# Thwarting Zero-Day Polymorphic Worms With Network-Level Length-Based Signature Generation

Lanjia Wang, Zhichun Li, Yan Chen, Zhi (Judy) Fu, and Xing Li

***Abstract*—It is crucial to detect zero-day polymorphic worms and to generate signatures at network gateways or honeynets so that we can prevent worms from propagating at their early phase. However, most existing network-based signatures are specific to exploit and can be easily evaded. In this paper, we propose generating vulnerability-driven signatures at 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 the worms exploiting buffer overflow vulnerabilities[1]. The signatures generated are intrinsic to buffer overflows, and are very difficult 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.**

***Index Terms*—length-based signature, polymorphic worm, worm signature generation, zero-day vulnerability.**

## I. Introduction

COMPUTER worms are serious threats to the Internet, causing billions of dollars in economic loss. Searching the network traffic for known patterns, or *signatures*, is an important way to defend against worms, implemented in intrusion detection systems (IDSes) [2], [3]. Since generating signatures manually is too slow to defend against zero-day self-propagating worms, approaches have been proposed to automate the process of worm signature generation [4]–[6]. However, *polymorphic* worms which change their byte sequences at every successive infection can disable these schemes.

Recently, some polymorphic worm signature generation schemes are proposed. Based on their characteristics, the signatures can be broadly classified into two categories—*exploit-specific* signatures and *vulnerability-driven* signatures. The former ones capture the features specific to a worm implementation, thus might not be generic enough and can be evaded by other exploits. Most of these signatures are content-based, aiming to exploit the residual similarity in the byte sequences of different polymorphic worm samples [7]–[11]. As mentioned in [11], there can be worms which have no content-based signature at all. Furthermore, various attacks have been proposed to evade the content-based signatures [12]–[16].

Unlike exploit-specific signature, vulnerability-driven signature captures the characteristics of the vulnerability the worm exploits, thus, it is inherent to the vulnerability and hard to evade. However, existing vulnerability-driven schemes are mostly host-based [17]–[20]. Both network-based approaches and host-based approaches have pros and cons. Host-based approaches work on end users' hosts or honeypots. They can be more accurate than network-based approaches, since they have better knowledge of the software (binary, source code, or even runtime information of the program). However, it is difficult for host-based systems to achieve a good installation coverage, since some users might disable them for various reasons (e.g., high overhead, system exception, confliction with existing programs, or just uncomfortableness). Honeypots also face scalability problems. On the contrary, network-based approaches can have better coverage, protecting all the users in the enterprise as a whole and thwarting worms fast. However, they can only see limited information, which makes complete vulnerability analysis a challenging job. Moreover, network-based approaches usually need to collect a substantial amount of worm samples before a reasonable signature can be generated, and also maintain an unbiased pool of normal traffic. In summary, we believe the best security practice needs to combine these two together as two-layer defense. They are good complement to each other.

Therefore, the goal of this paper is to find a network-based vulnerability-driven signature generation approach for zero-day polymorphic worms, which will work at the network level and possess the high accuracy of vulnerability-driven schemes. As the first step toward this ambitious goal, we propose length-based signature generator (called LESG) which generates *length-based signatures* which cannot be evaded. That is, even when the attackers know what the signatures are and how the signatures are generated, they 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 vulnerability, 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 for the particular field of the protocol to trigger the buffer overflow, the length of that field is usually much longer than those of the normal requests.

Manuscript received October 26, 2007; revised July 17, 2008; December 04, 2008; and February 26, 2009; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Yan. First published August 11, 2009; current version published February 18, 2010.

L. Wang and X. Li are with the Tsinghua University, Beijing 100084, China.

Z. Li and Y. Chen are with the Northwestern University, Evanston, IL 60201 USA.

Z. Fu is with Motorola Labs, Schaumburg, IL 60196 USA.

Color versions of one or more of the figures shown in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2009.2020431

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

Thus we can use the field length to detect the attacks. This is intrinsic to the buffer overflow, and consequently it is very hard for worm authors to evade.

Among existing vulnerability-driven signatures, a subset has been defined as "vulnerability signatures" specifically [19], [20]. Our length-based signatures might not be so accurate as them, since length-based signatures are not directly based on the exact vulnerability analysis and lack many vulnerability details. But length-based signatures essentially are driven by vulnerabilities, which determines LESG can be fairly accurate and hard to evade. The evaluation in Section VIII-I shows that LESG can handle most (95%) of the buffer overflow vulnerabilities.

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 [21]–[24] invariably suffer from false positives that lead to noise (normal traffic) in the worm traffic pool. Noise is also an issue for honeynet sensors [4], [9], [25]. For example, attackers may send legitimate traffic to honeynets to pollute the worm traffic pool. 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 [12].

**Efficient Signature Matching**. 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. By investigating more than ten protocols and buffer overflow vulnerabilities, we find that in most cases, length-based signature matching can be implemented as regular expression matching, which is a highly developed technique and can be done very fast.

In the rest of the paper, we first survey related work in Section II and discuss the LESG architecture in Section III. Then we formally define the length-based signature generation problem in Section IV, propose a generation algorithm toward it in Section V, and prove the attack resilience bound of LESG in Section VI. Some practical issues including signature matching are discussed in Section VII. After that, we evaluate LESG in Section VIII. Finally, Section IX concludes the paper.

## II. Related Work

Early automated worm signature generation efforts include Honeycomb [4], Autograph [6], and EarlyBird [5], but they do not work well with polymorphic worms.

The classification of existing work on automated polymorphic worm signature generation and LESG is shown in Table I, depending on whether it is vulnerability-driven or exploit-specific, and host-based or network-based.

**Exploit-specific signatures**. We have discussed content-based schemes in Section I, [7]–[11]. In addition, Nemean [25] generates connection and session signatures, which can be misled by deliberate noise injection [12]. Another approach CFG [26] is based on exploit code structure analysis and also can be evaded. Furthermore, it is computationally expensive. In comparison with most recent work in this category, such as Hamsa [7], LESG has better attack resilience, e.g., it has better bounds for deliberate noise injection attacks [12].

TABLE I
Comparison With Other Polymorphic Worm Signature Generation Schemes

| Signature properties | Signature generation mechanisms | |
|---|---|---|
| | Network-based | Host-based |
| Exploit-specific | Polygraph [8]<br>Hamsa [7]<br>PADS [9]<br>Nemean [25]<br>CFG [26] | Taint check [10]<br>DACODA [11] |
| Vulnerability-driven | **LESG** | COVERS [17]<br>Packet Vaccine [18]<br>Vulnerability signature [19]<br>Vigilante [20] |

**Vulnerability-driven signatures**. In this category, some are defined as "vulnerability signatures" [19], [20]. Brumley *et al.* presented the concept of vulnerability signature in [19]. Vigilante [20] proposed a vulnerability signature which is similar to the MEP symbolic constraint signatures in [19]. Liang *et al.* proposed the first host-based scheme to generate length-based signatures [1], [17]. However, the signature generated by COVERS [17] based on a small number of samples may be too specific to represent the overall worm population, thus may have high false negatives. Packet Vaccine [18] further improves the signature quality by using binary search. These schemes are all host-based.

**Other related work**. There are other previous research efforts on network-level detection of buffer overflow exploits. For example, some aim to detect different components or features of exploit code ([27]–[31]), some focus on payload-based anomaly detection ([22], [23]). Although significant progress has been made, most of these approaches suffer from high false positives or false negatives under newly developed attacks [31], [32], or cannot work at high-speed links. Unlike those research efforts, this paper focuses on signature generation for buffer overflow attacks, for which we can afford a relatively expensive process to generate signature but want the signature matching with low overhead and high accuracy. Another category of related work is about misleading attacks [12]–[16]. We discuss them in Section VI-B in detail.

## III. Architecture of LESG

As shown in Fig. 1, LESG can be connected to various 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.

Similar to the basic framework of Polygraph [8] and Hamsa [7], we first need to sniff traffic from networks and classify the traffic as different application level protocols, based on port numbers or other protocol identifiers. Next, for each protocol, we filter out known worms and separate the traffic into a suspicious traffic pool and a normal traffic reservoir using an existing flow classifier [6], [21]–[24].

That existing flow classifier may use various techniques (such as honeynet [33]–[35], port scan detection [6], [36], byte frequency detection [22], [23], and other advanced techniques) to identify suspicious flows. Note that the flow classifiers can operate at line speed of routers [36]. The scan detection based flow classifiers first detect hosts scanning a particular port number, and then classify successful TCP connections from any

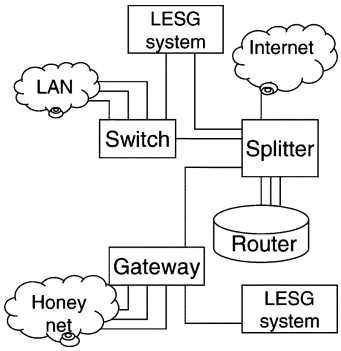



Fig. 1. Deployment of LESG.

