Network-based and Attack-resilient Length Signature Generation for Zero-day Polymorphic Worms

Zhichun Li, Lanjia Wang†, Yan Chen and Zhi (Judy) Fu‡
Northwestern University, Evanston, IL, USA
†Tsinghua University, Beijing, China
‡Motorola Labs, Schaumburg IL, USA

***Abstract*—It is crucial to detect zero-day polymorphic worms and to generate signatures at the edge network gateways or honeynets so that we can prevent the worms from propagating at their early phase. However, most existing network-based signatures generated are not vulnerability-based and can be easily evaded by attacks. In this paper, we propose generating vulnerability-based signatures on the network level without any host-level analysis of worm execution or vulnerable programs. As the first step, we design a network-based Length-based Signature Generator (LESG) for worms based on buffer overflow vulnerabilities[1]. The signatures generated are intrinsic to buffer overflows, and are very hard for attackers to evade. We further prove the attack resilience bounds even under worst case attacks with deliberate noise injection. Moreover, LESG is fast and noise-tolerant and has efficient signature matching. Evaluation based on real-world vulnerabilities of various protocols and real network traffic demonstrates that LESG is promising in achieving these goals.**

I. Introduction

Computer worms are serious threats to the Internet causing billions of dollars in economic loss. Recently, zero-day worm attacks that exploit unknown vulnerabilities have become popular [2]. Intrusion detection/prevention systems (IDSes) [3], [4] are proposed to defend against malicious worm attacks by searching the network traffic for known patterns, or *signatures*. Such signatures for the IDSes are currently generated manually or semi-manually, a process too slow to defend against self-propagating computer worms.

Thus it is critical to automate the process of worm detection, signature generation and signature dispersion in the early phase of worm propagation, especially at the network level (gateways and routers). There is some existing work towards this direction [5]–[7]. However, to evade detection by signatures generated with these schemes, attackers can employ *polymorphic* worms which change their byte sequence at every successive infection. Recently, some polymorphic worm signature generation schemes are proposed. Based on characteristics of the generated signatures, they can be broadly classified into two categories – *vulnerability-based* and *exploit-based*. The former signature is inherent to the vulnerability that the worm tries to exploit. Thus it is independent of the worm implementation, unique and hard to evade, while exploit-based signatures capture certain characteristics of a specific worm implementation. However, schemes of both categories have their limitations.

Existing vulnerability-based signature generation schemes are host-based and cannot work at the network router/gateway level. These schemes [8]–[10] either require exploit code execution or the source/binary code of the vulnerable program for analysis. However, such host-level schemes are too slow to counteract the worms that can propagate at exponential speed. Given the rapid growth of network bandwidth, today's viruses/worms can propagate quickly and infect most of the vulnerable machines on the Internet within ten minutes [11] or even less than 30 seconds with some highly virulent techniques [12], [13] at near-exponential propagation speed. At the early stage of worm propagation, only a very limited number of worm samples are active on the Internet, and the number of machines compromised is also limited. Therefore, signature generation systems should be network-based and deployed at high-speed border routers or gateways where the majority of traffic can be observed. Such a requirement for network-based deployment severely limits the design space for detection and signature generation as discussed in Section II.

Existing exploit-based schemes are less accurate and can be evaded. Some of these schemes are network-based and are much faster than those in the former category. However, most of such schemes are content-based, which aim to exploit the residual similarity in the byte sequences of different instances of polymorphic worms [14]–[18]. As mentioned in [18], there can be some worms which do not have any content-based signature at all. Furthermore, various attacks have been proposed to evade the content-based signatures [19]–[22]. The rest of the schemes in this category [23], [24] generate signatures based on exploit code structure analysis, which is not inherent to the vulnerability exploited and can also be evaded [19].

Therefore, our goal is to design a signature generation system which has both the accuracy of vulnerability-based schemes and the speed of exploit-based schemes so that we can deploy it at the network level to thwart zero-day polymorphic worm attacks. As the first step towards this ambitious goal, we propose LEngth-based Signature Generator (called $LESG$) which is a network-based approach for generating efficient and length-based signatures which cannot be evaded. That is, even when the attacker knows what the signatures are and how the signatures are generated, they still cannot find an efficient and effective way to evade the signatures.

Length-based signatures target buffer overflow attacks which constitute the majority of attacks [1]. The key idea is that in order to exploit any buffer overflow vulnerabilities, the length of certain protocol fields must be long enough to overflow the buffer. A buffer overflow vulnerability happens when there is a vulnerable buffer in the server implementation and some part of the protocol messages can be mapped to the vulnerable buffer. When an attacker injects an overrun string input for the particular field of the protocol to trigger the buffer overflow, the length of such an input for that field is usually much longer

[1]It is reported that more than 75% of vulnerabilities are based on buffer overflow [1].

than those of the normal requests. Thus we can use the field input length to detect attacks. This is intrinsic to the buffer overflow, and thus it is very hard for worm authors to evade.

In addition to being network-based and having high accuracy, LESG has the following important features.

Noise tolerance. Signature generation systems typically need a flow classifier to separate potential worm traffic from normal traffic. However, network-level flow classification techniques [7], [25]–[28] invariably suffer from false positives that lead to noise in the worm traffic pool. Noise is also an issue for honeynet sensors [5], [16], [23]. For example, attackers may send some legitimate traffic to a honeynet sensor to pollute the worm traffic pool and to evade noise-intolerant signature generation. Our LESG is proved to be noise tolerant or even better, attack resilient, *i.e.*, LESG works well with maliciously injected noise in an attempt to mislead NIDS [19].

Efficient Signature Matching. Many users patch their systems slowly due to the fact that they may have to restart the applications or reboot the machines. It is more efficient and effective to deploy signatures at the NIDS/firewall to filter out the malicious traffic of an entire enterprise network. Since the signatures generated are to be matched against *every flow* encountered by the NIDS/firewall, it is critical to have fast signature matching algorithms.In the LESG system, the length-based signatures can be matched at the network level with a protocol length parser without any host-level analysis.

In the rest of the paper, we first survey related work in Section II and discuss the LESG architecture in Section III. Then we present the length-based signature generation problem in Section IV, generation algorithm in Section V, and its attack resilience in Section VI. After that, in Section VII, we use real Internet traffic and seven real exploit code (enhanced with polymorphic capabilities) on five different protocols to test the performance of LESG prototype. Results show that LESG is highly accurate, noise tolerant, capable of detecting multiple worms in the same protocol pool, and capable of online signature generation with small memory consumption. Finally, we discuss some practical issues in Section VIII and conclude in Section X.

II. Related Work

Property of signatures	Signature generation mechanisms	
	Network-based	Host-based
Exploit-based	Polygraph [15], Hamsa [14], PADS [16], Nemean [23], CFG [24]	DACODA [18], Taint check [17]
Vulnerability-based	**LESG**	Vulnerability signature [10], Vigilante [29], COVERS [8], Packet Vaccine [9]

TABLE I
Comparison with other polymorphic worm signature generation schemes.

Early automated worm signature generation efforts include Honeycomb [5], Autograph [7], and EarlyBird [6], but they do not work well with polymorphic worms.

Existing work on automated polymorphic worm signature generation can be broadly classified into vulnerability-based and exploit-based. Based on signature generation input requirements, we can further categorize these schemes on another axis: host-based vs. network-based. The former requires either exploit code execution or the source/binary code of the vulnerable program . On the other hand, the network based approaches rely solely on network-level packets. The classification of existing schemes and LESG is shown in Table I.

Exploit-based schemes. We have discussed most of them in the introduction [14]–[18], [23], [24]. For example, Christopher *et al.*proposes using the structural similarity of the Control Flow Graph (CFG) to generate a fingerprint as signatures [24]. However, their approach can be evaded when the worm body is encrypted. Furthermore, compared with length-based signatures, it is much more computationally expensive to match the fingerprint with the network packets. Thus it cannot be applied to filter worm traffic at high-speed links.

