LISABETH: Automated Content-Based Signature Generator for Zero-day Polymorphic Worms

Lorenzo Cavallaro, Andrea Lanzi, Luca Mayer, and Mattia Monga
Dip. di Informatica e Comunicazione
Università degli Studi di Milano, Italy
{sullivan,andrew,luca,monga}@security.dico.unimi.it

ABSTRACT

Modern worms can spread so quickly that any countermeasure based on human reaction might not be fast enough. Recent research has focused on devising algorithms to automatically produce signature for polymorphic worms, required by Intrusion Detection Systems. However, polymorphic worms are more complex than non-mutating ones as they also require the identification of mutated instances. To this end, we propose LISABETH*, our improved version of Hamsa, an automated content-based signature generation system for polymorphic worms that uses invariant bytes analysis of network traffic content. We show an unknown attack to Hamsa's signature generator that is contrasted by* LISABETH*. Moreover, we show that our approach is able to generally improve the resilience to poisoning attacks as supported by our experiments with synthetic polymorphic worms.*

Categories and Subject Descriptors

K.6.5 [**Computing Milieux**]: Security and Protection—*Invasive software*

General Terms

Security

1. INTRODUCTION

A *worm* program is an independently replicating and autonomous infection agent, capable of seeking out new hosts and infecting them via the network [13]. Because of evermore pervasive Internet connections and software monoculture, worms with their typical scan/compromise/replicate pattern can infect all the vulnerable hosts in a matter of few hours or even minutes [21]. To contrast such a threat, the research community has proposed different kind of Intrusion Detection Systems [20] (IDSs). For example, a misuse-based IDS, deployed at the gateway between its network and the Internet, may filter incoming and outcoming network traffic for known *signatures* that correspond to malicious flows samples [14, 24]. In the past, signatures used by IDS have been generated manually with the supervision of security experts which studied network traces and inferred worms signatures. However, in the last years, the frequency and virulence of worms outbreaks increased dramatically thanks to their improved efficiency and evasion methods, and became well-understood that signatures generation processes that involve human labor were not feasible [5, 6] anymore.

To face the low pace of this approach in signatures generation and to speed up this task, automatic signature generation systems have been proposed in the past and recent years. Systems like Honeycomb [4], Autograph [8], and EarlyBird [22] monitor network traffic to identify new Internet worms and produce signatures for them. All these systems perform a so-called *content-based* analysis, i.e., they produce signatures by extracting common recurrent and invariant byte patterns across different suspicious flows. The main weakness of these generation systems is that they require that at least a single pattern of a significant length has to belong to different network streams. Unfortunately, *polymorphic* Internet worms [18, 17], probably the next generation of worms, are able to change their binary representation during the spreading process. By using polymorphism techniques[1], like self-encryption or semantic-preserving techniques [3, 7, 15], these worms are able to modify their payload and so the bytes sequence sent on the network, thus evading systems using single substring signatures.

To face polymorphism, recent models, like Polygraph [11] and its improvement Hamsa [25], use novel techniques to identify different variations of the same polymorphic worm. Although these systems, in normal conditions, can identify different instances of a polymorphic worm in an efficient and effective ways, new studies demonstrate the presence of some vulnerabilities exploitable by a set of old and new attacks. These attacks allow instances of polymorphic worms to evade detection and containment systems by misleading the signature generation process by injecting crafted packets into normal traffic [19, 1, 12]. To give a concrete example of these weaknesses, we designed a new attack which could be employed by an attacker to evade Hamsa [25], the state-of-the-art model. In particular, this paper makes the following contributions:

1. We devised a new suspicious poisoning pool attack against Hamsa's signature generation approach.

[1]In this paper we refer to both polymorphism and metamorphism techniques as polymorphism. Thus, we consider worms where cryptography and obfuscation techniques may be applied on.

SESS'08, May 17–18, 2008, Leipzig, Germany.

2. We proposed and implemented LISABETH[2], a new *automated content-based signature generator* model that improves (i) resilience to innocuous and suspicious pool poisoning attacks, and (ii) signature generation performance by using a new signature generation algorithm; our experiments show the our prototype is as effective as the state-of-the-art models and LISABETH, under the invariant presence assumption, is resilient to all suspicious poisoning pool attacks.

We proceed in the remainder of the paper as follows. In §2, we introduce the anatomy of polymorphic worms, provide evidence of the existence of invariant payload bytes in samples used in real worms exploits and motivate the class of signatures used in our model. We continue in §3 by setting the context in which LISABETH will be used and stating our design goals. Next, in §4, we describe the high level architecture of our network monitor before analyzing our signatures generation algorithm in §5. We discuss possible attacks against our system in §6, experimental results in §7, related literature work in §8, and concluding remarks in §9.