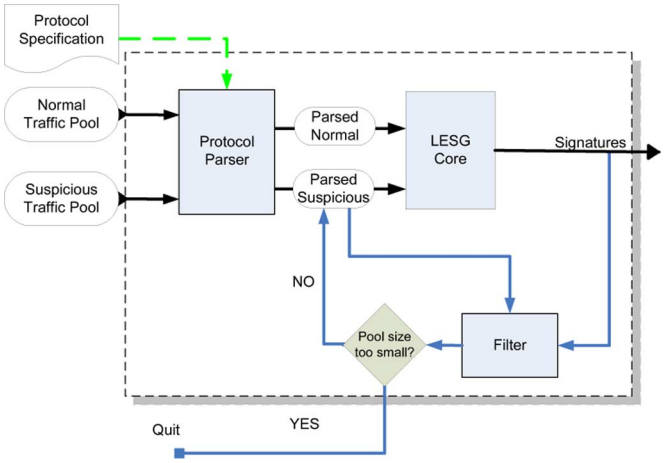



Fig. 2. LESG signature generator.

of the scanning hosts as suspicious flows. It is effective against scanning worms. 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 [7], we can create the normal pool. The suspicious pool and the normal pool are inputted to the signature generator as shown in Fig. 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 type and a length. The field length information of both the suspicious pool and the normal pool is given as input to the "LESG core"(signature generation algorithm) module to generate the signatures.

### A. Protocol Parsing

As emphasized in [37], protocol parsing is an important step in any semantic analysis of network traffic, such as network monitoring, network intrusion detection systems [2], [3], smart firewalls, etc. Recently, many research efforts [38]–[40] have been done on the protocol reverse engineering, making the parsing of close protocols also possible.

We have analyzed six text-based protocols (HTTP, FTP, SMTP, POP3, IRC, IDENT) 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 [37], has studied how to ease the job of writing a protocol parser. BINPAC actually works as a parser generator. Its input is a script which is a protocol specification written in BINPAC language. The output is a parser code for that protocol. Currently, BINPAC is executed in connection with Bro [3]. 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. For these advantages, we use BINPAC and Bro for packet flow reassembling and protocol parsing in our research.

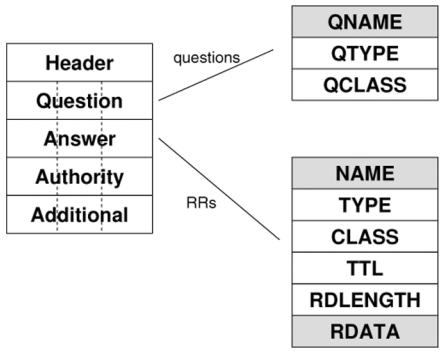



Fig. 3. Illustration of DNS PDU.

## IV. Length-Based Signature Definition and Problem Statement

In this section, we model each network flow 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 network 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. 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 subsequence 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. Fig. 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 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.).

Among all the fields in PDUs, some, e.g., QNAME (denoted as field $A$), NAME (field $C$, $F$, $I$) and RDATA (field $E$, $H$, $K$), as in Fig. 4, are *variable-length fields*; others are *fixed-length fields*, of which the fixed lengths are defined in the protocol standard. The continuous fixed-length fields can be combined as one field, for example, field $B$ in Fig. 4 represents the combination of QTYPE and QCLASS, and field $D$ represents the combination of TYPE, CLASS, TTL, and RDLENGTH.

TABLE II
TABLE OF NOTATIONS

| | |
|---|---|
| $\mathcal{M}$ : 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 normal flows in $\mathcal{N}$ |
| $\mathcal{M}^1$ : set of true worm flows in $\mathcal{M}$ | $\mathcal{M}^2$ : set of noise flows in $\mathcal{M}$ |
| $\alpha$ : 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 $\mathcal{S}$ |
| $\mathrm{COV}_0$ : minimum coverage requirement for a signature candidate | $\mathrm{FP}_0$ : maximum false positive rate for a signature candidate |
| $\gamma'$ : minimum coverage increase requirement for a signature to be outputted in the first loop of the Step 3 algorithm | $\gamma$ : minimum coverage increase requirement for a signature to be outputted in the second loop of the Step 3 algorithm |


Fig. 4. Abstraction of DNS PDU.

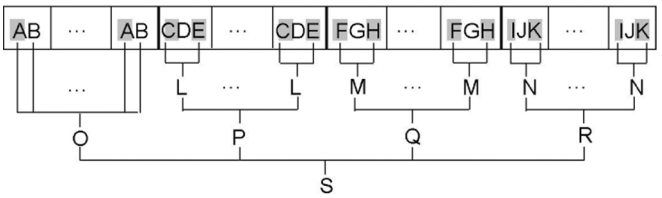



Fig. 5. Hierarchical Structure of DNS PDU.

We make the following observation on such a representation of PDU. The concatenation of multiple fields may be defined as another higher-level field with special semantic meanings, and stored in one buffer in certain server implementation. That is, if the server has an overflow vulnerability related to this buffer, it is the instance (i.e., corresponding subsequence of bytes) of that higher-level field in a flow that can overflow the buffer. For example, imagine a DNS server stores the entire PDU in a buffer when receiving a DNS PDU.

With these considerations, we design a hierarchical model to describe the structure of fields in a PDU. As Fig. 5 shows, we denote QUESTION section, the concatenation of repeated fields $A$ and $B$, as another field $O = (AB)^*$. Similarly, field $P = L^* = (CDE)^*$, and the highest level field $S$ is the entire PDU. We call the fields which are formed by combining consecutive fields together *compound fields*. For the fields which cannot be further decomposed, we call them *simple fields*.

In short, we include all possible variable-length fields that potentially correspond to vulnerable buffers. Such a hierarchy can be built for every protocol technically.

Suppose there is a total of $K$ variable-length fields in the hierarchy constructed for a certain protocol. We use an index set $E = \{1, 2, \ldots, K\}$ to denote these $K$ fields. For one flow, 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 uniquely to represent the field lengths for each variable-length field in this flow. In the rest of the paper, we refer to variable-length fields simply as fields for the sake of brevity.

### B. Length-Based Signature Definition

Based on the length vector representation of a flow above, 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 also called 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 $x^{f_j}$ with $l_j$. If $x^{f_j} > l_j$, flow $X$ is labeled as a worm flow; otherwise it is labeled as normal. More than one signature corresponding to different fields can possibly be generated for a given protocol, resulting in a *signature set* $\mathcal{S} = \{S_1, S_2, \ldots, S_J\}$. A flow will be labeled 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 mapped exactly to a vulnerable buffer. Let $L_B$ be the length of the vulnerable buffer, $f_B$ be the corresponding field ID, we define signature $B = (f_B, L_B)$ as the *ground truth signature*. If multiple server implementations have vulnerable buffers corresponding to the same field (the probability of this case should be small), we choose the *minimum* buffer length as the ground truth.

### 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 contain some normal flows. 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 known worms, there can be multiple new 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 $\mathcal{S} = \{S_1, S_2, \ldots, S_J\}$. 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)]

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** $\frac{|\mathcal{M}_{l_j}|}{|\mathcal{M}|} \geq \mathrm{COV}_0$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{(f_j, l_j)\}$;
  **end**
**end**
**Output** S ;

INPUT: Suspicious traffic pool $\mathcal{M} = \{M_1, M_2, \ldots\}$ and normal traffic pool $\mathcal{N} = \{N_1, N_2, \ldots\}$; value $\beta < 1$.
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 $\mathrm{COV}_\mathcal{S} \geq \beta$.

**Hardness** For a buffer overflow worm in the suspicious pool, in absence of noise, generation of a set of length-based signatures is a polynomial time problem, since we know the size of the set is one. However, with noise and multiple worms, the computational complexity of the problem has significantly changed.

*Theorem 1:* NLBSG is NP-Hard

*Proof Sketch:* The proof is by reduction from Minimum $k$ Union, which is equivalent to Maximum $k$-Intersection [41]. ■

## V. Signature Generation Algorithm

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-steps algorithm to generate length-based signatures. Although the problem NLBSG is NP-Hard in general, for buffer overflow worms, the algorithm is fast and has fair accuracy even in the worst case scenarios, which is proved in Section VI. To the best of our knowledge, this is the *first* network-based signature generation approach that has the accuracy bound even with adversaries' injected noise.

**Step-1:** *Field Filtering* select candidate signatures.
**Step-2:** *Signature Length Optimization* optimize the signature length for each field.
**Step-3:** *Signature Pruning* find the optimal subset of candidate signatures with low false positive rate and false negative rate.

### A. Field Filtering

In this step of the algorithm, we make the first selection on the fields that could be candidate signatures. The goal is to narrow down the searching space. Two parameters are set as the input: $\mathrm{FP}_0$ and $\mathrm{COV}_0$, which indicate the basic requirements on the false positives and detection coverage.

In the Step-1 algorithm, $\mathcal{N}_{l_j}$ and $\mathcal{M}_{l_j}$ denote the flows detected by signature $(f_j, l_j)$ in pools $\mathcal{N}$ and $\mathcal{M}$ respectively.

We process each field separately. For every field, 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. If the coverage $\alpha$ of true worm samples in the suspicious pool $\mathcal{M}$ is more than $\mathrm{COV}_0$, and the ground truth signature $B$ has no false positive, we are always able to find a *vulnerable field signature*, of which the field ID is the vulnerable field ID (i.e., field ID of ground truth signature).

