2008

Encrypted mal-ware detection

Bhuvaneswari Ramkumar
Iowa State University

Encrypted mal-ware detection

by

Bhuvaneswari Ramkumar

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Yong Guan, Major Professor
Daji Qiao
Ying Cai

Iowa State University

Ames, Iowa

2008

TABLE OF CONTENTS

LIST OF TABLES v

LIST OF FIGURES vi

ABSTRACT vii

CHAPTER 1. INTRODUCTION 1

1.1 Introduction 1

1.2 Related Work 2

1.2.1 Mal-ware detection 2

1.2.2 Identifying encrypted content 3

1.2.3 A Glimpse at Searchable Encryption 4

CHAPTER 2. PROBLEM STATEMENT 6

2.1 System Model 6

2.2 Threat Model 7

CHAPTER 3. OUR SOLUTION 9

3.1 Classes of signature 9

3.2 Need for a Generalized Signature Grammar 11

3.3 A proposal for a Generalized Mal-ware Signature Grammar 12

3.3.1 A New Notation 12

3.3.2 Illustration of the Generalized Mal-ware Signature Grammar 13

3.3.3 Token Subsequence Format or Ordered Signatures 13

3.3.4 Conjunction Format or Unordered Signatures 14

3.3.5 Bayes Format or Ranked Signatures 15

3.4 A Glance at Bloom Filters . . . 16
3.4.1 Introduction to Bloom-Filters . . . 16
3.4.2 What is a Bloom-Filter ? . . . 16
3.4.3 False-Positives in Bloom-Filter . . . 16
3.4.4 Optimizing Bloom-Filters . . . 17
3.4.5 Alternate Design Choices: Hash Table Vs Bloom-Filters . . . 17
3.4.6 Bloom Filters for Signature matching: Related work . . . 18
3.5 System Architecture and Mechanism . . . 18
3.5.1 Sender Side . . . 19
3.5.2 Monitor Side . . . 20
3.5.3 Sender's Bloom Filter: Some design choices and decision . . . 22
3.5.4 Worm-detection at the Monitor: A detailed look at signature-detection 24
3.5.5 Dealing with False-negatives and False-positives . . . 24
3.5.6 A possible scenario for false-negatives occurrence . . . 24
3.5.7 Absence of false negatives in our model . . . 25
3.5.8 Converting a possible false-negative case into a definite false-positive case 25
3.5.9 A false-positive aware model : Erring on the safe side . . . 26

CHAPTER 4. Optimization . . . 27

4.1 Bloom Filter Optimization . . . 27
4.1.1 Parallel Bloom Filters . . . 27
4.1.2 Hashing Algorithm design . . . 28
4.2 Query Optimization . . . 29
4.2.1 A Matrix representation . . . 29
4.2.2 A Survey of Sparse-Matrix Optimization Techniques . . . 30
4.2.3 Algorithms to optimize a square sparse matrix . . . 31

CHAPTER 5. Experiments and Discussions . . . 36

5.0.4 Experimental Set-up . . . 36
5.0.5 Sender Side Implementation . . . 36

5.0.6 Monitor Side Implementation 37
5.0.7 The working of our scheme with details 37
5.0.8 Experiment 1 38
5.0.9 Experiment 2 39
5.0.10 Experiment 3 41
5.0.11 Experiment 4 42

CHAPTER 6. DISCUSSION AND CONCLUSION 45

6.1 A Discussion on Alternative design choices and considerations 45
6.1.1 Index propagation Model 45
6.1.2 Exclusive Encryption Layer 45
6.1.3 Communicative Model 46
6.2 Conclusions and Future work 46

BIBLIOGRAPHY 48

LIST OF TABLES

Table 5.1 Effect of changing sender's Bloom-filter size on the false positive rate of our model 38

Table 5.2 Effect of changing Monitor's Bloom filter Size on the false positive rate of our model 40

Table 5.3 Results of first set of detection experiments for different number of worms sent 41

Table 5.4 Results of second set of detection experiments for different number of worms sent 43

Table 5.5 Comparison of system overhead in querying sparse matrix and its optimized representation 44

LIST OF FIGURES

Figure 2.1 A General System Model with a network-sniffer Monitor (M) in a Sender-Receiver system. 7

Figure 3.1 Our proposed Architecture with Bloom-filters 15

Figure 3.2 A typical IP packet as used in our detection mechanism 19

Figure 3.3 Layer of operation for our detection mechanism 19

Figure 3.4 A detailed diagram of the Sender-side functionality. 21

Figure 3.5 A detailed Flow-chart of Monitor-side functionality. 22

Figure 4.1 A Model of our keyword-matrix. 29

Figure 4.2 A Step-by-Step Illustration of Sparse Matrix optimization. 33

Figure 5.1 Simulation results of Table 1 experimental data: Effect of changing Sender's Bloom filter Size on False Positive Rate of our scheme 39

Figure 5.2 Simulation results of Table 2 experimental data: Effect of changing Monitor's Bloom filter Size on False Positive Rate of our scheme 40

Figure 5.3 Simulation results of Table 3 experimental data: Results of detection experiments for different number of worms sent at a specific sender (S)'s bloom filter BF_2 size=32 bits 41

Figure 5.4 Simulation results of Table 3 experimental data: Results of detection experiments for different number of worms sent at a specific sender (S)'s bloom filter BF_2 size=64 bits. 42

ABSTRACT

Mal-ware such as viruses and worms are increasingly proliferating through out all networks. Existing schemes that address these issues either assume that the mal-ware is available in its plain-text format which can be detected directly with its signature or that its exploit-code execution is directly recognizable. Hence much of the development in this area has been focussed on generating more efficient signatures or in coming up with improved anomaly-based detection and pattern matching rules. However with "secure data" being the watch-word and several efficient encryption schemes being developed to obfuscate data and protect its privacy, encrypted mal-ware is very much a clear and present threat. While securing resources from encrypted threats is the need of the hour, equally critical is the privacy of content that needs to be protected. In this paper we discuss encrypted mal-ware detection and propose an efficient IP-packet level scheme for encrypted mal-ware detection that does not compromise the privacy of the data but at the same time helps detect the presence of hidden mal-ware in it. We also propose a new grammar for a generalized representation of all kinds of malicious-signatures as described in [27]. This signature grammar is inclusive of even polymorphic and metamorphic signatures which do not have a straight-forward one-to-one mapping between the signature string and worm-recognition. In a typical system model consisting of several co-operating hosts which are un-intentional senders of mal-ware traffic, where a centralized network monitor functions as the mal-ware detection entity, we show that for a very small memory and processing overhead and almost negligible time-requirements, we achieve a very high detection rate for even the most advanced multi-keyword polymorphic signatures.

CHAPTER 1. INTRODUCTION

Mal-ware such as viruses and worms are considered to be one of the chief threats to the security of a system and is one of the most widely studied topics in the field of Network Security. In this paper, we propose a method to efficiently and accurately detect Encrypted Mal-ware without compromising on the privacy of the data which is encrypted.

1.1 Introduction

The term mal-ware (an abbreviation of the phrase "malicious software") can collectively refer to viruses, worms, Trojans or any other piece of code that intentionally perform malicious tasks on a computer system. Mal-ware propagation and detection is one of the most extensively studied fields in the field of network security. Mal-ware can occur in different forms, viruses that come as an attachment with some other application, worms that propagate independently through-out the network or even malicious scripts embedded in the middle of plain-text that can be used to exploit a vulnerability (as is the case with mal-ware like trojans and back-doors). Encrypted mal-ware is a class of mal-ware whose payload data and the signature used to identify it are concealed as randomly distributed encrypted strings and hence cannot be detected with simple signature-based detection schemes.Infact according to [22] almost 80 to 90 percent of all mal-ware occurs in the encrypted form. Current methods to detect mal-ware can be broadly classified into two-types, static-signature- based detection schemes and dynamic-anomaly based detection schemes. At times a combination the two schemes is employed in which mutual verification is used to reduce the error-rate of the detection. In the former method, looking for fixed strings or keywords uniquely identifying the mal-ware by its signature forms the crux of the detection mechanism.In the latter method, 2 patterns

of behavior such as -control or looping-constructs of a piece of code are examined and these pattern-matches as opposed to fixed strings identify the presence of mal-ware.

1.2 Related Work

1.2.1 Mal-ware detection

Signature-based detection schemes include [27], [13]and have the advantage that they are simple and can be deployed online to detect live-attacks. But their disadvantage is that these cannot detect zero-day attacks and need to be updated frequently with the most-recently developed signatures. Besides unless otherwise programmed they cant detect advanced polymorphic, metamorphic or self-encrypting worms which change their signatures routinely. Anomaly based detection schemes such as [30], [36], [11], [37] have the merit that they do not require any fixed signature-database and can detect any zero-day attack, but these are much more complex systems to be deployed since behavior analysis is not a quick one-to-one matching process. Sometimes it needs to be configured to suit the specific nature of a network and its underlying traffic pattern. Often, this cannot be done on-line resulting in delay and considerable overhead. Also, according to [35] these also tend to generate more false positives. Papers such as [35] attempt to strike a balance between the two methods to arrive at the more desirable trade-off. However both these detection mechanisms will fail in the presence of encrypted mal-ware. Signature-based schemes will straightforwardly fail to recognize matching strings since they are encrypted, unless we have some way to match signatures with all of its encrypted variants. But this could be easier said than done, considering the vast diversity of encryption mechanisms including the new ones being developed. Anomaly based methods that have been proposed so far do not consider control that includes possible decryption mechanisms and directly only look for matching patterns of exploit-code execution. Methods such as, [19] which do a content-based payload-partition to develop a signature or earliest- pattern-matching schemes like, [2] would not help us identify encrypted ma-ware at all. In the former, the authors look for the most prevalent signature byte in the stream, but this would fail if the worm morphs itself for each instance of its infection. In the latter, the authors conclude that

exploit-code detection should be recognized at the earliest possible execution time of an entity. On the other hand, most encrypted mal-ware come with multiple advanced and inter-changing encryption and decryption segments or sometimes the mal-ware even gets interleaved between these flows and hence such a scheme would also fail in this scenario.