2. POLYMORPHIC WORMS

To support the validity of our model, we now consider the anatomy of polymorphic worms and motivate the *invariant bytes* presence assumption. After a brief characterization of polymorphic worms structure, we show that different samples of the same worm often share some invariant content due to the constraints that an attacker has to respect to exploit a given vulnerability, as shown in [25, 11] as well. Then, after a short examination of existing signatures classes, we motivate why in LISABETH we adopt Hamsa-like signatures.

2.1 Polymorphic Worm Structure

As stated in [19, 11, 1], in a sample of polymorphic worm we can identify the following components:

Protocol framework. To infect new hosts and continue their spread, worms have to exploit a given vulnerability. This vulnerability, in many cases, is associated with a particular application code and execution path in this code. This execution path can be activated by few, or more often one, types of particular protocol request.

Exploit bytes. These bytes are used by the worm to exploit the vulnerability. They are necessary for the correct execution of the attack.

Worm body. These bytes contain instructions executed by the worm instances on new infected victims. In polymorphic worms these bytes can assume different values in each instance.

Polymorphic decryptor. The polymorphic decryptor decodes the worm body and starts its execution.

Others bytes. These bytes do not affect the successfully execution of both the worm body and exploit bytes.

Content-based signature generation approaches rely upon the presence of *invariant bytes* in some of the identified components. Some of these components, for their nature, offer high chance of finding these invariant sequences which are useful for the signature generation purpose.

[2]LISABETH IS A BETter Hamsa.

2.2 Invariant Bytes

In a polymorphic worm sample we can classify three kind of bytes: invariant, code and wildcard [11, 25, 19, 1].

Invariant bytes are those with a fixed value in every possible instance. If their value is changed, the exploit no longer works. They can be part of the protocol framework and exploit bytes but in some cases also of the worm body or the polymorphic decryptor. Such bytes are very useful in signatures generation because they are absolutely necessary for the exploit to work and their content is replicated across worm instances. *Code bytes* come from components like the worm body or decryption routine in which there are instructions to be executed. Although code section of worm samples can be subjected to polymorphism and encryption techniques, and thus they can assume different shapes in each instance, polymorphic engines are not perfect and some of these bytes can present invariant values. Lastly, *wildcard bytes* are bytes that may take any value without affect worms spreading capabilities.

Our analysis and others studies conducted in [11] and [25] demonstrate that invariant bytes presence assumption is indeed a sensible one. The idea on which Hamsa [25], Polygraph [11] and our system are based, is to capitalize this invariant bytes presence across different worms instances to characterize the worms itself.

2.3 Signature Classes for Polymorphic Worms

Signatures for polymorphic worms can be classified into two broad categories: *content-based signatures* that aim to use similarity in different instances of byte sequences to characterize a given worm, and *behavior-based signatures* that aim to characterize worms understanding the semantics of their byte sequences.

Our approach focuses on content-based signatures that allow us to treat worms as strings of bytes. In this way, we obtain a protocol independent system that does not require any final host information and that requires very short time to perform the signature generation task. Moreover, content-based systems can be easily incorporated in firewalls or network-based IDS (NIDS) because their signatures can be verified using fast signatures matching algorithms [23].

There are some different classes of content-based signatures, each of one with a different level of expressiveness [11]. The signatures generated by LISABETH are called *multi-set signatures* [25]. Multiset signatures are multi-sets of tokens, where a *token* is a sequence of bytes that recur in some network flows, and are characterized by a list of tokens each with its number of occurrences.

Formally speaking, a multi-set signature s takes the following form $\{(t_1, n_1), (t_2, n_2), \ldots, (t_k, n_k)\}$ where t_j is a token and n_j its number of occurrences.

We say that a network flow $\mathcal{G}$ *matches* the given multi-set signature s if it contains at least n_j copies of the t_j token of s, $\forall j \in [1, \ldots, k]$. If $\mathcal{A}$ is a set of flows and s a multiset signature, with $\mathcal{A}_s$ we denote the largest subset of flows in $\mathcal{A}$ that match with s.

It is important to note that this class of signatures does not consider any kind of token ordering. The invariant bytes presence assumption imposes to the attacker to use all worm invariant bytes, in all flow samples, but nothing is said about the invariants order. By introducing token ordering in signatures we make these vulnerable to *coincidental-pattern* attacks [11], and so, easy to evade by inserting spurious instances of the invariant tokens in the variant part of the worm flows misleading signatures generator about true order of the invariant tokens. Moreover, by specifying invariants occurrence numbers, it is possible to build more specific signatures than the so-called *conjunction signatures* proposed in [11], thus reducing the overall false positives rate.

3. PROBLEM STATEMENT AND SYSTEM REQUIREMENTS

As stated in the previous section, our approach is based on the observation of all the network traffic in transit across a monitoring point, such as between an edge network and the broader Internet, trying to generate multiset signatures that characterize the worms which flow samples are sent across the monitored network. While we believe that a distributed approach will be more effective, in this work we consider only a single centralized strategy.

Like Hamsa [25], our system analyzes network traffic, defragments network packets into contiguous flows of bytes, classifies re-assembled flows in *suspicious*, presumably sent by a worm instance, and *innocuous*, probably belonging to a common application, and tries to extract signatures that characterize flows classified as suspicious.

