Defending Against Internet Worms: A Signature-Based Approach

Yong Tang Shigang Chen

Department of Computer & Information Science & Engineering University of Florida, Gainesville, FL 32611-6120, USA

{yt1, sgchen}@cise.ufl.edu

Abstract-With the capability of infecting hundreds of thousands of hosts, worms represent a major threat to the Internet. The defense against Internet worms is largely an open problem. This paper attempts to answer two important questions. Can a localized defense system detect new worms that were not seen before and, moreover, capture the attack packets? How to identify polymorphic worms from the normal background traffic? We have two major contributions. The first contribution is the design of a novel double-honeypot system, which is able to automatically detect new worms and isolate the attack traffic. The second contribution is the introduction of position-aware distribution signature (PADS), which fits in the gap between the traditional signatures and the anomaly-based systems. We propose two algorithms based on Expectation-Maximization (EM) and Gibbs Sampling for efficient computation of PADS from polymorphic worm samples. The new signature is capable of handling certain polymorphic worms. Our experiments show that the algorithms accurately separate new variants of the MSBlaster worm from the normal background traffic.

Key Words: System Design

I. INTRODUCTION

An Internet worm is a self-propagated program that automatically replicates itself to vulnerable systems and spreads across the Internet. It represents a huge threat to the network community [1], [2], [3], [4], [5], [6], [7]. While much recent research concentrates on worm propagation models [8], [9], [10], [11], [12], the defense against worm attacks is largely an open problem due to the fact that worms are able to spread substantially faster than human can respond. In most cases, the defense against worm attacks can only be done reactively after the damage has already happened.

Moore et al. studied the effectiveness of worm containment technologies (*address blacklisting* and *content filtering*) and concluded that such systems must react in a matter of minutes and interdict nearly all Internet paths in order to be successful [2]. Williamson proposed to modify the network stack so that the rate of connection requests to distinct destinations is bounded [13]. The main problem is that this approach becomes effective only after the majority of all Internet hosts is upgraded with the new network stack. For an individual organization, although the local deployment may benefit the Internet community, it does not provide immediate anti-worm protection to its own hosts, whose security depends on the rest of the Internet taking the same action. This gives little incentive for the upgrade without an Internet-wide coordinated effort. In our previous work, a distributed anti-worm architecture (DAW) has been proposed [3]. By tightly restricting the connection-failure rates from worm-infected hosts while allowing the normal hosts to make successful connections at any rate, DAW is able to significantly slow down the worm's propagation in an ISP and minimize the negative impact on the normal users.

Most known worms have very aggressive behaviors. They attempt to infect the Internet in a short period of time. This type of worms is actually easier to be detected becuase their aggressiveness stands out from the background traffic. The future worms may be modified to circumvent the rate-based defense systems and purposely slow down the propagation rate in order to compromise a vast number of systems over the long run without being detected [2].

Intrusion detection has been intensively studied in the past decade. *Anomaly-based* systems [4], [14], [15] profile the statistical features of normal traffic. Any deviation from the profile will be treated as suspicious. Although these systems can detect previously unknown attacks, they have high false positives when the normal activities are diverse and unpredictable. On the other hand, *misuse detection* systems look for particular, explicit indications of attacks such as the pattern of malicious traffic payload. They can detect the known worms but will fail on the new types.

Most deployed worm-detection systems are *signature-based*, which belongs to the misuse-detection category. They look for specific byte sequences (called *attack signatures*) that are known to appear in the attack traffic. The signatures are manually identified by human experts through careful analysis of the byte sequence from captured attack traffic. A good signature should be one that consistently shows up in attack traffic but rarely appears in normal traffic.

The signature-based systems [16], [17] have advantage over the anomaly-based systems due to their simplicity and the ability of operating online in real time. The problem is that they can only detect known attacks with identified signatures that are produced by experts. Automated signature generation for new attacks is extremely difficult due to three reasons. First, in order to create an attack signature, we must identify and isolate attack traffic from legitimate traffic. Automatic identification of new worms is critical, which is the foundation of other defense measures. Second, the signature generation must be general enough to capture all attack traffic of certain type while at the mean time specific enough to avoid overlapping with the content of normal traffic in order to reduce falsepositives. This problem has so far been handled in an ad-hoc way based on human judgement. Third, the defense system must be flexible enough to deal with the polymorphism in the attack traffic. Otherwise, worms may be programmed to deliberately modify themselves each time they replicate and thus fool the defense system.

This paper attempts to address the above problems. We design a novel double-honeypot system which is deployed in a local network for automatic detection of worm attacks from the Internet. The system is able to isolate the attack traffic from the potentially huge amount of normal traffic on the background. It not only allows us to trigger warnings but also record the attack instances of an on-going worm epidemic. We summarize the polymorphism techniques that a worm may use to evade the detection by the current defense systems. We then define the *position-aware distribution signature* (PADS) that is capable of detecting polymorphic worms of certain types. The new signature is a collection of position-aware byte frequency distributions, which is more flexible than the traditional signatures of fixed strings and more precise than the position-unaware statistical signatures. We describe how to match a byte sequence against the "non-conventional" PADS. Two algorithms based on Expectation-Maximization [18] and Gibbs sampling [19] are proposed for efficient computation of PADS from polymorphic worm samples. Experiments based on variants of the MSBlaster worm are performed. The results show that our signature-based defense system can accurately separate new variants of the worm from the normal background traffic by using the PADS signature derived from the past samples.