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

**for** signature $(f_j, l_j) \in \mathcal{S}$
  sort $\mathcal{M}^{f_j}$ in ascending order;
  find $m_0$ such that $x^{f_j}_{m_0-1} < l_j < x^{f_j}_{m_0}$;
  $max\_score \leftarrow 0$;
  **for** $m' = m_0$ to $|\mathcal{M}|$
    $l'_j \leftarrow x^{f_j}_{m'} - 1$;
    **if** $(max\_score < Score(\mathrm{COV}_{l'_j}, \mathrm{FP}_{l'_j}))$
      $max\_score \leftarrow Score(\mathrm{COV}_{l'_j}, \mathrm{FP}_{l'_j})$;
      $l_j \leftarrow l'_j$;
      $m \leftarrow m'$;
    **end**
  **end**
  **while** $(l_j > \frac{x^{f_j}_{m-1} + x^{f_j}_{m}}{2})$
    **if** $(Score(\mathrm{COV}_{l_j}, \mathrm{FP}_{l_j}) == Score(\mathrm{COV}_{l_j-1}, \mathrm{FP}_{l_j-1}))$
      $l_j \leftarrow l_j - 1$;
    **else**
      update $\mathcal{S}$ with $l_j$; **Break**;
    **end**
  **end**
**end**
**Output** $\mathcal{S}$;

### B. Signature Length Optimization

The first step has selected candidate signatures to meet the basic requirements. In the second step, we try to optimize the length value of each candidate signature. There is often tradeoff between the detection coverage and false positive rate. We need a method to compare different lengths to determine which one is "better." For the sake of brevity, let $\mathrm{FP}_{l_j}$ denote the false positive rate (computed with $\mathcal{N}$) of signature $(f_j, l_j)$ and let $\mathrm{COV}_{l_j}$ denote its coverage on $\mathcal{M}$. This step aims to maximize a *score function* $\mathrm{Score}(\mathrm{COV}_{l_j}, \mathrm{FP}_{l_j})$ for each field $f_j$. The notion *score function* is proposed in [7], to determine the best tradeoff between the false positive and detection 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 $X_m = (x^1_m, x^2_m, \ldots, x^K_m)$, $m = 1, 2, \ldots, |\mathcal{M}|$, is the length of each field in a flow $m$. We define $\mathcal{M}^k = \{x^k_1, x^k_2, \ldots, x^k_{|\mathcal{M}|}\}$. Signature set $S$ generated in Step-1 is the input of this step.

In $\mathcal{M}^{f_j} = \{x^{f_j}_1, x^{f_j}_2, \ldots, x^{f_j}_{|\mathcal{M}|}\}$, if $x^{f_j}_m$ is in the ascending order, it is obvious that between any two consecutive elements, namely $x^{f_j}_{m-1}$ and $x^{f_j}_m$, the score $\mathrm{Score}(\mathrm{COV}_{l_j}, \mathrm{FP}_{l_j})$ is monotonically non-decreasing in $l_j$. Thus we only need to search among all the $x^{f_j}_m - 1$, $m = m_0, \ldots, |\mathcal{M}|$, for the maximum score, i.e., the total number we need to try is at most $|\mathcal{M}|$. This search is done by the first loop of Step-2 algorithm.

The result of that search is $l_j = x^{f_j}_m - 1$, $m \in [m_0, |\mathcal{M}|]$. If the length distributions of field $f_j$ in normal flows and in worm flows are well separated, $l_j$ would be much larger than $x^{f_j}_{m-1}$. Then in the second loop of Step-2 algorithm, $l_j$ decreases until the score changes (decreases actually) or $l_j$ reaches the median of $[x^{f_j}_{m-1}, x^{f_j}_m]$. In Section VI-B, we will discuss the advantage of performing this loop.

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

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

$m \leftarrow |\mathcal{M}|$; $\Omega \leftarrow \emptyset$;
$\mathcal{S}_1 \leftarrow \{s | s \in \mathcal{S}, \mathrm{FP}_s = 0\}$; $\mathcal{S}_2 \leftarrow \{s | s \in \mathcal{S}, \mathrm{FP}_s > 0\}$;
*LOOP1:*
**while** ($\mathcal{S}_1 \neq \emptyset$)
  Find $s \in \mathcal{S}_1$ such that $\frac{|\mathcal{M}_s|}{m}$ is the maximum one in $\mathcal{S}_1$;
  **If** ($\frac{|\mathcal{M}_s|}{m} \geq \gamma'$)
    $\Omega \leftarrow \Omega \cup \{s\}$; $\mathcal{S}_1 \leftarrow \mathcal{S}_1 - \{s\}$;
    Remove all the samples which match $s$ in $\mathcal{M}$;
  **else**
    **Break**;
  **end**
**end**
*LOOP2:*
**while** ($\mathcal{S}_2 \neq \emptyset$)
  Find $s \in \mathcal{S}_2$ such that $\frac{|\mathcal{M}_s|}{m}$ is the maximum one in $\mathcal{S}_2$;
  **If** ($\frac{|\mathcal{M}_s|}{m} \geq \gamma$)
    $\Omega \leftarrow \Omega \cup \{s\}$; $\mathcal{S}_2 \leftarrow \mathcal{S}_2 - \{s\}$;
    Remove all the samples which match $s$ in $\mathcal{M}$;
  **else**
    **Break**;
  **end**
**end**
**Output** $\Omega$;

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 $|\mathcal{S}| \leq K$, 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 signatures. Usually, the more signatures we use, the more false positives there might be, since all the signatures are to be matched against flows in the detection phase. In this step, we will find an optimal subset of the candidate signatures to be the final signature set.

Among all the signatures generated by Step-2 algorithm, we denote the vulnerable field signature (defined in Section V-A) as $B'$. The Step-3 algorithm contains two loops, denoted as $\mathrm{LOOP}_1$ and $\mathrm{LOOP}_2$, where $\gamma'$ and $\gamma$ are parameters and $\gamma' < \gamma$. In $\mathrm{LOOP}_1$, we try to find the signatures which can improve the detection coverage by $\gamma'$ at least, without generating any false positives. Usually, $\gamma'$ is small. Therefore, if $B'$ has no false positive, this loop can help improve the true positives even when adversaries are present. Then in $\mathrm{LOOP}_2$, we use a similar process to find signatures which can improve the detection coverage by $\gamma$, but may cause false positives.

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 $O(K(K \log |\mathcal{M}| + |\mathcal{M}|))$.

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 positive rate and false negative rate. For nonbuffer-overflow worms, the algorithm will output an empty set, since no signature meets the minimal requirement ($\mathrm{FP}_0$ and $\mathrm{COV}_0$) on accuracy.

## VI. Attack Resilience Analysis

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

TABLE III
Worst Cases With Different Assumptions

| Attackers can fully craft the worms and | $\mathrm{FN}_{\{B'\}} = 0$ and | |
|---|---|---|
| | $\mathrm{FP}_{\{B'\}} = 0$ | $\mathrm{FP}_{\{B'\}} \leq \mathrm{FP}_0$ |
| can craft noises | Theorem 2 | Theorem 3 |
| cannot craft noises | Theorem 4 | Theorem 5 |

### A. Worst Case Performance Bounds

The *ground truth signature* $B = (f_B, L_B)$ (defined in Section IV-B) is the best possible signature we could obtain, theoretically having no false positive ($\mathrm{FP}_{\{B\}} = 0$) and no false negative ($\mathrm{FN}_{\{B\}} = 0$). The *vulnerable field signature* $B' = (f_B, L'_B)$ (defined in Section V-A) may not be as perfect, since the length $L'_B$ may slightly differ from the buffer length $L_B$. In Step-1 and Step-2 of the signature generation algorithm, we tend to choose a more conservative signature than the ground truth signature $B$, i.e., $L'_B \leq L_B$, therefore $\mathrm{FN}_{\{B'\}} = 0$ and $\mathrm{FP}_{\{B'\}} \leq \mathrm{FP}_0$. Actually, for most worms, the vulnerable field length distributions of normal flows and worm flows are well apart, i.e., there is a noticeable gap between the two distributions, so by selecting an appropriate score function, we can achieve $\mathrm{FP}_{\{B'\}} = 0$.

Since we do not know which field is vulnerable *a priori*, the Step-1 algorithm might select the vulnerable field and some other fields (nonvulnerable fields) as well, especially in the case that an attacker can inject crafted noises. Note that in these noise flows, the nonvulnerable fields can be arbitrarily long to mislead the signature generator, but the instances of vulnerable field must be shorter than the corresponding buffer, otherwise these noises will trigger the overflow and become worm samples actually. Our algorithm has an accuracy bound even when crafted noises are injected. Theorem 2 to Theorem 5 proved below present the accuracy bounds for four different cases, as summarized in Table III.

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 the set of noises. Let the fraction of worm flows in $\mathcal{M}$ be $\alpha$, i.e., $\frac{|\mathcal{M}^1|}{|\mathcal{M}|} = \alpha$. Except Theorem 2, the proofs of all the following theorems can be found in [43].

*1) Performance Bounds With Crafted Noises:* In Theorems 2 and 3, we prove the worst case performance bounds of our system under the deliberate noise injection attacks, in which the attackers not only fully craft the worms but also inject the crafted noises. This is the worst case. The difference between Theorem 2 and Theorem 3 is that Theorem 2 assumes there is a noticeable gap between the length distributions of normal flows and worm flows, which is the most common case in reality. Theorem 3 considers even more general cases, in which the two distributions might not be well apart so that we get $0 < \mathrm{FP}_{\{B'\}} \leq \mathrm{FP}_0$. Due to the space limit, we only present the proof of Theorem 2 here. Please refer to our technical report [43] for all the other proofs.