In comparison with most recent work in this category, such as Hamsa [14], LESG has better attack resilience, *e.g.*, it has better bounds for deliberate noise injection attacks [19].

Vulnerability-based and host-based schemes. Brumley *et al.*presents the concept of a vulnerability signature in [10] and argues that the best vulnerability signatures are Turing machine signatures. However, since the signature matching for Turing machine signatures is undecidable in general, they reduce the signatures to symbolic constraint signatures or regular expression signatures. Their approach is a heavyweight host-based approach, which has high computational overhead and also needs some information such as the vulnerable program, multiple execution traces, and the vulnerability condition. Similarly, Vigilante [29] proposed a vulnerability-based signature which is similar to the MEP symbolic constraint signatures in [10].

Liang *et al.*proposed the first host-based scheme to generate length-based signatures [1], [8]. Packet Vaccine [9] further improves the signature quality by using binary search. Unfortunately, both of them are host-based approaches and are subject to the limitations mentioned before and some additional shortcomings. First, they need to know the vulnerable program. Sometimes, they have to try many different implementation versions to find the vulnerable ones. Second, the signature generated by [8] based on a small number of samples may be too specific to represent the overall worm population. Therefore, detection based on their generated signatures tends to have high false negatives. Moreover, the protocol specification language they used is not expressive enough for many protocols.

Other related work. There are previous research efforts on network-level detection of buffer overflow exploits. However, they do not generate any effective signatures due to high matching overhead and high false positives. TCTP [30] detects buffer overflow attacks by recognizing jump targets within the sessions. Approaches like SigFree [31] detect exploit codes based on control flow and data flow analysis.

III. Architecture of LESG

As shown in Figure 1, $LESG$ can be connected to multiple networking devices, such as routers, switches and gateways via a span (mirror) port or an optical splitter. Most modern switches are equipped with a span port to which copies of entire packets in the traffic from a list of ports can be directed. In addition, $LESG$ can also be used to monitor traffic for a large-scale honeynet/honeyfarm by sniffing traffic at its gateways. The honeynet/honeyfarm can be either centralized or distributed [32]–[34].

Similar to the basic framework of Polygraph [15] and

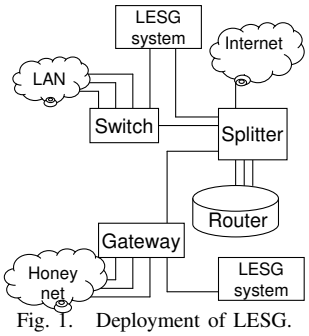

Fig. 1. Deployment of LESG.

Hamsa [14], we first need to sniff traffic from networks and classify the traffic as different application level protocols. Next, we filter out known worms and then further separate the traffic into a suspicious traffic pool and a normal traffic reservoir using an existing flow classifier [7], [25]–[28].

Similar to Polygraph [15] and Hamsa [14], we use an existing flow classifier that may use various techniques (such as honeynet/honeyfarm [32]–[34], port scan detection [7], [35], byte frequency detection [26], [27], and other advanced techniques) to identify suspicious flows. Note that the flow classifiers can operate at the line speed of routers as achieved in our previous work [35]. The scan detection based flow classifiers first detect the hosts scanning a particular port number and then classify successful TCP connections from any of the scanning hosts with that particular destination port number as suspicious flows. It is effective against any scanning worm. Meanwhile, the honeynet/honeyfarm based approach considers any traffic caught in the honeynet/honeyfarm as suspicious flows.

Leveraging the normal traffic selection policy mentioned in [14], we can create the normal pool. The suspicious pool and the normal pool are inputted to the signature generator as shown in Figure 2. We first specify the protocol semantics and use a protocol parser to parse each protocol message into a set of fields. Each field is associated with a length and a type. The field length information of both the suspicious pool and the normal pool are given as input to the "LESG core"(signature generation algorithm) module to generate the signatures.

A. Protocol Parsing

As emphasized in [36], protocol parsing is an important step in any semantic analysis of network traffic, such as network monitoring, network intrusion detection systems [3], [4], smart firewalls, *etc.*. We analyzed three text-based protocols (HTTP, FTP, and SMTP) and seven binary protocols (DNS, SNMP, SMB, WINRPC, SUNRPC, NTP, SSL). We find that, in general, it is much easier and faster to parse the lengths of the protocol fields than full protocol parsing.

Some recent research, such as BINPAC [36], has studied how to ease the job of writing a protocol parser. BINPAC is a yacc-like tool for writing application protocol parsers. It has a declarative language and compiler, and actually works as a parser generator. Its input is a script which is actually a protocol specification written in BINPAC language. The output is a parser code for that protocol. Currently, BINPAC is executed in connection with Bro [4], which implements other necessary traffic analysis at lower levels. With BINPAC, writing a protocol parser has been greatly simplified. Furthermore, not only can the available scripts provided by Bro be reused, but also many people can potentially contribute and produce more reusable protocol specifications for BINPAC as an open source tool. Because of these advantages, we use BINPAC and Bro for packet flow reassembling and protocol parsing in our research.

IV. Length-Based Signature Definition and Problem Statement

In this section, we model each application message as a field hierarchy, and present it as a vector of fields. Based on this model, we formally define the length-based signatures and the length-based signature generation problem.

A. Field Hierarchies

Each of the application sessions (flows) usually contains one or more Protocol Data Units (PDUs), which are the atomic processing data units that the application sends from one endpoint to the other endpoint. PDUs are normally specified in the protocol standards/specifications, such as RFCs. A PDU is a sequence of bytes and can be dissected into multiple *fields*. Here, a field means a sub-sequence of bytes with special semantic meaning or functionality as specified in the protocol standard. Typically, a field encodes a variable with a certain data structure, such as a string, an array *etc.*. Take the DNS protocol as an example. Figure 3 shows the format of the DNS PDUs. It has a header and four other sections – QUESTION, ANSWER, AUTHORITY and ADDITIONAL. Each section is further composed of a set of fields. The QUESTION section contains one or more DNS queries that are further composed of field class QNAME, QTYPE and QCLASS. The other three sections contain one or more Resource Records (RRs), and each RR is composed of six lower level fields (NAME, TYPE, *etc.*). Borrowing terms from the object model, we call the type of fields, such as QNAME and QTYPE, the *field class*, and each concrete instance of a certain field an *instance* of the field.

Among all the field classes in PDUs, some, *e.g.*, QNAME, NAME and RDATA, are *variable-length fields*; others are *fixed-length fields*, in which the instances all have the same length as defined in the protocol standard.

We make the following two observations on such a representation of PDU. First, the number of instances of one field class in a PDU may vary. For example, one PDU may contain one instance of field A, and another PDU may contain two. Second, in certain server implementations, it is possible that the concatenation of multiple field instances (of the same field class or not) are stored in one buffer. That is, if the server has an overflow vulnerability related to this buffer, it is the concatenation of several field instances that can overflow the buffer. For example, imagine a DNS server receives a DNS PDU and stores the entire PDU in a vulnerable buffer. What overflows the buffer is the concatenation of all the field instances. These two observations have been further validated on other protocols such as SNMP and WINRPC.

With these considerations, we design a hierarchical model to describe the possible field classes in a PDU. As Figure 5 shows, we denote the QUESTION section as a new field O, a concatenation of all the instances of field A and B[2], $O = (AB)^*$. In short, we include all possible variable-length fields that potentially correspond to vulnerable buffers. We build such a hierarchy for every flow.

In the rest of the paper, we refer to variable-length fields simply as fields for the sake of brevity. Suppose there is a total of K classes of fields in the hierarchy constructed for a certain

[2]we denote the variable-length field QNAME as A, and the concatenation of fixed-length field QTYPE and QCLASS as B.