1.2.2 Identifying encrypted content

Identifying encrypted content and differentiating it from content that is available in its plain-text as-is form has been studied before. [21] , [33], [24] are some of the papers which have been published in this area. But most often these papers aim to recognize and distinctly identify encrypted traffic by looking at HTTP port values or protocol deviations but do not deal with identifying mal-ware signatures in encrypted traffic. [24] actually speaks about detecting encrypted SSL traffic which involves storing private-key sensors that recognize these in the network. But in this technique the authors attempt to decrypt the packet and then detect mal-ware signatures. This is not a feasible option since in the system model, protecting the privacy of hosts is an important condition. Several statistical methods like the use of entropy, randomness and even byte-distribution have been used to identify encrypted data from plain-text data but as mentioned before, these stop at detecting encrypted traffic and do not proceed beyond that. In the secure system model that we are considering, all or most of the traffic can be assumed to be encrypted and hence such schemes will not be needed. [22] deals with identifying encrypted mal-ware using entropy as a metric. The authors evaluate degree of randomness as a determining factor to decide whether content is encrypted or not. Apart from not catering specifically to our problem-statement, the authors have also designed a technique that works only for Windows PE files. Such a platform-dependent solution may not be very useful for extensive-deployment in modern-day heterogenous networks. Also, there is a class of mal-ware called self-decrypting mal-ware which contain a decryption routine that un-packs the encrypted exploit code part, both of which are present in the same file or segment. We do not know how entropy or any other information- 4 distribution metric can clearly identify such mixed-content payloads. [39] which specifically deals with self-decrypting exploit code

is based on the assumption that the decryption routine can be successfully identified as a signature. The authors then proceed to develop a back-ward tracing method from this to actually detect the mal-ware. But the decryption routine itself is a simple, and ubiquitously occurring sequence of bytes such as a routine XOR decryption. They usually have a fixed key pre-configured within the routine for the decryption. Using this as a signature to identify or detect a self-decrypting exploit code can lead to a huge number of false-positives since such a piece of code is widely prevalent even in non-malicious packets. Also the presence of decryption routine itself is common only to this specific class and does not include content which only have the encrypted payload alone. Several other methods to detect encrypted packet content like the chi-square distribution and cumulative-byte-frequency distribution have been discussed before as methods to detect encrypted content.

1.2.3 A Glimpse at Searchable Encryption

Considering the requirements of our system, we propose an idea extending from a popular area of research generally called as "Searchable encryption ". A huge body of work has been done in this field and in this paper, we discuss how we can efficiently apply indexing as a means to detect encrypted mal-ware traffic as opposed to secure-indexing where the word in the index is itself encrypted and stored. Searchable encryption and Secure Indexing have been widely discussed in [3], [1], [4], [6], [20] , [10], [14], [17], [18], [28], [32]. Some of these papers propose a method by which data whose privacy needs to be preserved is encrypted in one the searchable-encryption schemes and before it gets encrypted, the data is indexed. This index along with the searchable encryption helps detect content in an encrypted packet. In fact [32] does not even require that data needs to be indexed before its encryption, it uses a simple XOR encryption, search and decryption to serve the same purpose. [6] and [10] have proposed schemes for provisioning symmetric encryption with search capabilities. [1] and [6] discuss a typical application,where a user publishes a public key while multiple senders send e-mail. This also works in the domain of searchable-encryption. Searchable symmetric encryption allows a party to out-source the storage of its data to another party (a server) in a private manner,

while maintaining the ability to selectively search over it and hence secure indexing finds an application here. For our system-model, we realized that constraining the nodes to be only searchable-encrypted might be a limiting factor that we are imposing on all the hosts in the network. Instead if we just ask that the nodes simply index their data before they encrypt and send the index in its plain-text form along with the encrypted data, then such a problem could be easily over-come. This follows directly from our assumption that while mal-ware might be present in the packet in its encrypted form, there is no real threat to the packet itself or its transmission and it is just the packet's encrypted payload whose privacy needs to be protected. So securing the index is not an issue for us. In-fact in most cases all that is needed is the optional field in the IP packet for us to put in a bloom-filter, this has absolutely no additional memory requirement. Once indexed this way, the method now becomes encryption-protocol independent which can work for any kind of secure encrypted data transfer. This gives more autonomy to every node in the network to choose a secure encryption of its choice.

CHAPTER 2. PROBLEM STATEMENT

The goal of our proposal can be summed up in the following statement: Given a set of non-malicious, co-operative and un-intentional senders of encrypted malicious traffic, how would we detect the presence of the mal-ware in the packet payload while also ensuring that the privacy of the data-transfer in packet is unaffected.

2.1 System Model

The System model we are considering has the following components:

- Multiple Hosts in the network acting as both Senders and Receivers of encrypted traffic from other hosts in the network.These are typical thin clients that are resource-constrained.
- A Monitor that sits in the middle of the network that is capable of sniffing and analyzing all packets that pass through the network.
- The Monitor is usually powerful enough to be aware of the exact encryption mechanism of all the hosts in the network, it is a trusted mal-ware sniffer and might be even in possession of the required keys to break the encryption but does not do so to meet our design goal of privacy.
- An extensive Worm-Signature database that can be dynamically updated and to which the Monitor alone has access to.

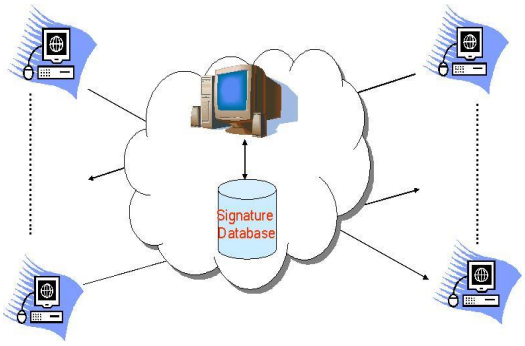

Figure 2.1 A General System Model with a network-sniffer Monitor (M) in a Sender-Receiver system.

2.2 Threat Model

The threat model which we are pursuing consists of the following components:

- A Sender wanting to transfer data to another host in the network decides to encrypt the data in a mutually agreed upon encryption mechanism.
- However the naive sender or the host is not aware of the nature of its content and is incapable of evaluating whether it is malicious or not since it has no access to an extensive worm-signature database.
- The Sender might not be always considered as the source generating the encrypted traffic, it could simply be a gateway node that encrypts and forwards traffic into its domain network.Hence the sender can be a non-malicious entity.
- Multiple innocent nodes communicating to each other in a secure fashion (either through symmetric or asymmetric means), these are typical thin clients that are resource-constrained.
- A Single centralized all-aware Monitor equipped with a dynamically updated signature-database without any critical resource constraints. This could typically be a router or even the Internet Service Provider or even a network fire-wall

This system model is one of the most-commonly prevalent and is typical of corporate networks, secure Local Area Network (LANs) and even Publish-subscribe systems where many

of the clients are thin subscribers while a server acts as the all-powerful system and is usually not as resource-constrained as the other hosts of the system. This is also a popular case in a real-time networks like Internet also, because very few nodes intentionally spread malicious traffic and predominant nodes do malicious acts without their will or intention. Examples include stepping stones which are used as intermediate levels between the source and the destination to spread worm traffic to avoid trace-back detection or bot-nets where slave bots routinely act under the will of controlling bots to spread the infection.It is a very realistic scenario that an all-powerful network sniffer sits in the middle of the network. Sometimes, even the router that flags each packet can act as the Monitor or the same functionality can be done by any general packet-logger or in some real-time cases, a network fire-wall or a gateway.The overhead of our scheme, as we will show in our experimental results in the later sections, is almost negligible compared to the variety of mal-ware that this scheme is capable of detecting.

CHAPTER 3. OUR SOLUTION

As a part of the solution to the encrypted mal-ware detection problem, we propose two important features:

- A Novel system model that uses the existing IP packet structure to efficiently detect encrypted mal-ware while keeping their privacy intact.
- A proposal for a generalized all-encompassing signature grammar that captures malicious attributes and the object defined in a simple format.

3.1 Classes of signature

In this paper we propose a new grammar for all the different types of polymorphic mal-ware signatures based on the classification given by [27]. According to the authors, samples of the same polymorphic worm often share some invariant content due to the fact that they exploit the same vulnerability. Its because we implement this advanced signature scheme in our grammar, we are also able to detect such advanced polymorphic worms also in our scheme. It general the byte-pattern of any exploit can be classified into 3 types:

- Invariant bytes are those fixed in value, which if changed, cause an exploit no longer to function. These byte-patters are useful as static signatures.
- Wildcard bytes are those which may take any value without affecting the correct functioning of a worm.
- Code bytes are the actual polymorphic code executed by a worm through its engine.

Based on the above given classification and combined with the fact that all polymorphic variants of the worm contain some invariant bytes in one fashion or the other, the authors have arrived at the following three classes of signatures:

- Conjunction signatures: A signature that consists of a set of tokens, and matches a payload if all tokens in the set are found in it, in any order.
- Token Subsequence Signature: A signature that consists of an ordered set of tokens.
- Bayes Signature: A signature that consists of a set of tokens, each of which is associated with a score, and an over-all threshold.

For instance consider the case where a mal-ware has as a part of its signature, tokens $t_1....t_m$ appearing in the packet payload to get exactly identified as that entity. Each of these tokens can be a multi-byte keyword that chains up in a certain way to form the mal-ware. Now we can enforce three different rules on these m tokens. We can say that the tokens should have a sequential ordering, that is t_1 should be followed by t_2 and so on till t_m. This is the token-subsequence or ordered format. When we only require that all of t_1 to t_m appear in the payload but we ignore the ordering or sequence arrangement of the tokens themselves, this is a conjunction signature. When each of the tokens are assigned a weighted value and a threshold assigned for the mal-ware, the presence of enough tokens that meet the threshold imply the presence of a worm and this is called the bayes format. In this case, we do not need all of the tokens to be present nor do we have a positional ordering for them. In the coming sections, we will describe these in more detail. For a simple illustration, consider various classes of signature and their representation of the Apache-Knacker worm as given in [27]:

Token Subsequence:

$$W_i = (GET. * HTTP/1.1/r/n * /xFF/xBF * /r/nHost) \tag{3.1}$$

Conjunction:

$$W_i = (GET., HTTP/1.1/r/n, /xFF/xBF, /r/nHost) \tag{3.2}$$

Bayes:

$$W_i = (GET. : 0.0035, /xFF/xBF : 3.1517, /r/nHost : 0.0022, Threshold : 1.9934) \quad (3.3)$$

3.2 Need for a Generalized Signature Grammar

Much of the work in the field of signature-based detection has been focussed on generating more efficient and accurate signatures which characterize and quickly identify a mal-ware entity's signature with the lowest possible false-positives and false-negatives. The other dimension is to automate the signature extraction method by heuristic analysis, behavior modeling and training. In this paper we look at another very critical but often over-looked dimension of Mal-ware signatures, viz how are these signatures going to get represented in a generalized, efficient and all-encompassing format. One of the first questions that might arise in the minds of the readers is why a generalized grammar format for mal-ware signature is required in the first place. Lets look at some common signature-storing utilities like Snort [31] and Bro [7]. Snort's rule-database is a huge set of individual text files that have different fields that characterize every possible anomalous behavior of the mal-ware. Each of these rule-files include several attributes of the packet like header length, round-trip time,packet-counters and such, all of which have to show some specific behavior for the intrusion to be matched and detected according to the rule-set. Though we are not analyzing the efficiency of the detection itself, we have to understand that such a representation is obviously not the most efficient one we can come up for storing the signatures. Snort has millions of such rules and a large number of files each storing in its format, the signature of a specific exploit and these files classified into multiple folders differentiated by the intrusion or attack type. Bro is not very different either. Also both these formats do not accommodate polymorphic or metamorphic variants as different forms of the same exploit. So due to all these draw-backs, there is a critical need to come up with a much more memory-efficient grammar to act as a common representation.

