

Honeypot-based Signature Generation for Polymorphic Worms

Sounak Paul¹ and Bimal Kumar Mishra²

¹*Dept. of Computer Sc. & Engineering
Birla Institute of Technology
Mesra, Ranchi, India*

²*Dept. of Applied Mathematics Birla Institute of Technology
Mesra Ranchi, India
paul.sounak@gmail.com, drbimalmishra@gmail.com*

Abstract

With the growing sophistication of computer worms, information security has become a prime concern for individuals, community and organizations. Traditional signature based IDS, though effective for known attacks but failed to handle the unknown attack promptly. This paper describes a novel honeypot system which isolates the suspicious traffic from normal traffic, and capture most useful information regarding the worm's activities, without attacker's knowledge. Our system will be used for critical study of structure and behavior of most sophisticated worms and then forwards the necessary input to Signature Generation Module for automatically generating signature of unknown polymorphic worms. Our attempt is to analyze the invariant content of polymorphic worms and using a probabilistic approach we compute the signature of worm with low false positive. Evaluation based on synthetically generated polymorphic worms demonstrate that our system is able to enhance the capability of IDS signature library and increases the probability of detecting polymorphic worms with efficiency, accuracy.

Keywords: *polymorphic worm, signature, honeypot, probability, false positive, token, intrusion detection*

1. Introduction

Worms spread in internet within a very short span of time. Traditional intrusion detection system fails to detect unknown worms, primarily due to following two reasons. First, internet consists of large number of autonomous network systems. Each system manages their defense on their own way. Therefore, a coordinated defense against any worm in internet is not possible. Second, during initial phase of spreading, there is not much difference between normal traffic and malicious traffic. Malicious activities become apparent after the worm infects a large number of machines. By that time it is too late for any defense mechanism to stop the worms from infecting hundreds and thousands of vulnerable hosts in the network. Some typical worms [1-10] in recent times showed exponential growth while spreading in internet. Therefore the response of the defense system must be very fast. In contrast to some existing defense system, that requires wide deployment and coordinated efforts to be successful, we propose a

This paper is an extended and revised version of our paper "Honeypot based signature generation for defense against polymorphic worms in networks," in proc. of IEEE IACC, Feb, 2013, DOI: 10.1109/IAdCC.2013.6514213

honeypot based fast signature generation system, whose location of implementation may be limited to local autonomous system to be effective. Also our system is able to detect previously unknown worms.

Our work is based on honeypot technology [11-12] to automatically generate signature of polymorphic worms. "A honeypot is an information system resource whose value lies in being probed, attacked or compromised" [13]. Depending on the level of interaction, honeypot can be classified as low-interaction honeypot and high-interaction honeypot. Low-interaction honeypot works by simulating one or multiple operating system and other real services. Such systems are easy to implement, High interaction honeypot on the other hand works with real operating system and services. Normally any traffic directed towards honeypot is considered as suspicious. Their activities can be captured in the honeypot, which can be analyzed in future. But often information captured at honeypot is mixture of information from both normal as well as anomalous traffic, as normal traffics are sometimes directed towards honeypot by mistake. It takes hours, sometimes days by the security experts to manually analyze the information captured at honeypot. But we can't afford this much time, because worms may infect a large part of internet by this duration.

Most of the defense mechanism developed against the worms work reactively, after full or partial damage has already occurred. Research Community has continuously tried to build and improve IDS to defend against malicious code attack. Research on worm defense may be broadly classified into two categories: Detection and Containment. Further Detection algorithm can be broadly divided into two categories Anomaly based detection and Signature based detection.

Anomaly based system [14-15] observe the traffic statistics and host behavior to detect previously unknown worms. To detect malicious traffic it require to understand normal traffic behavior, which in turn require efficient training which in real time scenario is much difficult to achieve, as the behavior of legitimate activities are largely unpredictable. Though this method is found to be effective in detecting unknown worms [4], it generates high false alarm.

Signature based detection [16-17] does not take interest in propagation or transmission scheme; neither they look into host behavior. They look for specific byte sequence in each packet. If any match found with signature stored in database it will be identified as malicious. When compared with anomaly based detection method signature based approach found to be more accurate and easier to implement online in real time. The difficulty is every signature needed to be stored in a signature pool. Over a period of time, the signature pool grows larger, making it complex for comparing any new signature with the existing one in the database. Moreover the early signature based methods are mostly manual; therefore this method consumes huge system resource, reducing the overall performance of the system and largely depends on human expertise of finding signature. Another problem with signature based approach is that it can detect only known worms with the signature generated manually by experts by carefully studying the network traces. The slow pace of manual signature generation led the researcher focus on automatically generating signature of both known as well as unknown worms.

Fortunately we can combine signature based detection with anomaly based detection, and get the advantage of both schemes in automated signature generation.

In this paper we have combined honeypot technology to isolate suspicious traffic and normal traffic to collect the information of captured traffic and forward these information to generate automated signature of polymorphic worms.

The rest of the paper is organized as follows. Section 2 surveys related works. Section 3 describes the structure of polymorphic worms. We propose the novel honeypot architecture and described data control and data capture mechanism in section 4. Section 5 presents a signature generation scheme for polymorphic worms, Section 6 discusses the experiments and result. Section 7 draws the conclusion.