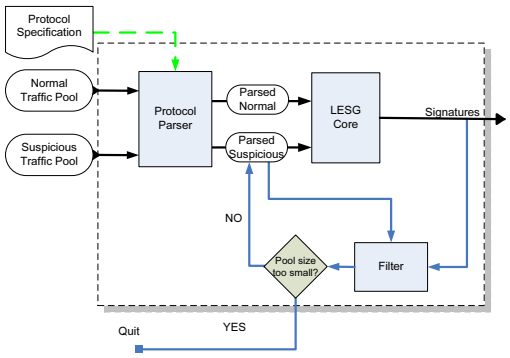

Fig. 2. LESG signature generator

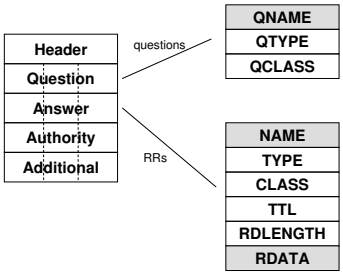

Fig. 3. Illustration of DNS PDU

Fig. 4. Abstraction of DNS PDU

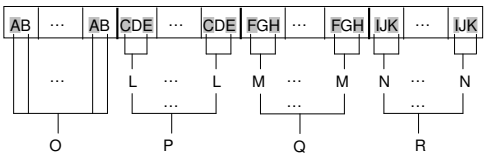

Fig. 5. Hierarchical Structure of DNS PDU

protocol. We use an index set $E = \{1, 2, \ldots, K\}$ to denote these K fields. Let $x_k, k = 1, 2, \ldots, K$, be the maximum among the lengths of potentially multiple instances of field k, then a vector $X = (x_1, x_2, \ldots, x_K)$ is generated to represent the field lengths for each field in a session (flow).

B. Length-based Signature Definition

Based on the length vector representation of a session, we formally define the concept of a *length-based signature* in this section. A signature is a pair $S_j = (f_j, l_j)$, where $f_j \in E$, f_j is the signature field ID, and l_j is the corresponding signature length for field f_j.

When using the signature to detect the worms, the matching process is as follows. For a flow $X = (x_1, x_2, \ldots, x_K)$, we compare x_{f_j} and l_j. If $x_{f_j} > l_j$, then the flow X is labelled as a worm flow; otherwise it is labelled as a normal one. More than one signature corresponding to different fields can possibly be generated for a given protocol, resulting in a *signature set* $S = \{S_1, S_2, \ldots, S_J\}$. A flow, which may contain one or more PDUs, will be labelled as a worm if it is matched by at least one signature in the set.

The length-based signatures are designed for buffer overflow worms. The signature field should be exactly mapped to a vulnerable buffer. In this case, the field of this instance must be longer than the buffer to overflow it, while normal instances must be shorter than the buffer. Note that different servers may implement different buffer lengths if the maximal length is not specified in the RFC. Here we focus on popular implementations because the spread speed and scope of worms will be significantly limited if they only target unpopular implementations. We define the minimum buffer length of popular implementations as *the ground truth signature*, denoted as $B = (f_B, L_B)$ where L_B is the vulnerable buffer length. Even with multiple different implementations, for the field related to the vulnerable buffer, the distributions of normal flows and worm flows should be well apart. That is, the lengths of normal flows should be less than L_B because for a popular server implementation (*e.g.*, FTP), there are often various client programs communicating with it without knowing its buffer length. So L_B should be large enough for most of the normal flows. On the other hand, those of worm flows should obviously be larger than L_B.

As elaborated below, our algorithm will not output any signatures for non-buffer-overflow worms because our algorithm ensures that all generated signatures have low false positives.

C. Length-Based Signature Generation Problem Formulation

If the flow classifier is perfect, all the flows in the suspicious pool are worm samples. If the worm is a buffer overflow worm, finding a length-based signature amounts to simply finding the best field and the field length with minimal false negatives and minimal false positives. However, in practice, flow classifiers at the network level are not perfect and always have some false positives, and therefore, the suspicious pool may have some normal flows. Finding signatures from a noisy suspicious pool makes the problem NP-Hard (Theorem 1). On the other hand, due to the large volume of traffic on the Internet, we assume the noise (worm flows) in the normal pool is either zero or very limited, and thus it is negligible.

After filtering existing known worms, there can be multiple worms of a given protocol in the suspicious pool, though the most common case is a single worm having its outbreak underway in the newly generated suspicious pool. The output of the signature generation is a signature set $S = \{S_1, S_2, \ldots, S_J\}$. A flow matched by any signature in this set will be labelled as a worm flow.

In Table II, we define most of the notations used in the problem formulation and theorems.

Problem 1 (Noisy Length-Based Signature Generation (NLBSG))**.**
INPUT*: Suspicious traffic pool* $\mathcal{M} = \{M_1, M_2, \ldots\}$ *and normal traffic pool* $\mathcal{N} = \{N_1, N_2, \ldots\}$*; value* $\gamma < 1$.
OUTPUT*: A set of length-based signatures* $\mathcal{S} = \{(f_1, l_1), \ldots, (f_J, l_J)\}$ *such that* $\mathrm{FP}_{\mathcal{S}}$ *is minimized subject to* $\mathrm{COV}_{\mathcal{S}} \geq 1 - \gamma$.

Theorem 1. *NLBSG is NP-Hard*

Proof Sketch: The proof is by reduction from Minimum k Union, which is equivalent to Maximum k-Intersection [37]. ■

V. Signature Generation Algorithm

Although the problem NLBSG is NP-Hard in general, for buffer overflow worms, the algorithms we proposed are fast and have fair accuracy even in the worst case scenarios. We formally proved the theoretical false positive and false negative bounds with or without adversaries to inject intentionally crafted noise. To the best of our knowledge, we are the *first* network based signature generation approach that has the accuracy bound even with adversaries' injected noise.

The protocol parsing step generates (field id, length) pairs for all flows in the normal traffic pool and suspicious traffic pool respectively. Based on that, we design a three-step algorithm to generate length-based signatures.

Step 1: *Field Filtering* Select possible signature field candidates.

Step 2: *Signature Length Optimization* Optimize the signature lengths for each field.

Step 3: *Signature Pruning* Find the optimal subset of candidate signatures with low false positives and false negatives.

A. Field Filtering

In this step of the algorithm, we make the first selection on the fields that could possibly be signature candidates. The goal is to limit the searching space. Two parameters are set

$\mathcal{M}$: suspicious traffic pool	$\mathcal{N}$: normal traffic pool
$\lvert\mathcal{M}\rvert$: number of suspicious flows in $\mathcal{M}$	$\lvert\mathcal{N}\rvert$: number of noise flows in $\mathcal{N}$
$\mathcal{M}^1$: set of true worm flows in $\mathcal{M}$	$\mathcal{M}^2$: set of noise flows in $\mathcal{M}$
α : coverage of true worms	K : number of variable length fields
$\mathcal{M}_\mathcal{S}$: set of suspicious flows covered by signature set $\mathcal{S}$	$\mathcal{N}_\mathcal{S}$: set of normal flows covered by signature set $\mathcal{S}$
$\mathrm{COV}_\mathcal{S}$: $\frac{\lvert\mathcal{M}_\mathcal{S}\rvert}{\lvert\mathcal{M}\rvert}$ for a signature set $\mathcal{S}$	$\mathrm{FP}_\mathcal{S}$: $\frac{\lvert\mathcal{N}_\mathcal{S}\rvert}{\lvert\mathcal{N}\rvert}$ for a signature set $\mathcal{S}$
COV_0 : minimum coverage requirement for a signature candidate	FP_0 : maximum false positive ratio for a signature candidate
γ' : minimum coverage increase requirement for a signature to be outputted in the first loop of the Step 3 algorithm	γ : minimum coverage increase requirement for a signature to be outputted in the second loop of the Step 3 algorithm

TABLE II
TABLE OF NOTATIONS

Algorithm **Step 1** *Field filtering* ($\mathcal{M}$,$\mathcal{N}$)

$S \leftarrow \emptyset$;
for field $f_j = 1$ to K
  find l_j such that $\frac{|\mathcal{N}_{l_j}|}{|\mathcal{N}|} \leq \mathrm{FP}_0 < \frac{|\mathcal{N}_{l_j-1}|}{|\mathcal{N}|}$;
  if $\frac{|\mathcal{M}_{l_j}|}{|\mathcal{M}|} \geq \mathrm{COV}_0$
    $S \leftarrow S \cup \{(f_j, l_j)\}$;
  end
end
Output S ;

as the input: FP_0 and COV_0, which indicates the most basic requirement on the false positives and detection coverage. For example, in our experiments, we choose $\mathrm{FP}_0 = 0.1\%$ and $\mathrm{COV}_0 = 1\%$.