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

*Proof:* Let the vulnerable field signature be $s$. We know $\frac{|\mathcal{M}^1_{\{s\}}|}{|\mathcal{M}^1|} = 1$ and $\mathrm{FP}_{\{s\}} = 0$. Let the signature set we find

in $\mathrm{LOOP}_1$ be $\Omega_1$, and the signature set found in $\mathrm{LOOP}_2$ be $\Omega_2 = \Omega - \Omega_1$.

After $\mathrm{LOOP}_1$, the residue of true worm samples $|R| < \gamma' \cdot |\mathcal{M}|$, which can be proved as follows. If $|R| \geq \gamma' \cdot |\mathcal{M}|$, $s$ should be taken as the output since it will be better. Then there is no true worm samples left. Therefore, $|R| < \gamma' \cdot |\mathcal{M}|$.

Then we have $|\mathcal{M}^1_\Omega| \geq |\mathcal{M}^1_{\Omega_1}| = |\mathcal{M}^1 - R| = |\mathcal{M}^1| - |R| > |\mathcal{M}^1| - \gamma' \cdot |\mathcal{M}|$. Since $\frac{|\mathcal{M}^1|}{|\mathcal{M}|} = \alpha$, $|\mathcal{M}^1_\Omega| > |\mathcal{M}^1| - \gamma' \cdot |\mathcal{M}| = |\mathcal{M}^1| - \gamma' \cdot \frac{|\mathcal{M}^1|}{\alpha} = (1 - \frac{\gamma'}{\alpha}) \cdot |\mathcal{M}^1|$. Hence, $\frac{|\mathcal{M}^1_\Omega|}{|\mathcal{M}^1|} > 1 - \frac{\gamma'}{\alpha}$. Therefore, $\mathrm{FN}_\Omega < \frac{\gamma'}{\alpha}$. Suppose the first output signature in $\mathrm{LOOP}_1$ is $s'$; then $\frac{|\mathcal{M}_{\{s'\}}|}{|\mathcal{M}|} \geq \frac{|\mathcal{M}_{\{s\}}|}{|\mathcal{M}|} = \alpha$. Therefore after $\mathrm{LOOP}_1$, the remaining suspicious pool size $|\mathcal{M}'| \leq (1 - \alpha) \cdot |\mathcal{M}|$.

Since $\mathrm{FP}_{\Omega_1} = 0$, we have $\mathrm{FP}_\Omega = \mathrm{FP}_{\Omega_2}$. Since in $\mathrm{LOOP}_2$ each iteration needs to improve coverage by $\gamma$, there at most are $\lfloor \frac{|\mathcal{M}'|}{\gamma \cdot |\mathcal{M}|} \rfloor \leq \lfloor \frac{(1-\alpha) \cdot |\mathcal{M}|}{\gamma \cdot |\mathcal{M}|} \rfloor = \lfloor \frac{1-\alpha}{\gamma} \rfloor$ iterations. Each iteration may introduce false positive rate $\mathrm{FP} \leq \mathrm{FP}_0$. Therefore, the final false positive rate is bounded by $\mathrm{FP}_0 \cdot \lfloor \frac{1-\alpha}{\gamma} \rfloor$ ■

*Theorem 3:* If the vulnerable field signature has no false negative and the false positive rate is bounded by $\mathrm{FP}_0$, the three-steps algorithm outputs a signature set $\Omega$ such that $\mathrm{FN}_\Omega < \frac{\gamma}{\alpha}$ and $\mathrm{FP}_\Omega \leq \mathrm{FP}_0 \cdot (\lfloor \frac{1-\alpha}{\gamma} \rfloor + 1)$.

These bounds are still tight, as shown in the example of deliberated noise injection attacks in Section VI-B. Furthermore, under the noise injection attacks, the experimental accuracy results obtained in Section VIII-G are much better than these bounds.

*2) Performance Bounds Without Crafted Noises:* Since injected noises will slow down the worm propagation, the worm authors might not want to do so. Suppose the noise ratio is 90% (i.e., 90% of traffic from a worm is crafted noise), the worm will propagate at least 10 times slower than before based on the RCS worm model [43]. For example, the Code Red II may take 140 h (6 d) to compromise all vulnerable machines instead of 14 h.

Without crafted noises, i.e., the noises are all from normal traffic, we are able to prove even tighter performance bounds for our system. Here, Theorem 4 assumes the length distributions of normal flows and worm flows are well apart, while 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 vulnerable field signature has no false positive and no false negative, the three-steps algorithm outputs a signature set $\Omega$ such that $\mathrm{FN}_\Omega = 0$ and $\mathrm{FP}_\Omega = 0$.

In this case, the output signature set $\Omega$ contains the vulnerable field signature.

*Theorem 5:* If the noise in the suspicious pool is normal traffic and not maliciously injected and the vulnerable field signature has no false negative and a false positive rate bounded by $\mathrm{FP}_0$, the three-steps algorithm outputs a signature set $\Omega$ 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 VIII-B are consistent with the theorems and are better than the bounds proved in the theorems.

*3) Discussions:* In this part, we discuss some issues related to the above theorems on attack resilience. *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 bounds. *Assumptions for theorems on attack resilience*. There are three general assumptions for all the theorems above.

First, there is a direct mapping between a field and the vulnerable buffer, thus that field of any worm must be longer than the buffer. This has been validated by our wide investigation on real-world buffer overflow vulnerabilities (in Section VIII-I).

Second, the attackers cannot change the field length distributions of normal traffic, which is also generally true.

Third, the vulnerable fields of all or most normal flows are shorter than the buffer, no matter the flow is destined to that vulnerable application or other applications. The reason for the former is that if the buffer is so small that many ($> \mathrm{FP}_0$) normal requests can overflow it, it will be noticed and fixed quickly, otherwise the application cannot be popular; while the reason for the latter is that normally the users' requests are independent of the server implementation, thus the profile of traffic destined to different server applications should be similar. If this assumption is invalid in a scenario, LESG might generate no signature (in the case that many normal flows overflow the vulnerable buffer) or inaccurate signature which will cause false positives (in the case that the normal pool is not a representative sample of the entire normal traffic). Actually, the vulnerable fields of most normal flows are much shorter than the buffer, since the application writers tend to allocate a buffer long enough for normal use.

With above assumptions, in most cases we can generate accurate signatures. Compared with the recent Hamsa system [7], 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 [12]–[16].

*Deliberate noise injection attack* can mislead most existing worm signature generators [12], and is also the most threatening attack toward our approach. However, even against this attack, our approach can perform reasonably well (proved by Theorem 2 and Theorem 3), especially in the case that the vulnerable field signature has no false positive. For example, if $\mathrm{FP}_0 = 0.1\%$, $\gamma' = 1\%$ and $\gamma = 5\%$, even with 90% crafted noise, $\mathrm{FN}_\Omega$ is bounded by 10% and $\mathrm{FP}_\Omega$ by 1.8%, according to Theorem 2. The experiment result in Section VIII-G is even much better than that bound: the evaluation FN is 6.3%, and FP is 0.14%. To the best of our knowledge, this is the *first* network-based approach that can achieve this performance. The *suspicious pool poisoning attack* proposed in Paragraph [13] is similar to deliberate noise injection attack.

*Randomized red herring attack* proposed in Paragraph [13] misleads the signature generators by the spurious tokens contained in all the worm samples in suspicious pool coincidentally. So the worm flows which do not contain the tokens are false negatives. However, it is hard to launch an effective similar attack toward our approach. Since the Step-3 algorithm will choose only one among the true (vulnerable) signature and the spurious ones, the probability of generating a spurious signature which can cover all the worm samples in suspicious pool

and produce many false negatives is quite low. Moreover, constructing plenty of long fields is much more difficult than tokens. In Paragraph [13], 400 spurious tokens are constructed.

*Dropped red herring attack* [13] includes some tokens at the beginning of the worm spread and drops those tokens in later propagation of the worm. This attack is also hard to implement, because obviously the true signature is more probable to be generated than the changing spurious signatures.

*Length dropping attack* is a similar attack which can be designed specially for length-based signatures. The attackers can inject a long input $L$ at the beginning and gradually decrease it in each run of infection to $L_B$. However, in our design we choose the signature length to be $l_j = \frac{x_{m-1}^{f_j} + x_m^{f_j}}{2}$, where $x_{m-1}^{f_j}$ is comparable to $L_B$ and $x_m^{f_j}$ is comparable to $L$. Therefore, in the worst case we only need to regenerate the length signature $O(\log(L - L_B))$ times.

*Innocuous pool poisoning* is to pollute normal traffic pool. However, this is very hard in general. First, the amount of normal traffic is so huge that even to poison 1% is hard. Second, using the random selection policy of normal traffic [7], it is very hard for attackers to poison the traffic in the right time to perform an effective evasion during the worm breakout.

*Type I and II allergy attacks* make the IDS generate signatures which can deny current normal traffic and future normal traffic respectively [14]. The type I attack does not work for our approach since we check the false positive rate against the normal traffic. The type II attack is also ineffective toward our approach, because unlike the contents of normal traffic, which may change a lot, the field length profile of normal traffic is quite stable.

The *blending attacks* [15] cannot work for our approach because the worms have to use a longer-than-normal input for the vulnerable field and they cannot mimic the normal traffic.

Recent research effort [16] has made a unified analysis of the learning-based signature generation algorithms. Under the assumptions discussed in Section VI-A3, LESG can work well according to their analysis.

