International Journal of u- and e- Service, Science and Technology
Vol.7, No.3 (2014), pp.129-150
http://dx.doi.org/10.14257/ijunesst.2014.7.3.12

Survey of Polymorphic Worm Signatures

Sounak Paul[1] and Bimal Kumar Mishra[2]

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

Abstract

Worms are self –replicating, fast moving malicious codes, capable of spreading themselves without human interaction. It's a weapon of choice for those, who like to launch destructive attacks on network or internet as a whole. Recently there emerge more sophisticated worms such as polymorphic worm which vary their payload in every infection attempt. Polymorphic worms have more than one mutated instances. It is very important to detect and prevent such worms quickly and accurately at their early phase of infection. There are several worm detection and containment methods available. This paper surveys recent automated signature based detection of polymorphic worms and presents a classification for various signature approaches. There are two main categories of worm signatures, exploit specific and vulnerability driven. Both categories of signature are either network-based or host based. Each signature generation scheme is described and discussed in appropriate category. We analyze and compare different signature schemes. The paper concludes with challenge and scope of future research directions.

Keywords: *polymorphic worm, worm signature, survey, Network Base, Host based, detection, network security*

1. Introduction

Worms are the malicious codes, which propagate themselves without human intervention. They are capable of replicating themselves in the host, when required. Worms first find targets using different scanning methods available. Most popular among them are blind-scan, hit-list, topological and web search. Once the target is found, the worm sends a copy to the next victim. Depending on the nature of the worm, propagation may be straightforward or worm payload may be downloaded from internet or from the infected host. Worm payload may also be embedded with legitimate traffic. Propagation may be epidemic as observed in many research [1, 2, 3, 38]. It can spread faster than human response. Depending on the type of transmission, worm may either require TCP connection or UDP connections. TCP worms are found to be latency limited, whereas UDP worms are bandwidth limited [4]. The payload of worm has been evolved from traditional monomorphic to much complex polymorphic and metamorphic. Polymorphic worms are hard to detect as they change their payload dynamically, maintaining the same functionality. Metamorphic worms on the other hand not only change their appearance but they change their behavior too on every infection, making it difficult for traditional IDSes [5, 6] to detect them [13].

ISSN: 2005-4246 IJUNESST

Most of the defense mechanism developed against these 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 primarily focused on detection and containment. Detection mechanisms are broadly classified into anomaly based detection and signature based detection [4, 7, 39].

Anomaly based system [8, 9] observe the traffic statistics and host behavior to detect previously unknown worms. To detect malicious traffic, it requires to understand normal traffic behavior, which in turn requires efficient training. 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, it generates high false alarm.

Signature based detection 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 method, signature based approach found to be much simpler 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.

The problem with manual signature based approach is that, it can detect only known worms with the signature generated manually by experts by carefully studying the network traces [10, 11, 12]. The slow pace of manual signature generation led the researcher focus on automatically generating signature of both known as well as unknown worms. Automated signature generation [40] scheme need to address the following challenges:

- It must understand well the difference between malicious and legitimate traffic, so that false alarm is minimum. There lacks a systematic solution to this problem.
- The signature must be flexible enough to defend against polymorphic worms that change their appearance in every infection. So single payload substring may not be invariant across worm connections [10].
- It should take care of system resource such as CPU time of generating signature and comparing them with network traffic. Signature storage space must not be stressed.
- The signature must be reliable enough to detect wide variety of zero day attacks.
- The signature must be noise tolerant, attack-resilient and accurate.
- The system must be easily deployable.
- The signature generation must be fast enough to handle spread of new worms.
- The system must produce signature with low false positive and false negative.

Several recent researches focus on defense against unknown polymorphic worms. They tried to address the above mentioned issues, but still it remains largely an open problem due to diversity of the modern worms. In this paper we surveys the signature based approaches of polymorphic worm defense. Our aim is to help better understanding of the polymorphic worm signature and point out the open issues that may be addressed in further research.

2. Anatomy of Polymorphic worms:

Typically a polymorphic worm and its instances are composed of following components [10, 11, 13].

Protocol Framing: Protocol frame part is responsible for branching down the code execution path, where software vulnerability exists.[10] The protocol framing string is invariant across all instances of polymorphic worms. This part is the prime source of invariant strings. We denote this part as ε .

Example of such protocol framing string in Apache–Knacker exploits: "GET", "HTTP/1.1", and "Host:" twice [10].

Return Address: Return address or function pointers are the values used to overwrite a jump target to redirect the server execution [10].Typically a 32 bit integer, of which first 23-bit are normally same across all worm samples [11]. Return address is another invariant part in polymorphic worms. We denote this control data part as γ .

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. We denote this part as π .

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

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: Wild card 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.

3. Classification of Polymorphic Worm Signatures

The polymorphic signature generation schemes presented in literature are broadly classified as exploit-specific and vulnerability-based. Exploit-specific signatures are based on feature specific to worm's implementation. They are mostly content based. Vulnerability-based signature looks for characteristics of the vulnerability the worm exploits. Depending on deployment both the exploit specific and vulnerability based signature may be classified as host based and network based. Although host based signature generation is more accurate, network based approach shows better coverage and ability of protecting most users in the network as a whole, enabling detecting the worm fast and at early stage [14]. The classification is shown in Table-I below [13] with their year of publication.

Table 1. Polymorphic Worm Signature Generation Schemes

Signature Properties	Signature Generation	
	Network Based	Host Based
Exploit - Specific	Nemean [15](2005) CFG [16](2005) Polygraph [10](2005) Hamsa [11](2006) PADS [22](2007) Polytree[23](2011)	Taint Check [37] DACODA [38]
Vulnerability - driven	LESG [14](2010)	COVERS [29](2005) Vigilante [32](2005) Malware Detection[33](2005) Packet Vaccine [34](2006) Vulnerability Signature [35] (2006)

4. Exploit-specific (Network Based) Signature Generation:

Nemean:

Yegneswaran *et al.*, presented Nemean [15], an automated signature generation for misuse detection, with low false-alarm by balancing specificity and generality. This balance is achieved by understanding session layer and application layer semantics. Nemean architecture composed of two components- Data Abstraction component (DAC) and signature generation component as shown in Figure 1 redrawn from [15].

Data abstraction component (DAC) aggregates and transforms packet traces (collected from a honeypot deployed on an unused IP address space) into a well defined data structures, without knowledge of application protocol or application level semantics. This aggregation is called semi-structured-session tree (SST). The aggregation step group packet data between two hosts into sessions.

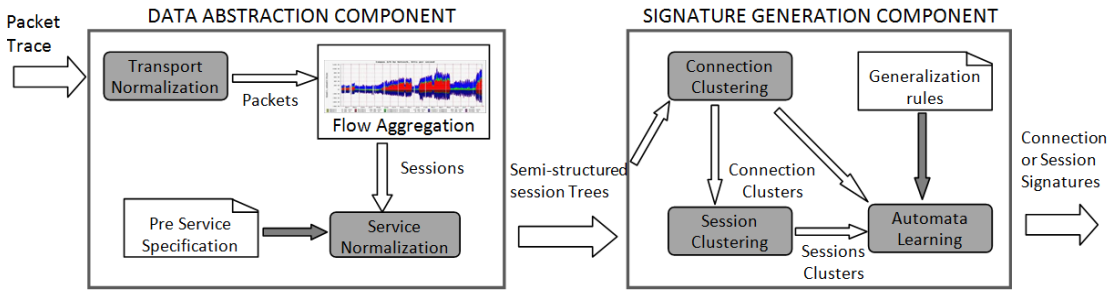

Figure 1. Components and Data Flow Descriptions of Nemean Architecture

Signature generation component groups sessions and connections of similar adversaries into clusters as per similarity metrics. An automata learning module, with the help of machine learning algorithm such as clustering and finite state machine generalization create signature from clusters of sessions.

