should k be? The answer depends on how long the used documents are. And, how large the set of typical characters is [24].

For worm instances, choosing k is very critical - if k is too small, this will produce high false positive, while choosing large k produces high false negative.

Many modeling choices should be taken into account when dealing with worm samples. Here is a list of the important ones:

White characters: e.g., I am Norah. vs. I am (new line)

Norah.

Case sensitivity: Norah vs. norah.

Punctuation: e.g. them, Norah vs. them Norah.

Number of occurrences: Should we count the number of replicas of shingles or not?

Articles, pronouns, and conjunctions: Words like (you, for, the, to, and, that, it, is, ... etc.) are very common. Should we omit these words or not?

3.3 Jaccard with Shingles

Let us consider the (k=2)-shingles for each of D_1, D_2, D_3, and D_4:

D_1: [I am], [am Norah]

D_2: [Norah I], [I am]

D_3: [I do], [do not], [not like], [like beef], [beef and], [and red], [red fish]

D_4: [I do], [do not], [not like], [like them], [them Norah], [Norah I], [I am]

The Jaccard similarities are as follows:

JS(D1, D2) = 1/3 ≈ 0.333

JS(D1, D3) = 0

JS(D1,D4) = 1/8 = 0.125

JS(D2, D3) = 0

JS(D3, D4) = 2/7 ≈ 0.286

JS(D3, D4) = 3/11 ≈ 0.273

4. Design and Implementation of Our Proposed Scheme

4.1 Design Phase

Our detection algorithm is designed in a way to detect most of the polymorphic worm [34] instances. MATLAB [26] interactive environment is used to implement the technique practically. With the work, we will examine sufficient number of polymorphic worms mixed within a normal packet traffic. False positive and false negative will also be determined to measure the proposed technique's accuracy. In general, the final signature will be generated based on the most common substrings between worm instances. As shown in Figure 2, substrings formulate the signature which can be found in two or more worm instances. However, this does not mean that the signature substrings can be found in all of the worm instances.

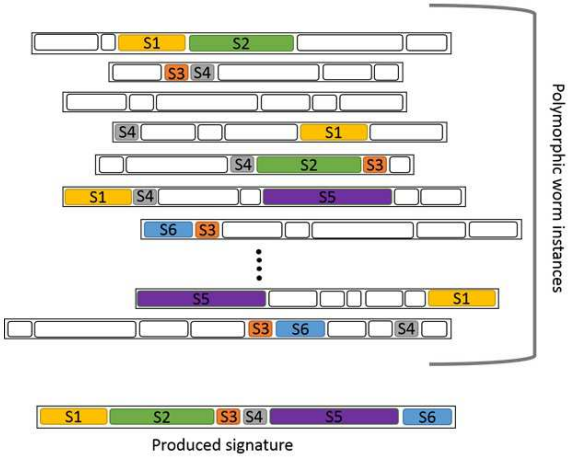


Figure 2. General format of the produced signature.

The proposed solution is implemented with four phases:

1. Shingles divider
2. Signature extractor
3. Signature reducer
4. Flow checker

The following sub-section will describe the design in a general form. After that, we will discuss each phase in detail.

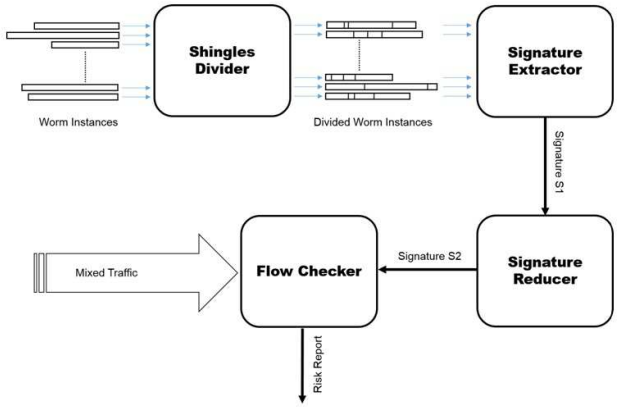
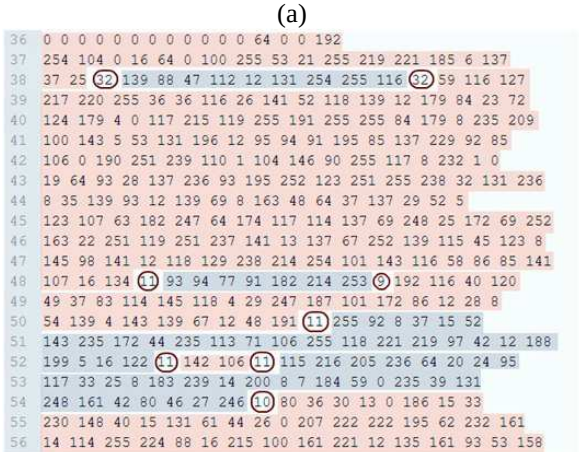


Figure 3. Polymorphic worm detection system overview.

4.2 Detection System Overview

An overview of our polymorphic worm detection system is shown in Figure 3. The system contains four components: shingles divider, signature extractor, signature reducer, and flow checker. Here, different worm instances belong to one kind of polymorphic worm and a mixed network traffic comes form the system input. The system output is the manager's risk report.

(a)



(b)

Figure 4. (a) Segment of a worm. (b) Segment of a worm after fragmentation.

First phase is the shingles dividing process, where each worm instance is divided into diverse shingles. The worm instances should be from the same type of worm; for instance, all of the instances are Sasser worm [4], [27]. However, the instances may have different sizes.
Second phase is the signature generation process, where common shingles among different instances are considered as the polymorphic worm signature, which should be achieved after computing the intersection result between different shingles.
Third phase is the signature reduction process. After generating worm signature from the previous phase, the signature still has rubbish data that need clipping. The signature will passes through three filters to be completely polished. After these three filters, the signature will be ready for use for the following phase.
Fourth phase is a network traffic checker, where the usual network traffic (i.e., innocuous packets) is examined against worm signature. Risk report will be generated to provide the network administrator with the level of danger.