3.3 A proposal for a Generalized Mal-ware Signature Grammar

The signature representation in both [31] and [7] is in-efficient at-least in terms of memory utilization. Firstly it requires multiple files to store the different types of as opposed to tuples in a table or database that can have a much lower requirement. This also saves a lot of data-transfer and network load since Snort has a mechanism to issue a new set of rule-set whenever a much updated set of rule-set is issued. Most critically , for multiple variants of the same mal-ware as is the case with most polymorphic signatures, Snort or Bro provides only for an individual representation of each of these variants as a new rule thought it does say that they might be similar by nomenclature. So at present there is no common format or general grammar that can encompass all the attributes of a signature, including the importance of some tokens being present over other, the way the tokens need to be evaluated and the threshold required to be satisfied to confidently say whether a mal-ware is present or not. To fill this void and provide for a signature-format that can accommodate all types of signature-classes including those for polymorphic worms as mentioned in [27], we come up with a generalized grammar format for mal-ware signatures. The format is explained as follows:

3.3.1 A New Notation

The generalized notation or grammar representation for any Worm W_i's signature can be expressed in the following form:

$$W_i = (K_i, R_i, Type, T) \ , \tag{3.4}$$

This above given equation acts as a general format to represent all kinds of worm signatures. This can be explained as follows:

Each mal-ware signature has a list of m tokens or keywords denoted by:

$$K_i = (k_1, k_2, \cdots, k_m) \ , \tag{3.5}$$

It also has a set of m rank-values associated with each token it denoted by:

$$R_i = (r_1, r_2, \cdots, r_m) \ , \tag{3.6}$$

Each of these keywords and their values need to be evaluated according to the rules governed by their type which is given by:

$$Type = (O/U) \ , \tag{3.7}$$

Here O implies Ordered Signature Type and U implies Unordered Signature Type.

- If the Signature type is Ordered and $Type = O$ corresponding to a certain worm W_i , then it would check if the keywords are indeed present in the same order as given in K_i. If this evaluation returns true, then the worm is assumed to be present
- If the Signature type is Unordered and $Type = U$ corresponding to a certain worm W_i then it would check if the keywords are present in the bloom-filter and their rank-sum is greater than the threshold or equal to it and nothing more .If this evaluation returns true, then the worm is assumed to be present in either static single keyword type or Conjunction type or Bayes type.

3.3.2 Illustration of the Generalized Mal-ware Signature Grammar

Let us consider the signature of the popular Apache-Knacker worm as directly taken from [27].This Worm has in its list of tokens, words such as GET, HTTP/1.1, Host:,0xFFBF. This worm expresses itself as polymorphic worm and depending on which form it takes, it can have any of the following variants of its signature.

3.3.3 Token Subsequence Format or Ordered Signatures

When the Worm occurs as its first variant, it occurs as a Token-Subsequence form.As per our grammar we can represent it in the following way:

$$K_i = (GET., HTTP/1.1/r/n, /xFF/xBF, /r/nHost) \ , \tag{3.8}$$

$$R_i = (1, 1, 1, 1, 1) \ , \tag{3.9}$$

$$Type =' O' \tag{3.10}$$

$$T = 5 \tag{3.11}$$

This implies that the above listed keywords are present in the same order as indicated by the (*) operator, viz, strictly ordered in the same format. In its ordered form, should be strictly followed by HTTP/1.1 and so on till all the strings have been found in the same sequence and only then the worm is assumed to be found in this polymorphic form.

3.3.4 Conjunction Format or Unordered Signatures

When the Worm occurs as its second variant, it occurs as a Conjunction form. As per our grammar, we can illustrate this format of the signature in the following way.

$$K_i = (GET., HTTP/1.1/r/n, /xFF/xBF, /r/nHost) \ , \tag{3.12}$$

$$R_i = (1, 1, 1, 1, 1) \ , \tag{3.13}$$

$$Type =' U' \tag{3.14}$$

$$T = 5 \tag{3.15}$$

This implies that the above listed keywords are present in the same order as indicated by the (,) operator. In this class, all the above mentioned tokens GET, HTTP/1.1 should be present in the payload, their order does not matter and the worm is assumed to be present if just all tokens are present in any order.

3.3.5 Bayes Format or Ranked Signatures

When the Worm occurs as its third and final variant, it occurs in the Bayes form. According to our formal notation, we can represent this as:

$$K_i = (GET., HTTP/1.1/r/n, /xFF/xBF, /r/nHost) \ , \tag{3.16}$$

$$V_i = (0.0035, 0.0022, 3.1517, 0.1108) \tag{3.17}$$

$$Type =' U' \tag{3.18}$$

$$T = 1.9934 \tag{3.19}$$

This implies that the above listed keywords are present in the same order as indicated by the (:) operator. When the worm occurs in the Bayes for, each token is assigned a weight or value and the sum of the value of the tokens found in the payload should exceed the threshold. If so, this alone is just a sufficient condition for the worm to be present. It does not require that all keywords should be present but just enough keywords to meet the threshold.

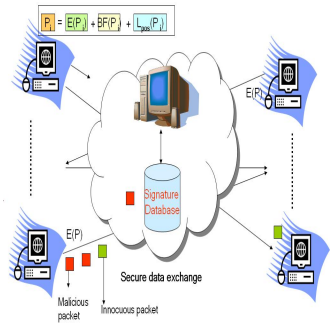

Figure 3.1 Our proposed Architecture with Bloom-filters

3.4 A Glance at Bloom Filters

3.4.1 Introduction to Bloom-Filters

A Bloom filter is a simple space-efficient randomized data structure for representing a set in order to support membership queries. Bloom filters allow false positives but the space savings often outweigh this drawback when the probability of an error is made sufficiently low. Burton Bloom introduced Bloom filters in the 1970s [5] and ever since they have been very popular in database applications. Recently they started receiving more widespread attention in the networking literature. [9] gives us a very extensive survey of the network applications of Bloom-filters.

3.4.2 What is a Bloom-Filter ?

A Bloom filter for representing a set S = x_1, x_2, . . . , x_n of n elements is described by an array of m bits, initially all set to 0. A Bloom filter uses k independent hash functions h_1, . . . , h_k with range 1, . . . , m. For mathematical convenience, we make the natural assumption that these hash functions map each item in the universe to a random number uniform over the range 1, . . . ,m. For each element x of S, the bits hi(x) are set to 1 for 1 = i = k. A location can be set to 1 multiple times, but only the first change has an effect. To check if an item y is in S, we check whether all h_i(y) are set to 1. If not, then clearly y is not a member of S. If all h_i(y) are set to 1, we assume that y is in S, although we are wrong with some probability. Hence, a Bloom filter may yield a false positive, where it suggests that an element x is in S even though it is not. Figure 1 provides an example.

3.4.3 False-Positives in Bloom-Filter

For many applications,including ours, false positives may be acceptable as long as their probability is sufficiently small. To avoid trivialities we will silently assume from now on that $k * n$ less than m. [17] gives us the false positive rate of the bloom-filter as a function of the total unique strings hashed (n), the size of the bloom-filter array (m) and also the number of hash-functions (r) used to hash each item into the bloom-filter.

3.4.4 Optimizing Bloom-Filters

Bloom-filters can be tuned to give a certain desirable false positive rate. The parameters that represent this false-positives are influenced by two factors, the number of hash functions used for each element and the size of the bloom-filter array. In most cases, the size of the database or the number of unique strings (n) to be hashed is more or less a constant. Suppose that we are given m and n and we wish to optimize for the number of hash functions. There are two competing forces: using more hash functions gives us more chances to find a 0 bit for an element that is not a member of S, but using fewer hash functions increases the fraction of 0 bits in the array.

3.4.5 Alternate Design Choices: Hash Table Vs Bloom-Filters

While thinking of Bloom-Filters as a design choice for our model, we also considered other alternatives. One of the most important one was a Hash-table. Indeed, a Bloom filter with just one hash function is equivalent to ordinary hash-table.

Hashing is among the most common ways to represent sets. Each item of the set is hashed into T(log n) bits, and a (sorted) list of hash values then represents the set. This approach yields very small error probabilities. For example, using 2 log n base 2, bits per set element, the probability that two distinct elements yield the same hash value is 1/n2. Hence, the probability that any element not in the set matches some hash value in the set is at most n/n2 = 1/n by the standard union bound.

Bloom filters can be interpreted as a natural generalization of hashing that allows more interesting tradeoffs between the number of bits used per set element and the probability of false positives. Bloom filters yield a constant false positive probability even for a constant number of bits per set element. For example, when m = 8n, the false positive probability is just over 0.02. For most theoretical analyzes, this tradeoff is not useful: typically, one needs an asymptotically vanishing probability of error, which is achievable only when we use T(log n) bits per element. Hence, Bloom filters have received little attention in the theory community. In contrast, for practical applications, a constant false positive probability may

well be worthwhile in order to keep the number of bits per element constant.

3.4.6 Bloom Filters for Signature matching: Related work

[15] deals with using parallel bloom-filter search engines to perform signature-matching. However the authors assume that the signature of each entity is a single contiguous word that exactly indicates an entity's presence.We will show in the later sections, the short-comings of such an approach.This might be true for simple static cases but fails miserably for any kind of mutable or complex exploits.Further, built on the lines of Snort, it uses rule-sets as opposed to strings of signatures as we consider in our system. Also the authors have a hardware-implementation of the bloom-filter search engine, this might provide a performance advantage, but FPGAs need to be re-configured and the circuits changed for every addition or modification we need to do. For applications that are dynamic and growing like a mal-ware signature database, we might want a solution that has a low overhead for change or growth. [12] gives an improved algorithm over the Bayer-Moore used by Snort to perform quicker pattern-matching. The authors arrive at ways to efficiently skip over some strings thereby improving the performance but once again they fail to address the problem of complex signature matches as mentioned in [27]. [16] also deals with improving the efficiency of string-matching in packets for intrusion detection but does not deal with encrypted mal-ware specifically.lets take a detailed and more closer look at Bloom-Filters.