In the algorithm below, $\mathcal{N}_{l_j}$ and $\mathcal{M}_{l_j}$ denote the flows detected by signature (f_j, l_j) in pools $\mathcal{N}$ and $\mathcal{M}$ respectively.

We set a low FP_0 as the basic low-false-positive requirement. A conservatively small value is chosen for COV_0 initially because attackers may inject noise into the suspicious pool. We will further optimize the values in the subsequent steps.

We process each field class separately. For every field class, the algorithm takes $O(|\mathcal{N}| \log |\mathcal{N}|)$ time to find the signature length, and then takes $O(|\mathcal{M}|)$ time to calculate the detection coverage on $\mathcal{M}$. Therefore, the total running time is $O(K|\mathcal{N}| \log |\mathcal{N}| + K|\mathcal{M}|)$. Since $|\mathcal{M}|$ is usually far smaller than $|\mathcal{N}|$, the overall time cost is $O(K|\mathcal{N}| \log |\mathcal{N}|)$.

This step makes use of the fact that, for buffer overflow worms, the true worm samples should have longer lengths on the vulnerable fields than the normal flows and the noise in the suspicious pool that is not injected by attackers should have a similar length distribution to traffic in $\mathcal{N}$. If the coverage α of true worm samples in the suspicious pool $\mathcal{M}$ is more than COV_0, the vulnerable field length with small false positive ratio FP_0 should have a larger coverage than COV_0 in the suspicious pool. The COV_0 and FP_0 are the very conservative estimates of the coverage and the false positive of the worm.

B. Signature Length Optimization

The first step selected candidate signatures to meet the most basic requirements. In the second step, we try to optimize the length value of each candidate signature to find the best tradeoff between the coverage and false positives. If the length signature selected is too long, there will be less coverage of malicious worm flows. On the other hand, if the length selected is too short, there will be a lot of false positives. The first step is a very conservative estimate of coverage. Sometimes a length does improve a lot on the coverage of the suspicious pool but also increases false positives. We need to have a method to compare different lengths to determine which one is a "better" signature. For the sake of brevity, let FP_{l_j} denote the false positive ratio of signature (f_j, l_j) and let COV_{l_j} denote its coverage on $\mathcal{M}$. This step aims to maximize $Score(\mathrm{COV}_{l_j}, \mathrm{FP}_{l_j})$ for each field f_j. We used the notion score function, which is proposed in [14], to determine the best tradeoff between the false positive and coverage. For example, we need to make a choice between $\mathrm{COV} = 70\%$, $\mathrm{FP} = 0.9\%$ and $\mathrm{COV} = 68\%$, $\mathrm{FP} = 0.2\%$.

In the Step 2 algorithm, $\mathcal{M} = \{X_1, X_2, \ldots, X_{|\mathcal{M}|}\}$, where $X_m = (x_m^1, x_m^2, \ldots, x_m^K),\ m = 1, 2, \ldots, |\mathcal{M}|$ is the length of each field in a flow m. We define $\mathcal{M}^k = \{x_1^k, x_2^k, \ldots, x_{|\mathcal{M}|}^k\}$. Signature set S generated in Step 1 is the input of this step.

With the sorted lengths as input, for candidate signature fields, each length above the candidate length selected at Step 1 will be tested for its goodness according to the score function, and the best one with the maximum score will be selected. The first loop picks a longer length value with the best score. Then in the second loop, we further optimize it by finding a smaller length with the same score.

In $\mathcal{M}^{f_j} = \{x_1^{f_j}, x_2^{f_j}, \ldots, x_{|\mathcal{M}|}^{f_j}\}$, if $x_m^{f_j}$ is in the ascending order, it is easy to know that between any two consecutive elements, namely $x_{m-1}^{f_j}$ and $x_m^{f_j}$, the score is monotonically non-decreasing in l_j. Thus we only need to search among all the $x_m^{f_j} - 1,\ m = m_0, \ldots, |\mathcal{M}|$ for the best score, *i.e.*the total number we need to try is at most $|\mathcal{M}|$.

The rationale for the second loop is as follow. The signature length too close to the edge of the lengths of worm flows is not a good choice, especially when the length distributions of normal field instances and of malicious field instances are well separated. So in the second loop of the algorithm Step 2, l_j decreases until the score changes (decreases actually) or l_j reaches the median of two consecutive elements in $\mathcal{M}^{f_j}$. In the Section VI-B, we will analyze the advantages of this choice.

To sort each $\mathcal{M}^{f_j}$ needs $O(|\mathcal{M}| \log |\mathcal{M}|)$. To search the best score from m_0 to $|\mathcal{M}|$ needs at most $O(|\mathcal{M}| \log |\mathcal{N}|)$. In the worst case, to find the best signature in the gap between $x_{m-1}^{f_j}$ and $x_m^{f_j}$, half of the gap needs to be searched. Since $|S| \leq K$, the total running time is $O(K(|\mathcal{M}| \log |\mathcal{M}| + |\mathcal{M}| \log |\mathcal{N}| + G))$. G is the possible maximum gap among all the fields.

C. Signature Pruning

Still we have a set of candidate field and length signatures. Too many length signatures will cause unnecessary false positives because we try to match any of the length signatures in the detection phase. Therefore, in this final step, we will find an optimal small subset of signature candidates to be the final signature set. Usually, the more signatures we use, the more false positives there are but with better coverage.

As proved in Section IV-C, to select the optimal small set of signatures in general is NP-Hard. The algorithm proposed here is not to search for the global optimum but to find a good solution with bounded false positives and negatives. In the Step 3 algorithm, γ' and γ are parameters and $\gamma' < \gamma$. The Step 3 algorithm has two stages. The first one is the opportunistic stage. We opportunistically find the signatures which can improve at least γ' percent of the initial suspicious

Algorithm **Step 2** *Signature Length Optimization* (S,$\mathcal{M}$,$\mathcal{N}$,$Score(\cdot,\cdot)$)

```
for signature (f_j, l_j) ∈ S
    sort M^{f_j} in ascending order;
    find m_0 such that x^{f_j}_{m_0-1} < l_j < x^{f_j}_{m_0};
    max_score ← 0;
    for m' = m_0 to |M|
      l'_j ← x^{f_j}_{m'} − 1;
      if (max_score < Score(COV_{l'_j}, FP_{l'_j}))
        max_score ← Score(COV_{l'_j}, FP_{l'_j});
        l_j ← l'_j;
        m ← m';
      end
    end
    while (l_j > (x^{f_j}_{m-1} + x^{f_j}_m)/2)
      if (Score(COV_{l_j}, FP_{l_j}) == Score(COV_{l_j-1}, FP_{l_j-1}))
        l_j ← l_j − 1;
      else
        update S with l_j; break;
      end
    end
end
Output S ;
```

Algorithm **Step 3** *Signature Pruning* (S,$\mathcal{M}$,$\mathcal{N}$)

```
m ← |M| ; Ω ← ∅ ;
S_1 ← {e|e ∈ S; FP_e = 0} ; S_2 ← {e|e ∈ S; FP_e > 0} ;
LOOP1:
while (S_1 ≠ ∅)
    Find s ∈ S_1 such that |M_s|/m is the maximum one in S_1 ;
    If (|M_s|/m ≥ γ')
      Ω ← Ω ∪ {s} ;S_1 ← S_1 − {s} ;
      Remove all the samples which match s in M ;
    else
      Break ;
    end
end
LOOP2:
while (S_2 ≠ ∅)
    Find s ∈ S_2 such that |M_s|/m is the maximum one in S_2 ;
    If (|M_s|/m ≥ γ)
      Ω ← Ω ∪ {s} ; S_2 ← S_2 − {s} ;
      Remove all the samples which match s in M ;
    else
      Break ;
    end
end
Output Ω ;
```

pool coverage than the existing signature set without generating any false positives. Usually, γ' is small. If the best signatures we can find for each worm have no false positive, the opportunistic stage can help improve the true positives even when adversaries are present. Then, we use a similar process to find other signatures with a marginal improvement requirement γ.