The rest of the paper is organized as follows. Section II proposes a double-honeypot system that can detect worm activities. Section III studies the worm polymorphism. Section IV proposes a position-aware distribution signature, and presents the algorithms for calculating such a signature and using the signature to identify worm in a byte sequence. Section V presents the experiment results. Section VI draws the conclusion and discusses the future work.

II. DOUBLE-HONEYPOT SYSTEM

A. Motivation

The spread of a malicious worm is often an Internet-wide event. The fundamental difficulty in detecting a previously unknown worm is due to two reasons. First, the Internet consists of a large number of autonomous systems that are managed independently, which means a coordinated defense system covering the whole Internet is extremely difficult to realize. Second, it is hard to distinguish the worm activities from the normal activities, especially during the initial spreading phase. Although the worm activities become apparent after a significant number of hosts are infected, it will be too late at that time due to the exponential growth rate of a typical worm [8], [9], [10], [11], [12]. In contrast to some existing defense systems that require large-scale coordinated efforts, we describe a double-honeypot system that allows an individual autonomous system to detect the ongoing worm threat without external assistance. Most importantly, the system is able to detect new worms that are not seen before.

Before presenting the architecture of our double-honeypot system, we give a brief introduction of honeypot. Developed in recent years, honeypot is a monitored system on the Internet serving the purpose of attracting and trapping attackers who attempt to penetrate the protected servers on a network [20], [21]. Honeypots fall into two categories. A high-interaction honeypot operates a real operating system and one or multiple applications. A low-interaction honeypot simulates one or multiple real systems. In general, any network activities observed at honeypots are considered as suspicious and it is possible to capture the latest intrusions based on the analysis of these activities. However, the information provided by honeypots is often mixed with normal activities as legitimate users may access the honeypots by mistake. Hours or even days are necessary for experts to manually scrutinize the data logged by honeypots, which is insufficient against worm attacks because a worm may infect the whole Internet in such a period of time.

We propose a double-honeypot system to detect new worms automatically. A key novelty of this system is the ability to distinguish worm activities from normal activities without the involvement of experts. Furthermore, it is a purely local system. Its effectiveness does not require a wide deployment, which is a great advantage over many existing defense systems [2], [13].

The basic idea is motivated from the worm's self-replication characteristics. By its nature, an worm infected host will try to find and infect other victims, which is how a worm spreads itself. Therefore, outbound connections initiated from the compromised hosts are a common characteristic shared by all worms. Suppose we deliberately configure a honeypot to



Inbound Honeypo

Fig. 1. Using double-honeypot detecting Internet worms

never initiate any outbound connections. Now if the honeypot suddenly starts to make outbound connections, it only means that the honeypot must be under foreign control. If the honeypot can be compromised, it might try to compromise the same systems on the Internet in the way it was compromised. Therefore, the situation is either a real worm attack or can be turned into a worm attack if the attacker behind the scene chooses to do so. We shall treat the two equally as a worm threat.

B. System Architecture

Figure 1 illustrates the double-honeypot system. It is composed of two independent honeypot arrays, the *inbound array* and the *outbound array*, together with two address translators, the *gate translator* and the *internal translator*. A honeypot array consists of one or multiple honeypots, which may run on separate physical machines or on virtual machines simulated by the same computer [21]. Each honeypot in the array runs a server identical to a local server to be protected. A honeypot in the inbound (outbound) array is called an *inbound (outbound) honeypot*. Our goal is to attract a worm to compromise an inbound honeypot before it compromises a local server. When the compromised inbound honeypot attempts to attack other machines by making outbound connections, its traffic is redirected to an outbound honeypot, which captures the attack traffic.

An inbound honeypot should be implemented as a highinteraction honeypot that accepts connections from outside world in order to be compromised by worms that may pose a threat to a local server. An outbound honeypot should be implemented as a low-interaction honeypot so that it can remain uninfected when it records the worm traffic.

The gate translator is implemented at the edge router between the local network and the Internet. It samples the unwanted inbound connections, and redirects the sampled connections to inbound honeypots that run the server software

the connections attempt to access (e.g., connections to ports 80/8080 are redirected to a honeypot running a web server). There are several ways to determine which connections are "unwanted". The gate translator may be configured with a list of unused addresses. Connections to those addresses are deemed to be unwanted. It is very common nowadays for an organization to expose only the addresses of its public servers. If that is the case, the gate translator can be configured with those publicly-accessible addresses. When a connection for a specific service (e.g., to port 80 for web access) is not made to one of the servers, it is unwanted and redirected to an inbound honeypot. Suppose the size of the local address space is N and there are h publicly-accessible servers on a particular destination port. Typically, N >> h. For a worm which randomly scans that port, the chance for it to hit an inbound honeypot first is $\frac{N-h}{N}$, and the chance for it to hit a protected server first is $\frac{h}{N}$. With a ratio of $\frac{N-h}{h}$, it is almost certain that the worm will compromise the inbound honeypot before it does any damage to a real server within the network.