## VII. Discussions on Length-Based Signature Matching

The operation of length-based signature matching has two steps: protocol parsing of the flows and field length comparison with the signatures. The latter is trivial. A straightforward way for protocol parsing is using a general parser. Currently, Bro and BINPAC based parsing can achieve 50–200 Mbps. Recently, Schear *et al.* proposed to simplify the protocol parsing for vulnerability signatures, which can achieve 1 Gbps or more [44]. 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 [45]. Therefore, with hardware support, protocol parsing can be done fast.

Actually, since the only aim of protocol parsing here is to get the lengths of the signature fields in the flows, it can be highly optimized and much faster than full parsing performed by general parsers mentioned above. We will discuss the schemes that speed up the process of protocol parsing, and consequently signature matching.

### A. Length-Based Signature in Regular Expression Format

We found it prevalent that in the worm flows targeting a certain vulnerability, the corresponding vulnerable field is prefixed and suffixed with a byte sequence that matches a unique regular expression respectively. For signature matching, we could match those two regular expressions against a flow to locate the vulnerable field, instead of fully parsing the entire flow. In such a case, the signature $S = (f_S, L_S)$ can be converted to *length-based signature in regular expression format* (abbreviated as *L-RE signature*). L-RE signature can be denoted as a three-tuple $S_r = \langle \mathrm{RE}_p, \mathrm{RE}_s, L_S \rangle$, where $\mathrm{RE}_p$ and $\mathrm{RE}_s$ are the prefix and suffix regular expressions respectively, and $L_S$ is signature length, i.e., the lower bound of the distance between $\mathrm{RE}_p$ and $\mathrm{RE}_s$ in worm flows. Regular expression matching is a highly developed technique. Recent work [46] has achieved 10 Gbps while matching thousands of regular expressions. Specifically, IDS Snort has already included signatures similar to L-RE signature to detect buffer overflow attacks. For example, rule sid-3087 uses a regular expression "/w3who.dll\x3F[^\r\n]{519}/i"[2] to match malicious inputs prefixed with "/w3who.dll\x3F/i" and longer than 519 bytes [2].

Given a length-based signature $S = (f_S, L_S)$, we discuss the methods for generating $\mathrm{RE}_p$ (or $\mathrm{RE}_s$) in two different cases. The first case is that field $f_S$ has a prefix (or suffix) defined by the protocol, which we call *protocol-defined prefix (or suffix) regular expression*, denoted as $\mathrm{RE}'_p$ (or $\mathrm{RE}'_s$). The $\mathrm{RE}'_p$ (or $\mathrm{RE}'_s$) is known and can be directly used as $\mathrm{RE}_p$ (or $\mathrm{RE}_s$), i.e., $\mathrm{RE}_p = \mathrm{RE}'_p$ (or $\mathrm{RE}_s = \mathrm{RE}'_s$). In this case, $\mathrm{RE}_p$ (or $\mathrm{RE}_s$) is *field-specific*, since the instances of field $f_S$ in any flows must be prefixed (or suffixed) with a string matching $\mathrm{RE}_p$ (or $\mathrm{RE}_s$).

L-RE signatures with field-specific $\mathrm{RE}_p$ or $\mathrm{RE}_s$ are prevalent in text-based protocols. Text-based protocols are usually *line-based*, which means each flow or part of each flow is composed of multiple lines. Usually, each line can be interpreted as a command with parameters. Each line is composed of three string components: $\mathrm{str}_{\mathrm{cmd}}$, $\mathrm{str}_{\mathrm{para}}$, and "\r\n". The $\mathrm{str}_{\mathrm{para}}$ can be a list of parameters separated by a delimiter, such as white space. To simplify the discussion we ignore the internal structure of $\mathrm{str}_{\mathrm{para}}$ here. But in the real implementation we do consider the internal structure which makes the $\mathrm{RE}_p$ and $\mathrm{RE}_s$ a little bit more complex. $\mathrm{str}_{\mathrm{para}}$ might correspond to certain field $f \in E_l \subseteq E$ ($E$ is the set of field IDs of variable length fields in a given protocol, as defined in Section IV-A). We call these lines *field-related lines*. $E_l$ usually contains tens of elements (namely, field IDs) at least, and any field $f_x \in E_l$ has the features that: i) The $\mathrm{str}_{\mathrm{para}}$ of certain field-related lines are instances of field $f_x$; ii) field $f_x$ has a unique protocol-defined prefix $\mathrm{RE}'_p =$ ``\r\n$\mathrm{RE}_p^0$", and the $\mathrm{str}_{\mathrm{cmd}}$ component of a line matches $\mathrm{RE}_p^0$; iii) field $f_x$ has an invariant protocol-defined suffix $\mathrm{RE}'_s \equiv$ ``\r\n". For example, in HTTP protocol, field `GET-REQUEST` has $\mathrm{RE}'_p =$ ``/\r\nGET\s/i" and $\mathrm{RE}'_s =$ ``\r\n". For an example GET-request line "GET http://www.tsinghua.edu.cn/index.html HTTP/1.1\r\n", $\mathrm{str}_{\mathrm{cmd}} =$ ``GET" matches $\mathrm{RE}_p^0 =$ ``/GET\s/i", and $\mathrm{str}_{\mathrm{para}} =$ "http://www.tsinghua.edu.cn/index.html HTTP/1.1" is the instance of field `GET-REQUEST`.

Compared with text-based protocols, binary protocols have fewer fields with protocol-defined prefix or suffix, therefore L-RE signatures with field-specific $\mathrm{RE}_p$ or $\mathrm{RE}_s$ are less prevalent but still available. We will present two examples of

[2]Modifier "/i" means doing case-insensitive matching.

```
RE_p ← NULL;
for k = k_0 − 1 to 1
  if field k has same content in all the flow in M_S
    let content be the content of field k;
    RE_p ← content + RE_p;
  else if the field k in all the flows in M_S has same length l;
    RE_p ← .{l} + RE_p;
  else
    remove the unnecessary .{l} at the beginning of RE_p;
    Break;
Output RE_p;
```

Fig. 6. $\mathrm{RE}_p$ generation algorithm.

L-RE signatures with field-specific $\mathrm{RE}_p$ and $\mathrm{RE}_s$ for binary protocols in the evaluation section.

The second case is that there is no apriori known prefix or suffix for the signature field. But it is still possible that the worm samples in the suspicious pool have a common prefix and suffix, and we try to find out that. Using signature $S = (f_S, L_S)$, we can obtain all the worm flows $\mathcal{M}_S$ which match signature $S$ from suspicious pool $\mathcal{M}$. Then we search in $\mathcal{M}_S$ to generate the prefix pattern $\mathrm{RE}_p$ with the algorithm in Fig. 6. The basic idea of the algorithm is to find the longest possible common regular expression pattern prefixing field $f_S$ among all the worm flows.

We label all the simple fields preceding the signature field including both fixed-length fields and variable-length fields from 1 to $k_0 - 1$. The algorithm searches for the longest index subsequence $\langle k_j, k_j + 1, \ldots, k_0 - 1\rangle$, in which each field has either invariant content or just invariant length in all the worm flow $\mathcal{M}_S$, and concatenates these fields into one regular expression $\mathrm{RE}_p$. In Fig. 6, "+" is string concatenation operator. We remove the unnecessary expression "$.\{l\}$" at the beginning of $\mathrm{RE}_p$. The algorithm for generating $\mathrm{RE}_s$ is similar, except that the simple fields following the vulnerable field are searched.

*a) Analysis on Accuracy and Attack Resilience:* Since generating length-based signature $S$ and converting it to L-RE signature $S_r$ are two separated processes, we just need to analyze the false negatives and false positives produced by $\mathrm{RE}_i, i \in \{p, s\}$ generated in the latter process. We term them *RE-caused false negatives* and *RE-caused false positives*.

Field-specific $\mathrm{RE}_i, i \in \{p, s\}$ never produce false negatives. $\mathrm{RE}_i$ generated by the algorithm in Fig. 6 is either *vulnerability-specific* or *exploit-specific*. Vulnerability-specific $\mathrm{RE}_i$ is required for exploiting certain vulnerability and also produces no false negative, while exploit-specific $\mathrm{RE}_i$ can be replaced by other byte strings. We have discussed that length-based signature has a great advantage that it is vulnerability-driven, but exploit-specific $\mathrm{RE}_i$ will make L-RE signature exploit-specific, which might be employed by the attackers to produce false negatives. Suppose that one worm has exploit-specific $\mathrm{RE}_{p1}$ at first, and LESG generates L-RE signature $S_{r1}$. Later, the worm replaces $\mathrm{RE}_{p1}$ by $\mathrm{RE}_{p2}$, thus false negatives are produced.

We propose a simple scheme to resist such an attack. Since LESG is fast, in the case above, it will generate a new signature $S_{r2}$ soon, from the later worm samples in the suspicious pool. Our scheme is that once an L-RE signature is generated by the algorithm in Fig. 6, we compare its field ID with the existing L-RE signatures. If we find a signature having the same field ID, we replace these two signatures by a length-based signature $S = (f_S, L_S)$, where $L_S$ is the average of the two signatures' lengths. In this way, false negatives can only be produced in the short time before the second exploit-based signature is generated. Moreover, we have found in the experiments (in Section VIII-H) that exploit-specific $\mathrm{RE}_i$ is not prevalent, therefore the design of L-RE signature is still effective overall.