4.3 Design Specification Details and Experiments

In this section, we describe each system component in details.
Phase (1) - Shingles Divider: The worm instance is constructed from a set of characters, like any type of document. There is no benefit from calculating Jaccard at character level, since the intersection between the instances becomes equal to the union. Therefore, Jaccard value will be equal to one, indicating that all instances are the same, which is logically false.
Shingling is the first step to find document similarity in natural language processing (NLP) [24]. It is the process to divide documents to a series of tokens based on a fixed size and/or special delimiter(s).
***Fragment worm instance to shingles*:** The same concept that is used for ordinary documents will be used on polymorphic worms' instances. The provided polymorphic worm samples are basically a set of characters converted to the corresponding Unicode integer values. A segment of a worm is shown in Figure 4(a).
The fragmentation will be delimiter based. Newline, vertical tab, horizontal tab and space characters are used as delimiters. They have the following Unicode representations respectively (10, 11, 9, 32). After processing the segment in Figure 4(a), the fragmentation will be as shown in Figure 4(b), where the segment is divided into nine tokens.
The following pseudo code describes the fragmentation process:

```
for each worm instance do
   W=worm instance
   W'=W.split(' 32 ', ' 10 ', ' 09 ', ' 11 ')
   Store W'
end
```

Phase (2) - Signature Extractor: The main objective of this phase in the system is to generate a shared worm signature. It should meet the following conditions:

1. The signature is common among the worm instances as much as possible.
2. The generated signature should be long enough to avoid false positive, and short enough to avoid false negative.
3. The signature must be flexible enough to defend against polymorphic worms that change their payloads in every infection attempt. So, considering single substring as the unique signature may not be invariant across worm instances.
4. The signature must be reliable enough to detect wide variety of Zero-day attacks.
5. It should take into account the system resource limitation and computation overhead such as; CPU (Central Processing Unit) time of generating signature and comparing them with the network traffic. In addition, signature storage space must not be stressed.

The last point will not be covered in this context. The signature extractor will be rather divided into multiple stages to make it simple and more efficient.
***Jaccard similarity calculation*:** As first step of signature extraction process, Jaccard similarity will be calculated between each pair of worm instances. The calculation will be as follows:
Jaccard similarity of two worm instances $\boldsymbol{W_1}$ and $\boldsymbol{W_2}$ is:

$$JS = \frac{|\text{number of shingles in } \boldsymbol{W_1} \cap \text{number of shingles in } \boldsymbol{W_2}|}{|\text{number of shingles in } \boldsymbol{W_1} \cup \text{number of shingles in } \boldsymbol{W_2}|} \quad (1)$$

This will be calculated for each instance pairs. So JS($\boldsymbol{W_1}$, $\boldsymbol{W_1}$), JS($\boldsymbol{W_1}$, $\boldsymbol{W_2}$), JS($\boldsymbol{W_1}$, $\boldsymbol{W_3}$),…JS($\boldsymbol{W_1}$, $\boldsymbol{W_n}$), JS($\boldsymbol{W_2}$, $\boldsymbol{W_1}$), JS($\boldsymbol{W_2}$, $\boldsymbol{W_2}$), JS($\boldsymbol{W_2}$, $\boldsymbol{W_3}$),… JS($\boldsymbol{W_2}$, $\boldsymbol{W_n}$),…JS($\boldsymbol{W_n}$, $\boldsymbol{W_n}$) are calculated for $\boldsymbol{n}$ worm instances. For simplicity, the result could be represented with a two dimensional matrix as shown in Table 1.

```
wormFile=import all worm instances after fragmentation
% nested For loop to traverse two dimensional matrix, that is used to store Jaccard similarity values
% Loop starts from 1 to the end of folder that contains the worm instances
for i = 1:size(wormFile);
  for j = 1:size(wormFile);

% JacFor2Doc function takes 2 worm instances as input and returns the similarity between them and the intersected segments as a vector

[intersectVec,jacMatrix(i,j)]=JacFor2Doc(wormFile(i),wormFile(j));
% intersectMatrix to store the intersectVector for each iteration
        intersectMat{i,j}=intersectVec;
  end
end
```

Any element on the diagonal surely will be 1, since it is a Jaccard comparison between the document and itself. To reduce computational overhead to half, we can calculate elements above the main diagonal only.
The following pseudocode describes the process of Jaccard similarity calculation for a set of worm instances. Worm instances in this step should pass the fragmentation process which has been described before.
Presented below is a part of *JacFor2Doc* function pseudocode, which compares two worm samples (after fragmentation process) and returns Jaccard value for them. In addition, it returns the intersected parts as a vector of segments:

```
JacFor2Doc(W1',W2')
% intersection of two worm instances stored as a vector
int=intersect(W1',W2');
% union of two worm instances stored as a vector
uni=union(W1',W2')

L = length(int);
L2 = length(uni);

%percentage between length of the intersection vector and unit
vector is Jaccard value
jaccardValue=L/L2;
return (int,jaccardValue);
```

Table 1. Representation of JS for all instances of one type of worm.

	W_1	W_2	W_3	.	W_n
W_1	JS(W_1, W_1)=1	1≥JS(W_1, W_2) ≥0	1≥JS(W_1, W_3) ≥0	.	1≥JS(W_1, W_n) ≥0
W_2	1≥JS(W_2, W_1) ≥0	JS(W_2, W_2)=1	1≥JS(W_2, W_3) ≥0	.	1≥JS(W_2, W_n) ≥0
W_3	1≥JS(W_3, W_1) ≥0	1≥JS(W_3, W_2) ≥0	JS(W_3, W_3)=1	.	1≥JS(W_3, W_n) ≥0
.	.	.	.	.	.
W_n	1≥JS(W_n, W_1) ≥0	1≥JS(W_n, W_2) ≥0	1≥JS(W_n, W_3) ≥0	.	JS(W_n, W_n)=1

***i. Jaccard for Blaster worm*:** We have calculated Jaccard similarity for 42 instances of Blaster worm [6], [7]. MATLAB®interactive environment has been used for the implementation. The result is shown with the bar chart in Figure 5(a). The bar chart clearly shows that the diagonal elements have 1 for Jaccard value, since worm sample has compared with itself. Elements around the diagonal with opposite position have the same Jaccard similarity. Therefore, obviously JS(W_3, W_2) =JS(W_2, W_3). So, we can conclude that Jaccard algorithm is a commutative relation. Detailed Jaccard similarity values are shown in Table 2. For brevity, 10 samples are shown from the 42.