Once an inbound honeypot is compromised, it will attempt to make outbound connections. The internal translator is implemented at a router that separates the inbound array from the rest of the network. It intercepts all outbound connections from an inbound honeypot and redirects them to an outbound honeypot of the same type, which will record and analyze the traffic.

We give the following example to illustrate how the system works. Suppose that the IP address space of a network is 128.10.10.0/24, with one public web server Y to be protected. The server's IP address is 128.10.10.1. No public web server runs on any other address. Suppose an attacker outside the network initiates a worm attack against systems of type Y. The worm scans the IP address space for victims. It is highly probable that a different IP address, e.g. 128.10.10.20, will be attempted before 128.10.10.1. The gate controller redirects the packets to an inbound honeypot of type Y, which is subsequently infected. As the compromised honeypot participates in spreading the worm, it will reveal itself by making outbound connections and provide the attack traffic that will be redirected to an outbound honeypot of the system.

We should emphasis that, the proposed double-honeypot system is greatly different from a conventional honeypot. A conventional system receives traffic from all kinds of sources, including traffic from the normal users. It is a difficult and tedious task to separate attack traffic from normal traffic, especially for attacks that are not seen before. It is more than often that, only after the damage of the new attacks is surfaced, the experts rush to search the recorded data for the trace of attack traffic. In our system, when an outbound honeypot receives packets from an inbound honeypot, it knows for sure that the packets are from a malicious source. The outbound honeypot does not have to face the potentially huge amount of normal background traffic that a conventional honeypot may receive.

III. POLYMORPHISM OF INTERNET WORMS

The double-honeypot system provides a means to capture the byte sequences of previous unknown Internet worms without manual analysis from the experts. The captured byte sequences can be used to generate worm signatures, and future connections carrying them will be automatically blocked. This is a great advantage over the current systems because the defense can be carried out automatically before new worms deal a significant damage to the network.

The attackers will try every possible way to extend the life time of Internet worms. In order to evade the signature-based system, a polymorphic worm appears differently each time it replicates itself. This section discusses the polymorphism of Internet worms, while the next section provides a solution against some common polymorphism techniques.

There are many ways to make polymorphic worms. One technique relies on self encryption with a variable key. It encrypts the body of a worm, which erases both signatures and statistical characteristics of the worm byte string. A copy of the worm, the decryption routine, and the key are sent to a victim machine, where the encrypted text is turned into a regular worm program by the decryption routine. The program is then executed to infect other victims and possibly damage the local system. While different copies of a worm look different if different keys are used, the encrypted text tends to follow a uniform byte frequency distribution [22], which itself is a statistical feature that can be captured by anomaly detection based on its deviation from normal-traffic distributions [4], [15]. Moreover, if the same decryption routine is always used, the byte sequence in the decryption routine can serve as the worm signature.

A more sophisticated method of polymorphism is to change the decryption routine each time a copy of the worm is sent to another victim host. This can be achieved by keeping several decryption routines in a worm. When the worm tries to make a copy, one routine is randomly selected and other routines are encrypted together with the worm body. The number of different decryption routines is limited by the total length of the worm. For example, consider a buffer-overflow attack that attempts to copy malicious data to an unprotected buffer. Oversized malicious data may cause severe memory corruption outside of the buffer, leading to system crash and spoiling the compromise. Given a limited number of decryption routines, it is possible to identify all of them as attack signatures after enough samples of the worm have been obtained.

Another polymorphism technique is called garbage-code insertion. It inserts garbage instructions into the copies of

a worm. For example, a number of nop (i.e., no operation) instructions can be inserted into different places of the worm body, thus making it more difficult to compare the byte sequences of two instances of the same worm. However, from the statistics point of view, the frequencies of the garbage instructions in a worm can differ greatly from those in normal traffic. If that is the case, anomaly-detection systems [4], [15] can be used to detect the worm. Furthermore, some garbage instructions such as nop can be easily identified and removed. For better obfuscated garbage, techniques of executable analysis [23] can be used to identify and remove those instructions that will never be executed.

The instruction-substitution technique replaces one instruction sequence with a different but equivalent sequence. Unless the substitution is done over the entire code without compromising the code integrity (which is a great challenge by itself), it is likely that shorter signatures can be identified from the stationary portion of the worm. The code-transposition technique changes the order of the instructions with the help of jumps. The excess jump instructions provide a statistical clue, and executable-analysis techniques can help to remove the unnecessary jump instructions. Finally, the register-reassignment technique swaps the usage of the registers, which causes extensive "minor" changes in the code sequence.

The space of polymorphism techniques is huge and still growing. With the combinations of different techniques, a cure-all solution is unlikely. The pragmatic strategy is to enrich the pool of defense tools, with each being effective against certain attacks. The current defense techniques fall in two main categories, misuse/signature matching and anomaly detection. The former matches against known patterns in the attack traffic. The latter matches against the statistical distributions of the normal traffic. We propose a hybrid approach based on a new type of signatures, consisting of position-aware byte frequency distributions. Such signatures can tolerate extensive, "local" changes as long as the "global" characteristics of the signature remain. Good examples are polymorphism caused by register reassignment and modest instruction substitution. We do not claim that such signatures are suitable for all attacks. On the other hand, it may work with executableanalysis techniques to characterize certain statistical patterns that appear after garbage instructions and excess jumps are removed.