The main issue in which we are interested in is generation of signatures by examining the suspicious and innocuous flows pools. Flows reassembly and traffic normalization at a monitor level [24] and identification of anomalous or suspicious traffic with more or less accurate techniques [8, 4, 10] are typical topics in IDS design and development and we do not cope with them here.

We only assume that the flow classifier is imperfect and may mis-classify innocuous flows as suspicious and viceversa. Moreover, the classifier is also not able to discriminate flows depending on the worm who generated them.

As said before, the approach leverages on the hypothesis that every worm has its invariants set and that an attacker must insert, in all worm samples, all the invariants bytes in order for the attacker to be successful. This means that, to allow a rapid spread of the worm, there will be a lot of flows in which all the invariant bytes occur. However, some of the same invariant bytes could appear also in innocuous flows or it will be quite simple for an attacker to inject designed noise (like bogus worms) in suspicious flows pool or fake invariants in worm samples in order to mislead generation of the signatures. These evasion techniques are known as *poisoning attacks.*

As we will see in Section 6, this issue is very important in design phase of new systems. Prior generation models, like Hamsa [25] and Polygraph [11], even if equipped with signatures generators effective also in presence of high noise ratios, will be led astray in presence of some ad-hoc forged traffic because of an incorrect approach to the issue.

In conclusion, given a suspicious traffic pool $\mathcal{M}$ and an innocuous traffic pool $\mathcal{N}$, our goal is to find a set $\mathcal{S}$ of signatures s_i each of which covers many flows in $\mathcal{M}$ but not so many in $\mathcal{N}$. So, the false positives rate $FP_{s_i} = \frac{|\mathcal{N}_{s_i}|}{|\mathcal{N}|}$ of each s_i must be low, while the coverage (true positive rate) $COV_{s_i} = \frac{|\mathcal{M}_{s_i}|}{|\mathcal{M}|}$ high.

4. HIGH LEVEL ARCHITECTURE

The high level architecture of Lisabeth is very similar to Hamsa's, from which it is derived from. In the following, we remind the key ideas. For an exhaustive description of common phases and algorithms, the reader should refer to [25]. Our new signature generation algorithm is, instead, described in §5.

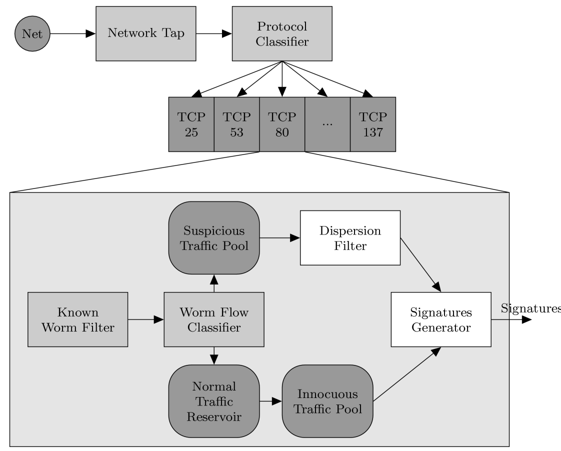

Figure 1: High level architecture of the new model. (In white the differences with Hamsa)

4.1 Global Overview of the System

Figure 1 depicts the architecture of Lisabeth, where the components that differ from Hamsa are depicted in white. We first need to *sniff* network traffic, reassemble flows of network packets and *classify flows* in terms of different protocols (TCP/UDP) and port numbers. For each (*port*,-*protocol*) pair, we filter out *known worm* samples and then, using the *worm flow classifier*, we separate flows into suspicious ($\mathcal{M}$) and innocuous ($\mathcal{N}$) pools.

The next step concerns the selection of suspicious and innocuous flows to send to the signatures generator and from which signatures will be created.

To avoid poisoning attacks from a single attacker, i.e., with network packets coming from a single network address or at most from a limited network address set, we propose to use a **dispersion filter**. The goal of this filter is to perform a dispersion analysis on suspicious flows in order to send to the generator a well dispersed set of flows. By using this technique only fewer flows for each source address are sent to the signature generator. Thus, to be successful, worm instances that want to perform *suspicious pool poisoning* attacks against our system *must* synchronize themselves with respect to the features to use in flows. This, as a consequence, makes the attack harder to carry out.

On the innocuous flows pool, instead, the idea is to use a good selection policy to decide which flows should be employed in the signature generation. Even if, as we will see in §6, our system is less sensible to *innocuous pool poisoning* attacks than Hamsa, we suggest the use of a dispersion-based flow selection policy also on the innocuous pool.

The selected suspicious and innocuous flows are given as input to the *signatures generator* which generates signatures as described in the following section.