3.5 System Architecture and Mechanism

Now lets take a detailed look at each of the components mentioned in our solution and explain where and how exactly Bloom-Filters are used in our system. The primary merit of the scheme we are proposing is, as mentioned earlier, the ability to use the existing IP packet on an as-is basis without any modification or special needs to help detect encrypted mal-ware.Its options fields are filled with the sender's bloom-filter that helps us detect mal-ware.This is explained in detail in the following sections. A typical IP packet in our scheme is illustrated in the figure below.

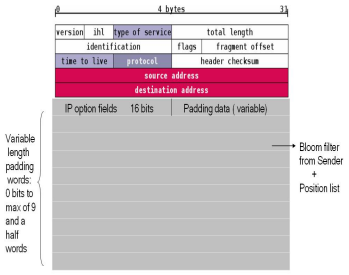

Figure 3.2 A typical IP packet as used in our detection mechanism

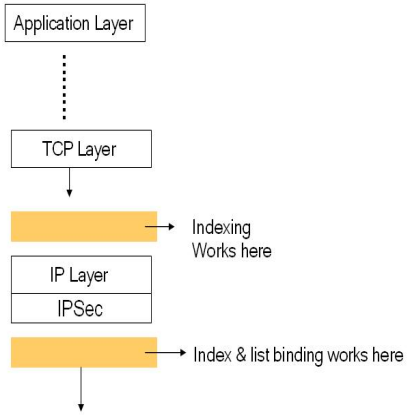

Figure 3.3 Layer of operation for our detection mechanism

3.5.1 Sender Side

The sender as shown in the figure does the following functions:

- Sender Receives a Bloom-Filter of all the signature keyword hashes from Monitor
- Whenever it has a packet to be encrypted and sent, it quickly uses the Bloom filter it received in step 1 and checks if and how many keywords are present in its payload that match the worm-signature
- All the matching keywords are then hashed into a new bloom-filter that the sender generates on a per-packet basis.

- Also parallel to this operation, it stores all bytes or byte-sequences that match an entry in the Monitor's bloom filter into an array and their position in the packet in another array.
- when the entire packet has been parsed this way,it sorts the keyword array in ascending order, by the natural value of the byte and orders the position array's values to correspond with the sorted array of keywords matched.
- This position array is what will be sent along with the packet and the array storing the keywords matched is discarded and no longer used after the sort is done.Lets call this list L_{pos}.
- The Sender that actually builds its encrypted packet and then attaches this bloom-filter in plain-text to the encrypted packet, to the options field of an IP packet. Reasons for choosing this and other alternatives are discussed in a later section.
- Once this is done, after the 32 bits parsed in the options field, whatever the size of L_{pos} is, those many bits from the remaining of the options field are used and the list inserted after the bloom-filter. We are hoping that the size of L_{pos} is never greater than 272 bits which is 304-32 bits. Even assuming that each number is represented as a 8 bit char, we can have a maximum of 34 keywords indexed this way. Now the task of sender is done.
- The sender needs to ensure that the indexing should be done before the encryption actually happens, no matter in which or how many layers and attached in its plain-text form to the outgoing encrypted packet without which the monitor will not achieve its purpose and the index of an encrypted packet will not provide any meaningful security of the system.

3.5.2 Monitor Side

The Monitor's functionality as shown in the figure below is as follows:

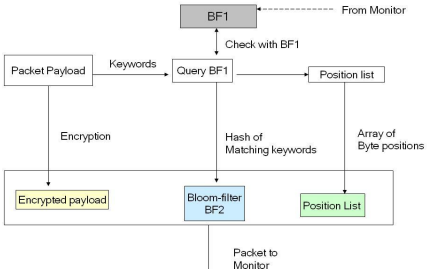

Figure 3.4 A detailed diagram of the Sender-side functionality.

- The Monitor receives the encrypted packet and from it extracts the sender's bloom filter alone.
- Once this is done, the monitor then queries the received Bloom-Filter for every worm-keyword in its database.
- It then populates an array that is filled with all the keywords that the monitor believes are present in the sender's bloom-filter. We should note that there is a risk of false-positives because of the basic nature of the bloom-filter and a small probability for false-negatives for which we will describe later on how we intend to tackle it.
- For every keyword found, the monitor queries its database to see if the keyword occurs as a single, static keyword signature for a worm. If this is true, the packet is immediately marked infected and dropped.
- If instead the keyword happens to be a part of a polymorphic token, we identify in which type of signature, this keyword is found and see if the position array has it, if not we quickly abandon this and move on to the next keyword.
- Whenever there is a need to detect an ordered or token-subsequence worm, that is the case when L_{pos} will be used. Since BF_2 is always 32 bits, the monitor is aware that 32 bits of the start of payload, the position list starts. All the keywords found in Monitor are sorted by their natural ordering in the same array. For now with some naivety we

assume that the position values stored in L_{pos} directly correspond to the ordered keyword found array formed at the monitor.

- Of-course this is based on the assumption that there are no false-positives at the Monitor's side. This matching one-to-one position information, along with the list of keywords found is then used to detect the presence of ordered mal-ware signatures.In the section below we describe how to deal with false-positives and a possibility for false-negatives arising out of this detection.

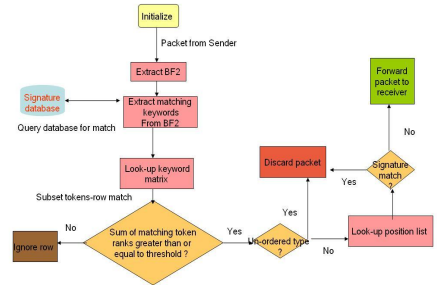

Figure 3.5 A detailed Flow-chart of Monitor-side functionality.

3.5.3 Sender's Bloom Filter: Some design choices and decision

So essentially the detection happens in two phases: the bloom-filter phase and the database-detection phase. In the first phase, Monitor (M) send its keyword database Bloom-filter (BF_1) to sender on an "on-updated" basis. Whenever the Sender (S) intends to send data to another message, it first looks at its payload part.It takes every byte of its payload, hashes it and verifies if there is a match found in the Monitor's database and if so, it constructs a Bloom-Filter (BF_2) of its own on a per-packet basis, hashes that particular keyword and all other keywords that match an entry in BF_1 into BF_2. Here, we assume that the both S and M know each other's index hashing technique since they mutually query each other's Bloom Filter at different stages. Now the question is how to communicate this BF_2 to M. We considered several design choices, if we were to send BF_2 separately as a correlated Bloom Filter for a certain

packet, we would be unnecessarily increasing the traffic on the network and besides M would have the additional task of correlating each packet with its corresponding BF_2. So the best choice would be to send this along with the packet itself. But S usually encrypts its packets but BF_2 should not be encrypted or it will not be read and parsed by M without breaking the encryption of S which is not allowed in this case. We can probably add an un-encrypted BF_2 to the encrypted packet but that would only increase the packet size and we need to also change the payload length and if adding BF_2 meant that data in the packet would exceed the IP packet size, then we had to handle those issues also. Hence we had to insert BF_2 in the packet in the IP packet but not in the payload.So we decided to use the options field in IP packet. The IP option data field consists of 9 and a half words, where each word is 32 bits and is a total of 304 bits excluding the option header. It may or may not be transmitted with every packet, though it should be implemented by every supporting system. Since we are implementing this scheme in a specific LAN or corporate environment we can enforce a simple constraint that all systems use the optional field mandatorily to index a packet. Most mal-ware have a signature not exceeding 100 bytes apart from the payload. In fact our signature database implementation also has an average keyword length and from our experimental results also we have proved that taking the options field to insert BF_2 is indeed a good choice since for most normal cases we arrive at acceptable false positive rates from the conclusions we draw from the experiments. Even considering that we have an unusually high number of matching strings in which the BF_2 might fail with an unacceptable false-positive rate, we have also tested an implementation which has 64 bits length for BF_2, using 64 bits of the options field data payload part leaving the optional fields to be set as the need may be. Of-course we see from experimental statistics that this does not create any huge difference in our detection scheme results for any normal IP packet carrying network such as ours. The payload is then inserted into the IP packet and sent through IPSec which generates an encrypted IP packet either in transport mode or tunnel mode, in both of which the IP header options field can still be used. BF_2 is inserted as 32 bits of options field in this encrypted packet and sent by S. Once at the monitor,the database detection phase begins and the first phase is assumed to be over.

3.5.4 Worm-detection at the Monitor: A detailed look at signature-detection

A packet is received at the Monitor, its options field are taken, and queried for every matching keyword in the database. For every matching keyword, all mal-ware signatures that contain this token are considered and BF_2 is again searched to see if the other tokens matching this mal-ware are present. If the mal-ware being considered has a symbol "O" on its signature type, then we just look for a plain presence of all tokens and if present the packet is reported as malicious and dropped here. If the symbol says "R" , then its matching keywords till their values meet the threshold are alone searched. Once the threshold has been reached, other keywords are not searched for and the packet directly dropped. But if the threshold has not been reached even after querying all the keywords of the worm, then that mal-ware is assumed to be absent and the bloom-filter is searched for other keywords and other matching mal-ware. The first phase is a probabilistic phase since there is a possibility of False-positives. But the second phase is a deterministic phase, for all the keywords assumed to be present in the sender's packet, a worm-signature query is initiated to the database and any matching result is reported for appropriate action directly without any mismatch or errors.

3.5.5 Dealing with False-negatives and False-positives

We know that the bloom-filters both at the sender side and the Monitor side do not produce any false-negatives at all, but there is a non-zero possibility that false-negatives are produced in our scheme itself. This is the case when a worm that is present in the packet is detected as not being present and the packet getting tagged as non-malicious and sent to the receiver. Lets now look at how this happens.

3.5.6 A possible scenario for false-negatives occurrence

Lets say a Worm W_i has in its ordered signature tokens t_1,t_2 and t_m where m is greater than 1 and 2 but less than or equal to n, the total number of keywords in the database. In short if W_i were to be present in the sender's packet,all the above mentioned keyword tokens would be detected at Sender along with some more additional ones due to false positives. All

of their positions are marked in L_{pos}.