When we use L-RE signature, additional false positives also might be produced, due to the non-vulnerable fields of normal flows matching $S_r$ by chance. Therefore, when signature $S_r$ is generated, we check its false positive rate FP against the normal pool. If $\mathrm{FP} \leq \mathrm{FP}_0$, $S_r$ is applied; otherwise the original signature $S$ is applied. Therefore, although the attackers can craft exploit-specific $\mathrm{RE}_i$ to increase FP in some cases, the performance bound discussed in Section VI is unchanged. In addition, for *strictly line-based* protocols (e.g., FTP control channel), the data of which are purely field-related lines, field-specific $\mathrm{RE}_i$ never produce RE-caused false positive, since field-specific $\mathrm{RE}_i$ is exactly defined for locating the field by the protocol.

### B. Partial Protocol Parsing

For the length-based signatures that cannot be converted to L-RE signatures, we still could speed up the matching process by partial protocol parsing, especially for binary protocols. We summarize four cases in which certain fields need not be parsed, as follows:

- A protocol often has multiple types of PDUs, and the type is indicated by a type field. If the signature generator finds that all the attacking PDUs have a common type, then during signature matching, PDUs identified as other types need not be parsed any more. More generally, besides the type field, any field common to all the attacking PDUs can work in a similar way.
- The fields positioned after the signature fields need not to be parsed.
- For continuous fixed-length fields, as long as none of them is used to determine the length of the following variable-length fields, the parser can simply skip over them. A large portion of fields are fixed-length fields in many protocols, for example, DNS, as Fig. 3 shows.
- Suppose part of the field hierarchy of a protocol is like $P = LXY = L(AB)Y$, where $Y$ is the signature field, $X$ is a higher level field composed of another two variable-length fields $A$ and $B$, and $L$ indicates the length of $X$. Obviously, to obtain the length of field $Y$, we just need to parse $L$ and $X$, without identifying the detailed structure of $A$ and $B$. Examples can be found in protocols SSLv2 and SSLv3.

Therefore, we can see that the overhead of protocol parsing for length signature matching is far less than full protocol parsing. In the implementation, the parser should be capable of parsing all the fields, while the real extent of parsing is determined according to the provided signatures.

## VIII. 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 27 GB of Internet traffic plus 123 GB of email SPAM. To test LESG's effectiveness, we used completely different datasets for LESG signature generation

TABLE IV
SUMMARY OF WORMS

| Protocol | DNS | SNMP | $SNMP_t$ | $FTP_1$ | $FTP_2$ | $FTP_3$ | 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) |
| vulnerable field length | fixed | variable | variable | variable | variable | variable | variable | variable |

(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 false positive rate and worm traffic to test false negative rate. For attack resilience testing, we tested the performance of our system under deliberate noise injection attack with different noise ratios.

*1) Polymorphic Worm Workload:* To evaluate our LESG system, we created eight polymorphic worms based on real-world vulnerabilities and exploits from `security-focus.com`, as shown in Table IV, by modifying the real exploits to make them polymorphic. The eight worms use six different protocols, DNS, SNMPv1, $\text{SNMPv1}_{\text{trap}}$, 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 nonvulnerable simple fields, we randomly chose the lengths with the same distribution as those in normal traffic; for the vulnerable simple fields, the lengths in the original exploit codes are mostly much longer than the buffer lengths, 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 IV); moreover, for some exploits that have a rigid exploit condition about field length, we kept that fixed length. In Table IV, the row titled "vulnerable field length" specifies whether the overflowing field length is fixed or not. For the vulnerability for which we cannot find the ground truth by literature searching, 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.

TABLE V
DATASET SUMMARY FOR EVALUATION

| Number of Fields | Normal pool | | | Evaluation dataset | | |
|---|---|---|---|---|---|---|
| | Bytes | Flows | Hours | Bytes | Flows | Hours |
| DNS: 15 | 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: 13 | 840M | 210K | 24 | 122G | 31M | 744 |
| HTTP: 7 | 2G | 77K | 7 | 11G | 360K | 40 |

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

**HTTP worm**. It exploits the IIS vulnerability used by a 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 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 any form of sampling. We used another 123 GB SPAM dataset from some open relay servers at a research organization in the U.S. for the SMTP. The datasets are summarized in Table V. Since an 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 separated 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 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 they are normal traffic.

*3) Experiment Settings:* In the Step-1 algorithm, we conservatively set $\text{FP}_0$ to be 0.1%, i.e., the server should be able to handle 1000 normal requests without crashing (buffer overflow triggered). We believe this is reasonable for most popular implementations of any protocol. We choose $\text{COV}_0 = 1\%$, a conservatively small value, because attackers may inject noise into the suspicious pool.

The score function in Step-2 is $\text{Score}(\text{COV}, \text{FP}) = (1/\log\text{FP} + 1) * \text{COV}$, which provides a good tradeoff between FP and COV, working well in practice. The score is directly proportional to COV. In Fig. 7, the three curves respectively correspond to $\text{COV} = 0.2$, 0.6 and 1. As the figure shows, the increment of score caused by changing FP from $10^{-2}$ to $10^{-3}$ is more than that of changing FP from $10^{-4}$ to $10^{-5}$. Therefore, when FP is high (e.g., $10^{-2}$), the function tends toward improving FP, while when FP is pretty low (e.g., $10^{-5}$), it tends toward improving COV.

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.

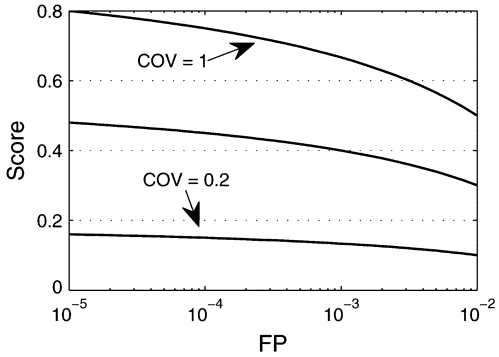



Fig. 7. Score function illustration.

TABLE VI
SIGNATURES AND ACCURACY UNDER DIFFERENT POOL SIZE AND NOISE

| 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 |
| $\text{SNMP}_t$ | (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 | $10^{-5}$ |

### B. Signature Generation for Single Worm With Noise

We evaluated the accuracy of LESG with the presence of noise. The noise is the set of flows randomly sampled from normal traffic, and mixed with worm samples to compose the suspicious pool. We chose DNS, SNMP, $\text{SNMP}_t$, SMTP, and HTTP protocols to demonstrate the case 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 sizes 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 test the sensitivity and accuracy.

Table VI shows the range of the signatures we generated and their accuracy. Tr. FN (FP) denotes false negative (positive) rate in the training dataset, while Ev. FN (FP) denotes false negative (positive) rate in the evaluation dataset. Under all the pool sizes and noise ratios, the same signature fields are generated, and all have excellent evaluation FN and FP. Because the size of suspicious pool is limited, the signature length varies in different tests. Note that generated signature lengths are smaller than the true buffer length, since most instances of the corresponding field in normal flows are much shorter than the buffer, as discussed in Section VI-A3.

### 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 VI.

TABLE VII
RESULT OF EACH STEP FOR THE DNS WORM

| | 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 VIII
SPEED OF PROTOCOL PARSING AND SIGNATURE GENERATION

| | Normal pool (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 |
| $\text{SNMP}_t$ | 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 |

### D. Evaluation of Different Stages of the LESG Algorithm

The LESG algorithm has three steps, and we evaluated the effectiveness of each step. Table VII illustrates the result of each step for the DNS worm with a suspicious pool of size 100 and a noise ratio 50%. Table VII shows that the false positive rate is largely decreased by refining each signature length in Step-2. And according to the ground truth shown in Table IV, we can see that in Step-3, the best 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 VI. This small size requirement validates LESG's ability of reacting to worm propagation rapidly.

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

### F. Speed and Memory Consumption

We evaluated the parsing speed 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 VIII shows that the speed of signature generation is quite fast, though it 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-h +period. For the suspicious pool, since it is much smaller than the normal pool, the protocol parsing can be done quickly. Moreover, as mentioned in [47], the BINPAC compiler can be built with parallel hardware platforms, which makes it much faster.

TABLE IX
MEMORY USAGE OF THE ALGORITHM

| Normal pool size | | Suspicious pool size | | |
|---|---|---|---|---|
| | | 100 | 200 | 500 |
| DNS (15 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 |

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

### G. Performance Under Deliberate Noise Injection Attacks

As discussed in Section VI, deliberate noise inject attack is the most threatening attack toward LESG, among all the proposed attacks. Therefore, we implemented one to evaluate the attack resilience of LESG, with assumptions that: 1) attackers know all the parameters used in LESG and can tailor their attacks against LESG; 2) attackers can obtain some normal traffic samples, and then estimate field length distributions based on the traffic samples.

We implemented this attack by modifying the Lion worm of the DNS protocol. There are 15 fields in the DNS protocol. Only one field $f_B$ has to be long enough to overflow the buffer, which cannot be controlled by attackers. The attackers can use the other 14 fields to craft arbitrary noise.

In the experiment, we simulated the situation in which the attacker captures the normal traffic with 100 K flows to optimize the attack. To make sure the attacker's normal traffic has similar length distributions as the training normal pool we used, in our experiment we randomly permutated the 320 K DNS normal flows shown in Table V, and divided them into the 100 K flow pool for attackers and the 220 K flow normal pool for the LESG system.