4.2 Signature Generator

Unlike Hamsa, the only assumption Lisabeth is based on

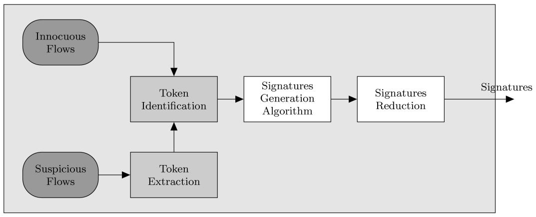

Figure 2: Signature generator architecture. (In white differences with Hamsa)

is that a true worm flow *must* contain all true invariants I_i of the invariant set $\mathcal{I}$ and that, in order to have a rapid spread, the worm sends worm samples across the network.

As it is possible to see in Figure 2, the first operation performed on the suspicious flows pool is *token extraction*. In this phase we find all sequences of at least ℓ bytes that occur in more than λ fraction of the suspicious pool. The constraint on sequences length is required to ignore too small tokens, while λ is used to take into account only those that occur in several flows.

To speed up the algorithm's execution time, all the extracted tokens are then **identified** in each innocuous flow, and all flows, innocuous and suspicious, are converted in sequences of tokens discarding all bytes sequences not included in the extracted tokens set.

Flows so obtained are then sent to the **signature generation algorithm** that, as we will see in §5, creates the required signatures.

The last phase performs **signatures reduction** on returned signatures to remove all tokens that, always appearing as subtokens of others ones, are not required to enhance signatures specificity.

5. SIGNATURE GENERATION ALGORITHM

Given suspicious ($\mathcal{M}$) and innocuous ($\mathcal{N}$) flows pools, the aim of the signature generation algorithm is to find a set $\mathcal{S}$ of signatures s_i such that $FP_{s_i} \leq FP_{max}$ and $COV_{s_i} \geq COV_{min}$. To this end Hamsa adopts a purely greedy approach. However, building the most specific signature including all possible invariants giving a good coverage of $\mathcal{M}$ without having any knowledge of the nature, real or fake, of the invariants or using a greedy algorithm with a restricted view of global situation without having a global overview of all possible signatures may weaken the strength of the detection, as we discuss in Section 6. To avoid generation of redundant signatures, we enforce an additional constraint in our algorithm:

$$s_i \in \mathcal{S} \Leftrightarrow \nexists s_j \in \mathcal{S} \mid \mathcal{M}_{s_i} \subseteq \mathcal{M}_{s_j}$$

The idea is to create all the signatures that match a considerable fraction of the suspicious pool, while avoiding the addition of new tokens to a partial signature when it has already an acceptable false positive rate (i.e., a rate less than FP_{max}).

In this way, we can have more signatures per worm but we have surely at least one specific enough signature containing *only* a subset of $\mathcal{I}$. The value used for FP_{max} may be smaller than the value used in Hamsa for the shortest signatures, i.e., composed by only one token, and so the maximum false positives rate, accepted for a single signature, is lower.

The generation of all the possible subsets of the extracted tokens and subsequent check of the given constraints would require a non-negligible computational effort, so another aim of the proposed algorithm is to avoid, whenever possible, to generate redundant or useless long signatures.

As we can easily think, for any given token there may be more occurrences of it in a single flow. To avoid worrying about this problem, we assign to each pair ($token$,$number\ of\ occurrence$) an unique identifier. In this way we consider these occurrences as different tokens. It is important to note that the same occurrence of the same token in different flows will have the same identifier. All these matching information between identifiers and related ($token, number\ of\ occurrence$) pairs are stored in an appropriate data structure, called TM, for subsequent use.

Once this identifier is assigned, for each of the aforementioned pair, we build a list of all suspicious flows in which it occurs, create a partial signature with that pair and related flows list and insert this new partial signature into the partial signatures set PS.

To avoid generation of redundant signatures caused by the high number of tokens that appear always as subtokens of other ones, we remove these subtokens from PS and take them into account at the end of the signature generation algorithm. We say that t_1 is a subtoken of t_2 if $t_1 \neq t_2$, t_1 is substring of t_2 and the occurrence number considered for t_1 is the same of that considered for t_2. Then, if a token t_1 occurs as a subtoken of t_2 we delete partial signature containing t_1 and add t_1 in the subtokens list of t_2 stored in subtokens data structure ST.

Once the partial signature set is build, our algorithm proceeds iteratively. First it evaluates the false positives rate of all the available partial signatures. Those with a value low enough are inserted into $\mathcal{S}$ and deleted from PS. The remaining signatures are then merged into each others and if each new partial signature has not a good coverage of $\mathcal{M}$, it is discarded along with those of the prior iteration. Coverage evaluation is simple and fast: the number of covered flows is given by the intersection between flows lists of the two partial signatures merged together. The merging of two partial signatures is performed only if the new one has just one more token than the two from which it is obtained.

Finally, iterations are stopped when the partial signature set is empty. Before returning it, to each generated signature are added the ignored subtokens of each included supertoken.