***ii. Jaccard for Sasser worm*:** Like the previous one, Jaccard similarity has been also calculated for 50 instances of Sasser worm. The same MATLAB®interactive environment has been used for simulation. The result is shown in the bar chart as in Figure 5(b). Again, the bar chart clearly shows that the diagonal elements have 1 for Jaccard value, since worm sample has compared with itself. Elements around the diagonal with opposite position have the same Jaccard similarity. Therefore, here also, JS(W_3, W_2) =JS(W_2, W_3). Detailed Jaccard similarity values are shown in Table 3 (10 samples are shown out of the 50). It should be noted here that the polymorphic worm samples used in this work were obtained from the Institute Eurecom in the French Riviera. We removed the signatures of the worms from the database of IDS so that these polymorphic worms play as Zero-day polymorphic worms.

***Choosing the superior Jaccard value*:** After producing Jaccard matrix for all worm instances of the same type, it is time to choose the best results and eliminate poor ones. For each row of Jaccard matrix, the maximum Jaccard value will be chosen as the best match. Apparently, any cell with equal to value 1 will be discarded since it indicates that the worm compared with itself.

Only the upper half of the matrix main diagonal will be considered at this stage to avoid redundant values, which will affect the final results. Figure 6 shows a general format of the process.

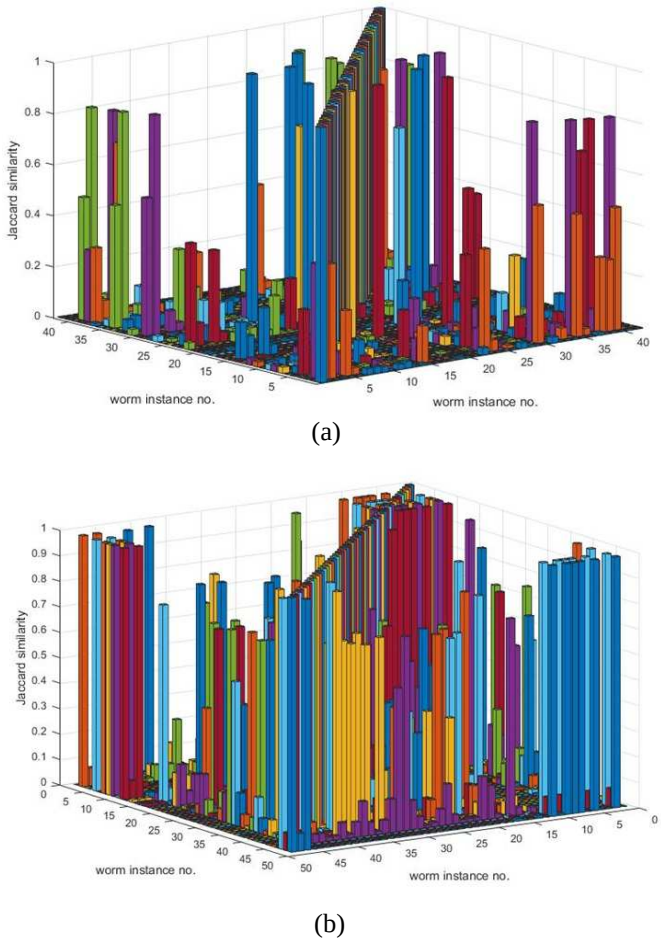


Figure 5. 3D bar chart representing (a) Jaccard matrix for 42 samples from Blaster worm. (b) Jaccard matrix for 42 samples from Sasser worm.

Let Jaccard matrix dimensions be $n \times n$. For each row r, cells c with positions between $(i, j+i+1)$ and (i, n) will be examined to choose the maximum value among them.

The following pseudocode shows the selection process:

```
IntersectMat contains the shared elements between each two
worm instances
% choosing the maximum value in each row of jacMatrix
for i = 1:size(wormFile);
      maxVal=0;
      for j = i+1 :size(wormFile);
          if (jacMatrix(i,j)) ~= 1 && jacMatrix(i,j) ~= 0)
             if(maxVal < jacMatrix(i,j))
               maxVal = jacMatrix(i,j);
               maxR=i;
               maxC=j;
% maxCell to store intersecting elements in each iteration from
original intersectMatrix
               maxCell(i)=intersectMat(maxR,maxC);
               maxInRow(i)=maxVal;
             end
          end
      end
end
```

Now, as we have the intersected elements for the maximum Jaccard value in each row, we have to compute the time of occurrence for each element. The element with

the highest occurrence value will be considered as a part of worm signature.

MATLAB *tabulate* function is used to create the *frequency of data* table (*maxCell*) which is presented in Table IV. *maxCell* contains the intersected elements among worms' samples with the highest Jaccard value. *sortrows* is used to sort the output of tabulate function in descending order. In general, the table generated using *tabulate* will look like Table 4.

Table 2. Part of Jaccard matrix (for 10 samples) of blaster worm.

	1	2	3	4	5	6	7	8	9	10
1	1	0.0014	0.0013	0.0044	6.7869e-04	6.8166e-04	0.0248	0.0262	0.0227	6.7994e-04
2	0.0014	1	0.4490	0.0044	0.2558	0.0274	0.0019	0.0018	0.0017	0.0245
3	0.0013	0.4490	1	0.0039	0.0874	0.0347	0.0016	0.0017	0.0015	0.0302
4	0.0044	0.0044	0.0039	1	0.0034	0.0041	0.0134	0.0077	0.0126	0.0034
5	6.7869e-04	0.2558	0.0874	0.0034	1	0.0377	8.7000e-04	8.9940e-04	7.3215e-04	0.0324
6	6.8166e-04	0.0274	0.0347	0.0041	0.0377	1	8.7489e-04	9.0463e-04	8.5900e-04	0.0896
7	0.0248	0.0019	0.0016	0.0134	8.7000e-04	8.7489e-04	1	0.0822	0.1481	8.7206e-04
8	0.0262	0.0018	0.0017	0.0077	8.9940e-04	9.0463e-04	0.0822	1	0.0725	7.7270e-04
9	0.0227	0.0017	0.0015	0.0126	7.3215e-04	8.5900e-04	0.1481	0.0725	1	8.5627e-04
10	6.7994e-04	0.0245	0.0302	0.0034	0.0324	0.0896	8.7206e-04	7.7270e-04	8.5627e-04	1

Table 3. Jaccard matrix (for 10 samples) of blaster worm.