Calculating $|\mathcal{M}_s|$ takes $O(\log|\mathcal{M}|)$, and thus finding the signature with maximum coverage takes $O(K\log|\mathcal{M}|)$. Furthermore, removing samples matched by signature s takes $O(|\mathcal{M}|)$. Therefore, the final running time for the Step 3 algorithm can be bounded by $O(K(K\log|\mathcal{M}|+|\mathcal{M}|))$.

With our three-step algorithm, we guarantee low false positives and false negatives on the generated signatures for buffer-overflow worms. For non-buffer-overflow worms, the algorithm will output an empty set, having found no signatures to meet the minimal requirements on false positives and false negatives.

VI. Attack Resilience Analysis

In this section, we analyze and prove attack resilience of our algorithm, *i.e.*, the quality of signatures generated (evaluated by false negatives and false positives) when attackers launch attacks to try to confuse and evade the LESG system. In particular, attackers may deliberately inject some noise into the suspicious pool to fool LESG.

A. Worst Case Performance Bounds

Note that the noisy length signature generation problem (NLBSG) is a NP-Hard problem and even *the global optimum solution* due to the limited input size can be different from the ground truth signature L_B as defined in Section IV-B. The signatures we generated are *approximated signatures*. In the Step 1 and Step 2 algorithms, we always select the field f_B in the signature candidate set if the worm coverage is larger than COV_0. However, in our algorithm, instead of getting the optimal length L_B, we might get L'_B. We denote the signature $B' = (f_B, L'_B)$. In the Step 2 algorithm, we tend to choose a more conservative signature than the ground truth signature B, *i.e.*, $L'_B \le L_B$. Therefore $\mathrm{FN}_{\{B'\}} = 0$ and $\mathrm{FP}_{\{B'\}} \le \mathrm{FP}_0$.

For most cases, the distributions of normal flows and worm flows are well apart, and there is a noticeable gap between the two distributions. In these cases we will get $\mathrm{FP}_{\{B'\}} = 0$, which has the same power as the ground truth signature. Without adversaries, our algorithm will output the signature B', which we call the *best approximated signature* because it has the tightest bound to the corresponding ground truth signature when compared with signatures generated with adversaries. Then with different adversary models and depending on whether the normal and worm flow length distributions have a noticeable gap, our algorithm will output different approximated signatures with different attack resilience bounds.

Let $\mathcal{M}^1$ be the set of true worm flows in $\mathcal{M}$ and let $\mathcal{M}^2 = \mathcal{M} - \mathcal{M}^1$, which is all the noise in $\mathcal{M}$. Let the fraction of worm flows in $\mathcal{M}$ be α, *i.e.* $\frac{|\mathcal{M}^1|}{|\mathcal{M}|} = \alpha$. Due to the interest of space, we ignore the proofs here. *Please refer to our technique report [38] for all the detailed proofs*.

1) Performance Bounds with Crafted Noises: In Theorems 2 and 3, we prove the worse case performance bounds of our system under the deliberate noise injection attacks, *i.e.*, with crafted noises. This is the worst case. The attackers not only fully craft the worms but also inject the crafted noises. The difference between Theorem 2 and Theorem 3 is that Theorem 2 assumes the length distributions of normal flows and worm flows are well apart which is the most common case in reality. Theorem 3 considers even more general cases, which the length distributions of normal flows and worm flows might not be well apart.

Theorem 2. *If the best approximated signature has no false negative and no false positive, the three step algorithm outputs a signature set* Ω *such that* $\mathrm{FN}_\Omega < \frac{\gamma'}{\alpha}$ *and* $\mathrm{FP}_\Omega \le \mathrm{FP}_0 \cdot \lfloor \frac{1-\alpha}{\gamma} \rfloor$.

Theorem 3. *If the best approximated signature has no false negative and the false positive ratio is bounded by* FP_0*, the three-step algorithm outputs a signature set* Ω *such that* $\mathrm{FN}_\Omega < \frac{\gamma}{\alpha}$ *and* $\mathrm{FP}_\Omega = \mathrm{FP}_0 \cdot (\lfloor \frac{1-\alpha}{\gamma} \rfloor + 1)$.

These bounds are still tight, as shown in the example of deliberated noise injection attacks in Section VI-A3.

2) Performance Bounds without Crafted Noises: Since injected crafted noises will slow down the worm propagation, the worm authors might not want to do that. For example, when the noise ratio is 90% (*i.e.*, 90% of traffic from a worm is crafted noises), the worm will propagate at least 10 times slower than before based on the RCS worm model [12]. For example, the Code Red II may take 140 hours (six days) to compromise all vulnerable machines instead of 14 hours.

Without crafted noises, *i.e.*, the noises are from normal traffic, we are able to prove even tighter performance bounds for our system. Here, the Theorem 4 below assumes the length distributions of normal flows and worm flows are well apart while the Theorem 5 removes this assumption. Both theorems assume the noises in the suspicious pool are randomly sampled from the normal traffic.

Theorem 4. *If the noise in the suspicious pool is normal traffic and not maliciously injected and the best approximated signature has no false positives and no false negatives, then the three-step algorithm outputs a signature set* Ω *such that* $\mathrm{FN}_\Omega = 0$ *and* $\mathrm{FP}_\Omega = 0$*; in other words, with no false negative and false positive.*

In this case, the outputted signature set Ω contains the best approximated signature.

Theorem 5. *If the noise in the suspicious pool is normal traffic and not maliciously injected and the best approximated signature has no false negative and a false positive ratio bounded by* FP_0*, then the three-step algorithm outputs a signature set* Ω *such that* $\mathrm{FN}_\Omega \leq \mathrm{FP}_0 \cdot \frac{1-\alpha}{\alpha}$ *and* $\mathrm{FP}_\Omega \leq \mathrm{FP}_0$.

The evaluation results in Section VII-B are consistent with the theorem and are often better than the bounds proved in the theorems.

3) Discussions: In this section, we discuss some issues related to the attack resilience theorems.

Multiple worms. For single worm cases, the theorems can be directly applied. In the case that multiple worms are in the suspicious pool, for each worm we treat the other worms as noises, and thus we have the same bound.

Parameter FP_0. From the theorems above, we can tell that FP_0 plays an important role on the bound. We have the following observations for its value. Usually given a standard protocol, a popular implementation of peer/server should be able to interoperate with various different implementations of peer/clients. Thus even for a server implementation with a buffer overflow vulnerability, in most cases the normal traffic should not trigger the buffer overflow. Here we assume FP_0 is no larger than 0.1%, and we conservatively set it to be 0.1%, *i.e.*, the server should be able to handle 1000 normal requests without crashing (buffer overflow triggered). This is equivalent to a server handling six requests per hour and not crashing for a week. We believe this is reasonable for most popular implementations of a protocol.

Assumptions for theorems on attack resilience. There are two general assumptions for all the theorems above. First, there is little or no overlap for the input length of vulnerable fields between the normal traffic and the worms. This is discussed in Section IV-B and also validated in our experiments. Secondly, the attacker cannot change the field length distribution of normal traffic, which is also generally true. Compared with the recent Hamsa system [14], we have fewer assumptions and allow crafted noises.