Now at the monitor, if we have a 3 element array and I query and detect t_1,t_2 and some t_r where r¡m. In this case, the third keyword sent by sender is m whereas we have detected m earlier than that. Since the position sent by sender is m and what we detect is r, both of them are three in number, matching the list sent by sender with L_{pos}. So in the position where monitor needs to see m it sees r and falsely concludes that there is no mal-ware when the packet is actually infected. This is the case where a false-negative might occur.

3.5.7 Absence of false negatives in our model

According to the above model, false negatives is a possibility as per our description of the working mechanism. But in truth there are no false-negatives in our model. This is because, for every Bloom-filter we receive, we query it for all possible keywords stored in the database and do not stop at a point when a worm is detected. Hence even if according to our scenario description t_r was detected as a false-positive, t_m would also definitely get detected since the bloom-filter itself has no false-negatives and it is the way we design our scheme that may or may not lead to one.

3.5.8 Converting a possible false-negative case into a definite false-positive case

Based on the scenario given above, we infer that at the monitor what we might get is 4 keywords and 3 positions in the L_{pos} array sent by sender. At this stage, the monitor's sorted keyword array will r followed by m. When such a situation occurs, monitor's keyword-found array will have r followed by m but only 1 position to match it with in the L_{pos}. At this stage, the monitor knows for a fact that it definitely has a false-positive since both r and m are present in its array competing for one position value. Now it can take any action that is deemed appropriate. It can see that with m, the packet becomes a worm and with r it does not. We err on the safe side and declare that the packet is malicious. But the knowledge about false-positives should not be ignored and hence might be broadcast to the sender or just stored and logged for statistical purposes.

3.5.9 A false-positive aware model : Erring on the safe side

In any case where a false-positive occurrence is detected and the inclusion of the false-positive case might affect our decision of whether the packet is malicious or not, we always detect it as a malicious packet being present. This might happen even in the reverse case where only r is actually present in the packet and m is detected as a false-positive. Even here, the packet is detected as malicious and this anyway adds to the existing false-positive rate inherent to the bloom-filter. This model follows the principle of erring on the safe side. Only instead of the general case, where a keyword might be detected as true or false positive without the user being aware, in this model we always know when a false positive has occurred because of L_{pos} though we might not know what the exact false-positive case at the monitor might be.The actual false-positively occurring keyword might be any value from t_1 to t_m in an m array keyword found index at the monitor. Hence all combinations of the set of keywords are found and checked if any of them lead to a worm being present. Though this might be trivial with the L_{pos} list already prepared and sent by the sender, this is a case where with a small deliberation in the monitor design, we overcome the possibility of false-negatives and instead make it into a false-positive aware mechanism. Besides this is an efficient trade-off between the sender sending all keywords with their indices and the sender just sending the bloom-filter alone to answer yes or no for the keywords.

CHAPTER 4. Optimization

4.1 Bloom Filter Optimization

There are several techniques to optimize the design of the bloom filters for efficient performance. In this section we discuss several of them. In most cases, a Bloom filter resides in the main memory to provide fast response to the query. Each membership query repeats the following procedure with a certain number of hash functions: hashing the query value for a memory address and one bit access. It is possible to speed up the membership query by reducing the number of hash functions. However, the false positive rate increases unless the space-efficiency is sacrificed as a tradeoff [25].

4.1.1 Parallel Bloom Filters

A parallel design for bloom filters has been introduced in [15].This system relies on a predefined set of signatures grouped by length and stored in a set of parallel Bloom filters in hardware. Each Bloom filter contains signatures of a particular length. At the time of membership query, a window whose length is equal to the maximum length of the data-members, is considered and bytes are filled into this. All the max length substrings starting from 1 are searched simultaneously in their respective length bloom filters in a single cycle and the entire network byte stream advanced by one byte in the window this way. The I/O and processing design is very specific and expensive to implement in a scalable way.

As mentioned in chapter 3, one of the chief drawbacks of this technique is the potential need for a hardware implementation to achieve the same kind of results as the authors. Such a designs restricts flexibility and dynamism in design.

[8] proposes a scheme where instead of hashing using an m bits array for all the k hash

functions, each hashing function has an output ranging of m/k consecutive bits and disjoint with the outputs of the others. Therefore, the k times one bit memory I/O can be done in parallel within one time memory I/O cycle. The chief disadvantage with this scheme is that the range of randomness is not sufficiently large. In short by limiting the range of each hash function to one of the m/k bits within the m array bits, the false positive rate increases in this scheme.

4.1.2 Hashing Algorithm design

[38] propose a new bloom filter design with pair-wise correlation among the hashed memory addresses by squeezing every two addresses into one memory I/O block. To be specific, once an address is computed, the next address computed must be within a certain offset away from the previous one. This is based on the premise that reducing the memory I/O latency is the chief way to reduce the delay created in bloom filter implementations. Therefore, each pair of hashed bits are accessed in one memory I/O cycle with the burst-type I/O mechanism provided in some contemporary memory designs such as Synchronous DRAM (SDRAM). The authors introduce two sets of hash functions, primary and secondary. For every primary hash function that is computed for a data member, the primary hash function indexes to any random address. For each primary hash function, there is a corresponding secondary hash function that hashes to a location within the set of memory bits or word accessed in the first iteration. Even though the secondary hash function's localized range seems to reduces the randomness and hence increase the false positive rate, the primary hash is completely random. So this scheme is a like a trade-off between a parallel bloom filter and the associated increased memory overhead associated with it. Compared with one completely random memory access for each hash result in the standard design of conventional bloom filters, the component of average membership query delay due to memory I/O latency is reduced by up to 50 percent in this new design.

4.2 Query Optimization

In this section, we deal with optimizing the actual implementation details at the Monitor Side corresponding to the signature database we have. The following sub-sections deal with how the querying process in Monitor can be optimized for efficient and fast look-ups.

4.2.1 A Matrix representation

We mentioned that we have a keyword database with matching keywords for each of mal-ware entities worms. Considering the fact that each mal-ware, say a worm, could have any number of keywords that form a part of its signature and any number of these keywords can be common to several worms, the best structure to represent such a complex inter-leaved structure is a mal-ware keyword matrix. This follows from a straight-forward assumption that several of the keywords occurring in a signature are commonly used keywords that can be present in any packet. Even the Apache-knacker worm that we illustrated in the previous section had words like GET and HTTP as a part of its signature which can routinely occur in any packet and stream. This is one of the common masquerading techniques used by worm-authors to prevent detection. Apart from this fact, a matrix structure saves us considerable amount of time with a O(1) look-up and is much more efficient than a standard tuple-populated database. A typical keyword matrix structure is shown below.

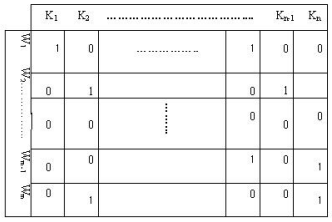

Figure 4.1 A Model of our keyword-matrix.

4.2.2 A Survey of Sparse-Matrix Optimization Techniques

When storing and manipulating sparse matrices on a computer, it is beneficial and often necessary to use specialized algorithms and data structures that take advantage of the sparse structure of the matrix. Operations using standard matrix structures and algorithms are slow and consume large amounts of memory when applied to large sparse matrices. Sparse data is by nature easily compressed, and this compression almost always results in significantly less memory usage. Indeed, some very large sparse matrices are impossible to manipulate with the standard algorithms.Further more depending on the nature of the matrix itself, there are different types of sparse matrix optimization techniques, for symmetric, square and asymmetric matrix structure.

4.2.2.1 Sparse Matrix and Diagonal Matrix

The naive data structure for a matrix is a two-dimensional array. Each entry in the array represents an element say $a(i, j)$ of the matrix and can be accessed by the two indices the key-id and the worm-id. For a m*n matrix we need at least enough memory to store $m * n$ entries to represent the matrix.

From observation we see that though its a $n * m$ matrix, its a matrix pre-dominantly with zeros, or in short its a sparse matrix. Sparse-matrices are a well studied phenomenon and occurs quite often in the several fields in real-time.

Many if not most entries of a sparse matrix are zeros. The basic idea when storing sparse matrices is to only store the non-zero entries as opposed to storing all entries. Depending on the number and distribution of the non-zero entries, different data structures can be used and yield huge savings in memory when compared to a naive matrix approach.

4.2.2.2 Direct Sparse Solvers

As discussed above, it is more efficient to store only the non-zero elements of a sparse matrix. There are a number of common storage formats used for sparse matrices, but most of them employ the same basic technique. That is, store all non-zero elements of the matrix into

a linear array and provide auxiliary arrays to describe the locations of the non-zero elements in the original matrix.

The storing of non-zero elements of a sparse matrix into a linear array is done by walking down each column (column-major format) or across each row (row-major format) in order, and writing the non-zero elements to a linear array in the order they appear in the walk. The Intel MKL sparse matrix storage format for direct sparse solvers is specified by three arrays: values, columns, and row index. The following table describes the arrays in terms of the values, row, and column positions of the non-zero elements in a sparse matrix.

- Value:A real or complex array that contains the non-zero elements of a sparse matrix. The non-zero elements are mapped into the values array using the row-major upper triangular storage mapping described above. columns
- Column: Element I of the integer array columns is the number of the column that contains the I-th element in the values array.
- Row-index: Element j of the integer array row-index gives the index of the element in the values array that is first non-zero element in a row j.

The length of the values and columns arrays is equal to the number of non-zero elements in the matrix.

4.2.3 Algorithms to optimize a square sparse matrix

4.2.3.1 Marriage problem and Latin Squares method

In mathematics, the stable marriage problem (SMP) is the problem of finding a stable matching a matching in which no element of the first matched set prefers an element of the second matched set that also prefers the first element. In short, given a square sparse matrix, this is the problem with which each row is assigned one unique column index which is 1 and vice-versa, ensuring a strict one-on-one match. Once such a match is obtained, we can easily have those index values as a diagonal matrix in which only the diagonals are non-zero and

thus store n elements from a $n * n$ matrix. This problem has been a very ancient one and is generally considered as the means by which efficient and unique one to one matches are made out of subsets of over-lapping matches.This is extensively discussed in [23].

This problem relates to our need to linearize our sparse-matrix , except that in this case, the sparse matrix is a square one and in our case, the matrix is not a square one. This square matching problem has also been called as the Latin Squares method. In the next section, we'll discuss in detail, an algorithm that solves this problem of creating a diagonal matrix.