Algorithm 1 describes in detail the generation algorithm developed to address the above requirements. The algorithm relies on the following methods' definitions:

getTokenList() Returns the list of extracted tokens and, for each of them, the multi-set of flows in which it appears.

maxOcc(t) Returns the maximum number of occurrences of the token t in a single flow, considering all the suspicious flows in which the extracted token occurs.

genNewId() Generates a new unambiguous identifier.

checkIfSubT(t, i) Checks if the occurrence i of token t occurs as a subtoken of some supertoken.

findSuperToken(t, i) Finds supertoken identifier for an occurrence i of the token t.

Algorithm 1 Signature generation algorithm

Input: $\mathcal{M}$, $\mathcal{N}$, FP_{max} and COV_{min}
Output: Signatures set $\mathcal{S}$ for worms in $\mathcal{M}$

```
TM = PS = ST = S = ∅
tokenList = M.getTokenList()
tokenList.sort() {by descending length}
for all t ∈ tokenList do
  for i = 1 to tokenList.maxOcc(t) do
    id = genNewId()
    if PS.checkIfSubT(t, i) then
      ST.append(PS.findSuperT(t, i), id)
    else
      PS.append(genNewId(), id,-
      tokenList.getFlowList(t, i))
    end if
    TM.append(id, t, i)
  end for
end for
for all e ∈ PS do
  if PS.calcCov(e, M) < COV_min then
    PS.delete(e)
  end if
end for
signLen = 1
while PS.isNotEmpty() do
  signLen = signLen + 1
  for all e ∈ PS do
    if calcFP(e, N) < FP_max then
      S.append(newSign(e, TM, ST))
      PS.delete(e)
    end if
  end for
  for all e ∈ PS do
    for all f ∈ PS ∧ f.id > e.id do
      tmp = merge(e, f)
      if tmp.tokenNum() = signLen then
        if calcCov(tmp, M) ≥ COV_min then
          PS.append(genNewId(), tmp.id, tmp.flow)
        end if
      end if
    end for
    PS.delete(e)
  end for
end while
return S
```

getFlowList(t, i) Returns the list of flows in which the occurrence i of token t occurs.

calcCov(e, P), calcFP(e, P) Return, respectively, the coverage and false positives of an element e on pool P.

merge(e, f) Returns a new element that contains the union of the tokens e and f and the intersection of their covered flows.

tokenNum() Returns the number of elements included into the token set on which is called.

newSign(e, TM, TS) Generates a signature containing tokens of e, all their subtokens, suggested by TS, and substitutes identifier of each token with the associated string, following TM hints.

6. ATTACK ANALYSIS

Although signature generation systems like Hamsa and Polygraph are able to build good signatures even in the presence of random noise, their behavior is not so accurate if the analyzed flows are provided by a malicious user that attempts to mislead worm signatures generator with forged invariants. In particular, three main potential adversary capabilities [12] might lead an attacker to achieve the desired outcome in systems based on an initial classifier:

Target feature manipulation. The adversary manipulates some characterizing features, like the worm code or the protocol framework bytes, in worm samples. There are many techniques to minimize or obfuscate required features or to include additional spurious features into worm samples to mislead signatures generator.

Suspicious pool poisoning. The adversary places some non-worm samples inside the suspicious pool. These samples are specially constructed to mislead the signatures generator.

Innocuous pool poisoning. Similarly, the adversary places specially crafted worm samples inside the innocuous pool to mislead signatures generation.

Systems like Hamsa and Polygraph suffer of some of these attacks [12, 19, 1]. As Hamsa is an improvement over Polygraph, in the following, we focus our discussion only on Hamsa.

The greedy approach used in Hamsa's signatures generation algorithm, with its incremental generation of partial signature, can be led to build useless signatures, thus making these unable to match any more actual worm samples. In fact, the greedy algorithm proceeds iteratively by selecting at every iteration the token that, added to previous ones, gives the best signature. Doing so, Hamsa creates signatures of incremental length, obtaining each of them by adding a token to the signature generated in the previous iteration.

The first signature contains only the token that maximizes COV rate within those offering a FP rate less than a given FP_{max} rate for a signature of that length. At each iteration, Hamsa's generation algorithm adds to the previous selected signatures the token with the FP rate lesser than a threshold with the maximum COV rate. When the maximum length for a signature is reached, Hamsa selects the best one by evaluating a score for each signature. This score takes into account FP and COV rate and signature's length. The one with the higher score is then selected and returned as signature for the given input. This approach presents some weaknesses.

Let $\mathcal{W}$ denote a worm and $\mathcal{I} = \{I_a, I_b, I_c, \ldots, I_x\}$ its invariant set. An attacker could try to introduce some fake invariants $\mathcal{F} = \{F_1, F_2, F_3, \ldots, F_y\}$, i.e., tokens found in suspicious flows but not really required by the exploit. In order to assure that Hamsa considers only fake invariants (and neglecting real ones) is enough that elements of $\mathcal{F}$ are forged according to the constraint

$$FP_{\{F_1,F_2,F_3,\ldots,F_i\}} \leq u(i) \quad \forall i \in [1..y]$$

where $u(i)$ is the function Hamsa uses to determine if the false positive rate of a given signature is low enough.