B. Resilience Against the Evading Attacks

In this section, we discuss the resilience of our schemes against several recently proposed attacks [19]–[22].

Deliberate noise injection attack In [19], deliberate noise injection is proposed to mislead the worm signature generator. Most other existing signature generators suffer under this attack. However, even with this attack, our approach can perform reasonably well, especially in the case when the best approximated signature with zero false positive exists. For example if $\gamma' = 1\%$ and $\gamma = 5\%$, even with 90% crafted noise, in most cases the false negative rate can be bound as 10% and the false positive rate as 1.8%. Note, this is the worst case theoretical bound, in practice it is very hard to approach. To the best of our knowledge, we are the *first* network-based approach that can achieve this performance.

There are several different attacks proposed in Paragraph [20]. Among them, the *suspicious pool poisoning attack* is similar to the deliberate noise injection attack. Next, we discuss other attacks.

Randomized red herring attack or coincidental attack is to inject unnecessary tokens into the content based approaches with a probability model so that these tokens are highly likely to be included in the signatures, producing more false negatives. A similar attack can be proposed for our approach, but it requires the attackers to use the "don't care" fields, which are the fields that can be manipulated without influencing the worm execution. Unlike the content-based signature generation approaches with which attackers can inject as many tokens as they want, there may be zero or only a small number of such "don't care" fields in a protocol, so the attack may not be applicable. Moreover, we use a signature set, so when any signature in this set matches the sample, we label the sample as a worm. This is more resilient than using the whole set as a signature.

Dropped red herring attack includes some tokens in the beginning of the worm spread and drops those tokens in later propagation of the worm. Again, a similar attack can be proposed for our approach, but there are several problems as well as countermeasures for such attacks. Firstly, this attack also requires "don't care" fields. Secondly, we can potentially still detect the worm with any disjunction in the signature set instead of using the conjunction. Thirdly, this attack is hard to implement because it requires the worm to dynamically change itself with synchronized actions. Fourthly, there are some dynamic update problems for signature change and signature regeneration. Since our signature generation is fast, it can alleviate the damage by this attack.

Moreover, there is another similar attack which can be designed specially for length-based signatures. We call it *length dropping attack*. Since the attackers have to inject an input longer than the buffer length L_B, they can inject a long input L at the beginning and gradually decrease the length by ΔL in each run of infection until L_B. However, if there is a gap between L and L_B, in our design we choose the signature length to be $l_j = \frac{x_{m-1}^{f_j} + x_m^{f_j}}{2}$ so that the $x_{m-1}^{f_j}$ is comparable to L_B and the $x_m^{f_j}$ is comparable to L. In other words, we will choose the median of L and L_B. Therefore, even when this attack is launched, we only need to regenerate the length signature $O(\log(L_1 - L_B))$ times where L_1 is the initial length that the attackers use.

Innocuous pool poisoning is the poisoning of the normal traffic pool. However, in general, this is very hard. First, the amount of normal traffic is huge, even to poison 1% is hard. Second, using the random selection policy of normal traffic [14], it is very hard for attackers to poison the traffic in the right time to have an effective evasion during the worm breakout.

In [21], Simon *et al.*propose two types of *allergy attacks*.

The type I attack makes the IDS generate signatures which can deny current normal traffic. The type II attack makes the IDS generate signatures which can deny future normal traffic. The type I allergy attack does not work for our approach because we check the false positive against the normal traffic. The type II attack may work in theory, but in practice it is very hard to happen. The contents of future traffic may change a lot more than that of the current normal traffic, but the length profile of fields in the protocol will still remain stable. Therefore, it is hard to find such a case. Even if there is such a case, it is very hard for the attack to predict.

The *blending attacks* [22] cannot work for our approach because the worm has to use a longer-than-normal input for the vulnerable field and they cannot mimic the normal traffic.

VII. Evaluation

We implemented the protocol parsing using Perl scripts with BINPAC and Bro, as mentioned in Section III-A, and implemented the LESG signature generator in *MATLAB*.

A. Methodology

We constructed the traffic of eight worms based on real-world exploits and collected more than 27GB of Internet traffic plus 123GB of email SPAM. To test LESG's effectiveness, we used completely different datasets for LESG signature generation (i.e. training dataset) and for signature quality testing (i.e. evaluation dataset). For the training dataset, we used a portion of the worm traffic plus some samples from the normal traffic (as noise) to construct the suspicious pool, and we used a portion of the normal traffic as the normal pool. For the evaluation dataset, we used the remaining normal traffic to test the false positives and worm traffic to test false negatives.

1) Polymorphic Worm Workload: To evaluate our $LESG$ system, we created eight polymorphic worms based on real-world vulnerabilities and exploits from `securityfocus.com`, as shown in Table III, by modifying the real exploits to make them polymorphic. The eight worms use six different protocols, DNS, SNMPv1, SNMPv1$_{trap}$, FTP, SMTP and HTTP. Since the original exploit code is not polymorphic and the field lengths are fixed, we modified them as follows: for the exploit unrelated fields, *i.e.*"don't care" fields, we randomly chose the lengths with the same distribution as those in normal traffic; for the signature related fields, the lengths in the original exploit codes are longer than the buffer lengths in most cases, so we used these values as the upper bound in the worms and used the hidden buffer length or a larger value that we believed was necessary to exploit the vulnerability as the lower bound (specified by the row "ground truth" in Table III); moreover, for some exploits that have rigid exploit conditions, we kept the fixed length. In the Table III, the row titled "signature related field length" specifies whether the overflowing field length is fixed or not. For the vulnerability for which we cannot find the ground truth by searching literature, we indicate such as "unknown".

The detailed descriptions of the worms we created are as follows.

DNS worm. It's a variant of the lion worm that attacks a vulnerability of BIND 8, the most popular DNS server. The exploit code constructs a UDP DNS message with a QUESTION section whose length is 493 bytes and difficult to make variable.

SNMP worm. It attacks a vulnerability in the NAI sniffer agent. The vulnerable buffer is 256 bytes long and stores the data transferred in the field ObjectSyntax.

SNMP Trap worm. The worm targets Mnet Soft Factory NodeManager Professional. When it processes SNMP Trap messages, it allocates a buffer of 512 bytes to store the data transferred in the field ObjectSyntax.

FTP worm I. It exploits a vulnerability in the Sami FTP Server. The content of the USER command must be longer than 228 bytes to overflow the buffer storing it.

FTP worm II. It targets a popular desktop FTP server, Serv-U. The content of the SITE CHMOD command plus a path name is stored in a buffer which is 419 bytes long.

FTP worm III. It targets the BulletProof FTP Client. The content of the FTP reply code 220 must be longer than 4104 bytes.

SMTP worm. This vulnerability resides in the RCPT TO command of the Ipswitch IMail Server.

HTTP worm. It exploits the IIS vulnerability also attacked by the famous worm Codered. The difference is that we varied the length of our created worm, while Codered has a fixed length.

2) Normal Traffic Data: The traffic traces were collected at the two gigabit links and another hundred-megabit link of the gateway routers at Tsinghua University campus network in China on June 21 - 30, 2006. All the traffic at Tsinghua University to/from DNS, SNMPv1 Trap, SNMPv1, HTTP and FTP control channel was collected without using any form of sampling. We used another 123GB SPAM dataset from some open relay servers at a research organization in the U.S. for the SMTP. The datasets are summarized in Table IV. Since a SNMPv1 Trap message is sent to port 162 and its format is different from other types of messages, we treat SNMPv1 Trap as a protocol separate from SNMPv1 on port 161. Also note that for evaluation purpose, in our prototype system, we only parsed the GET request for HTTP, which has the same effect as a complete parsing because the worm is only related to the GET request. The traces are checked by the Bro IDS system to make sure that the traces are normal traffic.