Advantages of Nemean:

- Nemean can generate signature where exploit is small part of the entire payload.
- It is Capable of generating signature for multistep attacks.
- Generalized signature may be produced from small number of input samples.

Limitations of Nemean:

- Nemean requires detailed protocol specifications for each and every application protocol.
- Nemean is not able to address automating the real time deployment of signatures.
- Nemean is not noise tolerant, *i.e.*, it can't generate accurate signature in presence of normal traffic in suspicious traffic pool.

CFG:

Christopher *et al.*, [16] present a network based approach based on structural similarity of control flow graph (CFG) to generate a fingerprint for detecting different polymorphic worms. They have assumed that worm have at least one part, that contain executable machine code and another part of the encrypted code. The first part is directly executable by victims machine, on the other hand the decryption routine present in the worm decrypt the encrypted part. They analyze the network flows to check for executable code. If any region with executable code found, this method generates the fingerprint for that region. These fingerprints are nothing but byte string extracted from network stream by content shifting approach [17].

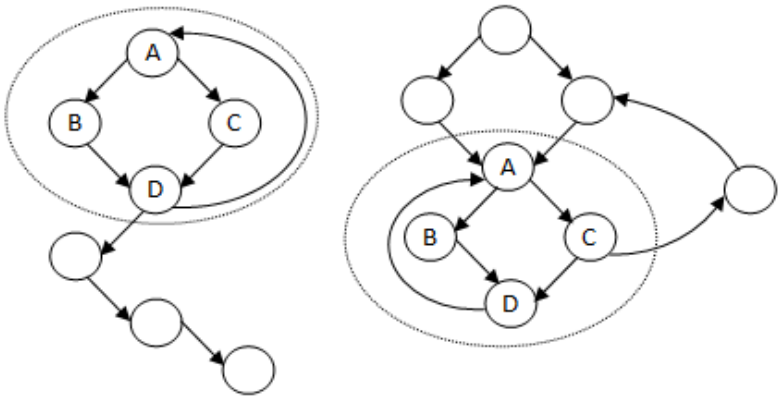

Figure 2. Two Control Flow Graphs with an Example of Common 4-subgraphs

The structure of executables described by control flow graph (CFG) as shown in figure 2 (redrawn from [16]). Nodes of the CFG form the basic building block. An edge from node u to v represents flow of control from u to v. In this paper a method has been developed to identify common substructures of two control flow graph. This is done by checking isomorphic connected subgraphs of size k (referred as k-subgraph) in all CFGs. If two subgraphs are isomorphic in two CFGs, they represent the instances of a polymorphic worm. For example in Figure 2, two CFG share common 4-subgraphs. The adjacency matrix is presented here in Figure 3(redrawn from [16]). The 16 bit string derived from this adjacency matrix act as fingerprint for the worm.

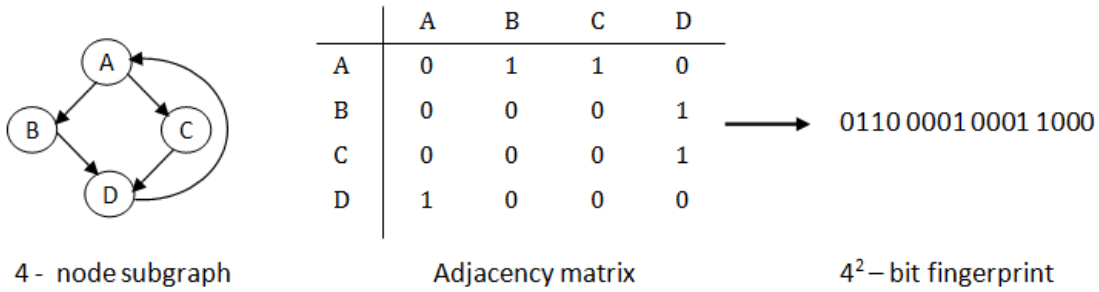

Figure 3. Deriving a Fingerprint from a Subgraph with 4 Nodes

Advantages of CFG:

- This approach is robust against polymorphic code modifications.
- The approach is resilient to insertion/ deletion to original executable.

Limitations of CFG:

- Current prototype of CFG operates offline.
- The approach is complex and expensive; therefore may not be useful in filtering worm traffic in high traffic link [11].
- The signature generated by this method may be evaded by using sophisticated encryption decryption (SED) [11].
- The technique cannot generate signature if the executable has a very small footprint

Polygraph:

Newsome et al., proposed Polygraph [10], a content based automated signature generation for polymorphic worms. Polygraph is an important milestone in research of signature of polymorphic worms. Prior to polygraph there were few automated signature generation schemes. Notable among them are Autograph [17], Honeycomb [18], and EarlyBird [19]. All these schemes used pattern based analysis to produce signatures. These signatures are single contiguous substring to match the worm. Polygraph find this assumption naïve and insufficient. Polygraph observed that multiple invariant substrings must often be present in all variant of polymorphic payloads. Motivated by insufficiency of single substring signatures, polygraph proposes three classes of signatures. There are conjunction signature, token-subsequence signature and Bayes signature.

Conjunction Signature: Conjunction signature consists of set of tokens in any order. If these set of tokens found in polymorphic payload it is said, a match is found.

Token subsequence signature: Token subsequence signature consists of tokens in some order. A matching is said to found with polymorphic payload if and only if it contains the same set of tokens in similar order.

Bayes Signature: Bayes signature consist of set of tokens. Each token is associated with a score and unlike exact matching by conjunction and token subsequence signature, Bayes signature provide probalistic matching information. Given a suspicious flow, the probability that flow is a worm is computed using the token scores present in the flow. If the result is over the threshold value the flow is classified as worms. Polygraph captures network traffic through a network tap. The captured traffic passes through a flow classifier, which reassembles them as contiguous byte flows. For same

IP protocol number and destination port, the flows are classified as suspicious or innocuous, which are used as input to polygraph signature generator. Figure 4 below (redrawn from [10]) depicts typical deployment of polygraph monitor.