Let A_1, ..., A_n be subsets of an n-set M. The family of sets A_1, ..., A_n has a system of distinct representatives (SDR) if and only if there exist distinct elements x_1, ..., x_n, such that x_i is in A_i, for each i = 1, ..., n.

Let A_1, ..., A_n be subsets of an n-set M. The family of sets A_1, ..., A_n satisfies the marriage condition if and only if the union of any k of the sets contains at least k elements, for all k = 1, ..., n.

MARRIAGE THEOREM

A family A_1, ..., A_n of sets has a system of distinct representatives if and only if the family satisfies the marriage condition.

Remark Strictly speaking, the proof below does not require the sets of boys and girls to be equipotent. However, this is a sensible assumption. For, if n less than M, the marriage condition fails. On the other hand, if n greater than M, then n minus M desperate girls willing to marry any boy can be added to the set without disturbing the truth or falsity of the marriage condition.

Proof

We can prove this theorem by describing a procedure for constructing a system of distinct representatives which succeeds if and only if the marriage condition is satisfied. Another elegant, though less procedural, proof illuminates various phases of the algorithm. It is convenient to describe everything in terms of zero-one matrices. Let $A = (a_{ij})$ be an n by n matrix whose columns are labeled by the sets A_1, ..., A_n and whose rows are labeled by the elements of M (= union of the Ai). For each i, j = 1,...,n, set $a_{ij} = 1$ if the i-th element of M is in A_j and set

$a_{ij} = 0$ otherwise.

Then the family has a SDR if and only if the matrix A has a transversal of 1's. Here is the procedure together with a proof that it constructs a transversal of 1's if and only if the family satisfies the marriage condition. Let us for instance consider that in the square sparse matrix , we have an upper triangle block A with a transversal of 1's and the lower triangle called as Block B.

GREEDY PHASE

If block B is empty (0 by 0), then we are done, the family has a SDR. If B contains a 1, interchange rows and columns (without disturbing the diagonal of ones in the upper left corner block) to put a 1 in the upper left corner of B, replace B with that obtained by using all but the first row and first column of B, and resume the greedy phase. Otherwise, we proceed to the LABEL PHASE

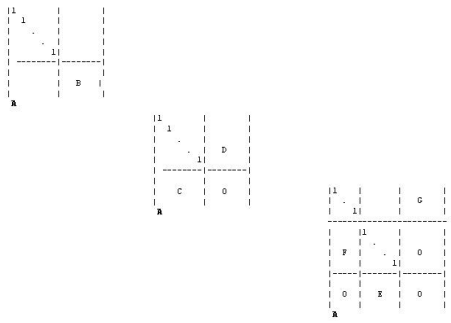

Figure 4.2 A Step-by-Step Illustration of Sparse Matrix optimization.

LABEL PHASE

Consider that the square matrix can be divided into multiple squares labeled from A to G. Having reached here, for every 1 in C there are only zeros in the corresponding row of D. Hence, after row and column interchanges, we have the picture above, with E all those columns of C which contain a 1.

If the block F contains only zeros, then we are done; the family has no SDR because the sets corresponding to the columns of F and one set corresponding to a column of G violates the marriage condition.

If $a_{ij} = 1$ in F and $a_{jk} = 1$ in G, for some k, then swap columns j and k of A. This puts a 1 in the zero block just below G. Now we can swap the k-th and i-th columns of A, putting a 1 in the lower right corner zero block. Resume the greedy phase.

If F has only zeros, then the sub-matrix of A consisting of the rows and columns of A other than the rows and columns of E violate the marriage condition. If for every nonzero column of F, the corresponding row of G has only zeros, then let E1 be the nonzero columns of F and let G_1 be the rows of G corresponding to the columns of E1. If G_1 has a 1, then we can interchange a column of G_1 with a column of E1 to put a 1 in a row that corresponds to a column of E; whence we can do another interchange to put a 1 into the lower right block of A and resume the GREEDY PHASE. Otherwise, let F_1 be the sub-matrix of A consisting of the columns left of E1 and whose rows are the rows of G_1. If F_1 has only zeros, then, as with F, the marriage condition is violated. Otherwise, let E2 be the nonzero columns of F_1 and let G2 be the rows of G corresponding to the columns of F2. If G2 contains a 1, we can do a series of column swaps to put a 1 to the right of E and resume the GREEDY PHASE. Otherwise, we continue the construction of F_i's and G_i's until some Gi has a nonzero entry unless G has all entries zero, a violation of the marriage condition (if a column of G is zero, then the union of one set has no elements, a violation of the marriage condition). Thus, either the marriage condition is violated or we can do a series of swaps that puts a 1 to the right of E so we can resume the GREEDY PHASE.

4.2.3.2 CuthillMcKee and Reverse CuthillMcKee Algorithm

In the mathematical subfield of matrix theory the CuthillMcKee algorithm is an algorithm to reduce the bandwidth of sparse symmetric matrices.

Algorithm

Given a symmetric $n*n$ matrix we visualize the matrix as the adjacency matrix of a graph. The Cuthill-McKee algorithm is then a relabeling of the vertices of the graph to reduce the bandwidth of the adjacency matrix.

The algorithm produces an ordered n-tuple R of vertices which is the new order of the

vertices.

First we choose a peripheral vertex x and set $R := (x)$.

Then for i=1,2,... we iterate the following steps while $|R| < n$

Construct the adjacency set A_i of R_i (with R_i the i-th component of R) and exclude the vertices we already have in R

Sort A_i with ascending vertex order. Append A_i to the Result set R. In other words, number the vertices according to a particular breadth-first traversal where neighboring vertices are visited in order from lowest to highest vertex order.

The reverse CuthillMcKee algorithm (RCM) is the same algorithm but with the resulting index numbers reversed. The authors have claimed that for several practical purposes, the reverse algorithms is a better solution.

Apart from this a huge body of work has been done on sparse-matrix solvers and ways to optimize their storage and improve efficiency. [35] is a research and development group that works on sparse matrix algorithms and provides a list of software packages that work on the different aspects of estimating and optimizing sparse matrices for various research application.

CHAPTER 5. Experiments and Discussions

In order to evaluate the effectiveness of our scheme in a real-time scenario, we conducted several experimental tests that measured the various metrics of the system.

5.0.4 Experimental Set-up

Our general system consists of the sender and the monitor working in a mutually helpful fashion to detect encrypted mal-ware.

5.0.5 Sender Side Implementation

In our implementation of the system, the Sender (S) is a thin-client FreeBSD host and the Monitor (M) is a powerful a Linux desktop system that sits in the same network. The Monitor has a signature database of a modest size of about 1604 signatures. We should note that this is an encrypted mal-ware signature database that matches an entity to its fixed or multiple string signature. Hence building such a database is different from directly taking the database of signatures from Snort or Bro which consist of rules that characterize different intrusions or anomalous behavior. Indeed a very small set of signatures in Snort alone exactly match our requirements for a mal-ware signatures, signatures that exhibit themselves in the payload of an entity, as opposed to specific values for the various protocol-header fields or packet metrics as they are in Snort. Snort has explicitly stated that it removed all virus-rules except some very few and so its entirely built on traffic flows in the network. Our database has been built in MySQL and consists of a tables of size not exceeding 200 Kilobytes.The average length of a mal-ware signature stored in our database is 107 bytes. Most of the signatures we obtained for our database are single-keyword static signatures and the only samples of

polymorphic signature useful for us would be those for which all the 3 classes of ordered, unordered and ranked signatures would have been proved. Hence we directly took from [27] the two polymorphic worm signatures that they generated with all 3 classes of detection. S is a Libnet application that builds an IP packet from scratch, hence we are able to specify the exact payload we want, including the number of keywords or tokens we want to be a part of the packet.

5.0.6 Monitor Side Implementation

Monitor (M) is a powerful a Linux desktop system that sits in the same network as the other hosts that it is trying to help. M has a Pcap application and a C program that runs on the Linux PC. The Pcap application essentially acts as a listener and a packet-filter, capturing all packets that come from nodes within the network. In our case though, it continuously listens for any packet that might come from the S. This can be implemented with a tcpdump filter which captures packets matching a certain expression, followed by a byte-parser that extracts for us the optional bytes of up-to 9 half-words.The header length of a packet can be used to determine how many bytes of the optional field have been used for the keyword position list by the sender.

5.0.7 The working of our scheme with details

S receives BF_1 from M. Along with it, it also receives an array of all possible lengths of its individual keywords. Both of these which can be defined as off-line setup information and configuration exchanges. For each packet that S generates, it then parses its bytes at lengths ranged by the length values given by monitor. S then checks with BF_1 for a matching entry and depending on the result of the string match, hashes the word into BF_2. Also in a parallel way, in 2 arrays it stores the keyword's natural value of the byte sequence and at the same index in the second array, its position in the packet. This procedure is repeated for the entire packet till the last set of bytes are analyzed for the highest keyword length as mentioned by the initialization data.

Table 5.1 Effect of changing sender's Bloom-filter size on the false positive rate of our model

BF_2 **Size in Bits**	$FalsePositiveRate$
200	1.00
300	0.80
400	0.20
500	0.0001

Once the packet is completely parsed, it then sorts both the arrays correspondingly based on the increasing order of the byte value found in the packet. The keyword position array is now inserted as mentioned before. It then generates the IP packet, creates a IPSec packet out of this IP packet, and then finally inserts BF_2 and L_{pos} along with the packet and transmits it forward. It should be noted here that whenever a signature database update happens at M and it generates a new BF_1, it is immediately sent to the sender. Before hashing any payload S verifies to see if it is the latest BF_1 and the length vector are the same or if there are any other recent updates it needs to consider. This has been implemented by the use of version number for the different BF_1's generated.

False Positives

A broad definition of false positives is the error of rejecting a null hypothesis when it is actually true. Plainly speaking, it occurs when we are observing a difference when in truth there is none. A false positive normally means that a test claims something to be positive, when that is not the case.

In our experimental system, a false positive is said to occur whenever a mal-ware gets reported as being present when it is actually a good instance or non-malicious packet. More formally, we define the false positive rate as the percentage of packets identified as containing mal-ware (incorrectly) over the total number of non-malicious packets sent by the sender.

The following are our experiments and their corresponding results:

5.0.8 Experiment 1

Impact of changing the size of Sender's Bloom filter on the overall False positive rate of our scheme

Goal

Our first set of experiments were motivated towards evaluating the effect of the size of the sender's bloom filter on the efficiency of the overall scheme. We evaluated the effect of this by keeping BF_1 constant and then varying the number of worms sent in both cases uniformly .