2. Related Work

Several algorithms have been proposed for anomaly based worm detection and signature based detection, but none can cover entire range of worms. A hybrid system which check for network anomaly and look for signature in worm's payload may give complete and broader view of detection with a honeypot system to collect and analyze worm activities.

One of the early work in this category is Honeycomb [18], proposed by Kreibich and Crowcroft. Honeycomb combines honeypot technology with automatic signature generation scheme. Honeycomb has implemented an extended version of open source honeypot *honeyd* for this purpose. Suffix tree has been used to implement the Longest Common Substring (LCS) algorithm in this case. Problem with Honeycomb is that it generates single contiguous substring of sufficient length of worm's payload to match the worms. This assumption is insufficient for polymorphic worm detection [19].

Hyang-Ah Kim and Brad Karp describes autograph [20] a distributed, automated worm signature generation scheme to detect polymorphic worms. Autograph takes the input from cross DMZ traffic that includes benign traffic and selects suspicious traffic using certain heuristic. The suspicious packet passes through flow reassembly. Payloads partition is done into different content block using COPP algorithm. The content blocks are analyzed and autograph selects most frequently occurring byte sequence across the flows in suspicious flow pool. Prevalence histogram is generated for each content block which acts as worm signature. Similar to Honeycomb, Autograph also relies on generating single contiguous substrings as signature of a worm's payload to match the worm instances. But polymorphic worm may change their payload in each infection. Autograph system fails to address this problem.

Our signature generation scheme is exploit-specific, network-based. We present below some of state-of-art work in this category.

J. Newsome et al. address the problems of autograph, honeycomb as discussed above in Polygraph [19]. Polygraph generates multiple disjoint content substrings to match all instances of a polymorphic worm. They observed that multiple invariant substrings often present in all variant payload of a polymorphic worm. Such invariant substrings include protocol framing bytes, return addresses and in some cases obfuscated code. Based on these facts, polygraph divides signatures into tokens, a contiguous byte sequence. The system extract tokens automatically and represents each suspicious flow as a sequence of tokens. Polygraph present three classes of signature suite: namely conjunction signature, token subsequence signature and Bayes signature.

Hamsa [21], a network based automated signature generation scheme has shown substantial improvement over polygraph in terms of speed, accuracy and attack resilience. Hamsa follows the polygraph token based approach, but replaced suffix tree method of token extraction with light weight suffix array method [22] which increases the speedup of token extraction process 100 fold than Polygraph. Hamsa uses a greedy approach of generating multiset token signature.

All these systems are effective in detecting polymorphic worms in normal situation. But they can be evaded by injecting well crafted fake anomalous flows into normal traffic, thereby misleading signature generation process [23].

Moreover token extracted in presence of noise are probably fragments, therefore polygraph and hamsa will generate wrong signature in presence of noise in suspicious pool [24]. Hamsa and LESG also proved that presence of noise in suspicious pool makes the problem NP hard. Still there is little or no attention is paid in filtering noise in suspicious pool.

Y. Tang, *et al.*, proposed position aware distribution signature (PADS) [25], a content based signature generation scheme for polymorphic worms. Expectation Minimization and Gibbs Sampling algorithms are used to compute PADS from polymorphic worm samples. It claims to bridge the gap between traditional signature scheme and statistical anomaly- based approach. While hamsa and polygraph signature based on invariant part in polymorphic worms PADS consider both invariant and variant part of worm's payload to find signature. However PADS can't assure the accuracy of signature in presence of noise [21].

Most of the signature generation approaches discussed above generate signature for polymorphic worms but encounter problem in presence of noise in suspicious pool. In this paper we propose a honeypot based a signature generation for zero day polymorphic worms. Our signature generation scheme is based on probabilistic approach.

3. Structure of Polymorphic Worms

Typically a polymorphic worm and its instances are composed of following components [26].

Protocol Framing: Protocol framing is necessary for branch down the code execution path, where software vulnerability exist. The protocol framing string is invariant across all instances of polymorphic worms.

Return Address: Return address or function pointers are the values used to overwrite a jump target to redirect the server execution [19]. Typically a 32 bit integer, of which first 23-bit are normally same across all worm samples. Return address is another invariant part in polymorphic worms.

Exploit Code: These invariant bytes are necessary for abusing vulnerability. It also activates decryption routines and ensures identical malicious activities in all attacks.

Encrypted worm code (Payload): It contains the code to perform malicious activities. In presence of strong encryption routines, the worm payloads take different values in different infection.

Decryption Routine: Its function is to decrypt the encrypted payload by decryption key and passes the control to worm's code to start execution. Decryption routines are obfuscated in different instances of polymorphic worms.

Decryption Key: Worm payload is encrypted by polymorphic engines by different keys in different instances. To decrypt the worm's payload, corresponding decryption key is required.

Wild Card bytes: These bytes may take any values without affecting the functioning of worms and their spreading capabilities.

In summary, polymorphic worms have two classes of bytes; invariant and variant bytes. Invariant bytes remain same across all instances of the worms while variant bytes change its value in every infection attempt. Typically invariant bytes are protocol

framing string, exploit code and return address. The other components are in general variant across different instances of a polymorphic worm.