3) Experiment Settings: In the Step 1 algorithm, we set $\mathrm{FP}_0 = 0.1\%$ and $\mathrm{COV}_0 = 1\%$. The score function in Step 2 is $Score(\mathrm{COV}, \mathrm{FP}) = (1/log\mathrm{FP} + 1) * \mathrm{COV}$, which works well in practice. The basic requirement of a score function is that the score should be monotonically increasing with COV and decreasing with FP. This function has another merit in that a large FP (eg. $\mathrm{FP} \in [10^{-3}, 10^{-2}]$) affects the score greater than a much smaller FP (eg. $\mathrm{FP} \in [10^{-5}, 10^{-4}]$) does. In Step 3, we choose $\gamma' = 1\%$ and $\gamma = 5\%$, indicating that we focus on the worms that cover more than 1% of the suspicious pool.

B. Signature Generation for A Single Worm with Noise

We evaluated the accuracy of LESG with the presence of noise. The noise is the flows randomly sampled from normal traffic, and mixed with worm samples to compose the suspicious pool. We chose DNS, SNMP, SNMP$_{trap}$, SMTP and HTTP protocols to demonstrate the cases of a single worm with noise. For HTTP we also tested our algorithm against the Codered worm.

For each protocol, we tested the suspicious pool size of 50, 100, 200 and 500, and at each size we changed the noise ratio from 0 to 80%, increasing 10% in each test. After signature generation, we matched the signatures against another 2000 samples of worms and an evaluation set of normal traffic to

Protocol	DNS	SNMP	$SNMP_{trap}$	FTP_1	FTP_2	FTP_3	SMTP	HTTP
Bugtraq ID	2302	1901	12283	16370	9675	20497	19885	2880
ground truth (fieldID,BufLen)	(2,493)	(6,256)	(7,512)	(1, 228)	(11,419)	(33, 4104)	(3, unknown)	(6, 240)
signature related field length	fixed	variable	variable	variable	variable	variable	variable	variable

TABLE III
THE SUMMARY OF WORMS

Number of Fields	Normal pool			Evaluation dataset		
	Bytes	Flows	Hours	Bytes	Flows	Hours
DNS: 14	120M	320K	21	960M	4.4M	120
SNMP: 10	12M	13K	20	282M	77K	120
$SNMP_t$: 11	21M	16K	72	67M	54K	218
FTP: 60	2.7G	66K	14	10G	373K	37
SMTP: 12	840M	210K	24	122G	31M	744
HTTP: 7	2G	77K	7	11G	360K	40

TABLE IV
DATASET SUMMARY FOR EVALUATION

test the sensitivity and accuracy.

Table V shows the range of the signatures we generated and their accuracy. Tr. FN/FP denotes the training false negatives and false positives in the training data. Ev. FN/FP denotes the evaluation false negatives and false positives in the evaluation data set. Under all the pool sizes and noise ratios, the same signature fields are generated. Because the size of suspicious pool is limited, the signature length varies in different tests. We checked these signatures against the evaluation datasets, and they all have excellent false negative and false positive ratio. It may be noticed that generated signature lengths are smaller than the true buffer length, because the length in normal flows are usually much smaller than the buffer length, which is reasonable since the buffer length is designed to be longer than the longest possible normal requests.

Worm	Signatures (ID,length)	Tr. FN	Tr. FP	Ev. FN	Ev. FP
DNS	(2, 284∼296)	0	0	0	0
SNMP	(6, 133∼238)	0	0	0	0
$SNMP_t$	(7, 304∼314)	0	0	0	0
SMTP	(3, 109∼112)	0	0	0	10^{-5}
FTP	(1, 128∼169) (11, 262∼300) (33, 2109∼2121)	0	0	0	0
HTTP	(6, 239∼240)	0	0	0∼1%	10^{-4}
CodeRed	(6, 339)	0	0	0	10^{-5}

TABLE V
SIGNATURES AND ACCURACY UNDER DIFFERENT POOL SIZE AND NOISE

C. Signature Generation for Multiple Worms with Noise

We also evaluated the case of multiple worms with noise using the FTP protocol. We have three FTP worms in total. We tested the suspicious pool sizes of 50, 100, 200 and 500, and at each size we changed the noise ratio from 0 to 70%, increasing 10% in each test.The result is also shown in Table V.

D. Evaluation of Different Stages of the LESG Algorithm

The LESG algorithm has three steps, and we evaluated the effectiveness of each step. Table VI illustrates the results of each step for the DNS worm with a suspicious pool of size 100 and a noise ratio 50%. Table VI shows that the false positive rate is largely decreased by refining each signature length in Step 2. And comparing with the ground truth shown in Table III, we can see that in Step 3, the best and most accurate signature is selected, further decreasing the false positives.

E. Pool Size Requirement

We tested the accuracy of our algorithm when only a small suspicious pool is available. We chose suspicious pools of size 10 with a noise ratio of 20% and of size 20 with a noise ratio of 50%. All the tests generated signatures within the range presented in Table V.

	Signature	Tr. FN	Tr. FP
Step 1	{(1,62), (2,66), (3,2), (4,15), (5,28), (6,47), (10,99),(11,2)}	0	0.32%
Step 2	{(1,68), (2,296), (3,21), (4, 99), (5,333), (6,543), (10,111), (11,2)}	0	0.15%
Step 3	{(2, 296)}	0	0

TABLE VI
RESULT OF EACH STEP FOR THE DNS WORM

We did similar tests for the DNS worm using normal pool sizes of 5K, 10K, 20K, and 50K, and we found that our approach is not sensitive to the size of the normal pool either.

F. Speed and Memory Consumption Results

	Normal pool (Bytes/Flows) (Bytes/Flows)	Protocol parsing (secs)	Signature generation (in different pool size) (secs)			
			50	100	200	500
DNS	120M/320K	58	2.1	3.6	9.4	18
SNMP	12M/13K	8	0.08	0.09	0.15	0.32
$SNMP_t$	21M/16K	4	0.12	0.24	0.37	0.88
FTP	2.7G/66K	95	0.20	0.29	0.54	1.20
SMTP	836M/210K	50	0.47	1.30	1.84	3.36

TABLE VII
SPEED OF PROTOCOL PARSING AND SIGNATURE GENERATION

We evaluated the parsing speed by using Bro and BINPAC and the speed of our signature generation algorithm. Since HTTP was not completely parsed, we only provide the results of the other five protocols. Table VII shows that the speed of the signature generation algorithm is quite fast, though the speed is influenced by the sizes of the suspicious pool and the normal pool. The protocol parsing for the normal pool can be done offline. We can run the process every once in a while (e.g. several hours). These datasets were collected over a 20-hour+ period. For the suspicious pool, since it is much smaller than the normal pool, the protocol parsing can be done very quickly. Moreover, as mentioned in [39], the BINPAC compiler can be built with parallel hardware platforms, which makes it much faster.

Normal pool size		Suspicious pool size		
		100	200	500
DNS (14 fields)	50K	5.64MB	5.66MB	5.71MB
	100K	11.26MB	11.28MB	11.33MB
FTP (60 fields)	50K	8.43MB	8.45MB	8.53MB
	100K	16.83MB	16.85MB	16.93MB

TABLE VIII
MEMORY USAGE OF THE ALGORITHM

The memory usage of the signature generation algorithm implemented in *Matlab* was evaluated under different pool sizes, shown in Table VIII. The memory usage is proportional to the normal pool size and the number of fields.