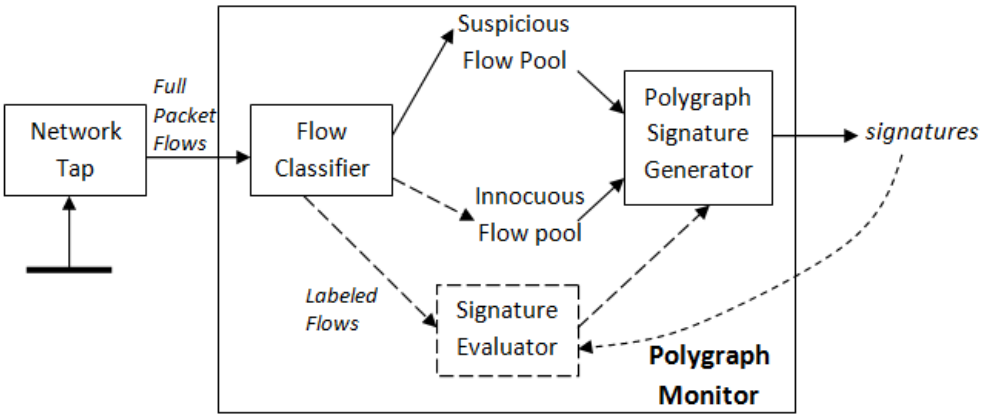

Figure 4. Deployment of Polygraph Architecture

Advantages of Polygraph:

- Although conjunction and token subsequence can't handle multiple types of worms, hierarchical clustering of worms gives the solution to the problem.
- Polygraph signature is resilient to coincidental pattern attack Bayes signature is resilient to red herring attack.

Limitations of Polygraph:

- Polygraph signature may not work if worm lacks invariant code.
- Polygraph uses naïve Bayesian model to generate signature.
- If the normal flow pool is too diverse the false positive rate in polygraph will be high.
- Polygraph is not memory efficient. Polygraph performs a top down traversal of the suffix tea to generate tokens. This operation takes asymptotically linear time but consume large memory space.
- Polygraph is not resilient to token fit attack described by hamsa [11].

Hamsa:

Z. Li et al., present Hamsa [11], a content based automated signature scheme for polymorphic worms. Hamsa uses protocol frame part (ε), control data (γ) and worm payload (π) to generate polymorphic worm signature. Hamsa uses similar architecture as polygraph to separate the network traffic into suspicious and normal flow pool. The architecture is shown in figure 5 (redrawn from [11]).

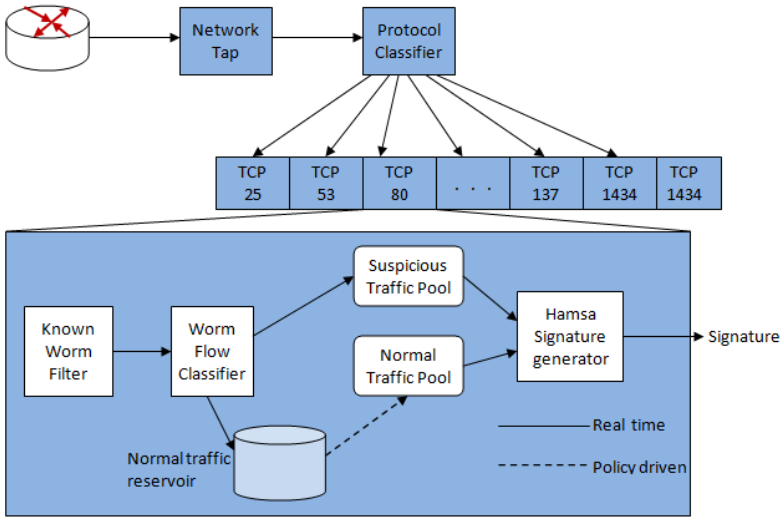

Figure 5. Architecture of hamsa

Given a suspicious flow, token is extracted using a suffix array based algorithm [20]. The algorithm finds tokens (byte sequence) that occur in at least λ fraction suspicious flow pool. Hamsa define the worm signature as multi-sets of tokens. This set is defined with tokens and their corresponding number of occurrences. A match with a worm is said to found, if it has all the tokens in desired number in any sequence. The first token the multi-set is chosen as the one with smallest false positive value. The token, which has least false positive value in conjunction with first token chosen as next token. Other tokens are similarly evaluated in the greedy approach and a multi-set token thus produced, act as worm signature. Hamsa claimed to produce better signature than polygraph with similar assumption that protocol frame part (ε) and exploit data (γ) will remain invariant in worm. Using the light weight suffix array algorithm hamsa achieves 100 times speedup for token extraction compared to polygraph. Figure 3(redrawn from [11]) describes hamsa signature generation.

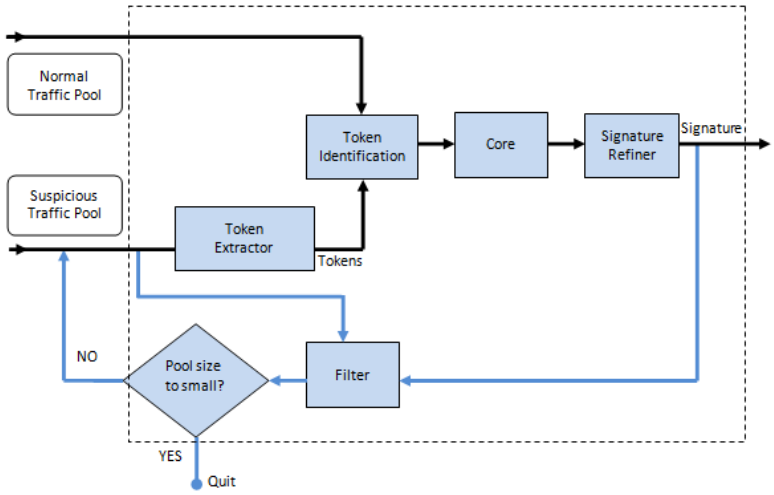

Figure 6. Hamsa Signature Generator

Advantages of Hamsa

- Hamsa achieves significant speed up for token extraction over polygraph.
- Hamsa is noise tolerant. It can generate signature in presence of noise in suspicious flow.
- Hamsa achieves significant accuracy and attack resilient over polygraph.

Limitations of Hamsa

- Although hamsa produces good signature, even in presence of noise, its behavior is not so accurate if malicious user mislead with forged invariant.
- Hamsa may not be resilient to few attacks such as Target feature manipulation, suspicious pool poisoning and innocuous pool poisoning attacks described in [21].

PADS:

Yong Tang *et al.*, [22] proposed position-aware distribution signature (PADS). PADS claims to fill the gap between traditional signature and anomaly-based intrusion detection systems. PADS is a collection of position-aware byte frequency distributions. It is proved to be more flexible than contiguous string based signature while more precise than statistical anomaly based approaches. The signature is generated by considering the positive aspect of both the systems. In addition to invariant content, PADS also considers the variant parts of the worm content that follow certain distributions. Figure 7(redrawn from [22]) describes variant of a polymorphic worm.