4. System Model

4.1. Architecture

Figure1 depicts the typical deployment of our proposed honeypot system. It is composed of two independent arrays of virtual honeypot – honeytrap1 and honeytrap2. Each honeypot array consists of multiple honeypot hosts. These are essentially low interaction honeypot, dedicated to the tasks of intrusion detection. These honeypots emulate different kinds of services such as DHCP, FTP, HTTP, NBT SMB, POP3, SMTP, SQL-Server, Telnet etc. The emulation is done at the application layer, top layer of OSI model. This enables uses of full network libraries. The machines where these honeypots are running can be treated as just another server on the network. Some security loopholes are added deliberately in the honeypot host's operating system and other software and some apparently useful information are left in the system. The intention is to lure the attacker inside the honeypot system and allow them to perform their normal activities, therefore capturing their every action and control their propagation without their knowledge. The system can also reveal the nature of the attack. The information captured through the system can be used to refine the rules of firewalls and generate new signatures of unknown worms.

4.2. Data Control

The layer two bridging device isolates the entire honeypot system from the rest of the network. The same device force all traffic going to and from honeypot system, first pass through as “invisible” layer two bridge. This bridge allows the attacker come into the honeypot system, but decide its fate if it want to come out of the honeypot [11]. The attacker can't trace any IP address, MAC-Address, Time to Live (TTL) reducing or routing path as the bridge functions in layer two.

When honeytrap1 is compromised it will try to make an outbound connection, to the intent of spreading the attack in other systems. The traffic is redirected to honeytrap2 through the internal translator1 (IT1) implemented at router. The honeytrap2 also does the same thing when the worm tries to make an outbound connection. The traffic is redirecting it to honeytrap1 through internal translator2 (IT2). We can decide a threshold of the number of outbound connections allowed. We set this threshold as 10 connections per hour. We will block further connections. This will serve two purposes. First we will be able to capture enough instances of malicious traffic, second it will reduce the chances of Denial of service attacks. The disadvantage of this method is that it can still launch attack in outbound limits. Instead of blocking all the outbound connections at outbound threshold we will allow some of them periodically to pass through honeynet sensor to forward them to the IDS which will modify and disable the attacks. Attacker may see the failure but will unable to recognize the reason. This way we can capture necessary details regarding the attack, at the same time reducing the chances of compromising remote systems.

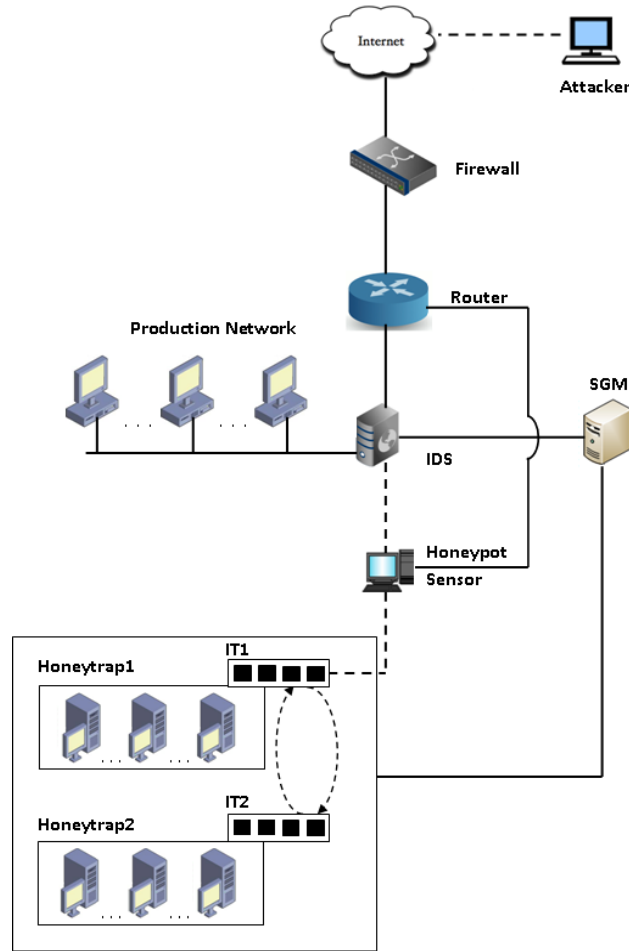


Figure 1. System Architecture

4.3. Data Capture

To analyze the malicious traffic effectively and generate signature from it, we require different types of information related to the attack. A single layer of defense can't capture all necessary information. Therefore we have implemented multilayer data capturing system in our proposed system. Entry layer of data capture is the firewall. Firewall logs preliminary header information such as source and destination address, attack timestamp, source and destination port no etc. Firewall can also filter known invasion. The second layer of data capture is the layer two bridge from router to honeypot sensor. This layer can sniff all network traffic and detect the suspicious traffic, their types and possible location using anomaly detection method. It can also trick the suspicious pool to enter into honeypot system. The honeypot is the final layer of data capture in our system. It captures different types of data, necessary for generating signature of worms. The honeypot opens the ports of the residing host, and wait for a connection to be made, in the same way some application server such as web server, SMTP server does. It listens to both TCP and UDP ports and interact with the visitors (malware or legitimate traffic that enters into honeypot system by mistake.) and

an event log is created corresponding to each events. A honeypot server is thus set up to capture the activities of the attacker.