IV. ALGORITHMS FOR SIGNATURE DETECTION

A. Background

Most deployed defense systems against Internet worms are signature-based. They rely on the exact matching of the packet payload with a database of fixed signatures. Though effective in dealing with the known attacks, they fail to detect new or variants of the old worms, especially the polymorphic worms whose instances can be carefully crafted to circumvent the signatures [23]. Moreover, manually identifying the signatures may take days if not longer.

To address these problems, several anomaly-based systems [4], [15] use the *byte frequency distribution* (BFD) to identify the existence of a worm. Their basic approach is to derive a byte frequency distribution from the normal network traffic. When a new incoming connection is established, the payload of the packets is examined. The byte frequency distribution of the connection is computed and compared with the byte frequency distribution derived from the normal traffic. A large deviation will be deemed as suspicious. The problem is that an intelligent attacker could easily cheat the system by attach the worm body to a lengthy normal, legitimate session. Since the majority of the payload is from legitimate operations, its byte frequency distribution will not vary much from the normal traffic, its statistic characteristics are smoothed out.

Both signature-based and anomaly-based systems have their pros and cons. The signature-based systems work well against the technique of attaching worm to normal traffic, but they are weak against polymorphism. On the other hand, the anomaly-based systems is able to handle polymorphism only when the worm is largely separated from the background and does not carry too much garbage instructions that distort the distribution.

Our system inherits the positive aspects of both signaturebased and anomaly-based systems. It is based on a new defense technique that is complementary to the existing ones. We define a relaxed, inexact form of signatures that have the flexibility against certain polymorphism. The new signature is called the *position-aware distribution signature* (PADS for short). It includes a byte frequency distribution (instead of a fixed value) for each position in the signature "string". The idea is to focus on the generic pattern of the signature while allowing some local variation.

Consider a polymorphic worm with register reassignment (Section III). Because registers are used extensively in executables, swapping registers is effective against traditional signatures. However, when a signature is expressed in positionaware distributions, not only are the static elements in the executable captured, but the set of likely values for the variable elements are also captured. Hence, PADS allows a more precise measurement of "matching". A similar example is instruction substitution, where the mutually replaceable instructions (or sequences) can be represented by the positionaware distributions.

The goal of our system is to use double honeypots to capture the worm attack traffic, based on which PADS is derived and used to detect inbound worm variants. It provides a quick and automatic response that complements the existing approaches

h	0	1	2		9	10
U	0	1	2		,	10
0x00	0.001	0.001	0.001		0.500	0.100
0x01	0.001	0.001	0.001		0.200	0.500
0x02	0.005	0.001	0.001		0.001	0.100
0xfe	0.100	0.001	0.001		0.001	0.001
0xff	0.001	0.700	0.700		0.001	0.001
TABLE I						

An example of a PADS signature with width W = 10

involving human experts. Based on PADS, the defense system will be able to identify the new variant of a worm at its first occurrence, even if such a variant has not been captured by the system previously.

B. Position-Aware Distribution Signature (PADS)

We first describe what is a PADS signature, then explain how to match a byte sequence against a signature, and finally motivate how to compute such a signature based on captured worm sequences.

At each byte position p of a PADS signature, the bytefrequency distribution is a function $f_p(b)$, which is the probability for b to appear at position p, where $b \in [0..255]$, the set of possible values for a byte. $\sum_{b \in [0..255]} f_p(b) = 1$. We use $(f_1, f_2, ..., f_W)$ to characterize the byte-frequency distribution of the worm, where W is the width of the signature in terms of the number of bytes. Let $f_0(b)$ be the byte frequency distribution of the legitimate traffic. The PADS signature is defined as $\Theta = (f_0, f_1, f_2, ..., f_W)$, which consists of a normal signature f_0 and an anomalous signature $(f_1, f_2, ..., f_W)$. Table I gives an example of a PADS signature with width W = 10.

Consider a set of byte sequences $S = \{S_1, S_2, ..., S_n\}$, where S_i , $1 \le i \le n$, is the byte sequence of an incoming connection. We want to decide whether S_i is a variant of the worm by matching it against a signature Θ . Let l_i be the length of S_i . Let $S_{i,1}, S_{i,2}, ..., S_{i,l_i}$ be the bytes of S_i at position 1, 2, ..., l_i , respectively. Let $seg(S_i, a_i)$ be the W-byte segment of S_i starting from position a_i . The matching score of $seg(S_i, a_i)$ with the anomalous signature is defined as

$$M(\Theta, S_i, a_i) = \prod_{p=1}^{W} f_p(S_{i,a_i+p-1})$$

which is the probability for $seg(S_i, a_i)$ to occur, given the distribution $(f_1, f_2, ..., f_W)$ of the worm. Similarly, the matching score of $seg(S_i, a_i)$ with the normal signature is defined as

$$\overline{M}(\Theta, S_i, a_i) = \prod_{p=1}^W f_0(S_{i,a_i+p-1})$$

We want to find a position a_i that maximizes $M(\Theta, S_i, a_i)$ and minimizes $\overline{M}(\Theta, S_i, a_i)$. To quantify this goal, we combine the above two scores in order to capture both the "similarity" between $seg(S_i, a)$ and the anomalous signature, and the "dissimilarity" between $seg(S_i, a_i)$ and the normal signature. For this purpose, we define $\Lambda(\Theta, S_i, a_i)$ as the matching score of $seg(S_i, a_i)$ with the PADS signature.