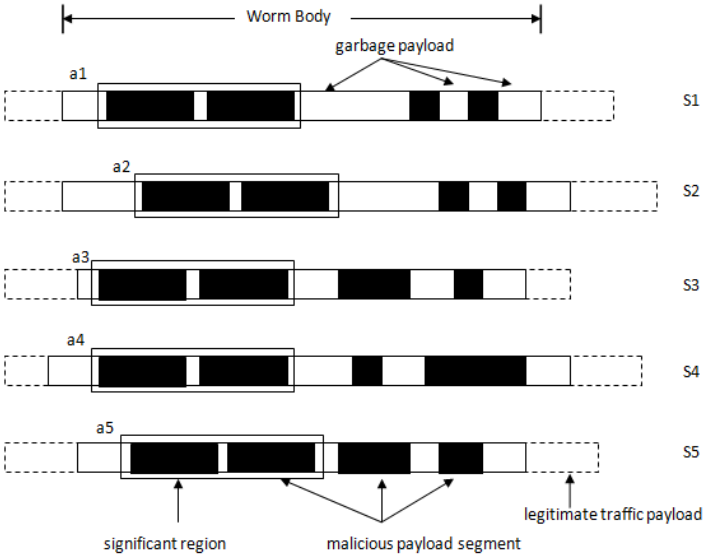

Figure 7. Variants of a Polymorphic Worm

The authors have used two algorithms, Expectation-Maximization (EM) and Gibbs Sampling to compute PADS signature from a collection of worm variants. Through an iterative method, EM computes maximum likelihood parameter estimation. EM follows three steps for PADS signature namely Initialization, Expectation and Maximization. In the Initialization step, starting position of significant regions of worm variants are assigned randomly. Maximum likely estimate of PADs signature is calculated based on initial guess of significant region. In the expectation phase the new guess on the locations of the regions are calculated based on estimated signature. Based on these new

guess on locations new maximum likelihood estimate of signal is calculated in maximization phase. This step is iterative. The process continues till it achieves a pre-defined threshold matching scores.

However there were few limitations observed in EM algorithm. It may struck in a local maxima. There is absolutely no guarantee that global maxima will be reached. These problems are addressed by Gibbs sampling algorithm through the following three steps: Initialization, Predictive Update and Sampling

Advantages of PADS:

- PADS bridge the gap between signature based approach and statistical anomaly based approaches.
- It is able to separate different polymorphic worms from a mixture and generate signature for each of them.

Limitation of PADS:

- In presence of noise in suspicious traffic the accuracy of PADS signature can't be assured.

Polytree:

Y.Tang et al., present Polytree [23], a network based signature generation scheme to defend against polymorphic worms. Polytree organizes worm signatures as a tree like structures, based on "more specific than" relation. Each node in the tree represents a signature. Signature is represented in the form of simplified regular expression (SRE). Child node must be *more specific* than the one of its parent node. Polytree signature extends the polygraph's token subsequence signatures with length constraints. Polytree consist of two main components- Signature generator and Signature selector.

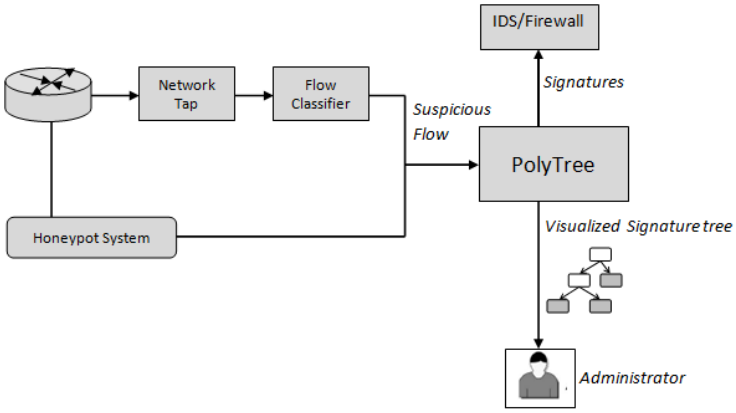

Figure 8. Deployment of Polytree

Polytree uses similar architecture as Polygraph [10], Hamsa [11] and LESG [14]. Figure 8 (redrawn from [23]) depicts typical deployment of Polytree. Polytree captures the traffic from a network tap and the traffic passes through an existing flow classifiers [17, 24, 25] or through honeypot [18, 26]. The traffic will be reassembled, transforming packets into contiguous byte flow. The flows are classified as suspicious or normal. Suspicious flows are sent to Polytree signature generator, which construct incrementally updated signature tree. Signature is generated using Multiple Sequence Alignment (MSA) algorithm. Signature selector selects a set of signature from signature tree and submits them to IDS for new worm detection. Polytree is capable of generating

signature of single polymorphic worm as well as multiple polymorphic worms. Figure 9(redrawn from [23]) describes Polytree signature generation.

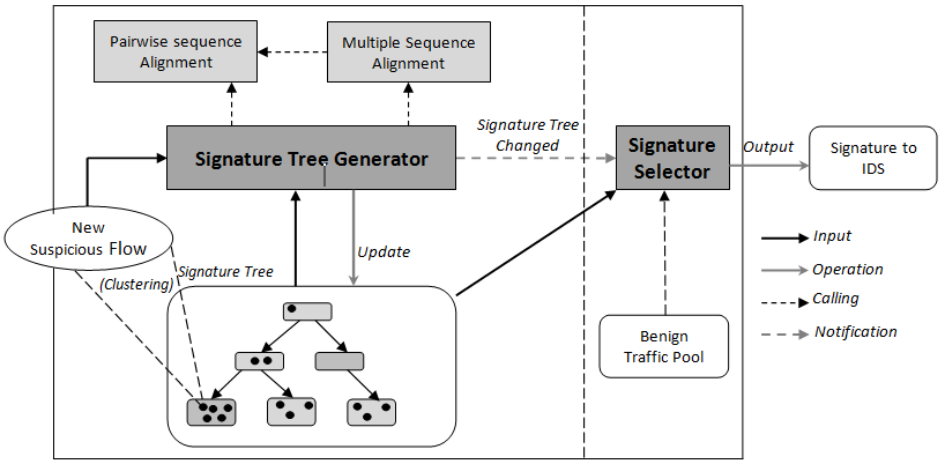

Figure 9. Polytree Signature Generation Architecture

Advantages of Polytree:

- Signature tree make it easy to organize and maintain worm signatures.
- It creates a balance between false positive rate and generalization ability of worm signatures.
- Signature tree gives the evolution map of a polymorphic worm. *i.e.*, how a worm evolves to a number of variants over a period of time.
- Signature tree provides a way to quick and online signature generation.
- Polytree signature is noise tolerant.
- Polytree is resilient to following attacks. Red herring attack, coincidental-pattern attack, tokenfit attack and fake anomalous flow attack.

Limitations of Polytree:

- Polytree may suffer from overhead of signature tree update when the number of worm sample is very large.
- Although it is in general attack resilient Polytree may not be appropriate defense tool for future sophisticated worms.

5. Exploit Specific (Host based) signature:

Taintcheck:

Newsome and Song proposed a method, called dynamic Taint Analysis [27], for automatic detection, analysis and generating signature of worms that exploit commodity software. The data originated from an untrusted source are labeled as tainted. The propagation of tainted data is monitored as the program executes. An attack is detected when the data is used in an anomalous way. The most common attack in network is overwrite attack, where valuable data such as return address, function pointers value format string are corrupted. This approach detects overwrite attacks. After the attack is

detected, dynamic taint analysis can provide the details about vulnerabilities. Such details are used to automatically generate signatures.

Following three components help dynamic taint analysis to detect an attack and generates signature from them. Figure 10(redrawn from [27]) describes the detection by TaintCheck.