VIII. DISCUSSIONS OF PRACTICAL ISSUES

a) Speed of Length Based Signature Matching: The operation of length-based signature matching has two steps: protocol parsing of packets and field length comparison with the signatures. The latter is trivial. The major overhead is for

protocol parsing. Currently, the Bro and BINPAC based parsing can achieve $50 \sim 200$ Mbps. As mentioned in [39], with parallel hardware platform support, BINPAC may achieve $1 \sim 10$ Gbps. On the commercial products side, Radware's security switch on an ASIC-based network processor can operate at 3 Gbps link with protocol parsing capability [40]. Therefore, with hardware support, the whole length-based signature matching can be done very fast, which is comparable to the current speed of pattern-based (string) signature matching techniques widely used in IDSs.

b) Relationship Between Fields and Vulnerable Buffers: The main assumption of length based signatures is that there is a direct mapping between variable length fields and vulnerable buffers. In addition to the vulnerabilities shown in the evaluation section, we further checked 11 more buffer overflow vulnerabilities from `securityfocus.com`. We found that the assumption holds for all cases except one. Therefore, we know in most situations that LESG should work. Next, we will first examine the normal cases and then check the special one.

In Section IV-A we show that the consecutive fields can be combined together to form a *compound field*. For the variable length fields which cannot be further decomposed, we call them *simple fields*. We found in 13 cases (out of the total of 18 cases that we examined) that the field mapped to the vulnerable buffer is a simple field while in 3 cases it is a compound field. There is one case we found in which two simple fields, which cannot be combined to form a compound field, are mapped to one vulnerable buffer. Therefore, either of the two fields can cause the buffer overflow to happen. In all these cases, we can get the accurate length-based signatures. However, we did find one case (again, 1 out of 18 cases) which does not have length-based signatures. It is a buffer overflow vulnerability present in versions of wu-ftpd 2.5 and below. The vulnerable buffer corresponds to the path of the directory, so if a very deep path is created by continuously making new directories recursively, the buffer will eventually be overflowed. From the protocol messages of the FTP, only a set of MKD (mkdir) commands can be seen, and the length of each directory could be normal. Therefore, no length-based signatures exist.

IX. Conclusions

In this paper, we proposed a novel network-based automatic worm signature generation method that generates length-based signatures for buffer overflow worms. Our approach has good attack resilience guarantees even under deliberate noise injection attacks. We further show that our approach is fast and accurate through experiments and evaluation based on real-world vulnerabilities and network traffic.

X. Acknowledgement

Support for this work was provided by the NSF grant CNS-0627751.

References

[1] Z. Liang and R. Sekar, "Automatic generation of buffer overflow attack signatures: An approach based on program behavior models," in *Proc. of Computer Security Applications Conference (ACSAC)*, 2005.

[2] The SANS Institute, "The Top 20 Most Critical Internet Security Vulnerabilities - PRESS UPDATE." http://www.sans.org/top20/2005/spring2006update.php.

[3] M. Roesch, "Snort: The lightweight network intrusion detection system," 2001, http://www.snort.org/.

[4] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, 1999.

[5] C. Kreibich and J. Crowcroft, "Honeycomb - creating intrusion detection signatures using honeypots," in *Proc. of the Workshop on Hot Topics in Networks (HotNets)*, 2003.

[6] S. Singh, C. Estan, *et al.*, "Automated worm fingerprinting," in *Proc. of USENIX OSDI*, 2004.

[7] H. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," in *Proc. of USENIX Security Symposium*, 2004.

[8] Z. Liang and R. Sekar, "Fast and automated generation of attack signatures: A basis for building self-protecting servers," in *Proc. of ACM CCS*, 2005.

[9] X. Wang *et al.*, "Packet vaccine: Black-box exploit detection and signature generation," in *Proc. of ACM CCS*, 2006.

[10] D. Brumley *et al.*, "Towards automatic generation of vulnerability-based signatures," in *Proc. of IEEE Security and Privacy Symposium*, 2006.

[11] D. Moore *et al.*, "The spread of the Sapphire/Slammer worm," http://www.caida.org, 2003.

[12] S. Staniford *et al.*, "How to own the Internet in your spare time," in *Proceedings of the 11th USENIX Security Symposium*, 2002.

[13] ——, "The top speed of flash worms," in *Proc. of ACM CCS WORM Workshop*, 2004.

[14] 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. of IEEE Security and Privacy Symposium*, 2006.

[15] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," in *Proc. of IEEE Security and Privacy Symposium*, 2005.

[16] Y. Tang and S. Chen, "Defending against internet worms: A signature-based approach," in *Proc. of IEEE Infocom*, 2003.

[17] J. Newsome and D. Song, "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software," in *Proc. of NDSS*, 2005.

[18] J. R. Crandall, Z. Su, and S. F. Wu, "On deriving unknown vulnerabilities from zeroday polymorphic and metamorphic worm exploits," in *Proc. of ACM CCS*, 2005.

[19] R. Perdisci *et al.*, "Misleading worm signature generators using deliberate noise injection," in *Proc. of IEEE Security and Privacy Symposium*, 2006.

[20] J. Newsome, B. Karp, and D. Song, "Paragraph: Thwarting signature learning by training maliciously," in *Proc. of RAID*, 2006.

[21] S. P. Chuang and A. K. Mok, "Allergy attack against automatic signature generation," in *Proc. of RAID*, 2006.

[22] P. Fogla *et al.*, "Polymorphic blending attacks," in *Proc. of USENIX Security Symposium*, 2006.

[23] V. Yegneswaran *et al.*, "An architecture for generating semantic-aware signatures," in *Proc. of USENIX Security Symposium*, 2005.

[24] C. Kruegel *et al.*, "Polymorphic worm detection using structural information of executables," in *Proc. of RAID*, 2005.

[25] Packeteer, "Solutions for Malicious Applications," http://www.packeteer.com/prod-sol/solutions/dos.cfm.

[26] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *Proc. of RAID*, 2004.

[27] K. Wang, G. Cretu, and S. J. Stolfo, "Anomalous payload-based worm detection and signature generation," in *Proc. of RAID*, 2005.

[28] R. Vargiya and P. Chan, "Boundary detection in tokenizing network application payload for anomaly detection," in *Proc. of ICDM Workshop on Data Mining for Computer Security (DMSEC)*, 2003.

[29] M. Cost *et al.*, "Vigilante: End-to-end containment of internet worms," in *Proc. of ACM Symposium on Operating System Principles (SOSP)*, 2005.

[30] F. Hsu and T. Chiueh, "Ctcp: A centralized TCP/IP architecture for networking security," in *Proc. of ACSAC*, 2004.

[31] X. Wang *et al.*, "Sigfree: A signature-free buffer overflow attack blocker," in *Proc. of USENIX Security Symposium*, 2006.

[32] V. Yegneswaran, P. Barford, and D. Plonka, "On the design and use of internet sinks for network abuse monitoring," in *Proc. of RAID*, 2004.

[33] M. Bailey *et al.*, "The internet motion sensor: A distributed blackhole monitoring system," in *Proc. of NDSS*, 2005.

[34] W. Cui, V. Paxson, and N. Weaver, "GQ: Realizing a system to catch worms in a quarter million places," ICSI, Tech. Rep. TR-06-004, 2006.

[35] Y. Gao, Z. Li, and Y. Chen, "A dos resilient flow-level intrusion detection approach for high-speed networks," in *Proc. of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2006.

[36] R. Pang *et al.*, "binpac: A yacc for writing application protocol parsers," in *Proc. of ACM/USENIX IMC*, 2006.

[37] S. A. Vinterbo, "Maximum k-intersection, edge labeled multigraph max capacity k-path, and max factor k-gcd are all NP-hard," Decision Systems Group,Harvard Medical School, Tech. Rep., 2002.

[38] Z. Li, L. Wang, Y. Chen, and Z. Fu, "Network-based and attack-resilient length signature generation for zero-day polymorphic worms," Northwetern University, Tech. Rep. NWU-EECS-07-02, 2007.

[39] V. Paxson *et al.*, "Rethinking hardware support for network analysis and intrusion prevention," in *Proc. of USENIX Hot Security*, 2006.

[40] Radware Inc., "Introducing 1000X Security Switching," http://www.radware.com/content/products/application_switches/ss/default%.asp.