The above constraint can be trivially respected. Given $\mathcal{I}$ and $\mathcal{F}$, an instance of $\mathcal{W}$ will generate two class of samples: worm samples $\mathcal{W}_1^{IF}, \mathcal{W}_2^{IF}, \ldots, \mathcal{W}_n^{IF}$ that contain real and fake invariants and non-worm samples $\mathcal{W}_1^{F}, \mathcal{W}_2^{F}, \ldots, \mathcal{W}_j^{F}$ that contain only fake invariants and so are not real working worms.

To assure attack achievement, an attacker must send worm and non-worm samples to the victim and drives the initial classifier to classify these as suspicious. To do so, it is sufficient to hold an anomalous behavior, where what anomalous means depends on the initial classifier type.

Suppose that $\mathcal{W}$ sends n samples of $\mathcal{W}^{IF}$ and j of $\mathcal{W}^{F}$ with $n + j \geq \lambda$, where λ is the minimum number of tokens occurrences in suspicious flows pool required to be considered in signatures generation.

The token extraction procedure will extract all fake invariant tokens F_i and, if $n \geq \lambda$, all the real invariants I_i.

In the signature generation algorithm, the first chosen token will be a fake invariant because there is at least one fake invariant, i.e., F_1, with false positive rate less than $u(1)$ that occurs more than any other real invariant I_i. Similarly, in subsequent iterations, the algorithm includes in the temporary signature a fake invariant because there is always a F_i that, added to the previous ones, respects $u(i)$ value and, with the others, occurs more times than any other true invariant.

To guarantee that the order in which fake invariants are chosen by the algorithm follows the predicted one, and to avoid that after some iterations the best token, and so that to include in the signature, will be a true invariant, is necessary to include in the non-worm samples y additional flows such that:

$$\mathcal{W}_i^{Ford} \quad \text{contains only } F_{[1..i]} \text{ fake invariants, } i \in [1..y]$$

In this way, if n is greater than the maximum length of a signature considered by the algorithm (Hamsa proposes a length of 15 tokens), it is possible to obtain a signature made only by fake invariants.

The execution of the signature refinement procedure, that includes in the selected signature all the tokens occurring in all the covered suspicious flows only if not already present in the signature, does not affect attack effectiveness: at most all remaining fake invariants, and only these, will be included in the selected signature.

This attack leads signature generation algorithm to build a signature that does not include any real invariant. The attacker is now able to send another burst of worm and non-worm samples without being detected. Even if more than one worm instance attacks the same host, this attack, unlike the well-known *red herring* attack [12], can work any way if the value of n is big enough with respect to the value of j. In the red herring attacks the adversary incorporates fake invariants into the worm samples to lead the generation system to create signatures that include those spurious features in addition to the necessary invariant tokens. Then the adversary can evade the resulting signature by not including some fake invariants in subsequently generated worm samples. So, if two or more not synchronized attackers send worm samples using different fake invariants sets, the signatures generation system will be able to create the correct signature. This is possible because the number of real invariants is greater in comparison to that of fake invariants and so, by offering better COV rate, they will be selected before the fake ones. The new signature probably contains only real invariants and so matches with all current and future flows.

6.1 Attack Effectiveness

We evaluated both Hamsa and our model for this new attack, injecting 20 fixed different tokens to the variant part of each worm and non-worm sample. Hamsa generated one signature built only with injected fake invariants. Due to the lack of real invariants presence in the signature, no new polymorphic instances of the same worm could be detected (100% false negative). In addition, all analyzed worm flows are then discarded, and so the system is not able to build a correct signature also in subsequent execution of the signature generation algorithm.

7. LISABETH EVALUATION

We evaluated the effectiveness and efficiency of LISABETH under several scenarios. To evaluate the effectiveness of our approach we first considered the case where the suspicious flows pool contained only flows of one worm. Next, we considered the case where suspicious flows pool contained some noise, and some innocuous flows as well. Finally, we considered the case where the suspicious pool contained flows from multiple worms. To evaluate LISABETH efficiency we ran our system with different amounts of data both for suspicious and innocuous pools and compared these results with those of our Hamsa implementation.

To accomplish our tests we used polymorphic versions of three real-world exploits, i.e., the Apache-Knacker, the ATPhttpd, and the Code-Red exploit, generating suspicious flows using a companion tool included in Polygraph's source code [9]. As innocuous flows we used HTTP traces collected from our laboratory's network gateway during normal usage. During the evaluation process we used several network flows pools of different sizes as input both for the suspicious and the innocuous pools.