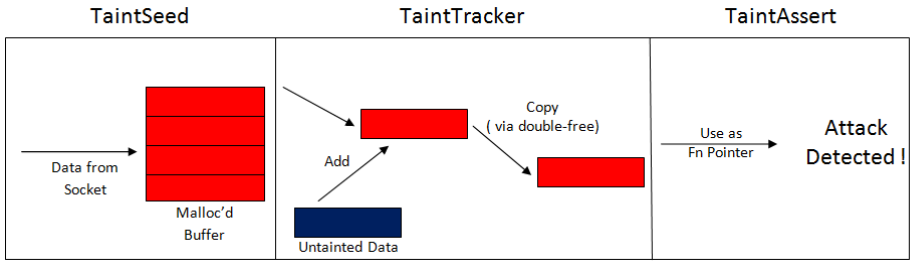

Figure 10. Taintcheck Detection of an Attack

TaintSeed: TaintSeed marks any data originates or derived from untrusted source as tainted. For example Input from network sockets, by default considered as tainted. TaintSeed examines the argument and result of each system call to check whether memory written by system call has been tainted. If the memory focused to be tainted, a taint data system is allotted to record system call number, current stack status etc. The shadow memory location is then set to a pointer to this data structure.

TaintTraker: TaintTracker tracks the propagation of tainted data. TaintTracker allocate a taint structure with relevant information and points back to previous taint structure. This chain structure is used to determine how the tainted data propagates through the memory.

TaintAssert: TaintAssert check whether the tainted data used in illegitimate ways, such as overwriting the value of jump address, format string, or system call arguments with attacker data. When TaintAssert confirms such anomaly, the exploit analyzer provides detail information about the exploit. Also it provides semantic information about attack payload. All these information used in automatic semantic analysis to generate signature.

Advantages of TaintCheck:

- Does not require source code or specially compiled binaries.
- Reliably detects most overwrite attacks
- Enable automatic semantic analysis based signature generation.
- Taintcheck can be used as flow classifier to reduce the false positive/false negative rate.

Disadvantages of TaintCheck:

- TaintCheck technique is very application specific; a certain version of servers must be deployed to monitor as vulnerability to discuss how the worm interacts with the server [11].

DACODA:

Based on the assumption that certain exploit vector must be present for the exploit to work Crandall *et al.*, present DACODA[28], a symbolic execution tool for polymorphic

worms. DACODA tracks data from attacker's network packet. Every byte of the packet labeled with an unique integer. These unique integers are stored in NE2000 device's memory pages, read into processor through I/O port. DACODA discovers equality predicates when a labeled byte or symbolic expression is used as jump call target. DACODA dynamically traces and correlates the network input to control flow changes to find the malicious input and infer the properties of worms [11].

6. Vulnerability – Driven (Network Based) signature:

LESG: Lanjia Wang *et al.*, [14] presented vulnerability driven, network-level, length-based, automated signature for zero day polymorphic worms. Buffer overflow is the most common type attacks in network. LESG signature exploits the intrinsic characteristics of buffer overflow attacks. One of the major characteristics of buffer overflow is that length of certain protocol field is longer than normal requests. Buffer overflow vulnerability occurs when there is a vulnerable buffer in the server implementation, and some part of the protocol messages can be mapped to the vulnerable buffer.

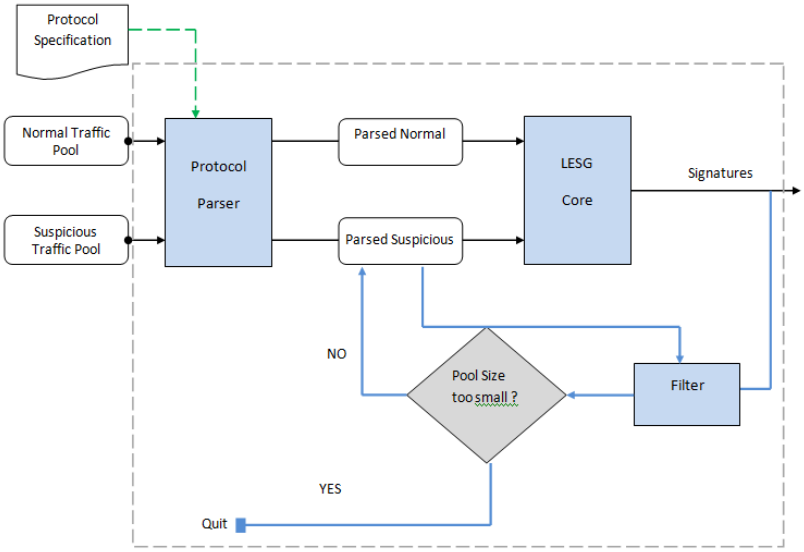

Figure 11. LESG Signature Generation Architecture

The architecture of LESG implementation is similar to Polygraph [10] and Hamsa [11]. The architecture is shown in figure 11(redrawn from [14]). LESG first sniffs the packets from network and classifies them as a different application level protocol based on port numbers and other protocol identifiers. For each protocol the traffic is then segregated as suspicious and normal traffic pool. A flow classifier such as honeynet, port scan detection or byte frequency detection helps in this job. The suspicious traffic pool and normal pool passes through a protocol parser. Protocol parsing generates (field ID, length of protocol field) pairs for both normal and suspicious traffic flow. Protocol semantics are specified and the length information of both the traffic pool given as input to LESG signature generation algorithm to generate length based signatures. An open source tool BINAPAC and Bro is used for packet flow re-assembling and protocol parsing in LESG.

LESG works in following three steps:

- Field Filtering: In this step the search space is narrowed by selecting field that could be candidate signature. Two parameters namely false positive (FP_0) and detection coverage (COV_0) are taken as a input to this step. This step use the fact that in buffer overflow attacks the true worm samples should have longer lengths on the vulnerable fields than the normal flows.
- Signature length optimization: In this step the length value of each candidate signature is optimized using a score function defined in [11], Score (COV_{l_j}, FP_{l_j}) for each field f_j . The algorithm resolves the tradeoff between detection coverage and false positive rate.
- Signature Pruning: This step finds an optimal subset of the candidate signature as the final signature set.

Advantages of LESG:

- Fast and Noise tolerant.
- Have analytical attack resilience bound under certain assumptions.
- LESG signatures can handle most of the buffer overflow attacks.

Limitations of LESG:

- Length-based signatures are not directly based on the exact vulnerability analysis and lack many vulnerability details.
- For the above reasons, signatures generated by LESG are less accurate, compared to other vulnerability based signatures [32][35].

7. Vulnerability- Driven (Host Based) signature:

COVERS:

COVERS [29], an automated signature generation method was introduced to give protection against large scale, repetitive attacks. COVERS utilize the fact that all memory error exploits involve corruption of pointer values, and this value must be included in attack input. It uses forensic analysis of victim server's memory to correlate attacks to input received over the networks. This approach is context based and vulnerability oriented. The signature generated by COVERS look for the vulnerability characteristics (such as length of a message field). COVERS can handle other attack includes polymorphic attack exploiting similar vulnerability.

COVERS follow following four steps for generate signature:

- *Attack Detection* – Buffer overflow attack is detected using memory error exploitation technique address-space-randomization (ASR) [30][31] .
- *Correlation to input*- This step look forwards to packets involved in the attack and more specifically the bytes responsible for it. In all memory attack pointer value must get corrupted. These values are important to get the clue of the attack. Therefore it is must be included in attack input. A forensic analysis is done of the victim process memory surrounding the corrupted pointer value, and matches this region with recent inputs.