$$\Lambda(\Theta, S_i, a_i) = \frac{M(\Theta, S_i, a_i)}{\overline{M}(\Theta, S_i, a_i)} = \prod_{p=1}^W \frac{f_p(S_{i, a_i+p-1})}{f_0(S_{i, a_i+p-1})}$$
(1)

The matching score of the byte sequence S_i with the signature is defined as the maximum $\Lambda(\Theta, S_i, a_i)$ among all possible positions a_i , that is,

$$\max_{a_i=1}^{l_i-W+1} \Lambda(\Theta, S_i, a_i)$$

Alternatively, we can use the logarithm of Λ as the score, which makes it easier to plot our experiment results. Our final matching score of S_i with the PADS signature Θ is defined as

$$\Omega(\Theta, S_i) = \lim_{a_i=1}^{l_i - W + 1} \frac{1}{W} \log(\Lambda(\Theta, S_i, a_i))$$
$$= \lim_{a_i=1}^{l_i - W + 1} \sum_{p=1}^{W} \frac{1}{W} \log \frac{f_p(S_i, a_i + p - 1)}{f_0(S_i, a_i + p - 1)}$$
(2)

The W-byte segment that maximizes $\Omega(\Theta, S_i)$ is called the *significant region* of S_i , which is denoted as R_i . The matching score of the significant region is the matching score of the whole byte sequence by definition.

For any incoming byte sequence S_i , if $\Omega(\Theta, S_i)$ is greater than a threshold value, a warning about a (possibly variant) worm attack is issued. Additional defense actions may be carried out, e.g., rejecting the connection that carries S_i . The threshold is typically set at 0. From the definition of Ω , above zero means that S_i is closer to the anomalous signature $(f_1, f_2, ..., f_W)$; below zero means that S_i is closer to the normal signature f_0 .

Next we discuss how to calculate Θ based on the previously collected instances of a worm. Suppose we have successfully obtained a number n of variants of a worm from the double-honeypot system. Each variant is a byte sequence with a variable length. It contains one copy of the worm, possibly embedded in the background of a normal byte sequence. Now let $S = \{S_1, S_2, ..., S_n\}$ be the set of collected worm variants. Our goal is to find a signature with which the matching scores of the worm variants are maximized. We attempt to model it as the classical "missing data problem" in statistics and then apply the expectation-maximization algorithm (EM) to solve it.

To begin with, we know neither the signature, which is the underlying unknown parameter, nor the significant regions of the variants, which are the missing data. Knowing one would allow us to compute the other. We have just showed how to compute the significant region of a byte sequence if the signature Θ is know. Next we describe how to compute the signature if the significant regions of the variants are known.

First we compute the byte frequency distribution for each byte position of the significant regions. At position $p \in [1...W]$, the maximum likelihood estimation of the frequency $f_p(x), x \in [0...255]$, is the number c(p, x) of times that x appears at position p of the significant regions, divided by n.

$$f_p(x) = \frac{c_{p,x}}{n}$$

One problem is that $f_p(x)$ will be zero for those byte values x that never appear at position p of any significant region. However, considering that our calculation is based on a limited collection of the variants and $f_p(x)$ is only the maximum likelihood estimation of the frequency, we are not absolutely confident that the actual frequencies are zero unless we obtain all variants of the worm. For better flexibility, we apply a "pseudo-count" to the observed byte count $c_{p,x}$, and the byte frequency $f_p(x)$ is estimated as

$$f_p(x) = \frac{c_{p,x} + d}{n + 256 \cdot d} \tag{3}$$

where d is a small predefined pseudo-count number.

We have established that the PADS signature and the significant regions can lead to each other. We do not know either of them, but we know that the significant regions are those segments that can maximize the matching score with the signature. This "missing data problem" can be solved by an iterative algorithm, which first makes a guess on the starting positions of the significant regions, computing the signature, using the signature to compute the new starting positions of the significant regions, and repeating the process until convergence.

In the following, we show how to use the expectationmaximization algorithm and the optimized Gibbs sampling algorithm to compute the PADS signature from a collection of worm variants captured by our double-honeypot system. We want to stress that, though comparing the signature with the payload of the incoming connections is online, the signature itself is computed off-line. There is no real-time requirement.

C. Expectation-Maximization Algorithm

Expectation-Maximization (EM) [18] is an iterative procedure that obtains the maximum-likelihood parameter estimations. Given a set S of byte sequences, we lack the starting positions $a_1, a_2, ..., a_n$ of the significant regions, which are the missing data in our problem. The underlying parameter Θ of our data set is also unknown. The EM algorithm iterates between the expectation step and the maximization step after the initialization.

The description of EM algorithm is given below.

Initialization. The starting positions $a_1, a_2, ..., a_n$ of the significant regions for worm variants $S_1, S_2, ..., S_n$ are assigned randomly. They define the initial guess of the significant regions $R_1, R_2, ..., R_n$. The maximum likelihood estimate of the signature Θ is calculated based on the initial significant regions.