7.1 Effectiveness

LISABETH is resilient to the attacks described in § 6 to which Hamsa is exposed. Our signature generation algorithm builds some more signatures than Hamsa but at least one of them with only real invariants, and so at least one is able to detect all subsequent worm flows. In order to evaluate how good was our performance under this kind of attack, we had to disable the dispersion filter, since if it was activated only few of all the flows sent by the worm would have been sent to the signature generator, because each flow would have been related to the same network source address. LISABETH is in fact resilient to all attacks of the *suspicious pool poisoning* family until the assumption of invariant presence holds. The dispersion filter by itself is able to neutralize all the suspicious pool poisoning attacks in which is required that, in a not synchronized environment, only one worm instance sends packets to a designated victim as in the dropped red herring attack.

Moreover, the innocuous flows selection policy also assures more resilience against *innocuous pool poisoning* attacks than Hamsa. Even if in some cases this countermeasure may be circumvented by a smart attacker, e.g., by using address spoofing on UDP traffic, the lack of constraints on the maximum false positives rate of partial signatures makes our model more resilient against innocuous pool poisoning attacks that aim to inject fewer real invariants in the innocuous traffic, as described in [12].

Finally, by not taking into account invariants order in signatures, unlike Polygraph, our model is also resilient against *coincidental-pattern* attacks [11].

The evaluation performed demonstrates the ability of our model to generate good signatures. Using a value of 1.875×10^{-2} for FP_{max} (as used in Hamsa as a threshold for signatures of 4 tokens), our model generated, in each test, at least one signature for each worm containing only a subset of its the invariant set.

In each test performed on suspicious pools containing only worm samples, and so without noise, the number of built signatures was very small: in the worst case LISABETH generated two signatures for a single worm but all generated signatures contained only real invariants. Multiple per worm signatures generation is due to the very low FP rate of these partial signature and so to their satisfying specificity. In tests with noise, LISABETH created correct signatures for each worm, so including only real worm's invariants, and a limited number of unwanted signatures containing only invariants coming from noise. Its important to note that these unwanted signatures present low false positive rate and so they do not heavily jeopardize the use of our system. Moreover, tokens included in these signature belong to a small set of strings and so this issue may be resolved by the use of a white-list of signatures.

Generating more than one, less specific signatures per worm than Hamsa, one problem of our model may be higher false positive rate. Our experiments however, prove the ability of the system to create specific enough signatures, giving an average false positives rate of 9.5×10^{-4} and so comparable with Hamsa's accuracy. This accuracy is due to the low false positive rate required for each single signature even if shorter than those produced by Hamsa.

We also assessed the efficiency of LISABETH by considering the execution time of the generation algorithm for our model compared with the execution time of our implementation of Hamsa's algorithm. In Figure 3, we show the runtime required by our model and by Hamsa to perform signatures generation for different innocuous flows pool sizes.

While spending the same amount of time, this improved efficiency allows us to use a bigger innocuous flows pool than Hamsa and so to have more accurate false positive rate evaluations on partial signatures during signatures generation algorithm execution.

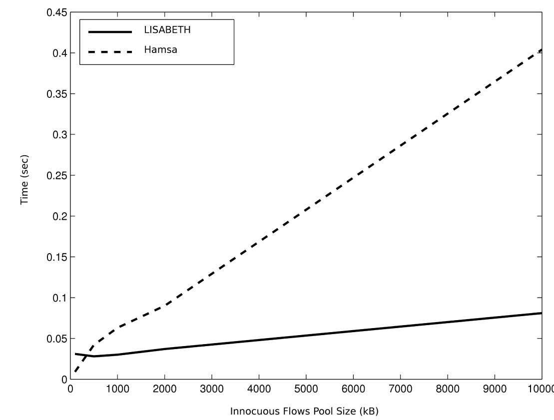

Figure 3: Requested time for generation algorithm execution in Hamsa and in Lisabeth for growing sizes of innocuous flows pool

8. RELATED WORKS

Even if early automated signatures generation systems [4, 8, 22] use different techniques to build worm signatures, all of them assume the presence of a single, specific enough, long invariant substring. Recently, there has been active research on polymorphic worm signatures generation, and new approaches have been proposed. New content-based systems like Polygraph and Hamsa have been deployed. As shown in this paper, our system is very similar to these systems, but it is a significant improvement over Polygraph [11] and Hamsa [25] in terms of speed and attack resilience.

Behavior-based systems, that use protocol and binary code information to characterize worm and subsequently build signatures, have been researched as well. In [2], Kruegel *et al.* propose an approach based on structural similarity of Control Flow Graph (CFG) to generate signatures for detecting different polymorphic worms. This approach, however, is computationally expensive and cannot detect worms with very small CFG or that apply special obfuscation techniques such as insertion of never exercised conditional branches. Of course, due to a more polymorphism resilience, it is also possible that this system detects worm that our approach misses. `TaintCheck` [10], working at host level, dynamically traces and correlates the network input to control flow changes to find the malicious input and derive worm properties. TaintCheck can understand worms and exploited vulnerability and it is able to automatically generate signatures. In [16], Christodorescu *et al.* model malware behavior and detect the code similar to an abstract model. Like the CFG-based approach, however, their approach is computationally expensive.