- *Identifying input Context* – This step identifies logical input context within which an attack appears. The approach use simple specification of message formats to guide the input context identification step.
- *Generating Signatures:* Cover generate signature by exploiting underlying vulnerabilities, such as length of message field (excessive long) or presence of binary data within text valued fields.

Figure 12 (redrawn from [29]) presents the implementation architecture of COVERS.

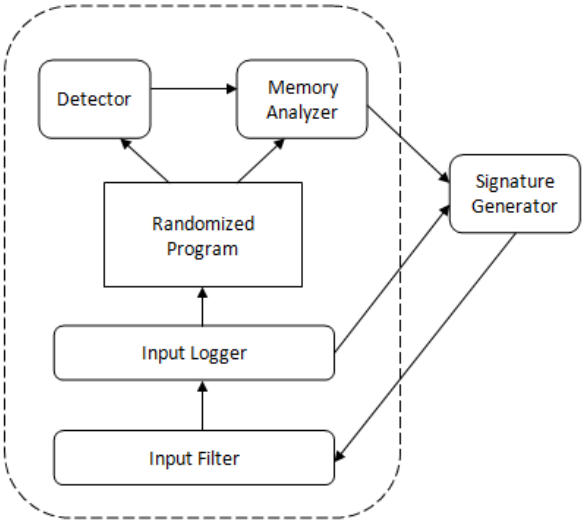

Figure 12. COVERS Implementation Architecture

Advantages of COVER:

- Cover signature claims to be fast and effective against attacks. It is effective against Denial of Service attacks and randomization attacks.
- Low overheads under normal operations.
- Cover does not require any modifications to the protected server software, or access to its source code.

Limitations of COVER:

- COVERS could be subject to a miscorrelation attack in which an intelligent attacker duplicates memory data collected from an unrelated exception into a packet which carries no exploit payload. As a result, a false signature could be generated to filter out legitimate packets [34].
- COVER is application specific.

Vigilante:

Vigilante [32] is a host based signature generation scheme that uses end-to-end approach to contain worms automatically. Any software running on a host is vulnerable to worm infection. Vigilante use self-certifying alert (SCA) to get the information of infection of a particular host. Host detects worms by instrumenting network facing services to analyze infection attempt. These analysis data are used to generate SCAs and are distributed among hosts. When a host receives SCA it verifies the infection

possibilities in it. Alerted host protect themselves by generating filters using dynamic data and control flow analysis of the execution path any worm follows when infecting a host. Each host follows this procedure of protecting by installing the filters. Figure 13(Redrawn from [32]) depicts the procedure of automatic containment of worms by vigilante.

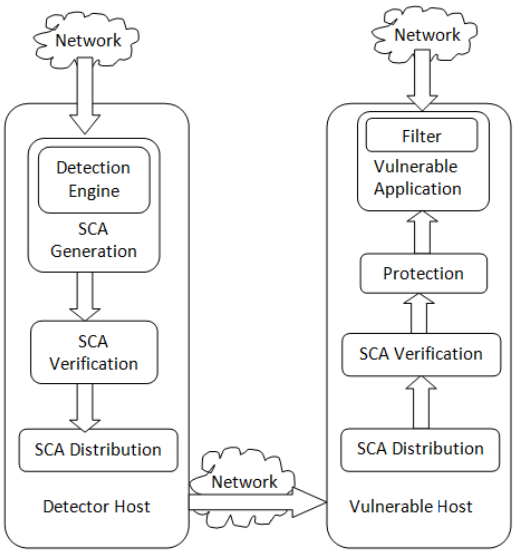

Figure 13. Worm Containment in Vigilante

To cover most of the vulnerability vigilante has developed three different types of alerts- Arbitrary execution control alert, arbitrary code execution alert, arbitrary function argument alert. SCAs are sequence of messages. When received by a host, which runs vulnerable services, causes it to reach a disallowed state. SCAs are verified by sending the messages to the service and checking whether it reaches to disallowed state. The SCAs have a common format: an identification of the vulnerable service, an identification of the alert type, verification information to aid alert verification and sequence of message with the network end points. Figure 14(redrawn from [32]) shows the SCA verification process by the host receiving it. The verification process runs on virtual machine of the host, and it involves sending the sequence of messages in the alert to a vulnerable service. Host generates SCAs whenever there is an infection attempt. The detection engine is enabled and it searches the log to generate candidate SCAs and run the verification process for each candidate. After the SCAs are being generated it broadcast to other host, so that they can protect themselves from similar vulnerability. Upon receiving SCAs the local host generates filter after verifying the SCAs.

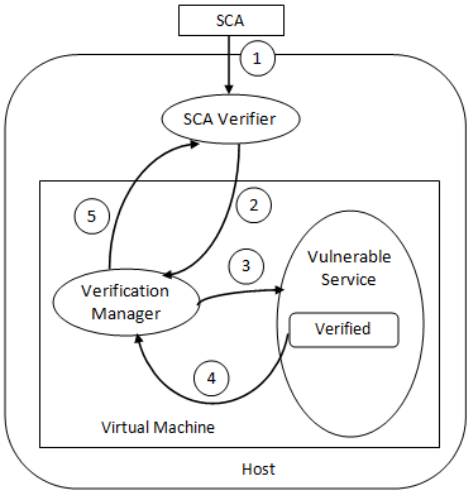

Figure 14. SCA Verification

Advantages of Vigilante:

- Vigilante claims to automatically contain fast-spreading worm including unknown worms.
- False positive rate is less.
- Memory and CPU overhead is small

Limitations of Vigilante:

- Vigilante signature may lead to false negative
- There may be replay attacks during broadcasting of SCAs to other hosts.

Semantic aware malware detection:

Mihai et al.,[33] present Semantic-aware malware detection. In this work author addressed the deficiency in pattern-matching approach used in commercial virus scanner. This approach incorporates instruction semantics to detect malicious program. It has described malicious behaviors using templets, which are instruction sequence with variables and constants used.

The malware detection works by finding, for each templet mode, a matching node in the program. A node in templet and in program matches if there exist an assignment to variables from the templet node expression that unifies it with the program node expression. If match found between templet node and program node, it is time to check whether *def-use* relationship that exist between templet node also hold true for corresponding program node. (Two nodes are said to be in *def-use* relationship if one node's definition of a variable reaches the other node's use of same variable) If all the nodes in the templet have matching counterparts under these conditions, the algorithm has found a program that satisfies the templet and produces a proof of this relationship. The algorithm is implemented as malware detection tool. The tool takes an input a templet and a binary program and determines whether a fragment of the suspicious program contains a behavior specified by the templet.

Advantages of Semantic aware malware detection:

- It can handle following obfuscation transformation: code reordering, equivalent instruction replacement, register renaming, and garbage insertion.

Limitations of Semantic aware malware detection:

- The algorithm is not able to handle equivalent functionality and reordered memory access obfuscation transformation
- The algorithm can detect only those program that exhibit the same ordering of memory updates.
- The approach is computationally expensive.