	1	2	3	4	5	6	7	8	9	10
1	1	0	0.0154	0.0134	0	0.0028	0	0	0	0
2	0	1	0	0	0.9856	3.2819e-04	0.9856	0.9856	0.9856	0.0074
3	0.0154	0	1	0.1623	0	0.0885	0	0	0	0
4	0.0134	0	0.1623	1	0	0.0248	0	0	0	0
5	0	0.9856	0	0	1	3.2819e-04	0.9856	0.9856	0.9856	0.0074
6	0.0028	3.2819e-04	0.0885	0.0248	3.2819e-04	1	3.2819e-04	3.2819e-04	3.2819e-04	3.2862e-04
7	0	0.9856	0	0	0.9856	3.2819e-04	1	0.9856	0.9856	0.0074
8	0	0.9856	0	0	0.9856	3.2819e-04	0.9856	1	0.9856	0.0074
9	0	0.9856	0	0	0.9856	3.2819e-04	0.9856	0.9856	1	0.0074
10	0	0.0074	0	0	0.0074	3.2862e-04	0.0074	0.0074	0.0074	1

Table 4. General format of tabulate function table.

Value	Count	Percentage
The unique values of x	The number of instances of each value	The percentage of each value
Ex. 99 97 110 110 111 116	19	0.0567
Ex. 1×37 char	23	0.1008

The second row in Table 4 shows the occurrence frequency for cell '99 97 110 110 111 116', where 19 is the number of its occurrence, and 0.0567 is the percentage of its occurrence among other cells in *maxCell*. The third row shows the occurrence frequency for the 1×37 char array of characters, where 23 is the number of its occurrence, and 0.1008 is the percentage of its occurrence among other cells.

The complement value of count column in Table 4 can be considered as false negative percentage if this substring is chosen to be the worm signature. Example: If we have X worm instances from one type and that worm has a common substring S with count value Y, $Y < X$. If S is chosen as a signature, $(X - Y)/100$ is the false negative percentage.

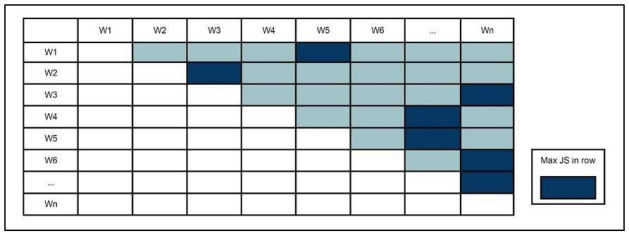


Figure 6. Choosing the superior JS value in each row.

We have tested the method of choosing the superior Jaccard values both for Blaster worm and Sasser worm. To put some examples, Figures 7(a) and 7(b) show parts of the *tabulate* table for Blaster worm before and after sorting, respectively. Similar snapshots could also be presented for Sasser worm.

Phase (3) - Signature Reducer: After generating the common signature for specific polymorphic worm, we notice that there are many common shingles to be considered as part of the signature. In other words, the common signature contains many substrings, and dealing with big amount of shingles/substrings can be difficult and ineffective in the next phase. It will take a huge amount of time to search and filter normal data flow from those shingles/substrings.

	1	2	3
1	'0'	17	0.0527
2	'032 0'	14	0.0434
3	*1x16 char*	1	0.0031
4	*1x16 char*	1	0.0031
5	*1x16 char*	1	0.0031
6	*1x16 char*	1	0.0031
7	*1x16 char*	1	0.0031
8	*1x16 char*	1	0.0031
9	*1x43 char*	1	0.0031
10	'0 0 0'	15	0.0465
11	*1x14 char*	2	0.0062
12	'0 0 032'	1	0.0031
13	*1x25 char*	1	0.0031
14	*1x26 char*	5	0.0155
15	*1x26 char*	5	0.0155
16	*1x26 char*	5	0.0155
17	*1x26 char*	5	0.0155
18	*1x26 char*	5	0.0155
19	*1x113 char*	2	0.0062
20	*1x240 char*	2	0.0062
21	*1x57 char*	1	0.0031
22	*1x26 char*	5	0.0155
23	'0 0 0 0 82 5...	5	0.0155
24	*1x47 char*	1	0.0031
25	*1x48 char*	1	0.0031

(a)

	1	2	3
1	'105 110'	35	0.1085
2	'112 114 111...	35	0.1085
3	'114 117 110'	35	0.1085
4	*1x37 char*	35	0.1085
5	'98 101'	35	0.1085
6	*1x22 char*	35	0.1085
7	'116 111'	19	0.0589
8	'110 111 116'	18	0.0558
9	'13'	18	0.0558
10	'0'	17	0.0527
11	'10'	17	0.0527
12	'102 111 114'	16	0.0496
13	'105 115'	16	0.0496
14	'32'	16	0.0496
15	'45'	16	0.0496
16	'97'	16	0.0496
17	'0 0 0'	15	0.0465
18	'116'	15	0.0465
19	'37 115'	15	0.0465
20	'60'	15	0.0465
21	'032 0'	14	0.0434
22	'101 110 111...	14	0.0434
23	*1x18 char*	14	0.0434
24	'115 112 97 ...	14	0.0434
25	*1x18 char*	14	0.0434

(b)

Figure 7. Part of tabulate table for Blaster worm (a) before sorting, (b) after sorting.

As a rule of thumb, to choose among multiple shared shingles/substrings, two conditions must be taken into account:

1. The chosen signature should be the most common shingles/substrings between worm instances as much as possible.
2. The chosen signature must be long enough to avoid false positive.

The first condition is already achieved (in section 4.3) by calculating the Jaccard values between worm instances to generate the signature. The second condition can be achieved by deleting those cells that contain short strings from the final signature (see Figure 8).

Another effective step will be added to assure signature efficiency. This step is to compare the resulted signature parts with a known pure normal traffic. This step basically searches for the signature shingles/substrings in the known pure normal traffic and if any part of the signature is found in the normal traffic, that part will be eliminated from the signature.

The main benefit of the latter step is that we can assure that any common part shared in specific type of files will be eliminated from the signature. For example, the header part of any *.exe* file is the same whether it is a worm file or a normal file. Another example is the beginning of any JAVA program, e.g., the same "*public class … {public static void main(String[] args) {…*". Hence, this step will guarantee that any kind of common string will not be excluded from the signature, and this will give a strong protection against the

false negative occurrences. Signature reduction process can be done in any sequence which is illustrated in Figure 8.