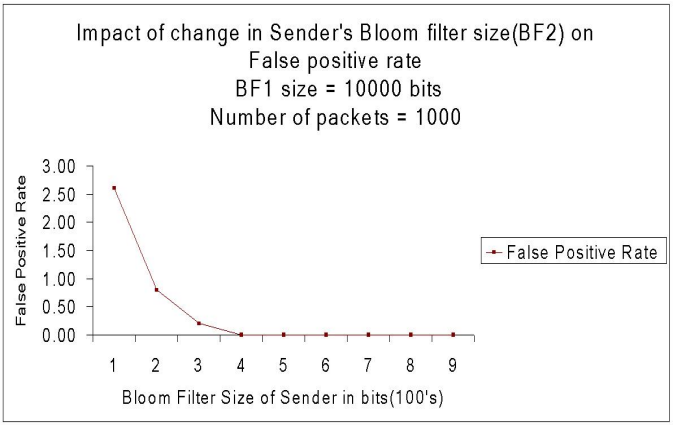

Figure 5.1 Simulation results of Table 1 experimental data: Effect of changing Sender's Bloom filter Size on False Positive Rate of our scheme

Inference

As it can be observed from the table 5.1 and the corresponding figure, we see that the Number of false positives detected for the same number of packets when BF_2 is varied from 200 bits to 1000 bits the false positives drop rapidly as BF_2 size is increased. Here we kept BF_1 size as a constant 20000 bits, since the monitor has no memory constraints about its size of the bloom filter. The number of packets is kept at 1000. For a size as low as 500, the false positive rate drops to 0.0001 percent and remains negligible for higher sizes.

5.0.9 Experiment 2

Impact of changing the size of Monitor's Bloom filter on the overall False positive rate of our scheme

Goal Our second set of experiments were motivated towards evaluating the effect of changing the Monitor's bloom filter on the efficiency of the overall scheme. We varied BF_1's size from

Table 5.2 Effect of changing Monitor's Bloom filter Size on the false positive rate of our model

BF_1 **Size in Bits**	$FalsePositiveRate$
15000	0.86
16000	0.71
17000	0.68
18000	0.16
19000	0.03
20000	0.01
21000	0.0001

15000 to 21000. The number of worms sent is always kept a constant at 1000 and the sender bloom filter size id the minimal value of 200 which can be easily fit into a IP packet.

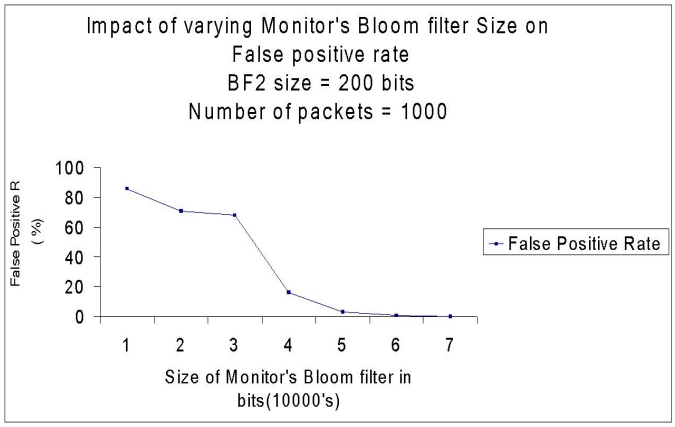

Figure 5.2 Simulation results of Table 2 experimental data: Effect of changing Monitor's Bloom filter Size on False Positive Rate of our scheme

Inference

From the table and graph in 5.2 we observe how as BF_1 improves,our false positive rate also improves in a significant way. For acceptable sizes of the bloom filter size of Monitor, say 20000 and that of sender at 200 bits, we get a false positive rate as low as 1 percent.So from this experiment we conclude that for our nature of experimental setup with a monitor and a sender, our scheme has been implemented successfully and tested to show that we can with a careful set-up and tolerable values of memory requirements, our false positive rate can be easily reigned in to provide for an effective detection mechanism for encrypted mal-ware.

Table 5.3 Results of first set of detection experiments for different number of worms sent

BF_2 Size in Bits	Number of worms sent at S per 1000 packets	Number of worms detected at M per 1000 packets
32	1000	1000
32	2000	2000
32	3000	3000
32	4000	6000
32	5000	9000

5.0.10 Experiment 3

Experiments on detection rate

Goal Our third set of experiments were motivated towards evaluating the effect of changing the Sender's Bloom filter size on the detection rate of our scheme or indirectly on the efficiency of our scheme. We kept BF_1's size a constant at 800 bits, had a scaled-down version of our signature database and changed the BF_2 size from 32 bits to 64 bits. The number of worms sent per packet was varied from 1 to 10 per packet and the detection rate calculated with it. So at each instance we sent 1000 packets with a variable number of worms in each one from 1 to 10. These results can be verified from Table 5.3 and its corresponding figure.

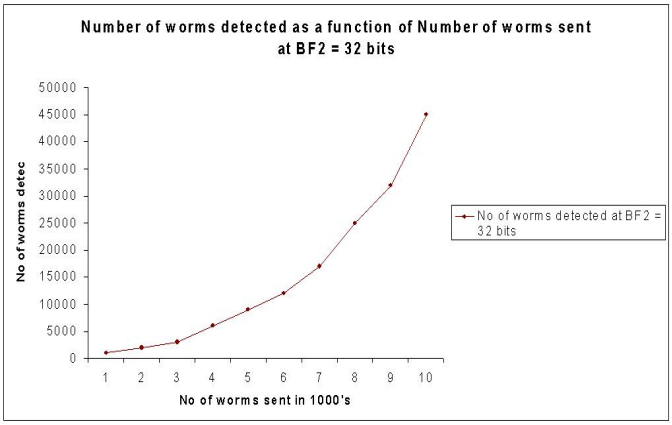

Figure 5.3 Simulation results of Table 3 experimental data: Results of detection experiments for different number of worms sent at a specific sender (S)'s bloom filter BF_2 size=32 bits

Inference

The detection rate of our scheme is always greater than or equal to 100 percent. We can

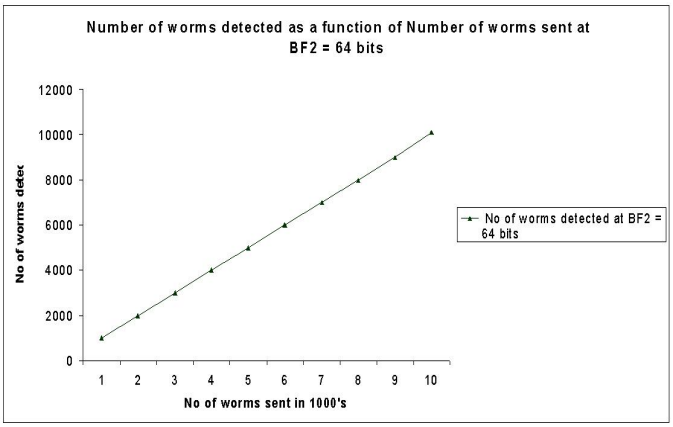

Figure 5.4 Simulation results of Table 3 experimental data: Results of detection experiments for different number of worms sent at a specific sender (S)'s bloom filter BF_2 size=64 bits.

confidently state this based on the design of our scheme. This is because, the Bloom-filters we use at both the sender and the monitor's end always have false-positives but will never have a false-negative. In short, whenever a keyword-token is sent, it will never escape detection at the Bloom-filter(absence of false negatives), but there are times when a worm is not present, but it would still get detected as a malicious-packet(presence of false-positives). From the tables 5.3, 5.4 given and their corresponding graphs we observe that when the Sender's bloom-filter size is 32 bits taken from the optional field, then the number of false positives we generate is within a controllable value till the rate of infection is 4 worms per packet. But the more interesting observation is for the 64 bit case, we see how the number of false positives is dramatically reduced by half for a small addition of 32 bits to the optional field. Even at the rate of 10 worms per packet, which is quite high considering conventional packets, we get a negligible 99 false positive reports which is still a controllable value.

5.0.11 Experiment 4

Keyword matrix optimization results

In the above sections we have studied a representation for the mal-ware signature database, considering that each mal-ware entity's signature is some kind of an rule-based computation over a set of keywords. This can be represented as a matrix mapping multiple keywords to many

Table 5.4 Results of second set of detection experiments for different number of worms sent

BF_2 **Size in Bits**	**Number of worms sent at S per 1000 packets**	**Number of worms detected at M per 1000 packets**
64	2000	2000
64	4000	4000
64	6000	6000
64	8000	8000
64	10000	10099

mal-ware entities, otherwise a keyword-matrix representation. Several mal-ware entities share keywords and sometimes the different mal-ware names are just variants of the same exploit or have common features. In this case, it is inefficient to represent each entity separately. This can be efficiently achieved by a matrix representation with say the mal-ware entities as the rows and the set of all possible keywords as the columns of the matrix. If instead of the keywords themselves, if we use a bit-map matrix and indicate the presence or absence of a keyword for a certain mal-ware by setting or resetting the corresponding row-column bit in the matrix, we might achieve huge savings in terms of the memory used, especially in cases where the mal-ware signature database can run up to millions of entries and keywords. Furthermore, even of if this keyword matrix happens to be extremely sparse, several algorithms exist that convert a sparse-matrix structure into a much more efficient one in its representation. A good survey of some of the sparse matrix re-ordering and optimization techniques is present in [26].

As a proof of concept, we tested an implementation of the same signature database represented as a keyword matrix. We had the same set-up for the system as mentioned earlier in this section except that at the Monitor side, instead of a SQL signature database, we had a bit-map matrix representation of the database. In the first instance of our experiment, we just allowed monitor to query the sparse matrix on an as-is basis and evaluated the overhead. In the second instance we tried an optimized representation of the keyword-matrix, choosing to store only the non-zero values in the keyword matrix as a table. We see from Table-5.4 that querying this optimized table has a much lesser overhead when compared with querying the entire sparse matrix. This experiment could be a good starting point for further optimizing the results of our design.

Table 5.5 Comparison of system overhead in querying sparse matrix and its optimized representation

Representation Type	**Processing overhead per packet time (ms)**
Sparse Matrix	0.1
Optimized Sparse Matrix	0.001

As a further improvement, we can use any of the algorithms mentioned in [26] and apply say, Cuthill-Mckee or Reverse Cuthill-Mckee algorithm to reduce our processing overhead by minimizing the querying delays.

CHAPTER 6. DISCUSSION AND CONCLUSION

6.1 A Discussion on Alternative design choices and considerations

This section contains ideas that were initially thought of as alternatives but due to lack of time, remain at the proposal stage itself, in-short they are the possible alternative strategies that we could come up for our present scheme.