Expectation. The new guess on the locations of the significant regions is calculated based on the estimated signature Θ . In our algorithm, the new starting position a_i of the significant region is the position that the significant region has the best match score with the signature Θ . In other words, we seek

$$a_i = \arg\max_{a_i} \Lambda(\Theta, S_i, a_i) \quad \forall i \in [1..n]$$

Maximization By formula (3), the new maximum likelihood estimate of the signature $\overline{\Theta}$ is calculated based on the current guess on the locations of the significant regions.

The algorithms terminates if the average matching score Ω is within $(1 + \varepsilon)$ of the previous iteration, where ε is a small predefined percentage.

Starting with a large signature width W, we run the above algorithm to decide the signature as well as the significant regions. If the minimum matching score of all significant regions deviates greatly from the average score, we repeat the algorithm with a smaller W. This process continues until we reach a signature that matches well with the significant regions of all collected worm variants.

D. Gibbs Sampling Algorithm

One main drawback of the EM algorithm is that it may get struck in a local maxima. There is no guarantee that the global maxima can be reached. In order to solve the problem, many strategies have been proposed. One approach is to start with multiple random parameter configurations and look for the best among different results obtained. Another is to preprocess the data with some other methods and choose "good" initial configuration. In recent years, the simulated annealing [24] approach attracted great attention. Simply speaking, the approach allows certain random selection of the parameter (with a small probability moving towards a worse direction), which provides a chance to jump out of a local maxima. One example of the simulated annealing is the Gibbs Sampling Algorithm [19], which we will use to compute the PADS signature below.

The algorithm is initialized by assigning random starting positions for the significant regions of the worm variants. Then one variant is selected randomly. This selected variant is temporarily excluded from S. The signature is calculated based on the remaining variants. After that, the starting position for the significant region of the selected variant is updated, according to a probability distribution based on the matching

scores at different positions. The algorithm continues with many iterations until a convergence criterion is met.

The description of the Gibbs sampling algorithm is given below.

Initialization. The starting positions $a_1, a_2, ..., a_n$ of the significant regions for worm variants $S_1, S_2, ..., S_n$ are assigned randomly.

Predictive Update. One of the *n* worm variants, S_x , is randomly chosen. The signature Θ is calculated based on the other variants, $S - S_x$.

The algorithm terminates if the average matching score is within $(1 + \varepsilon)$ of the previous iteration, where ε is a small predefined percentage.

Sampling. Every possible position $a_x \in [1..l_x - W + 1]$ is considered as a candidate for the next starting position for the significant region of S_x . The matching score for each candidate position is $\Lambda(\Theta, S_x, a_x)$ as defined in (1). The next starting position for the significant region of S_x is randomly selected. The probability that a position a_x is chosen is proportional to $\Lambda(\Theta, S_x, a_x)$. That is,

$$\Pr(a_x) = \frac{\Lambda(\Theta, S_x, a_x)}{\sum_{a_x=1}^{l_x - W + 1} \Lambda(\Theta, S_x, a_x)}$$

Go back to the predictive update step.

E. Signature with Multiple Separated Strings

Thus far the PADS signature is assumed to be a continuous string (where each position in the string is associated not with a byte value but with a byte frequency distribution). The definition can be easily extended for a signature to contain $k(\geq 1)$ separated strings, which may have different lengths. The significant region of a byte sequence also consists of multiple separated segments, each having a starting position and corresponding to a specific string in the signature. The matching score $\Lambda(\Theta, S_i, a_{i1}, a_{i2}, ...)$ should now be a function of a set of starting positions, and the significant region is defined by the set of starting positions that maximizes the matching score. Because it remains that the signature and the significant regions can be computed from each other, the EM algorithm and the Gibbs Sampling algorithm can be easily modified to compute a signature with k strings.

V. EXPERIMENTS

The effectiveness of our algorithms in detecting polymorphic worms is demonstrated by experiments. The malicious payload of MS Blaster worm, which is 1.8KB long, is used in the experiments. The worm exploits a vulnerability in Microsoft's DCOM RPC interface. Upon successful execution, MS Blaster worm retrieves a copy of the file msblast.exe from a previously infected host [25]. In the experiments, we artificially generate the variants of the MS Blaster worm based on some polymorphism techniques discussed in Section III.



Fig. 2. Variants of a polymorphic worm



Fig. 3. Influence of initial configurations

Figure 2 illustrates the polymorphic worm design with five variants, S1, S2, ..., and S5. Each variant consists of three different types of regions. The black regions are segments of the malicious payload in MS Blaster worm. Substitution is performed on 10% of the malicious payload. Garbage payloads, which are represented as the white regions with solid lines, are inserted at different random locations in the malicious payload. The default ratio of the malicious payload to the garbage payload is 9:1.¹ In addition to garbage payload, each variant is embedded in the legitimate traffic of a normal session, represented by the white regions with dotted lines. The length of the normal traffic carried by a worm variant is between 2KB to 20KB. The significant regions of these variants start at a1, a2, ..., and a5, respectively.

In the first experiment, 100 variants of MS Blaster worm are generated and they are used as worm samples for signature generation. EM and Gibbs each run three times with different initial configurations. Specifically, the initial starting points of significant regions are randomly selected each time. Figure 3 shows the quality of the PADS signature obtained by EM or

¹This ratio is not shown proportionally in Figure 2 for better illustration.