9. CONCLUSION

In this paper, we have presented LISABETH, an automated content-based signature generation approach for zero-day polymorphic worms. According to our experiments LISABETH achieves significant improvements with respect to performance and attack resilience over Hamsa [25], the state-of-the-art of the network-based signature generation model for zero-day polymorphic worms. Currently, our prototype is able to perform signature generation for a given pool of suspicious flows but does not implement some of the minor proposed modules like those for signatures reduction or flow selection policies. Future works will analyze potential advantages deriving from an extension of our system in a distributed environment in which many monitors will cooperate in traffic monitoring and signature generation.

Acknowledgements.

The authors would thank Roberto Perdisci for his insightful comments on preliminary versions of this paper. We would also like to thank Roberto Paleari for his constructive comments and suggestions.

10. REFERENCES

[1] Aleg Kolesnikov and Wenke Lee. Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic. Technical report, Georgia Tech College of Computing, 2004.

[2] C. Kruegel, E. Kirda, D. Mutz, W. Robertson and G. Vigna. Polymorphic Worm Detection Using Structural Information of Executables. In *Proc. of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 3858 of *LNCS*, pages 207–226, Seattle, WA, September 2005. Springer-Verlag.

[3] Christian Collberg, Clark Thomborson and Douglas Low. A Taxonomy of Obfuscating Transformations. Technical Report 148, July 1997.

[4] Christian Kreibich and Jon Crowcroft. Honeycomb - Creating Intrusion Detection Signatures Using Honeypots. In *Proc. of the Second Workshop on Hot Topics in Networks (Hotnets II)*, Boston, November 2003.

[5] Cliff Changchun Zou, Lixin Gao, Weibo Gong and Don Towsley. Monitoring and early warning for internet worms. In *Proc. of the 10th ACM conference on Computer and communications security*, pages 190–199, Washington D.C., USA, 2003. ACM Press.

[6] D. Moore, C. Shannon, G. Voelker and S. Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. In *Proc. of INFOCOM 2003*, April 2003.

[7] Frederic Raynal. Malicious Cryptography, May 2006.

[8] Hyang-Ah Kim and Brad Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. In *Proc. of the USENIX Security Conference*, 2004.

[9] James Newsome. Polygraph. [Online; last access 2007 january 28].

[10] James Newsome and Dawn Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *The* 12^{th} *Annual Network and Distributed System Security Symposium*, February 2005.

[11] James Newsome, Brad Karp and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proc. of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2005.

[12] James Newsome, Brad Karp and Dawn Song. Paragraph: Thwarting Signature Learning by Training Maliciously. In *Proc. of the Ninth International Symposium on Recent Advances in Intrusion Detection (RAID 2006)*, Hamburg, Germany, September 2006.

[13] Jose Nazario. *Defense and Detection Strategies against Internet Worms*. Artech House, 2004.

[14] Martin Roesch. Snort - Lightweight Intrusion Detection for Networks. In *LISA '99: Proceedings of the 13th USENIX conference on System administration*, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.

[15] Mihai Christodorescu and Somesh Jha. Static Analysis of Executables to Detect Malicious Patterns. In *Proceedings of the 12th USENIX Security Symposium (Security'03)*, pages 169–186, Washington, DC, USA, August 2003. USENIX Association, USENIX Association.

[16] Mihai Christodorescu, Somesh Jha, Sanjit A. Seshia, Dawn Song and Randal E. Bryant. Semantics-Aware Malware Detection. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 32–46, Washington, DC, USA, 2005. IEEE Computer Society.

[17] Peter Ferrie and Frederic Perriot. Detecting Complex Viruses, December 2004.

[18] Peter Szor and Peter Ferrie. Hunting for Metamorphic. In *Virus Bulletin Conference*, September 2001.

[19] Roberto Perdisci, David Dagon, Wenke Lee, Prahlad Fogla and Monirul Sharif. Misleading Worm Signature Generators Using Deliberate Noise Injection. In *Proc. of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*, Washington, DC, USA, 2006. IEEE Computer Society.

[20] Stefan Axelson. Intrusion Detection Systems: A Survey and Taxonomy. Technical Report 99-15, Chalmers University of Technology Department of Computer Engineering, Göteborg, Sweden, March 2000.

[21] Stuart Staniford, Vern Paxson and Nicholas Weaver. How to Own the Internet in Your Spare Time. In *Proc. of the 11th USENIX Security Symposium*, Oakland, CA, August 2002.

[22] Sumeet Singh, Cristian Estan, George Varghese and Stefan Savage. Automated Worm Fingerprinting. In *Proc. of Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[23] N. Tuck, T. Sherwood, B. Calder, and G. Varghese. Deterministic memoryefficient string matching algorithms fo intrusion detection.

[24] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Proc. of the 7th USENIX Security Symposium*, San Antonio, Texas, January 1998.

[25] Zhichun Li, Manan Sanghi, Yan Chen, Ming-Yang Kao and Brian Chavez. Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience. In *Proc. of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2006.