5. Signature Generation

Our signature generation scheme is based on the fact that multiple invariant strings must present in all variant of a polymorphic worm [19, 21, 26]. We consider each of these strings as tokens. A token is a byte sequence that is present in a significant member of flows. A signature is a set of tokens with their occurrence number in suspicious flows. Formally a signature takes the following form:

$\{(t_1, n_1), (t_2, n_2), (t_3, n_3), \dots, (t_k, n_k)\}$, where t_j is the token and n_j is the number of occurrence.

A signature is said to match a flow F if it contains at least n_j copies of t_j as substring. For example tokens of a polymorphic worm are.

$\{(\backslash'XFF\backslashBFB', 1), ('GET', 1), ('r\n', 5), ('HTTP/1.1\n', 1), ('r\n Host', 2)\}$

A signature is a set of (token, number of occurrence) taken in any order.

Our signature generation algorithm follows the following steps. Signature generation architecture and algorithm is depicted in Figure 2 and Figure 3 respectively.

Token Extraction:

Our signature generation extracts tokens that occur in at least λ fraction in suspicious traffic. The minimum length of token is set as 2, to avoid all characters to be included as tokens.

Hamsa [21] used deep-shallow-sort [27], a suffix array based algorithm to extract tokens. The use of suffix array has increased the speed of token extraction manifold, compared to Polygraph [19]. We have used suffix array induced sorting (SA-IS) [28] for token extraction in this algorithm. The use of this suffix array algorithm enhanced the speed of token extraction and memory efficiency [28]. The optimized code of SA-IS is available in [29]. The author of this paper compared performance of SA-IS algorithm with deep-shallow sorting that is used in token extraction and false positive calculation of Hamsa. The experiment show that SA-IS is most time and space efficient among all linier time suffix array construction algorithm. Token extracted may appear more than once in a flow. We note the number of occurrence of a token in a flow and store them as a pair (Token, number of occurrence).

Sub Token Consideration:

We define t_1 as sub token of t_2 , if $t_1 \neq t_2$ and t_1 is a substring of t_2 . We include every subtoken in signature if it satisfies the minimum length constraint and its coverage is larger than λ . Many of the token based signature generation schemes argued in favor of eliminating subtokens from signature. But we have observed by eliminating the subtokens we may miss some crucial invariants.

Sorting Tokens:

Sort the list of tokens in descending order of their length. We assign each (token, no of occurrence) pair a unique identifier. For each identifier we build a list of suspicious flows in which it occurs.

Probability Calculation and Signature Generation:

The flow classification in our architecture used double honeypot system. Therefore it is expected that the suspicious flow will contain mostly worms. Some innocuous flow will also enter into the honeypot system by mistake. The worm flows are constrained to include a multi-set of invariant in any order.

Say this multi-set invariant set is

$$T = \{t_1^{\hat{=}}, t_2^{\hat{=}}, t_3^{\hat{=}}, \dots, t_k^{\hat{=}}\}$$

Now we find out the probability of a token to appeared in a sample, given the classification of the sample (worm or benign). *i.e.*, for each token t_i we compute the probability & select the token with highest probability as first token.

Such that,

$$\Pr_{(t_i)} \geq \Pr_{(t_j)} \quad \forall i$$

$$\Pr_{(t_i, t_2)} \geq \Pr_{(t_1, t_j)} \quad \forall i > 1$$

$$\Pr_{(t_i, t_2, \dots, t_j)} \geq \Pr_{(t_1, t_2, \dots, t_{j-1}, t_j)} \quad \forall j \forall i > j - 1$$

This means $t_1^{\hat{=}}$ is the token with highest probability appearing in a sample among all tokens available in the list. $t_1^{\hat{=}}$ in conjunction with $t_2^{\hat{=}}$ highest probability value among all pairs in the list. This greedy process of computing probability will continue till we produce a signature S such that probability of all the tokens in the set of S in any order is crossing a threshold probability value. *i.e.*,

$$\Pr(S) \geq \Pr_{threshold}$$

The false positive value in normal traffic pool of the signature is calculated and is bound by following condition. In case the condition not met new threshold value of the probability to be decided.