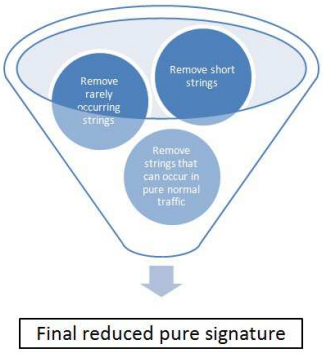


Figure 8. Signature reduction process.

The signature reduction has been tested for two types of worms as follows:

i. Signature reduction for Blaster worm: For Blaster worm, signature reduction step is achieved by the following sub-steps:

- From *tabulate* Table (see Figure 7), eliminate cells with (count < 5).
- Eliminate cells with 1, 2 or 3 characters only (value < 4).
- Eliminate any cell that can occur in normal traffic.

Here, (count < 5) is considered because average word size in English is about 5 letters, which is used in programming languages, hence, the worm samples. Words shorter than 5 letters appear very frequently. So, to build a system that runs as fast as possible, we can have a significant performance boost by ignoring words less than 5 letters; therefore, the worm signature will have less shingles and that can be noticeable especially in Blaster worm's case. In our experiment, normal traffic is 123 MB of pure data in the application layer tested against any type of malicious ware. Figure 9 shows a dramatic decrease in Blaster worm signature size from 20125 shingles/substrings to only 199.

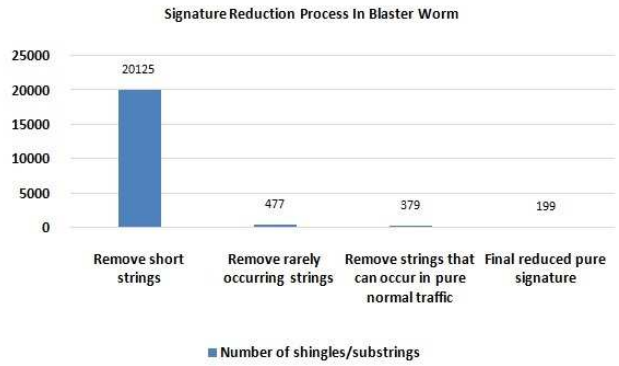


Figure 9. Signature reduction process for Blaster worm.

ii. Signature reduction for Sasser worm: Same steps as applied to Blaster worm's case have been applied for Sasser worm. Signature size after each reduction step is shown in Figure 10.

Phase (4) - Flow Checker: This phase will filter out normal traffic from polymorphic worms and will generate an alarm to the network administrator. Flow checker stage is basically a simulation module of what could happen in a real IDS implementation. It also is a stage where the quality of the generated signature can be tested. Parts of the flow checker phase simulation results are shown in Figure 11. First column determines the location of the scanned file. Second column shows the *degree of danger* of each file. This number indicates the number of discovered signature substrings in the file.

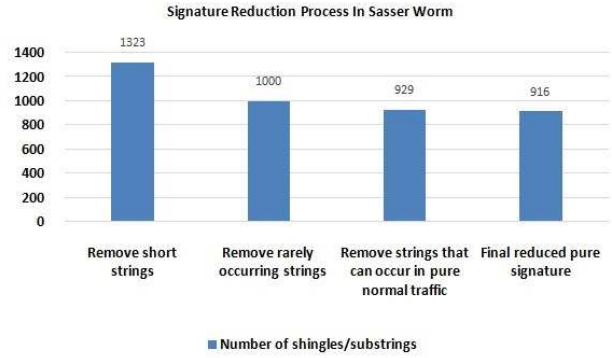


Figure 10. Signature reduction process for Sasser worm.

High number means that the file is definitely a worm, which requires a fast response from the network manager.

In this test case, first 80% of pure normal traffic and 20% of worm instances are blended to form the mixed traffic.

$$Mixed\ Traffic = 80\,\%\ Normal\ Traffic + 20\,\%\ Worm\ Instances$$

To ensure the fineness and reliability of the generated signature, worm instances that are used in this phase are new and completely different from those that were used to generate the signature.

	1	2
1	C:\Users\8\Desktop\Master Thesis\mixed traffic\1.txt	0
2	C:\Users\8\Desktop\Master Thesis\mixed traffic\2.txt	0
3	C:\Users\8\Desktop\Master Thesis\mixed traffic\3.txt	0
4	C:\Users\8\Desktop\Master Thesis\mixed traffic\4.txt	0
5	C:\Users\8\Desktop\Master Thesis\mixed traffic\5.txt	0
6	C:\Users\8\Desktop\Master Thesis\mixed traffic\6.txt	0
7	C:\Users\8\Desktop\Master Thesis\mixed traffic\7.txt	0
8	C:\Users\8\Desktop\Master Thesis\mixed traffic\8.txt	0
9	C:\Users\8\Desktop\Master Thesis\mixed traffic\Blaster 15.txt	780
10	C:\Users\8\Desktop\Master Thesis\mixed traffic\Blaster 21.txt	80
11	C:\Users\8\Desktop\Master Thesis\mixed traffic\Blaster 22.txt	26
12	C:\Users\8\Desktop\Master Thesis\mixed traffic\Blaster 33.txt	1
13	C:\Users\8\Desktop\Master Thesis\mixed traffic\Blaster 34.txt	768

Figure 11. Snapshot from part of the flow checker stage simulation in MATLAB.

After this step, mixed traffic will be scanned to detect worm instances. The scanning process will be done using the final reduced signature generated from the previous step. Let us see some practically implemented and tested cases in the following section.

5. Experimental Evaluation and Comparative Analysis

The following cases have been tested using the flow checker:

5.1 Flow Checker for Blaster Worm

In Table 5, we show the amount of normal traffic and worm instances that are used to generate mixed traffic, using number of files and total size of files in MB (Megabyte). The blending step was implemented using randomly chosen Blaster worms and randomly chosen files from the pure normal traffic.

Number of the detected Blaster worms in the mixed traffic is

roughly comparable to the ideal case, where all Blaster worm instances are found. Apparently, worm instances that have the lowest Jaccard similarity values fail the test. Figure 12 shows the number of detected Blaster worm instances from mixed traffic.