Packet Vaccine:

Inspired by biological vaccine, X.Wang et al.,[34] presented packet vaccine, a black box exploit detection and signature generation, especially for commodity software. It detects and analyzes an exploit using output of a vulnerable program. An exploit most likely causes an exception when attempt to hijack the control flow of a vulnerable program. Taking such exploit as a templet, packet vaccine creates set of new vaccine to investigate the condition, under which exploit happen. Using these conditions a signature is generated to prevent similar attacks. This approach does not require knowledge of vulnerable program's source and binary code. As packet vaccine work n black-box mode it may even work on commodity software, obfuscated for copyright protection.

Most exploits injects a jump address to redirect the control flow of a vulnerable program. Such an address points to somewhere in the stack or heap in a code-injection attack, or to a global library entry in an existing-code attack. Based on this assumption, the vaccine is generated in following ways:

- Check every 32 bit or 64 bit sequence in a packet's application payload.
- Randomize those which fall in the address range of the potential jump targets in a protected program. Randomization helps preserving exploit semantic tokens.
- These vaccines generated in this way cause an exception such as segmentation fault, illegal instruction fault, to a vulnerable program.
- The vulnerability is analyzed by identifying the jump address and its location in an exploit packet. The known exploit is taken as templet to generate faults (vaccines) and inject into a vulnerable program to collect required attributes of underlying vulnerabilities.

Signature Generation: After the vaccine injection the approach can generate signature with or without application level information. Signature when generated without knowledge of application protocol form a token sequence- an ordered sequence of byte strings (tokens) with their location information. Signature when generated with application protocol specification form a vulnerability-based signature. These signatures use the characteristics of buffer overflow exploit and format string exploit to describe vulnerabilities. Such signature takes the form (application, command, field.name, max.field.size).

Advantages of Packet Vaccine:

- Packet vaccine provides a fast mean to detect exploit attempt by injecting vaccine packets in a program.
- Packet vaccine does not require source code or recompiled binary information to generate signature. Therefore it can be used to protect even obfuscated commodity software.

- Packet vaccine signature use host information, therefore immune to interference from internet noise.
- Packet vaccine is light weight and easy to deploy.

Limitations of Packet Vaccine:

- Packet vaccine requires decoding a packet for vaccine generation, for this reason it may not be useful at the edge of a network to protect a wide range of Internet services and applications.
- Packet Vaccine will not work directly on packets with encrypted payload or checksums.

Vulnerability-based signature:

Burneley et al., presented concept of vulnerability signature [35]. They have given a formal definition of vulnerability signature. Vulnerability signature matches, a set of inputs which satisfy a vulnerability condition in the program. Vulnerability condition is nothing but specification of program bug. The authors in this paper explored the related design space and shown the tradeoff between signature matching and accuracy. Three classes of vulnerability signatures have been discussed: Turing machine signatures, symbolic constraint signature and regular expression signature.

Turing machine(TM) signature consists of instruction that lead to vulnerability point with vulnerability condition meet. The programs that do not reach vulnerability point are termed benign. Whereas program that reaches vulnerability point with vulnerability condition are exploits. *Symbolic constraint signature* is a set of Boolean formulas that approximate TM signatures. Symbolic constraint signature does not have loop. Such kind of signature is not as precise as TM signatures. *Regular expression signature* is least accurate but fastest among all signature classes.

8. Comparison of Polymorphic worm signature Generation techniques:

We extend the comparison of different polymorphic signature schemes presented by hamsa [11] in Table II below.

Table II. Comparison of Polymorphic Worm Signatures

	Network /Host based	Content/Behavior based	Noise Tolerance	Online Detection Speed	General purpose/ Application Specific	Provable attack resilience	Information Exploited
Nemean[15]	Host	Content based	No	Fast	Protocol Specific	No	ε
CFG[16]	Network	Behavior based	Yes	Slow	General	No	π
Polygraph[10]	Network	Content based	Yes (slow)	Fast	General	No	εγπ
Hamsa[11]	Network	Content based	Yes	Fast	General	Yes (Limited)	εγπ
PADS[22]	Host	Content	No	Fast	General	No	εγπ

		based					
Polytree[23]	Network	Content based	Yes	Fast	General	Yes (Limited)	εγπ
TaintCheck[27]	Host	Behavior based	Yes	Slow	Application Specific	No	γπ
DACODA[28]	Host	Behavior based	Yes	Slow	Application Specific	No	εγπ
LESG[14]	Network	Behavior based	Yes	Fast	General	Yes	γ
COVERS[29]	Host	Behavior based	Yes	Fast	Server Specific	No	εγ
Vigilante[32]	Host	Behavior based	Yes	Slow	General	No	γ
Malware Detection[33]	Host	Behavior based	Yes	Slow	General	No	π
Packet Vaccine[34]	Host	Behavior based	Yes	Slow	Application Specific	No	γπ
Vulnerability Signature[35]	Host	Behavior based	Yes	Slow	General	No	π

9. Conclusion and Future Work

In this survey we have presented the anatomy of polymorphic worms. We have discussed challenges of automatic signature generation for zero-day worms. We have classified the polymorphic worm signature generation based on their deployment at network level or at host level. Within this broad category we have further classified them as exploit-specific and vulnerability driven. We have discussed existing polymorphic signature generation schemes with their advantages and limitations. The merits and challenges of the existing scheme will provide future direction of research on polymorphic signature generation. As pointed out by David Burmley et al. [36], that a hybrid approach of proactive protection and reactive antibody defense may be most effective in protecting from sophisticated worm attacks. Signature based approach may proved to the most essential part of such approaches. Most of the new worms are not novel and are derived from some existing worms [37]. Therefore knowledge of existing worm defense will help in better equipped for protecting from most of the new worms.

Acknowledgements

This work was partially supported by Birla Institute of Technology, Mesra, Ranchi 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] B.K. Mishra and D. K. Saini, "SEIRS Epidemic Model with Delay for Transmission of Malicious Objects in Computer Network", Applied Mathematics and Computation, Elsevier, vol. 188, **(2007)**, pp.1476–1482.

[2] B.K. Mishra and S. K Pandey, "Fuzzy Epidemic Model for the Transmission of Worms in Computer Network", Nonlinear Analysis: Real World Applications, Elsevier, vol. 11, **(2010)**, pp. 4335-4341.

[3] O.A. Toutonji, S.-M. Yoo and M. Park, "Stability Analysis of VEISV Propagation Modeling for Network Worm Attack", Applied Mathematical Modelling, Elsevier, vol. 36, **(2012)**, pp.2751–2761.

[4] P. Li, M. Salour, and X. Su, "A Survey of Internet Worm Detection and Containment", Communications Surveys & Tutorials, IEEE, vol. 10, **(2008)**, pp. 20-35.

[5] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time", Computer Networks, vol. 31, **(1999)**, pp.23-24.

[6] M. Roesch, "Snort: The Lightweight Network Intrusion Detection System", http://www.snort.org/, **(2001)**.

[7] P. Vinod, V. Laxmi and M.S. Gaur, "Survey on Malware Detection Methods", Proc Hackers Workshop, IIT, **(2009)**; Kanpur.

[8] C. Kruegel and G. Vigna, "Anomaly Detection of Weh-based Attacks", Proc. ACM Conference Computer and Communication Security, ACM, **(2003)**, pp. 251-261.