In the experiment, we assume all the noise is deliberately injected. We test the noise ratio from 8% to 92%, increasing 7% in each test. We use a suspicious pool of size 200. For the Lion worm, the vulnerable field signature has no false positive. The Proof of Theorem 2 shows that the false negatives should be less than $\gamma'$ portion of the suspicious pool. Therefore, at most $200 \times \gamma' - 1 = 200 \times 0.01 - 1 = 1$ false negatives can be generated. Under a given noise ratio, among the 14 fields, we search all the possible combinations and choose the optimal set of fields to increase the false positives. Then we choose one of the remaining fields to increase the false negatives.

We use another 2000 worms to test the evaluation false negatives and the 4.4M DNS flows shown in Table V to test the evaluation false positives. The results are shown in Figs. 8 and 9. The training false negatives and the evaluation false negatives are very close, so the two lines coincide with each other. From the results we know that even with 90% deliberately injected noise, our system still only has a false negative rate of 6.3% and a false positive rate of 0.14%. This indicates that it is quite hard for attackers to increase the false positives. The reason of this phenomenon is that the worst case bound happens when each field can introduce false positives in a mutually exclusive way, which seldom happens and is hard to achieve in practice.

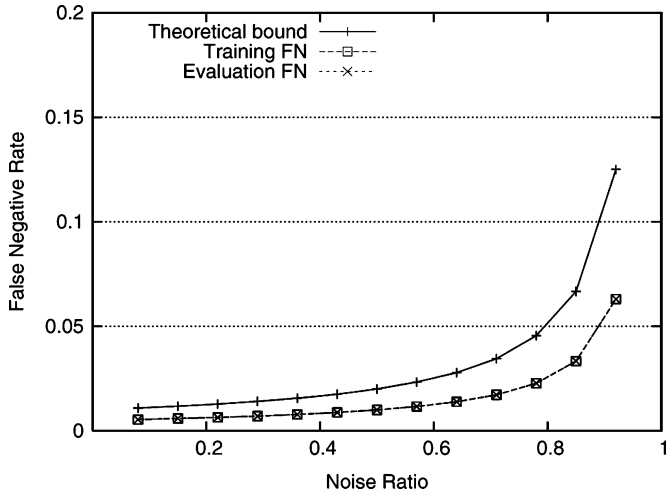



Fig. 8. False negative rate with different noise ratio.

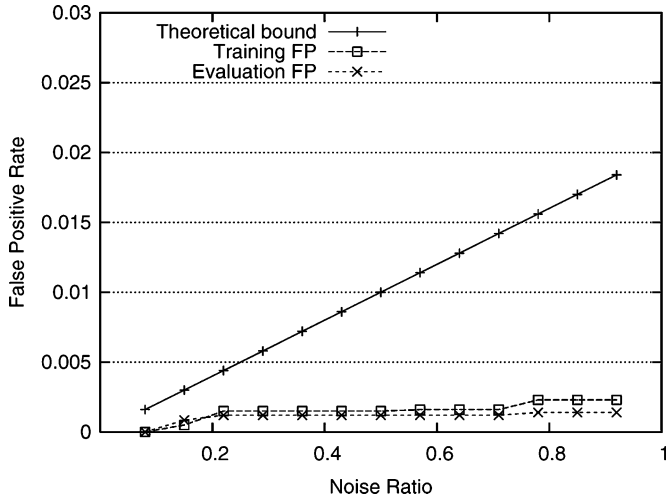



Fig. 9. False positive rate with different noise ratio.

### H. Length-Based Signature in Regular Expression Format

We evaluated the prevalence and accuracy of L-RE signatures (defined in Section VII) for both text-based protocols and binary protocols.

We analyzed six text-based protocols, including FTP, HTTP, SMTP, POP3, IRC, and IDENT. As discussed in Section VII-A, field-specific $\mathrm{RE}_i, i \in \{p, s\}$, never produces false negatives. Moveover, among the six protocols, FTP (control channel), IRC and IDENT are *strictly line-based*, thus also never have false positives produced by field-specific $\mathrm{RE}_i$, as discussed in Section VII-A. However, for the other three protocols, besides the *field-related lines* (defined in Section VII-A), there are also many types of *message bodies* (e.g., transferred files) which might match certain L-RE signature whose field ID belongs to $E_l$ (defined in Section VII-A) and generate false positives.

We investigated all the fields belonging to the $E_l$ of protocol HTTP and SMTP. The total number of fields is in column $|E_l|$ of Table X. Recall that for any field $f \in E_l$ of text-based protocols, protocol-defined suffix $\mathrm{RE}'_s$ is "\r\n" invariantly, and protocol-defined prefix has the form of $\mathrm{RE}'_p = ``\backslash \mathrm{r} \backslash \mathrm{n} \mathrm{RE}^0_p"$. Examples of $\mathrm{RE}^0_p$ could be "/GET\s/i" for HTTP field `GET-REQUEST`, and "/RCPT TO\s/i" for SMTP field `RCPT-TO-COM-`

TABLE X
FALSE POSITIVES OF L-RE SIGNATURES FOR TEXT-BASED PROTOCOLS

| Protocol | Trace Size | $\|E_l\|$ | $RE'_p$ Matches | False Matches |
|---|---|---|---|---|
| HTTP | 14GB | 46 | 3.5M | 0 |
| SMTP | 20GB | 12 | 6.9M | 0 |

TABLE XI
FALSE POSITIVES OF L-RE SIGNATURES FOR BINARY PROTOCOLS

| Protocol | DNS | SNMP | $\text{SNMP}_t$ |
|---|---|---|---|
| Trace Size | 1.1GB | 290MB | 88MB |
| RE type | *vulnerability-specific* | *field-specific* | *field-specific* |
| $RE_p$ | ^.{4}\x00.\x00{5}\x01[3] | \x30.(..)?\x06.\x2B\x06 | |
| $RE_s$ | \x00\x00\xFA\x00\xFF | \x30.(..)?\x06.\x2B\x06 or End of PDU | |
| $RE_i$ Matches | 0 | 3.4M | 210K |
| FP | 0 | 0 | 0 |

MAND. As discussed in Section VII-A, for a length-based signature whose field ID is $f$, its corresponding possible L-RE signature $S_r = \langle \text{RE}_p, \text{RE}_s, L \rangle$, which we term as field $f$'s *possible L-RE signature*, will have $\text{RE}_p = \text{RE}'_p$ and $\text{RE}_s = \text{RE}'_s$. In this evaluation, we matched $\text{RE}'_p$ of each investigated field against the normal traffic traces, and found that for each $\text{RE}'_p$, the matches are all in field-related lines (column "$\text{RE}'_p$ Matches" presents the total number of the matches of all fields), i.e., no match is in message bodies (column "False Matches"). Therefore, each field's possible L-RE signatures will also have no match in the message bodies, namely, no RE-caused false positive.

For binary protocols, we analyzed six vulnerabilities, including the three (DNS, SNMP and $\text{SNMP}_t$) discussed in this section before, one of NTP (Bugtraq ID 2540), one of SSLv2 (Bugtraq ID 5363, attacked by worm Slapper), and one of DCOM RPC (Bugtraq ID 8205, attacked by worm Blaster). Among them, the one of SSLv2 has no L-RE signature since the SSLv2 message is encrypted; the one of RPC may have exploit-specific $\text{RE}_i, i \in \{p, s\}$, for an incompletely polymorphic worm; and the other four vulnerabilities all have field-specific or vulnerability-specific $\text{RE}_i$, therefore no RE-caused false negative can be produced. In other words, at least 4 out of these 6 vulnerabilities have L-RE signatures. Since we chose these vulnerabilities randomly, they validate the prevalence of L-RE signature to some extent.

We generated $\text{RE}_p$ and $\text{RE}_s$ for the three vulnerabilities of DNS, SNMP and $\text{SNMP}_t$. We matched "$\text{RE}_p|\text{RE}_s$" against the normal traces, and the total number of matches is presented in row "$\text{RE}_i$ Matches" of Table XI. Note that since $\text{RE}_p$ and $\text{RE}_s$ of the vulnerability of DNS are vulnerability-specific, it is reasonable that there is no match in normal flows. For the vulnerabilities of SNMP and $\text{SNMP}_t$, we matched L-RE signature $\langle \text{RE}_p, \text{RE}_s, L \rangle$ ($L$ is generated in Section VIII-B) against the traces, and found no RE-caused false positive.

### I. Vulnerability Coverage

As discussed in Section VI-A3, our approach works well only for the case that a protocol field must be longer than normal lengths to perform a successful exploit. To find out the percentage of vulnerabilities we can handle, we made an investigation on real-world vulnerabilities. In securityfocus.com, we searched for the vulnerabilities whose headlines contain keyword "overflow" and another keyword which is one of some widely used protocols (including DNS, SNMP, NTP, SSL, HTTP, FTP, and SMTP) or corresponding applications (including bind, ntpd, apache, iis, *httpd, *ftpd, and sendmail). Among the total 385 search results, we randomly selected 172 for checking. Since the available references of many vulnerabilities are not detailed enough, finally we achieved conclusions for 83 vulnerabilities: only 4 (less than 5%) out of them have no accurate length-based signature. The four special cases include a buffer filled with cumulative PDUs, and some complicated heap overflows. Therefore, at least to some extent, this investigation proves that our approach has a high coverage of the buffer overflow vulnerabilities.