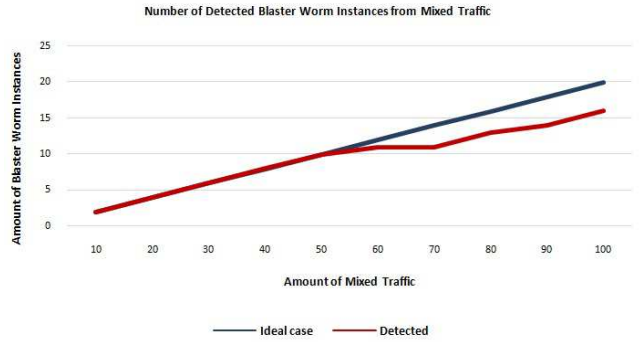


Figure 12. Number of detected Blaster worm instances from mixed traffic.

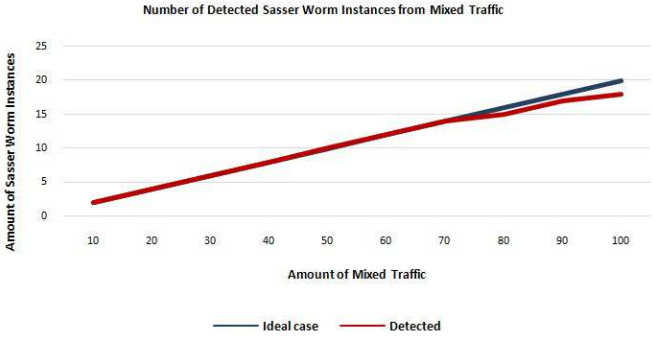


Figure 13. Number of detected Sasser worm instances from mixed traffic.

5.2 Flow Checker for Sasser Worm

Just like the previous example, Table 6 is the case for Sasser worm. Similar method is used to test this as well.

Number of detected Sasser worms in the mixed traffic is almost similar to the ideal case, where all Sasser worm instances are detected, without any false negative. Small amount of worm instances fail the test. Apparently, those have the lowest Jaccard similarity values. Figure 13 shows the number of detected Sasser worm instances from mixed traffic.

5.3 Percentages of False Positives and False Negatives

A good worm scanner should have two core aspects. Firstly, it should avoid false negatives. A false negative is a case where the detection system declares that some traffic is free from worm infection, but in fact it is not. Secondly, the scanner should avoid false positives as well. A false positive is the opposite of false negative which is wrongly declaring that a traffic contains a worm, but it is actually clean.

"*Avoiding both types of mis-detections is a worthy goal for virus software, but has been proved to be theoretically impossible*" [28], [32], [33]. Keeping a balance between them is needed. While a high level of false negatives is worse in the short term (since it leaves the system infected), high level of false positives means the network admin will be careless toward the IDS's warnings, possibly causing to ignore a real alarm. But, false negatives imply that an actual worm is crossing the IDS without action. Therefore, the percentage of false positives and false negatives represent the system sensitivity [29]. False negatives give the worms the opportunity to escape defense, while false positives may cause network shortage by preventing normal traffic [21].

We could assure that our algorithm is false positive free. False negative percentage cannot exceed 23% in case of Blaster worm (Figure 14) and 10% for Sasser worm (Figure 15).

False positives and false negatives percentages are calculated as follows:

$$FP\% = \frac{number\ of\ files\ mistakenly\ detected\ as\ a\ worm}{total\ amount\ of\ pure\ normal\ traffic} \times 100 \quad (2)$$

$$FN\% = \frac{number\ of\ miss\_detected\ worm\ instances}{total\ number\ of\ worm\ instances\ in\ the\ mixed\ traffic} \times 100 \quad (3)$$

5.4 Comparative Analysis

The main advantage of our system is that it generates a flexible signature. This signature contains different strings, of which each string has a different probability to appear in the polymorphic worm body. Assigning different probability to each sub string breaks the rule that the majority of the existing systems follow; which mean that all the signature parts must exist in the polymorphic worm body.

Different security levels can be maintained in a real-life application scenario. After the normal traffic passes through the IDS, our algorithm will give the network manager the presence percentage of a worm's signature parts.

The only disadvantage of our algorithm is the potential computational overhead. But, the effect of this disadvantage depends of many factors, such as the machine type, amount of the input, and the used programming language. Moreover, to ensure right level of security, any system could of course spend a bit of extra resources, if need be.

Table 5. Amount of normal traffic and Blaster worm instances blended with generated mixed traffic.

80 % Normal Traffic	20 % Worm Instances	Mixed Traffic
80 files (total size 104 MB)	20 files (total size 38 MB)	100
72 files (total size 90 MB)	18 files (total size 26.1 MB)	90
64 files (total size 67 MB)	16 files (total size 23.6 MB)	80
56 files (total size 62.1 MB)	14 files (total size 15.9 MB)	70
48 files (total size 45.2 MB)	12 files (total size 14.1 MB)	60
40 files (total size 41 MB)	10 files (total size 7.22 MB)	50
32 files (total size 29.3 MB)	8 files (total size 3.32 MB)	40
24 files (total size 23.3 MB)	6 files (total size 3.27 MB)	30
16 files (total size 14.3 MB)	4 files (total size 3.2 MB)	20
8 files (total size 1.62 MB)	2 files (total size 2.46 MB)	10

The comparative advantages and disadvantages for each polymorphic worm detection system are presented in Table 7. Regarding the false negative and false positive values shown in this table, the numbers are according to each work's reported data. Different systems used different metrics.

Table 6. Amount of normal traffic and Sasser worm instances blended with generated mixed traffic.