$$FP_s \leq FP_{max}$$

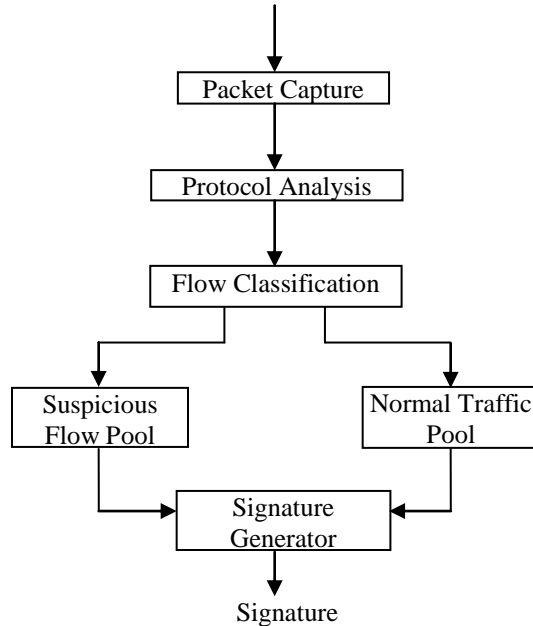


Figure 2. Signature Generation Architecture

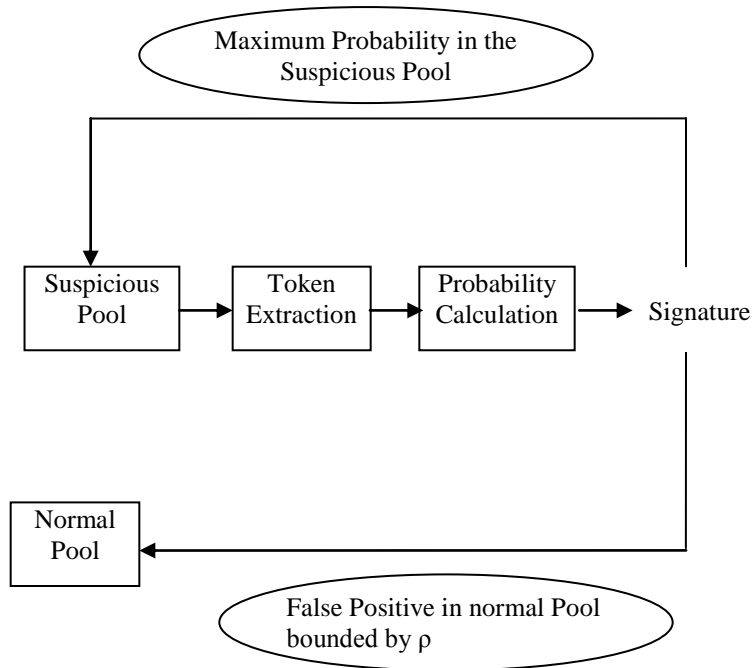


Figure 3. Signature Generation

6. Experiment and result Analysis:

We have setup our experiment in our institutional LAN. We have used KFSensor to setup our honeypot system as described in section 4.1. KFSensor is a Window based virtual honeypot. A virtual honeypot is essentially an emulated server. KFSensor is expected to be installed along with software WinPCap. The two main components of KFSensor are KFSensor Server. It listens to TCP and UDP ports and interacts with visitors (Malware or normal traffic) and generates events. KFSensor monitor is a GUI based interface, through which we can configure and monitor KFSensor Server. KFSensor is used with other security software. We have used network firewalls and Kaspersky antivirus suite. The basic idea is to generate signature of unknown worms through the information captured by honeypot system.

The port view of KFSensor shows which ports it is currently listening. Portview, visitor view and event view are interrelated. If we chose a particular port, say for example port 80, all events and visitors (malware or normal traffic) corresponding to the port will be displayed.

ID	Start	Duration	Pro...	Sens...	Name	Visitor	Sig. Message	Received
442	7/8/2014 11:32:08 AM...	0.000	UDP	4002	UDP Packet	172.15.3.51	<HBV1.0)Tagsys	
441	7/8/2014 11:32:07 AM...	0.000	UDP	4002	UDP Packet	172.15.3.52	<HBV1.0)Tagsys	
440	7/8/2014 11:32:07 AM...	0.000	UDP	12223	UDP Packet	172.15.31.20	[1C AA 07]n[C7	
439	7/8/2014 11:32:07 AM...	0.000	UDP	5246	UDP Packet	172.15.31.20	[00] [02 10 00 00	
438	7/8/2014 11:32:06 AM...	0.000	UDP	4002	UDP Packet	172.15.3.51	<HBV1.0)Tagsys	
437	7/8/2014 11:32:05 AM...	0.000	UDP	4002	UDP Packet	172.15.3.52	<HBV1.0)Tagsys	
436	7/8/2014 11:32:04 AM...	0.000	UDP	4002	UDP Packet	172.15.3.51	<HBV1.0)Tagsys	
435	7/8/2014 11:32:03 AM...	0.000	UDP	4002	UDP Packet	172.15.3.52	<HBV1.0)Tagsys	
434	7/8/2014 11:32:02 AM...	0.000	UDP	4002	UDP Packet	172.15.3.51	<HBV1.0)Tagsys	
433	7/8/2014 11:32:01 AM...	0.000	UDP	4002	UDP Packet	172.15.3.52	<HBV1.0)Tagsys	
432	7/8/2014 11:32:00 AM...	0.000	UDP	138	NBT Datagram...	dell4c4585.waljat.com		NBT DGRAM Pa
431	7/8/2014 11:32:00 AM...	0.000	UDP	67	DHCP	dellc4g354.waljat.com		DHCP: Boot Rec
430	7/8/2014 11:32:00 AM...	0.000	UDP	4002	UDP Packet	172.15.3.51	<HBV1.0)Tagsys	
429	7/8/2014 11:32:00 AM...	0.000	UDP	12223	UDP Packet	172.15.6.119	[E8]@[DE 19]	
428	7/8/2014 11:32:00 AM...	0.000	UDP	5246	UDP Packet	172.15.6.119	[00] [02 10 00 00	
427	7/8/2014 11:31:59 AM...	0.000	UDP	4002	UDP Packet	172.15.3.52	<HBV1.0)Tagsys	
426	7/8/2014 11:31:58 AM...	0.000	UDP	67	DHCP	gwdt327.waljat.com		DHCP: Boot Rec
425	7/8/2014 11:31:58 AM...	0.000	UDP	4002	UDP Packet	172.15.3.51	<HBV1.0)Tagsys	
424	7/8/2014 11:31:57 AM...	0.000	UDP	4002	UDP Packet	172.15.3.52	<HBV1.0)Tagsys	
423	7/8/2014 11:31:57 AM...	0.000	UDP	67	DHCP	dellb3g380.waljat.com		DHCP: Boot Rec

Figure 4. Ports View

Figure 4 shows the emulated services on different ports. Each event provides a detail analysis of an attack as shown in Figure 5. 6 and 7.