## IX. CONCLUSION

In this paper, we propose a novel network-based automatic worm signature generation method that generates length-based signatures for buffer overflow worms. This is the first attempt to generate vulnerability-driven signatures at network level. We build a field hierarchy model, and formally define the length-based signature generation problem based on it. The algorithm designed to solve that problem has good accuracy bound even under deliberate noise injection attacks. We also discuss the schemes to speed up signature matching. We further show that our approach is fast and accurate with experiments, using real-world vulnerabilities and network traffic.

## REFERENCES

[1] Z. Liang and R. Sekar, "Automatic generation of buffer overflow attack signatures: An approach based on program behavior models," in *Proc. Comput. Security Appl. Conf. (ACSAC)*, 2005, pp. 215–224.
[2] M. Roesch, "Snort: The lightweight network intrusion detection system," 2001 [Online]. Available: http://www.snort.org/
[3] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Comput. Netw.*, vol. 31, pp. 2435–2463, 1999.
[4] C. Kreibich and J. Crowcroft, "Honeycomb: Creating intrusion detection signatures using honeypots," *SIGCOMM Comput. Commun. Rev.*, vol. 34, pp. 51–56, 2004.
[5] S. Singh *et al.*, "Automated worm fingerprinting," in *Proc. USENIX OSDI*, 2004, p. 4.
[6] H. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," in *Proc. USENIX Security Symp.*, 2004, pp. 271–286.
[7] Z. Li, M. Sanghi, Y. Chen, M. Kao, and B. Chavez, "Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience," in *Proc. IEEE S&P*, 2006, pp. 33–47.
[8] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," in *Proc. IEEE S&P*, 2005, pp. 226–241.
[9] Y. Tang and S. Chen, "Defending against Internet worms: A signature-based approach," in *Proc. IEEE INFOCOM*, 2003, pp. 1384–1394.
[10] J. Newsome and D. Song, "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software," presented at the NDSS, 2005.
[11] J. R. Crandall, Z. Su, and S. F. Wu, "On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits," in *Proc. ACM CCS*, 2005, pp. 235–248.
[12] R. Perdisci *et al.*, "Misleading worm signature generators using deliberate noise injection," in *Proc. IEEE S&P*, 2006, pp. 17–31.
[13] J. Newsome, B. Karp, and D. Song, "Paragraph: Thwarting signature learning by training maliciously," in *Proc. RAID*, 2006, pp. 81–105.
[14] S. P. Chuang and A. K. Mok, "Allergy attack against automatic signature generation," in *Proc. RAID*, 2006, pp. 61–80.
[15] P. Fogla *et al.*, "Polymorphic blending attacks," in *Proc. USENIX Security Symp.*, 2006, Article No. 17.
[16] S. Venkataraman, A. Blum, and D. Song, "Limits of learning-based signature generation with adversaries," presented at the NDSS, 2008.

[17] Z. Liang and R. Sekar, "Fast and automated generation of attack signatures: A basis for building self-protecting servers," in *Proc. ACM CCS*, 2005, pp. 213–222.
[18] X. Wang *et al.*, "Packet vaccine: Black-box exploit detection and signature generation," in *Proc. ACM CCS*, 2006, pp. 37–46.
[19] D. Brumley *et al.*, "Towards automatic generation of vulnerability-based signatures," in *Proc. IEEE Security Priv. Symp.*, 2006, pp. 2–16.
[20] M. Cost *et al.*, "Vigilante: End-to-end containment of Internet worms," in *Proc. ACM SOSP*, 2005, pp. 133–147.
[21] Packeteer, "Solutions for malicious applications," [Online]. Available: http://www.packeteer.com/prod-sol/solutions/dos.cfm
[22] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *Proc. RAID*, 2004, pp. 203–222.
[23] K. Wang, G. Cretu, and S. J. Stolfo, "Anomalous payload-based worm detection and signature generation," in *Proc. RAID*, 2005, pp. 227–246.
[24] R. Vargiya and P. Chan, "Boundary detection in tokenizing network application payload for anomaly detection," in *Proc. DMSEC*, 2003, pp. 50–59.
[25] V. Yegneswaran *et al.*, "An architecture for generating semantic-aware signatures," in *Proc. USENIX Security Symp.*, 2005, p. 7.
[26] C. Kruegel *et al.*, "Polymorphic worm detection using structural information of executables," in *Proc. RAID*, 2005, pp. 207–226.
[27] T. Toth and C. Kruegel, "Accurate buffer overflow detection via abstract payload execution," in *Proc. RAID*, 2002, pp. 274–291.
[28] R. Chinchani and E. Berg, "A fast static analysis approach to detect exploit code inside network flows," in *Proc. RAID*, 2005.
[29] F. Hsu and T. Chiueh, "CTCP: A centralized TCP/IP architecture for networking security," in *Proc. Comput. Security Appl. Conf. (ACSAC)*, 2004, pp. 335–344.
[30] X. Wang *et al.*, "Sigfree: A signature-free buffer overflow attack blocker," in *Proc. USENIX Security Symp.*, 2006, Article No. 16.
[31] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos, "Emulation-based detection of non-self-contained polymorphic shellcode," in *Proc. RAID*, 2007, pp. 87–106.
[32] P. Fogla and W. Lee, "Evading network anomaly detection systems: formal reasoning and practical techniques," in *Proc. CCS*, 2006, pp. 59–68.
[33] V. Yegneswaran, P. Barford, and D. Plonka, "On the design and use of Internet sinks for network abuse monitoring," in *Proc. RAID*, 2004, pp. 146–165.
[34] M. Bailey *et al.*, "The internet motion sensor: A distributed blackhole monitoring system," presented at the NDSS, 2005.
[35] 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.
[36] Y. Gao, Z. Li, and Y. Chen, "A DoS resilient flow-level intrusion detection approach for high-speed networks," in *Proc. ICDCS*, 2006, Article No. 39.
[37] R. Pang *et al.*, "BINPAC: A yacc for writing application protocol parsers," in *Proc. ACM/USENIX IMC*, 2006, pp. 289–300.
[38] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: automatic extraction of protocol message format using dynamic binary analysis," in *Proc. ACM CCS*, 2007, pp. 317–329.
[39] G. Wondracek, P. M. Comparetti, C. Kruegel, and E. Kirda, "Automatic network protocol analysis," presented at the NDSS, 2008.
[40] Z. Lin, X. Jiang, D. Xu, and X. Zhang, "Automatic protocol format reverse engineering through conectect-aware monitored execution," presented at the NDSS, 2008.
[41] S. A. Vinterbo, "Maximum k-intersection, edge labeled multigraph max capacity k-path, and max factor k-GCD are all NP-hard," Decision Syst. Group, Harvard Med. School, Tech. Rep., 2002.
[42] Z. Li, L. Wang, Y. Chen, and Z. Fu, "Network-based and attack resilient length signature generation for zero-day polymorphic worms," Northwestern University, Tech. Rep. NWU-EECS-07–02, 2007.
[43] S. Staniford *et al.*, "How to own the internet in your spare time," in *Proc. 11th USENIX Security Symp.*, 2002, pp. 149–157.
[44] N. Schear, D. Albrecht, and N. Borisov, "High-speed matching of vulnerability signatures," in *Proc. RAID*, 2008, pp. 155–174.
[45] Radware Inc., "Introducing 1000X security switching," [Online]. Available: http://www. radware.com/content/products/application_switches/ss/default.asp
[46] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," in *Proc. ACM SIGCOMM*, 2006, pp. 339–350.
[47] V. Paxson *et al.*, "Rethinking hardware support for network analysis and intrusion prevention," in *Proc. USENIX Hot Security*, 2006, p. 11.

**Lanjia Wang** received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2003.

She is a Ph.D. degree candidate with the Department of Electronic Engineering, Tsinghua University. Her research interests mainly focus on network security, especially intrusion detection and malicious code analysis.

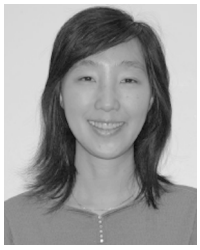
**Zhichun Li** received the B.S. degree in physics and the M.S. degree in computer science from Tsinghua University, Beijing, China.

He is a Ph.D. degree candidate with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL. His research interests are on network security, network measurement and monitoring, data streaming, and P2P systems.

**Yan Chen** received the Ph.D. degree in computer science from the University of California at Berkeley in 2003.

He is an Assistant Professor with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL. His research interests include network measurement, monitoring and security, and P2P systems.

Dr. Chen won the Department of Energy (DOE) Early CAREER award in 2005, the Air Force of Scientific Research (AFOSR) Young Investigator Award in 2007, and the Microsoft Trustworthy Computing Awards in 2004 and 2005 with his colleagues.



**Zhi (Judy) Fu** received the Ph.D. degree in computer science from North Carolina State University, Raleigh, in 2001.

She is a Principal Staff Researcher with Motorola Research Laboratories, Schaumburg, IL, where her work focuses on wireless network security research. Her research interests include wireless AAA, protocol vulnerability analysis, security policy, and intrusion detection.

**Xing Li** received the Ph.D. degree in electrical engineering from Drexel University, Philadelphia, PA, in 1989.

He is a Full Professor with the Department of Electronic Engineering, Tsinghua University, Beijing, China. He is the Deputy Director of the China Education and Research Network (CERNET) Center. His research interests include statistical signal processing, multimedia communication, and compute networks.

Dr. Li is a Fellow of China Communication Institute and a Senior Member of the China Electronic Institute.