Gibbs sampling algorithm after a certain number of iterative cycles. According to the algorithm description in Section IV, a run of either algorithm consists of iteration cycles (Expectation/Maximization steps for EM and Update/Sampling steps for Gibbs). During each iterative cycle, EM recalculates the significant regions of all variants, while Gibbs only modifies the significant region of one randomly selected variant. To make a fair comparison, we let the x axis be the number of per sequence iterations, which is defined as the average number of the iterations each sequence has undergone. The y axis is the average matching score of the 100 variants with the signature obtained so far. The matching score Ω is defined by (2). From the figure, the best matching score is around 7.5, which is likely to be the global maxima. EM tends to settle down at a local maxima, depending on the initial configuration. Gibbs is likely to find the global maxima but it does not stabilize even when it reaches the global maxima due to the randomness nature in its selection of starting points of significant regions.

Figure 4 and Figure 5 show the average matching score with respect to the signature width and the average length of the worm variants, respectively. In Figure 5, because the worm code has a fixed length, we increase the length of a variant by increasing the normal traffic it carries. In each experiment, we generate 200 variants of MS Blaster worm. We use 100 of them as samples for signature generation. The rest 100 variants are mixed with 100 normal-traffic byte sequences to test the quality of the signature. The left-hand plots in Figure 4 and Figure 5 show the average matching scores of sample variants after EM and Gibbs sampling algorithms converge to a final signature. The right-hand figures show the average matching scores of the testing worm/normal traffic sequences. The scores for worm traffic are always above zero and the scores for normal traffic are always below zero. Therefore, with a threshold of 0, worm variants are distinctively saperated from normal traffic. In our experiments, the generated PADS signature was always able to identify new variants of the worm without false positives.

Figure 4 also indicates that increasing the signature width will decrease the average matching score of worm variants. The reason is that a longer signature means a larger significant region, which increases the chance for the significant region to include garbage payload or normal traffic (that a worm variant carries), which in turn decreases the matching score. Figure 5 shows that increasing the length of the normal traffic carried by a worm variant, which has been widely used by some polymorphic worms to elude the anomaly-based systems, provides no help to avoid detection by our system. The reason is that our system identifies a significant region and only uses the significant region for signature generation.

For the purpose of comparison, we also perform experiments with some existing methods. Figure 6 shows the exper-



Fig. 4. Influence of different widths of the signatures



Fig. 5. Influence of different lengths of the sample variants



Fig. 6. The performance of signature-based system using longest common substrings.





Fig. 8. Byte frequency distributions of worm variants. Left-hand plot: malicious and normal payloads carried by a worm variant have equal length. Right-hand plot: normal payload carried by a worm variant is 9 times of malicious payload.

imental results based on the *longest common substring* method [20], which first identifies the longest common substring among the sample worm variants and then uses the substring as a signature to match against the test variants. Based on the left-hand plot, as the number of sample variants increase, the length of the longest common substring decreases. A shorter signature increases the chance for it to appear in normal traffic. Consequently, the false negative ratio decreases, but the false positive ratio increases dramatically (the right-hand plot). On the contrary, without the requirement of exact matching, a PADS signature is able to retain much more (particularly statistical) characteristics of a polymorphic worm.

Now consider the *position-unaware* byte frequency distributions that are used in some current systems. The left-hand plot of Figure 7 shows the position-unaware byte frequency distribution of 100 normal traffic sequences (from 100 normal sessions) and the right-hand plot shows the byte frequency

distribution of MS blaster payload. These two distributions are very different, which seems provide a way to detect the worm. However, if we create a worm variant by embedding the worm payload in normal traffic, the combined byte frequency distribution can be made very similar to that of normal traffic. Figure 8 shows the byte frequency distributions of two worm variants whose normal traffic payloads are 1 and 9 times of malicious payload, respectively. The right-hand plot is very similar to the left-hand plot of Figure 7. Therefore, using byte frequency distributions alone cannot handle worm variants. The proposed *position-aware* distribution signature works better against polymorphic worms.

VI. CONCLUSION AND FUTURE WORK

In this paper, we provide a new defense system to detect the attacks of malicious Internet worms. The key idea is to capture the samples of the Internet worm using proposed double-honeypot system before the protected server has been compromised. Those IP addresses that are unreachable from the outside are used to attract and trap the attackers. The system is especially useful in large networks where large number of unreachable IP addresses exist.

Our system is able to defend against polymorphic worms. After collecting a number of variants of polymorphic worm, the system uses iterative algorithms to find the PADS signature of the worm, which is used to detect future worm attacks even if new variants have not been captured before. In our experiment, a 100% accuracy has been achieved to detect the variants of MSBlaster worm which means all malicious traffic can be detected and all legitimate traffic can pass through the system with no false positives.

The system is completely automatic. It requires no involvement of human experts, which is typically the drawback of the regular signature-based system. The system also tolerates some modifications of the worm where both signature- and anomaly-based systems may fail.

In our future work, we plan to evaluate the system in a live environment. We also need some further improvement of our proposed iterative algorithms. For example, what should we do to distinguish several different worms from a mixture collection of the variants of these worms. The research in these directions will provide a more robust and reliable system to defend against future worm attacks.