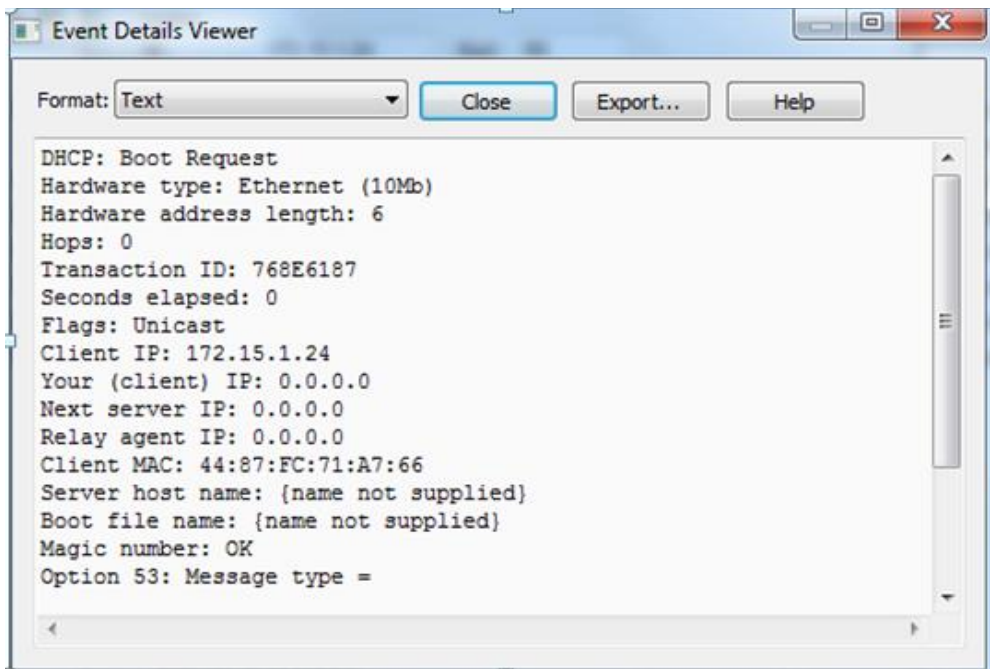


Figure 5. Event Analysis

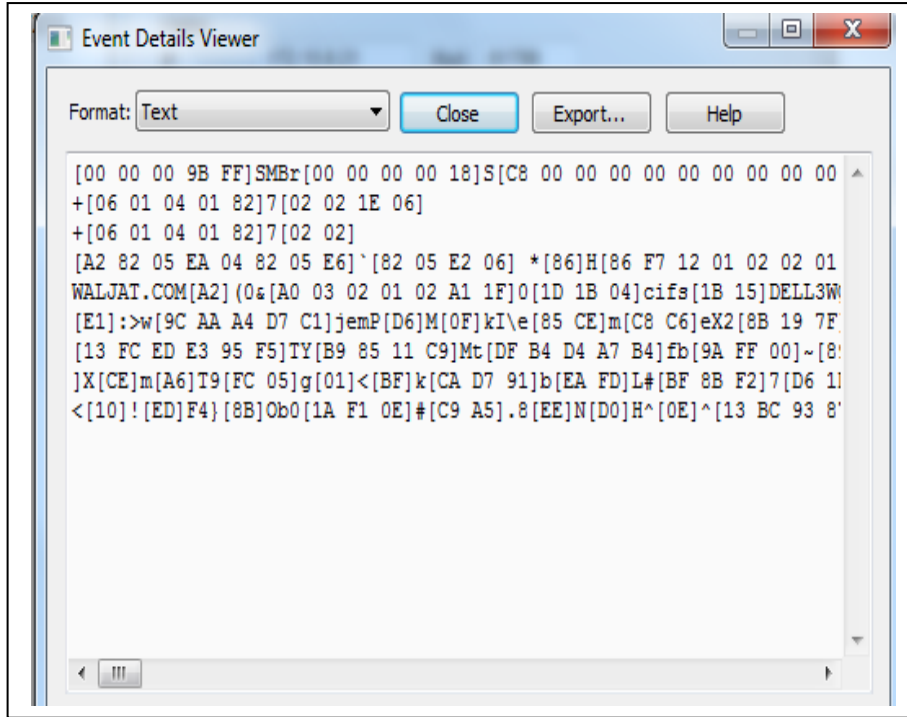


Figure 6. Event Details in Text

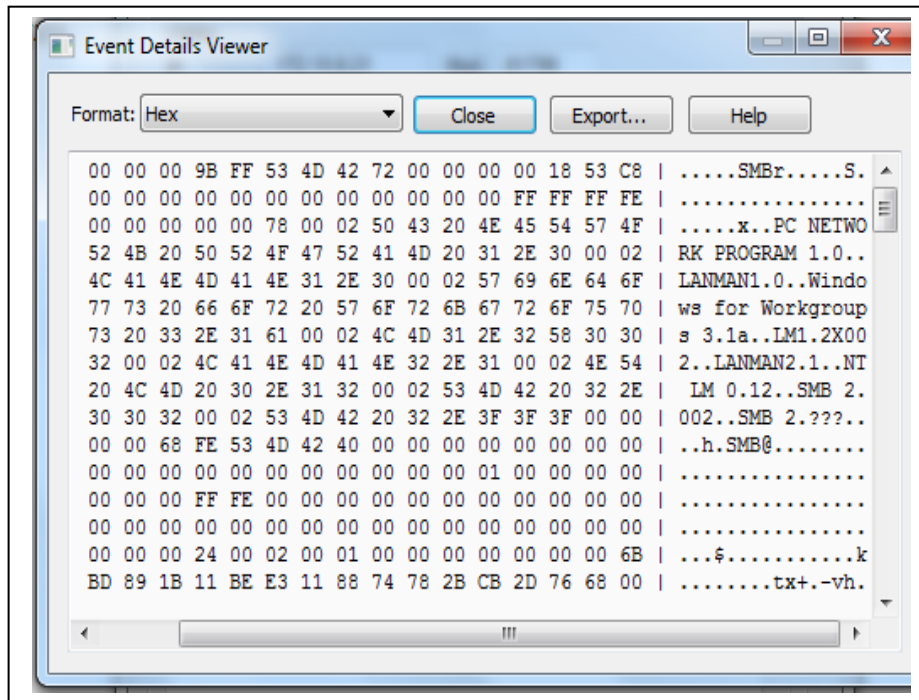


Figure 7. Event Details in Hex

My SQL is being configured to record the activity log in the database for analysis. For our experiments we have used two synthetically generated polymorphic worms;

Apache Knacker [30] and BIND-TSIG [31]. Apache Knacker uses text based HTTP protocol while BIND-TSIG uses binary based DNS protocol. BIND-TSIG is polymorphic version of Lion worm. For the sake of comparing we have used same worm that is used by Polygraph [19]. We have used 4 days network trace for both HTTP and DNS protocols. We have used our own institutional network to capture trace. The institute network is of medium size; consist of almost 500 hosts, with several different servers. For our experiments, we have considered only one polymorphic worm in suspicious traffic. As the worm replicate to make outbound connections, the instance samples will also be present in the suspicious pool. We test our signature generation algorithm using different numbers of worm samples of same polymorphic worm. We experimented with different suspicious pool size ranging from 3 to 50. Signature generated with 2 suspicious pools will be inaccurate and false negative is 100%.

We have considered minimum length of token as, $l_{min} = 2$ and require each token to cover at least 15% of suspicious pool. *i.e.*, $\lambda = 0.15$. We have considered the threshold probability value as $Pr_{threshold} = 0.75$ and the false positive maximum tolerable value as 0.01. *i.e.*, $FP_{max} = 0.01$. We reject the signature if the false positive value is more than 1% in normal traffic, and probability of appearance in a suspicious pool is less than 0.75.

We present here in Table-I and Table-II, the result of our experiment, for the two polymorphic worms; Apache Knacker and BIND-TSIG respectively.

Table I. Apache-knacker Worm Experimental Result

Class	False Positive	False Negative
Polygraph's Conjunction signature	0.0024%	0%
Polygraph's Token Subsequence signature	0.0008%	0%
Our Probabilistic signature	0.0020%	0%
<i>Tokens: '\xFF\xBF': 1, 'GET': 1, ':': 4, '\r\n': 5, 'HTTP/1.1\r\n': 1, '\r\nHost:': 2 Probability: 0.83</i>		