80 % Normal Traffic	20 % Worm Instances	Mixed Traffic
80 files (total size 104 MB)	20 files (total size 2.92 MB)	100
72 files (total size 90 MB)	18 files (total size 2.82 MB)	90
64 files (total size 76 MB)	16 files (total size 2.25 MB)	80
56 files (total size 62.2 MB)	14 files (total size 2.26 MB)	70
48 files (total size 45.2 MB)	12 files (total size 1.38 MB)	60
40 files (total size 41 MB)	10 files (total size 1.27 MB)	50
32 files (total size 29.3 MB)	8 files (total size 1.16 MB)	40
24 files (total size 23.2 MB)	6 files (total size 950KB)	30
16 files (total size 14.2 MB)	4 files (total size 842KB)	20
8 files (total size 1.62 MB)	2 files (total size 112KB)	10

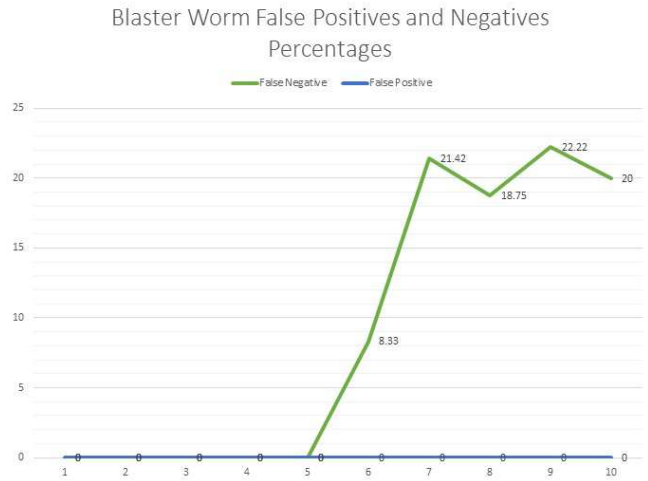


Figure 14. Blaster worm false positive and false negative percentages.

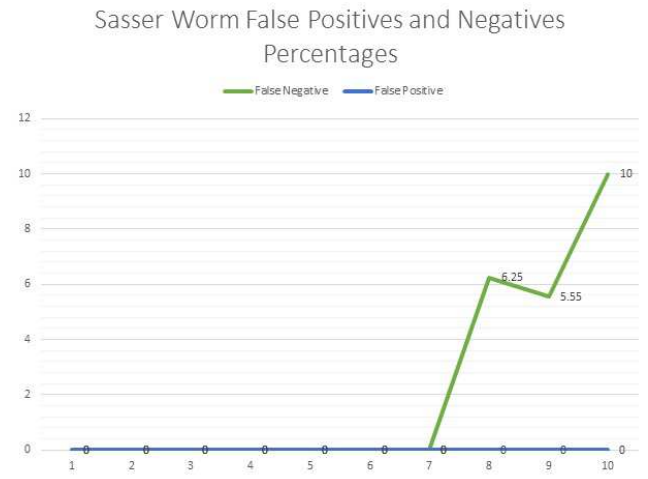


Figure 15. Sasser worm false positive and negative percentages.

There are many other factors that would influence the practical implementation cases as well. Some of them are: types of worms used in the experiments, amount of worm samples and percentage of odd samples, and so on. A standard performance testing platform is therefore needed to produce a more accurate comparison. We have tried to do the comparison as fairly as possible. Clearly, no detection system is fully safe, and all will have shortcomings. However, as we have tested several cases, we have found that our scheme performs pretty well compared to all other available solutions in this area.

6. Conclusions and Future Work

In this paper, we have shown that the network *content-based methodology* holds great promise for defending against Zero-day polymorphic worms. Moreover, we have proven that applying Jaccard similarity algorithm is an effective way to generate a fixable signature for polymorphic worms. We have observed that it is inefficient to make the polymorphic worm signature rigid. Our solution contains four phases. While the first three phases are to generate a precise signature, the last phase is to evaluate the efficiency of the generated signature. We used two types of polymorphic worms to test the proposed algorithm. The resulted signature was accurate and produced no false alarm. The signature generated by our system can be deployed with commonly used IDSs with ease.

Our main goal in the signature generation process was to achieve accurate signature that suits most of the polymorphic worm instances. There is a potential *trade-off* between computational overhead and generating a precise signature. For any kind of mobile system or any network with mobility, a critical issue would be the lower resource consumption. As stated in the subsection 4.3, we took into account the system resource limitation and computational overhead, which makes our design implementable in mobile networks and systems.

The performance of our signature could further be improved by applying code optimization techniques. In future, we would like to work in this direction to improve the proposed mechanism and test it under various kinds of dynamic and mobile settings.

References

[1] P. Szor. The Art of Computer Virus Research and Defense. Upper Saddle River, NJ: Addison-Wesley Professional, February 13, 2005.

[2] "A new era of computer worms: Wireless mobile worms", 2005. [Online]. Available: http://searchsecurity.techtarget.com/feature/A-new-era-of-computer-worms-Wireless-mobile-worms [last accessed: 22 October, 2016]

[3] F. Perriot and P. Szor, "An Analysis of the Slapper Worm Exploit," Symantec Security Response, 2003.

[4] F. Syed, "Understanding Worms, Their Behaviour and Containing Them," Project Report, 2009. [Onine]. Available: http://www.cse.wustl.edu/~jain/cse571-09/ftp/worms.pdf [last accessed: 11 March, 2016]

[5] "Global Information Assurance Certification Paper," SANS Institute, 2003. [Online]. Available: https://www.giac.org/paper/gsec/3091/ms-sql-slammer-sapphire-worm/105136 [last accessed: 11 March, 2016]

[6] M. Bailey, E. Cooke, F. Jahanian, and D. Watson, "The Blaster Worm: Then and Now," IEEE Sec. and Pri. Mag., vol. 3, no. 4, pp. 26-31, 2005.

[7] C. Dougherty, J. Havrilla, S. Hernan and M. Lindner, "W32/Blaster worm," CERT, 2003. [Online]. Available: http://www.cert.org/historical/advisories/ca-2003-20.cfm [last accessed: 11 March, 2016]