REFERENCES

- S. Staniford, V. Paxson, and N. Weaver, "How to 0wn the Internet in Your Spare Time," in *Proceedings of the 11th USENIX Security Symposium (Security '2002)*, San Francisco, California, USA, Aug. 2002.
- [2] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Internet Quarantine: Requirements for Containing Self-Propagating Code," in Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '2003), San Francisco, California, USA, Apr. 2003.
- [3] S. Chen and Y. Tang, "Slowing Down Internet Worms," in Proceeding of the 24th International Conference on Distributed Computing and Systems (ICDCS '2004),, Tokyo, Japan, Mar. 2004.
- [4] C. Kruegel and G. Vigna, "Anomaly Detection of Web-based Attacks," in *Proceedings of the* 10th ACM Conference on Computer and Communication Security (CCS'2003). Washington D.C., USA: ACM Press, Oct. 2003, pp. 251–261.
- [5] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the Slammer Worm," *IEEE Magazine of Security* and Privacy, pp. 33–39, July 2003.
- [6] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton, "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks," in *Proceedings* of the 7th USENIX Security Conference (Security '1998), San Antonio, Texas, USA, Jan. 1998, pp. 63–78.
- [7] M. Eichin and J. Rochlis, "With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988," in *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, Oakland, California, USA, May 1989, pp. 326–344.
- [8] C. C. Zou, W. Gong, and D. Towsley, "Code Red Worm Propagation Modeling and Analysis," in *Proceedings of the 9th ACM Conference* on Computer and Communications Security (CCS '2002). Washington, DC, USA: ACM Press, Nov. 2002, pp. 138–147.

- [9] C. C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and Early Warning for Internet Worms," in *Proceedings of the* 10th ACM Conference on Computer and Communication Security (CCS '2003). Washington D.C., USA: ACM Press, Oct. 2003, pp. 190–199.
- [10] Z. Chen, L. Gao, and K. Kwiat, "Modeling the Spread of Active Worms," in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM' 2003)*, San Francisco, California, USA, Mar. 2003, pp. 1890–1900.
- [11] David Moore and Colleen Shannon and K Claffy, "Code-Red: A Case Study on the Spread and Victims of an Internet Worm," in *Proceedings* of the 2nd Internet Measurement Workshop (IMW '2002), Marseille, France, Nov. 2002, pp. 273–284.
- [12] J. O. Kephart and S. R. White, "Directed-Graph Epidemiological Models of Computer Viruses," in *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, Oakland, California, USA, May 1991, pp. 343– 361.
- [13] M. M. Williamson, "Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code," in *Proceeding of the* 18th Annual Computer Security Applications Conference (ACSAC '2002), Las Vegas, Neveda, USA, Oct. 2003.
- [14] H. Javitz and A. Valdes, "The NIDES Statistical Component Description and Justification," Computer Science Laboratory, SRI International, Menlo Park, California, USA, Tech. Rep., 1994.
- [15] K. Wang and S. J. Stolfo, "Anomalous Payload-based Network Intrusion Detection," in 7th International Symposium on Recent Advances in Intrusion Detection (RAID '2004), Sophia Antipolis, French Riviera, France, Sept. 2004.
- [16] K. Ilgun, R. Kemmerer, and P. Porras, "State Transition Analysis: A Rule-based Intrusion Detection Approach," *IEEE Trans. Software Eng.*, vol. 2, pp. 181–199, 1995.
- [17] U. Lindqvist and P. Porras, "Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST)," in *Proceedings of the 1999 Symposium on Security and Privacy*, Oakland, California, USA, May 1999.
- [18] C. E. Lawrence and A. A. Reilly, "An Expectation Maximization (EM) Algorithm for the Identification and Characterization of Common Sites in Unaligned Biopolymer Sequences," *PROTEINS: Structure, Function* and Genetics, vol. 7, pp. 41–51, 1990.
- [19] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton, "Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy for Multiple Alignment," *Science*, vol. 262, pp. 208–214, Oct. 1993.
- [20] C. Kreibich and J. Crowcroft, "Honeycomb Creating Intrusion Detection Signatures Using Honeypots," in 2nd Workshop on Hot Topics in Networks (HotNets-II), Cambridge, Massachusetts, USA, Nov. 2003.
- [21] N. Provos, "A virtual Honeypot Framework," Center for Information Technology Integration, University of Michigan, Ann Arbor, Michigan, USA, Tech. Rep. CITI Technical Report 03-1, Oct. 2003.
- [22] C. Kaufman, R. Perlman, and M. Speciner, *Network Security: Private Communication in a Public World*. Upper Saddle River, NJ, USA: Prentice Hall, Inc., 2002.
- [23] M. Christodorescu and S. Jha, "Static Analysis of Executables to Detect Malicious Patterns," in *Proceedings of the 12th USENIX Security Symposium (Security '2003)*, Washington D.C., USA, Aug. 2003.
- [24] S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distribution, and the Bayesian Restoration of Images," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 6, pp. 721–741, 1984.
- [25] CERT/CC, "CERT Advisory CA-2003-20 W32/Blaster worm," 2003. [Online]. Available: http://www.cert.org/advisories/CA-2003-20.html