Table II. Bind- tsig Worm Experimental Result

Class	False Positive	False Negative
Polygraph's Conjunction signature	0%	0%
Polygraph's Token Subsequence signature	0%	0%
Our Probabilistic signature	0%	0%
<i>Tokens: '\xFF\xBF', '\x00\x00\xFA' Probability: 0.88</i>		

In final signature calculation, the above tokens may appear in any order. The probability value is 0.83 for Apache-Knacker and 0.88 for BIND-TSIG, which is more than the threshold value. The false positive value for Apache-Knacker is 0.0020%. while for BIND-TSIG is 0%. We see that our probabilistic signature is atleast as good as Polygraph signature. But the speed of signature generation and memory efficiency in our scheme will be much higher because of the SA-IS suffix array algorithm [28] as token extraction. Although this speed and memory performance is analytically proved in [28], we have to establish it experimentally. We keep it as future scope of our work.

7. Conclusion

In this paper we propose a honeypot based signature generation scheme for polymorphic worms using a probabilistic approach. We present here the novel architecture of our honeypot system. Our system captures the suspicious traffic analyzes them and sends it to the signature generator module for signature synthesis. We extract tokens from suspicious sample and using a greedy probabilistic approach we generate a multi-sets byte sequence as signature of polymorphic worm. While finalizing the signature we ensure false positive is less than a permitted value. We evaluate our proposed algorithm with synthetically generated polymorphic worms for accuracy, efficiency and effectiveness. All the flows are subjected to pass through the internal network. Suspicious flows are attracted towards the honeypot system. Event analysis is done at honeypot and subsequently log is being generated. The log is used in our signature generation module to generate signature.

Acknowledgements

This work is partially supported by Birla Institute of Technology, Mesra, Ranchi, India and Waljat College of Applied Science (Birla Institute of Technology, International Centre, Muscat, Oman). The authors would also like to thank the anonymous reviewers for their constructive comments and feedback on this paper.