6.1.1 Index propagation Model

The most important and limiting assumption we have made in our model is that encryption does not happen anywhere before or after the indexing layer. This assumption can also be overcome if we just ensure that no matter in how many layers we encrypt or as many times as we encrypt, we just need to ensure that the indexes are generated right before the first encryption happens and then propagated specially in an un-encrypted form and then finally attached to the encrypted packet in the last layer, say after the IPSec of IP layer happens, assuming that is the final level of encryption the packet goes through.

6.1.2 Exclusive Encryption Layer

Also we could add a separate thin layer called the indexing layer to the existing TCP/IP protocol stack model and make it as the exclusive-encryption layer. This might not be immediately feasible in conventional networks , but the idea of an exclusive indexing cum encryption layer might be welcomed in other kinds of networks like Wireless sensor networks, military communication or even any other kind of simple radio-based transmission (with extensive remote deployment and centralized control).

6.1.3 Communicative Model

An alternative proposal might be a scheme that allows the layer that indexes to talk to the final layer that provides encryption and attach the index after that is done. Say for instance, the indices were all generated by the system at the application layer, and kept there, then the data goes through multiple layers with or without encryption. At the last stage before the actual transmission happens, we capture each packet, attach the Bloom-filter generated at the application layer, directly to the IP-packet and then transmit it. This model avoids creating a new layer but requires new functionality though.

6.2 Conclusions and Future work

While we have proposed an efficient encrypted mal-ware detection scheme, this is by no means self-sufficient to address the different kinds of encrypted mal-ware threats that continue to evolve everyday. For instance, a separate class of mal-ware (usually as worms and exploits) exists in a self-encrypted form, that is, the encrypted exploit as well as the decryption routine, and or a hidden secret key, that is used to un-lock this encryption are all part of the same file or payload. In this case, deciding what we are going to recognize as a signature and index, deciding whether a node receiving this perceives it as a plain-text exploit that needs to be encrypted before transmission or not, are all some of the issues we have to resolve in the future.

There is also one more class of worms that spread in a time-lagged manner, these are called slow-propagating worms. So another important feature that we could like to work upon is that we are dealing with worms on a per-packet basis, we don't yet have a method to deal with stream-based data transfer, and also for worms that distribute themselves across multiple packets either in a time-lagged or memory-lagged fashion. In these cases we require some kind of a cross-packet correlation scheme that either does a time-based buffering (say buffer an hour's worth of packets and detect worms in the buffer, based on the average delay estimate of slow-propagating worms) or memory-based buffering (buffering, a certain amount of packets before detection to assimilate all possible tokens of a signature that are spread across say 1 MB

of buffered packets) or host-based buffering (say we buffer all packets that come from a specific source or a specific source-destination pair or ports identified as malicious or black-listed) , if we assume that we buffer and hold the packet at the Monitor while one of the 3 types of buffering happens, the system should be tolerant to such a kind of delay and should not suffer critically. We have shown experimentally that the hashing and detection itself needs negligible time, but the buffering-time and demands might critically affect several Quality-of-Service parameters causing undesirable network over-head and this might even lead to congestion or break-down in a reliable protocol delivery mechanism such as TCP. Also this requires that any other kind of attacks such as IP spoofing (for host-based buffering) or flooding (for memory-based buffering) does not occur.

Furthermore in this model we have assumed that all the nodes themselves are non-malicious and give accurate indexes, however the nodes can also be compromised or manipulated to give incorrect indexes , so this method is not resistant to such attack . Also this method is more of a attack-propagation-prevention scheme for encrypted mal-ware, so the point-of-contact infection might not be cured. To avoid this, we can probably come up with a distributed sender-monitor scenario that can run as a simple application in each host or better still, make the monitor as a part of all the ingress filters or incoming gate-ways.

In almost all areas of networking, securing the message transmission and also ensuring clean packets is not just a desirable but a critical feature. Implementing our indexing scheme for such alternative applications and networks is one of our future projects.

BIBLIOGRAPHY

[1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. M. Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In CRYPTO 2005, volume 3621 of LNCS, pages 205222. Springer, 2005.

[2] P. Akritidis, K. Anagnostakis, and E.P. Markatos: "Efficient Content-Based Detection of Zero-Day Worms "in*Proceedings of the International Conference on Communications (ICC 2005)* Seoul, Korea. May 2005.

[3] L. Ballard, S. Kamara, and F. Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In Proceedings of the Seventh International Conference on Information and Communication Security (ICICS 2005), pages 414426, 2005. information retrieval. In IEEE Symposium on Foundations of Computer Science, pages 364373, 1997.

[4] M. Bellare, A. Boldyreva, and A. ONeill. Efficiently-searchable and deterministic asymmetric encryption. Cryptology ePrint archive, June 2006. report 2006/186, http://eprint.iacr.org/2006/186. Thesis, 1992.

[5] B.Bloom. Space/time trade-offs in hash coding with allowable erros. ACM, 13(7):422-426, May 1970

[6] D. Boneh, G. di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In Proc. EUROCRYPT 04, pages 506522, 2004.

[7] http://bro-ids.org/

[8] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey,", Internet Mathematics, vol. 1, no. 4, pp. 485509, 2004.

[9] A. Broder and M. Mitzenmacher. Network applications of Bloom Filters: A survey. in*Proc. of Allerton Conference, 2002.*

[10] Y. C.Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In Applied Cryptography and Network Security Conference, 2005.

[11] R.Chinchani and E.Berg " A fast static analysis approach to detect Exploit Code Inside Network Flows", in*RAID 2005*

[12] J. Coit, S. Staniford, and J. McAlerney, "Towards Faster String Matching for Intrusion Detection or Exceeding the Speed of Snort", in, *Proc. 2nd DARPA Information Survivability Conference and Exposition (DISCEX II), IEEE CS Press,* 2001, pp. 367-373.

[13] M.Costa,J.Crowcroft,M.Castro,A.Rowstron,L.Zhou,L.Zhang and P.Barham, " Vigilante: End-to-End Containment of Internet Worms ", in *SOSP'05* oct-23-26,2005

[14] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. Cryptology ePrint archive, June 2006. report 2006/210, http://eprint.iacr.org/2006/210.

[15] Dharmapurikar, S., Krishnamurthy, P., Sproull, T., and Lockwood, J. "Deep Packet Inspection Using Parallel Bloom Filters." in,*Proc. 11th Symp. High Performance Interconnects* pages 44- 51, Stanford, California, 2003.

[16] M. Fisk and G. Varghese, " Fast Content- Based Packet Handling for Intrusion Detection",in *tech. report CS2001-0670, Univ. of California, San Diego,* 2001.

[17] E.-J. Goh. Secure indexes. Technical Report 2003/216, IACR ePrint Cryptography Archive, 2003. See http://eprint.iacr.org/2003/216.

[18] M.Jakobsson, M. Yung, and J. Zhou, editors, Applied Cryptography and Network Security Conference (ACNS), volume 3089 of LNCS, pages 3145. Springer-Verlag, 2004.

[19] H.A.Kim and B.Karp " Autograph: toward automated, distributed worm signature detection", in *Proceedings of the 13th USENIX Security Symposium* August 2004

[20] E.Kushilevitz,D. Boneh, R. Ostrovsky, and W. Skeith. Public-key encryption that allows PIR queries. Unpublished Manuscript, August 2006.

[21] M. Liberatore and B. N. Levine. "Inferring the Source of Encrypted HTTP Connections.",Department of Computer Science, University of Massachusetts, Amherst, MA 01003- 9264. 2006.

[22] R.Lyda and J.Hamrock "Using entropy analysis to find encrypted and packeted mal-ware "in *Proceedings of 2007 IEEE symposium on Security and Privacy* Vol.5, Issue 2, Mar-Apr 2007 pages: 40-45

[23] http://www.cut-the-knot.org/arithmetic/marriage.shtml

[24] McAfee Network Protection Solutions, White Paper." Encrypted Threat Protection Network IPS for SSL Encrypted Traffic.", October 27, 2007.

[25] M. Mitzenmacher,"Compressed bloom filters ", IEEE/ACM Transactions on Networking, vol. 10, no. 5, pp. 604612,October 2002.

[26] http://www.osl.iu.edu/ chemuell/new/graph-data-mining.php

[27] J.Newsome, B.Karp and D.Song, " Polygraph: Automatically generating Signatures for Polymorphic worms", in *Proceedings of 2005 IEEE symposium on Security and Privacy* pages 226-241, May 2005

[28] D. Park, K. Kim, and P. Lee. Public key encryption with conjunctive field keyword search. In 5th International Workshop WISA 2004, volume 3325 of LNCS, pages 7386. Springer, 2004.

[29] S.Singh, C.Estan,G.Varghese and S.Savage " The EarlyBird System for Real-Time Detection of Unknown Worms", in*Technical report CS2003-0761, CSE Department, UCSD* Aug.2003

[30] S.Singh, C.Estan,G.Varghese and S.Savage "Automated Worm fingerprinting ", in *Proceedings of the 6th ACM/USENIX Symposium on Opearting System design and Implementation (OSDI)* Dec.2004

[31] http://www.snort.org/

[32] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In Proceedings of 2000 IEEE Symposium on Security and Privacy, pages 4455, May 2000.

[33] Q. Sun, D. R. Simon, Y. Wang, W. Russell, V. N. Padmanabhan, L. Qiu." Statistical Identification of Encrypted Web Browsing Traffic ", in *Technical Report MSR-TR-2002-23, Microsoft Research, Microsoft Corporation* , March 8, 2002.

[34] Y.Tang and S.Chen" Defending Against Internet Worms: A Signature-Based Approach", in*INFOCOM 2005*

[35] http://www.cise.ufl.edu/research/sparse/

[36] K.Wang,G.Cretu and S.Stolfo "Anomalous Payload-based Worm Detection and Signature Generation ", in*RAID 2005*

[37] X.Wang, C.Pan, P.Liu, S.Zhu "SigFree:A signature free Buffer Overflow Attack Blocker", in *Proceedings of the 15th USENIX Security Symposium* July 2006

[38] Y.Chen, A.Kumar, J.Xu," A New Design Of Bloom Filter For Packet Inspection Speedup ", GLOBECOM'2007 pp.1-5

[39] Q.Zhang, D.Reeves, P.Ning,S.Iyer, " Analyzing Network Traffic To Detect Self-decrypting Exploit Code", in *ASIACCS'07* March 20-22, 2007