[9] K. Wang and S. J. Stolfo, "Anomalous Payload-based Network Intrusion Detection", Proc. 7th International Symposium on Recent Advances in Intrusion Detection, **(2004)**.

[10] J. Newsome, B. Karp, and D. Song. "Polygraph: Automatically Generating Signatures for Polymorphic Worms", Proc. IEEE Security and Privacy Symposium, **(2005)**.

[11] Z. Li, M. Sanghi, Y. Chen, M. Kao, and B. Chavez, "Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience", Proc. IEEE S&P, **(2006)**.

[12] 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.

[13] S. Paul and B.K. Mishra, "PolyS-Network based Signature Generation for Zero-day Polymorphic Worms", SERSC International Journal of Cloud and Distributed System, vol. 6, no. 4, **(2013)** August.

[14] L. Wang, Z. Li, Y. Chen, Z. Fu and X. Li., "Thwarting Zero-day Polymorphic Worms with Network-Level Length-Based Signature Generation", IEEE/ACM transactions on networking, vol. 18, no. 1, 2010, pp. 53-65.

[15] V.Yegneswaran J.T.Giffin, Paul. Barford, S.Jha, "An Architecture for Generating Semantic-aware Signatures", Proc. USENIX Security Symposium, **(2005)**, p.7.

[16] C. Kruegel, E.Kirda, "Polymorphic Worm Detection Using Structural Information of Executables", in Proc. RAID, **(2005)**, pp. 207–226.

[17] H. Kim and B. Karp, "Autograph: Toward Automated, Distributed Worm Signature Detection", In USENIX Security Symposium, **(2004)**.

[18] C. Kreibich and J. Crowcroft, "Honeycomb - Creating Intrusion Detection Signatures Using Honeypots", Proc. of the Workshop on Hot Topics in Networks (HotNets), **(2003)**.

[19] S. Singh, C.Estan, G.Varghese and S.Savage, "Automated Worm Fingerprinting", Proceedings of the 6th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI), **(2004)** December.

[20] G. Manzini and P. Ferragina, "Engineering a Lightweight Suffix Array Construction Algorithm", Algorithmica, vol. 40, no. 1, **(2004)**.

[21] L. Cavallaro, A. Lanzi, L. Mayer and M. Monga, "LISABETH: Automated Content-Based Signature Generator for Zero-day Polymorphic Worms", Proc. of the Fourth International Workshop on Software Engineering for Secure Systems, **(2008)**; Leipzig, Germany.

[22] Y Tang, S Chen," An Automated Signature-Based Approach against Polymorphic Internet Worms", IEEE Transaction on Parallel and Distributed Systems, **(2007)** July, pp. 879-892.

[23] Y.Tang, B. Xiao and X. Lu, "Signature Tree Generation for Polymorphic Worms", IEEE Transactions on Computers, vol. 60, no. 4, **(2011)**.

[24] K. Wang, S. J. Stolfo, "Anomalous Payload-based Network Intrusion Detection", Proc. 7th Infemtional Symposium on Recent Advances in Intrusion Detection, **(2004)**.

[25] K. Wang, G. Cretu and S. J. Stolfo, "Anomalous Payload-based Worm Detection and Signature Generation", Proc. 8th Infemtional Symposium on Recent Advances in Intrusion Detection, **(2005)**.

[26] Y. Tang, S. Chen, "Defending Against Internet Worms: A Signature-Based Approach", In Proc. IEEE INFOCOM, **(2005)**.

[27] J. Newsome and D. Song, "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software", NDSS, **(2005)**.

[28] J. R. Crandall, Z. Su, and S. F.Wu, "On Deriving Unknown Vulnerabilities from Zero-day Polymorphic and Metamorphic Worm Exploits", Proc. ACM CCS, **(2005)**, pp. 235–248.

[29] Z. Liang and R. Sekar, "Fast and Automated Generation of Attack Signatures: A Basis for Building Self-Protecting Servers", Proc. ACM CCS, **(2005)**, pp. 213–222.

[30] A. Baratloo, N. Singh, and T. Tsai, "Transparent Run-time Defense Against Stack Smashing Attacks", USENIX Annual Technical Conference, **(2000)**.

[31] S. Bhatkar, D. DuVarney, and R. Sekar, "Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits, USENIX Security, **(2003)**.

[32] M. Costa, J.Crowcroft, M.Castro and A. Rowstron, "Vigilante: End-to-end Containment of Internet Worms", Proc. ACM SOSP, **(2005)**, pp. 133–147.

[33] M. Christodorescu and S. Jha, "Semantics-Aware Malware Detection", In IEEE Symposium on Security and Privacy, **(2005)**.

[34] X. Wang, Z. Li and J.Y.Choi, "Packet Vaccine: Black-box Exploit Detection and Signature Generation", Proc. ACM CCS, **(2006)**.

[35] D. Brumley, J.Newsome, D.Song, H.Wang and S.Jha, "Towards Automatic Generation of Vulnerability based Signatures", Proc. IEEE Security Priv. Symp., **(2006)**.

[36] D. Brumley, Li Liu, D.Song and P.Poosankam, "Design Space and Analysis of Worm Defense Strategies", Proc. ACM ASIACCS, **(2006)**.

[37] M. E. D. Kienzle, "Recent Worms: A Survey and Trends," Proc. ACM WORM '03, **(2003)**.

[38] T. Fan. Y.Li, F.Gao, "Study of Virus Propagation Model in Cloud Environment", SERSC International Journal of Security and its Applications, vol. 7, no. 4, **(2013)**.

[39] J. Raiyn "A Survey of Cyber Attack Detection Strategies", SERSC International Journal of Security and its Applications, vol. 8, no. 1, **(2014)**.

[40] S.Paul and B.K. Mishra, "Honeypot Based Signature Generation for Defense against Polymorphic Worm Attacks in Networks", Proceeding of IEEE International Advanced Computing Conference (IACC), **(2013)** February.

Authors

Sounak Paul is an Assistant professor with Birla Institute of Technology (BIT), Mesra, Ranchi, India, presently deputed at the Department of Computer Science & Engineering, Birla Institute of Technology, International centre, Muscat, Oman (Waljat College of Applied Sciences). He received his Master in Technology in Computer science and Engineering from Indian Institute of Technology (IIT), Guwahati, India. He earned his Master in Computer Applications (MCA) from Birla Institute of Technology, Mesra, Ranchi, India. His research interests include network security specially malware analysis and defense, intrusion detection.

Bimal Kumar Mishra is a Professor with the Department of Applied Mathematics and Dean Student Welfare, Birla Institute of Technology (BIT), Mesra, Ranchi, India. He was Associate dean faculty and sponsored research before this assignment. He received his Master degree in Operational Research from University of Delhi, India (1992) and Masters in Mathematics too. He earned his Ph. D. degree from Vinoba Bhave University, Hazaribag, India (1997) and subsequently earned his D.Sc. Degree from Berhampur University, Orissa, India in 2007. His research interests include Nonlinear Analysis specifically mathematical modeling of cyber attack and its defense. He is editor in chief of International Journal of mathematical modeling, Simulations & Applications and International Journal of mathematical modeling and Computing. He is the member in editorial board of several international Journals. He has published more than 90 research papers in journals of repute and conference proceedings. Presently he is working in the area of cyber attack/crime and its defense mechanism.