References

- [1] E. Spafford, "The Internet Worm Program: An Analysis," *Computer Communication, Review*, (1989).
- [2] J. A. Rochlis and M. W. Eichen, "With Microscope and Tweezers: The Worm from MIT's Perspective", *Communication. ACM*, vol. 32, no. 6, (1989), pp. 689-698.
- [3] C. C. Zou, W. Gong and D. Towsley, "Code Red Worm Propagation Modeling and Analysis", In Proc. of the 9th ACM Conference on Computer and Communications Security (CCS '02), ACM Press, (2002) November, pp. 138-147, Washington, DC, USA.
- [4] D. Moore, C. Shannon and K. Claffy, "Code-red: a case study on the spread and victims of an internet worm", in Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement, IMW '02, ACM, New York, NY, USA, (2002), pp. 273-284.
- [5] "Computer Emergency Response Team", CERT Advisory CA-2001-23: "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL, (2001).
- [6] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford and N. Weaver, "Inside the slammer worm", *IEEE Security and Privacy*, vol. 1, no. 4, (2003) July, pp. 33-39.
- [7] M. Bailey, E. Cooke, F. Jahanian, D. Watson and J. Nazario, "The blaster worm: Then and now", *IEEE Security Privacy*, vol. 3, (2005), pp. 26-31.
- [8] P. A. Porras, H. Saidi and V. Yegneswaran, "An analysis of conficker's logic and rendezvous points," SRI International, Tech. Rep., (2009) March.
- [9] "An analysis of the ikee.b (duh) iPhone botnet," SRI International, Tech. Rep., (2009), December.
- [10] N. Falliere, L. O Murchu and E. Chien, "W32.stuxnet dossier," Website, Symantec Corp., Tech. Rep., (2011) February.
- [11] L. Spitzner, "The Honeynet Project: Trapping the Hackers", In proc. IEEE S&P, (2003), pp. 15-23.
- [12] Y. Tang and S. Chen, "Defending against Internet Worms: A Signature-Based Approach," In Proc. IEEE INFOCOM, (2005).
- [13] L. Spitzner, "Honeypots: Tracking Hackers", Addison-Wesley, (2002), www.tracking-hackers.com/book.
- [14] C. Kruegel and G. Vigna, "Anomaly Detection of Web-based Attacks", in Proc. ACM Conference Computer and Communication Security, ACM, (2003), pp. 251-261.
- [15] K. Wang and S. J. Stolfo, "Anomalous Payload-based Network Intrusion Detection", in proc. 7th international Symposium on Recent Advances in Intrusion Detection (RAID '2004).
- [16] U. Lindqvist and P. Porras, "Detecting Computer and Network Misuse through the Production-Based Expert System Toolset P-BEST," in Proc of Symposium on Security and Privacy, (1999).
- [17] K. Ilgun, R. Kemmerer, and P. Porras, "State Transition Analysis: A Rule-based Intrusion Detection Approach." *IEEE Trans. Software Eng.*, vol. 2. (1995), pp. 181-199.

- [18] C. Kreibich and J. Crowcroft, "Honeycomb - creating intrusion detection signatures using honeypots", In Proc. of the Workshop on Hot Topics in Networks (HotNets), (2003).
- [19] J. Newsome, B. Karp and D. Song. "Polygraph: Automatically generating signatures for polymorphic worms", In proc. IEEE Security and Privacy Symposium, (2005).
- [20] H. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection", In USENIX Security Symposium, (2004).
- [21] Z. Li, M. Sanghi, Y. Chen, M. Kao and B. Chavez, "Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience," in Proc. IEEE S&P, (2006), pp. 33-47.
- [22] G. Manzini and P. Ferragina. "Engineering a lightweight suffix array construction algorithm" *Algorithmica*, vol. 40, no. 1, (2004).
- [23] R. Perdisci, *et al.*, "Misleading worm signature generators using deliberate noise injection," in Proc IEEE S&P, (2006), pp. 17-31.
- [24] J. Wang and J. Wang, "An automated signature generation approach for polymorphic worms based on color coding," in proc. of IEEE ICC, (2009).
- [25] Y. Tang and S Chen, " An Automated Signature-Based Approach against Polymorphic Internet Worms," *IEEE Transaction on Parallel and Distributed Systems*, (2007) July, pp. 879-892.
- [26] S. Paul and B. K. Mishra, "PolyS-Network based signature generation for zero-day polymorphic worms", *International journal of grid and distributed computing*, vol. 6, no. 4, (2003), pp. 69-74.
- [27] G. Manzini and P. Ferragina. "Engineering a lightweight suffix array construction algorithm", *Algorithmica*, vol. 40, no. 1, (2004).
- [28] G. Nong, S. Zhang and W. Hong, "Two efficient algorithm for linear time suffix-array construction," *IEEE transactions on computers*, vol. 60, no. 10, (2011), pp. 1471-1484.
- [29] Y. Mori, "SAIS-An Implementation of the Induced Sorting Algorithm," (2008), <http://yuta.256.googlepages.com/sais>.
- [30] C. CAN-2003-0245, "Apache apr-psprintf memory corruption vulnerability", <http://www.securityfocus.com/bid/7723/discussion/>.
- [31] SANS Institute, "Lion worm", <http://www.sans.org/y2k/lion.htm>.