[8] Nazario. Defense and Detection Strategies against Internet Worms. Boston, MA: Artech House, October 2003.
[9] M.M.Z.E. Mohammed, H.A. Chan, N. Ventura, and A.-S.K. Pathan, "An Automated Signature Generation Method for Zero-day Polymorphic Worms Based on Multilayer Perceptron Model," ACSAT2013, December 22-24, 2013, Kuching, Sarawak, Malaysia, pp. 450-455.
[10] M. Mohammed and A.-S.K. Pathan. Automatic Defense against Zero-day Polymorphic Worms in Communication Networks. ISBN 9781466557277, CRC Press, Taylor & Francis Group, USA, 2013.
[11] H. Orman, "The Morris worm: A fifteen-year perspective," IEEE Security & Privacy Magazine, vol. 1, no. 5, 2003, pp. 35-43.
[12] J. Newsome, B. Karp, and D. Song, "Polygraph: automatically generating signatures for polymorphic worms," 2005 IEEE Symposium on Security and Privacy, 2005, pp. 226-241.
[13] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez. "Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience," IEEE Symp. on Sec. and Pri., Oakland, CA, 2006.
[14] L. Cavallaro, A. Lanzi, L. Mayer, and M. Monga, "LISABETH: Automated Content-Based Signature Generator for Zero-day Polymorphic Worms," Proc. of the fourth int. workshop on Software engineering for secure systems, Leipzig, Germany, 2008, pp. 41-48.
[15] Y. Tang and S. Chen, "An Automated Signature-Based Approach against Polymorphic Internet Worms," IEEE Transactions on Parallel and Distributed Systems, Volume 18, Issue 7, July 2007, pp. 879-892.
[16] Y. Tang and S. Chen, "Defending against Internet Worms: A Signature-Based Approach," IEEE INFOCOM, Volume 2, 2005, pp. 1384-1394.
[17] Snort, Available: https://www.snort.org/ [last accessed: 11 March, 2015]
[18] J. Newsome and D. Song, "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software," In Proc. of NDSS'05, 2005.
[19] Z. Liang and R. Sekar, "Fast and Automated Generation of Attack Signatures: A Basis for Building Self-Protecting Servers," Proc. of the 12th ACM CSS'05, pp. 213-222.
[20] Z. Liang and R. Sekar, "Automatic generation of buffer overflow signatures: An approach based on program behavior models," 21st Annual Computer Security Applications Conference, 2005.
[21] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang and P. Barham, "Vigilante," ACM Transactions on Computer Systems, vol. 26, no. 4, Article No. 9, 2008.
[22] M.M.Z.E. Mohammed, Automated Signature Generation for Zero-day Polymorphic Worms Using a Double-honeynet. PhD dissertation, University of Cape Town, 2012.
[23] C. Rong, W. Lu, X. Wang, X. Du, Y. Chen, and A.K.H. Tung, "Efficient and Scalable Processing of String Similarity Join," IEEE Trans. on Knowledge and Data Engineering, vol. 25, no. 10, 2013, pp. 2217-2230.
[24] J. Leskovec, A. Rajaraman, and J. D. Ullman, Mining of Massive Datasets. ISBN: 9781107077232, 2nd edition, Cambridge University Press, November 2014.
[25] R.B. Zadeh, and A. Goel, "Dimension independent similarity computation," The Journal of Machine Learning Research, Volume 14, Issue 1, January 2013, pp. 1605-1626.
[26] MATLAB - The Language of Technical Computing, Available at: http://www.mathworks.com/products/matlab/?requestedDomain=www.mathworks.com [Last accessed: 3 March, 2016]
[27] C. Fosnock, "Computer Worms: Past, Present, and Future," 2005. [Online]. Available: http://www.infosecwriters.com/text_resources/pdf/Computer_Worms_Past_Present_and_Future.pdf [last accessed: 11 March, 2016]
[28] "Beating the Superbug: Recent Developments in Worms and Viruses," 2002. [Online]. Available: https://www.sans.org/reading-room/whitepapers/malicious/beating-superbug-developments-worms-viruses-146 [last accessed: 11 March, 2016]
[29] C.P. Pfleeger, S.L. Pfleeger, and J. Margulies, Security in Computing. 5th edition, Prentice Hall, February 5, 2015.
[30] L. Wang, Z. Li, Y. Chen, Z. Fu, and X. Li, "Thwarting Zero-Day Polymorphic Worms With Network-Level Length-Based Signature Generation," IEEE/ACM Trans. on Net., vol. 18, no. 1, 2010, pp. 53-66.
[31] P. Li, M. Salour, and X. Su, "A survey of internet worm detection and containment," IEEE Com. Surv. & Tut., vol. 10, no. 1, 2008, pp. 20-35.
[32] I.A. Farag, M.A. Shouman, T.S. Sobh, and H.Z. El-Fiqi, "Intelligent System for Worm Detection," International Arab Journal of e-Technology, Volume 1, Number 1, 2009, pp. 58-67.
[33] R. Kaur and M. Singh, "Efficient hybrid technique for detecting zero-day polymorphic worms," 2014 IEEE IACC, 2014, pp. 95-100.
[34] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee, "Polymorphic blending attacks," Proc. 15th USENIX-SS'06 - Volume 15, Article No. 17, 2006.

Table 7. Comparisons among polymorphic worm detection systems.

	Network/ Host based	Content/ behavior based	False Positive	False Negative	Pros	Cons
Polygraph [12]	Network	Content	Low (between 92.5% and 0%, depends on the used method)	Low	• Applies greedy approach for signature generation process to minimize computational expense.	• Multiple fixed substrings should be found in all polymorphic worm instances.
Hamsa [13]	Network	Content	Low (average = 0.09%, maximum = 0.7%)	Zero	• The resulted signature does not get affected by the size of the normal traffic.	• The resulted signature gets affected by the size of the suspicious traffic. 100 samples of a worm as minimum.
LISABETH [14]	Network	Content	Low (Average= 0.095%)	Low	• Signature generation process is less than by 20%	• Same as Hamsa.
PADS [15]	Host	Content	Low (did not exceed 0.0003)	Low (did not exceed 0.0003)	• Able to capture any possible value of the variable elements in a worm.	• Not capable of detecting advanced worms. • Cannot be merged with other IDSs like for instance, Snort. • High computational overhead.
TaintCheck [18]	Host	behavior	Low (.0017%)	Low	• Capable of detecting any overwrite attack	• Signature generation process cannot be achieved automatically. • Very much application-specific: a certain type of server must be used.
COVERS [19]	Host	Content	Low	Low	• Fast generation of signatures.	• Cannot be used by IDSs or firewalls. • Not for general purpose - it is application-specific. • Need for manual involvement • Depends on application's source code.
ARBOR [20]	Host	behavior	Zero	Low (but more than COVERS)	• Fully automatic signature generation.	• Cannot evade attacks that are fragmented through multiple packets.
Vigilante [21]	Host	behavior	Zero	Low	• Can deal with three different worm infection mechanisms.	• Depends on threads scheduling order.
Double-honeynet [22]	Both	Content	Zero	Low (0.92%)	• produces accurate worm signatures. • Network-based and Host-based.	• Relatively long deployment time for the honeynets.
Our Scheme	Network	Content	Zero	Low	• Each sub-signature has a different occurrence probability. • Very accurate signature. • Different security levels.	• Expected computational overhead[4].

[4]Refer to the future work section, where we suggest some solution.