Operational Programme "Competitiveness"

R&D Cooperations with Organizations of non-European Countries

EAR: Early warning system for automatic detection of Internet-based cyberattacks
(Code G.S.R.T.: ΗΠΑ-022**)**

D4.1 "System Deployment and Evaluation"

Abstract: This document describes the deployment and evaluation of the EAR Early Warning System for Internet-Based Cyber-Attacks.

Contractual Date of Delivery	12 January 2006
Actual Date of Delivery	12 January 2006
Deliverable Security Class	Public
Editor	Periklis Akritidis

The EAR Consortium consists of:

FORTH	Coordinator	Greece
GA Tech	Principal Contractor	USA
FORTHnet	Principal Contractor	Greece

Contents

1 Introduction **5**

2 Implementation Update **6**
2.1 STRIDE Implementation 6
2.2 Updated Screenshots 7

3 Evaluation **9**
3.1 EAR Evaluation 9
3.1.1 Experimental Environment 9
3.1.2 Detection Delay and False Positives 9
3.1.3 Performance 15
3.2 STRIDE Evaluation 16
3.2.1 Experimental Environment 17
3.2.2 Detection Rate 17
3.2.3 False Positives 19
3.2.4 Performance 20

4 Deployment **23**
4.1 Monitored Network 23
4.2 Deployment Setup 23
4.3 Deployment Exprerience s 25

5 Conformance to the Requirements **26**
5.1 Functional Requirements 26
5.1.1 Detected Attacks 26
5.1.2 Detection Delay 27
5.1.3 False Positives 27
5.1.4 Configuration - Customization 27
5.1.5 Security Constrains 28
5.1.6 Privacy 29
5.2 Performance Constrains 29
5.2.1 Monitoring Capacity 29
5.3 Deployment 30
5.3.1 Software and hardware platform 30

5.3.2 Placement 30
5.3.3 Monitored area 30
5.3.4 Software Distribution 31
5.3.5 Initial Setup and Periodic Updates 31
5.3.6 Hardware performance requirements 31

6 Conclusions **32**

References **32**

List of Figures

2.1 Updated monitor command-line syntax 7
2.2 STRIDE-generated alerts. 7
2.3 Configuration dialogs. 8

3.1 EAR detection delay and false positives as a function of substring length. 11
3.2 EAR detection delay and false positives as a function of cache size. 12
3.3 EAR detection delay and false positives as a function of distinct destination threshold. 13
3.4 EAR detection delay and false positives as a function of offset limit. 14
3.5 EAR CPU-time as a function of bits set in sampling mask. 16
3.6 Distribution of URI lengths in the real trace used for the evaluation. 17
3.7 Detection rate for APE when applied to obfuscated trampoline sleds as a function of MEL. 18
3.8 Comparative effectiveness of the various sled detection schemes. . 20
3.9 False positives rate for APE and STRIDE with varying parameters. 21
3.10 Sensitivity of the APE method to various sled lengths of type-4 sleds. 22

4.1 Deployment diagram 24

List of Tables

3.1 Characteristics of the traces used in the experiments with EAR . . 10
3.2 EAR CPU time and extrapolated throughput. 16
3.3 Detection rates of the various sled detection schemes 18
3.4 Comparison of the processing cost of APE and STRIDE. 22

4.1 Number of hosts in FORTH deployment 24

Chapter 1

Introduction

In this document we discuss the deployment and evaluation of the EAR Early Warning System for Internet-Based Cyber-Attacks. We provide a detailed offline evaluation of the components using traces and describe a pilot deployment within the ICS-FORTH network. Deployments at several other sites, including FORTHNET and the University of Crete are under way.

This document should be read in conjunction with the "System Design" and "System Implementation" documents, which describe with detail the system evalutated in this document.

The rest of the document is organized as follows. In Chapter 2 we provide an update of the implementation. In Chapter 3 we provide an offline evaluation of the worm detection heuristics developed for this project using traces collected from ICS-FORTHs network during a worm epidemic. In Chapter 4 we describe the first deployment of the system. In Chapter 5 we tabulate the requirements of the system and evalutate the conformance of the system. Finally, we conclude in Chapter 6.

Chapter 2

Implementation Update

In this chapter we descrive some improvements made to the implementation used for the results in this document relative to the one described in the "System Implementation" document.

As described in the "System Implementtion" document, the system is composed of three modules: a monitor, programmed in the low-level C language for performance; a second component, programmed in the high-level Python language for flexibility, that further processes the generated alerts; and the Graphical User Interface (GUI), implemented in Python and GTK.

The main improvement compared to the version described in the "System Implementation" document is the integration of the STRIDE detection mechanism with the rest of the system. In the next sections we describe some implementation details of STRIDE and the updated system.

2.1 STRIDE Implementation

The design of STRIDE has been described in the "System Design" document, but was not included in the "System Implementation". Here we elaborate on its implementation.

STRIDE is passed a buffer and returns the offset of the first detected sled or -1 if no sled was found. As a first pass it will decode the buffer (using the decoder from [6]) and for each position it will determine the instruction (jump or invalid) terminating the sequence starting from each position and after how many bytes this instruction appears.

The second pass would verify for each offset that a sled of length n can start at that offset, by checking for a valid sequence of length n starting at that offset, then for a valid sequence of length $n-4$ starting at $offset+4$, and so forth. However, this would require $O(n^2)$ operations for *each* offset. Instead, we first use an approximation to STRIDE which can be applied incrementally. The approximation scans for $n-m$ bytes with valid sequences of length m. It is obviously a weaker heuristic, but can be applied incrementally. Only if the approximation finds a sled

will we run the full STRIDE to verify it.

2.2 Updated Screenshots

```
Usage: ear OPTIONS [ filter expression ]
  -f, --offset=INT         flow offset threshold
  -p, --period=INT         period threshold
  -s, --select-mask=HEX    select mask
  -t, --targets=INT        targets threshold
  -l, --length=INT         substring length
  --skip-nul               skip strings with ASCII nul characters
  -n, --home-net=NET       network under protection
  -r file                  read packets from file
  -i interface             capture packets from interface
  --disable-ear            disable the EAR detection heuristic
  --enable-stride          enable the STRIDE detection heuristic
  --stride-flow-depth=INT  how deep within flows to apply STRIDE
  --stride-sled-length=INT sled length parameter for STRIDE
  -h, --help               display this help message
```

Figure 2.1: Updated monitor command-line syntax.

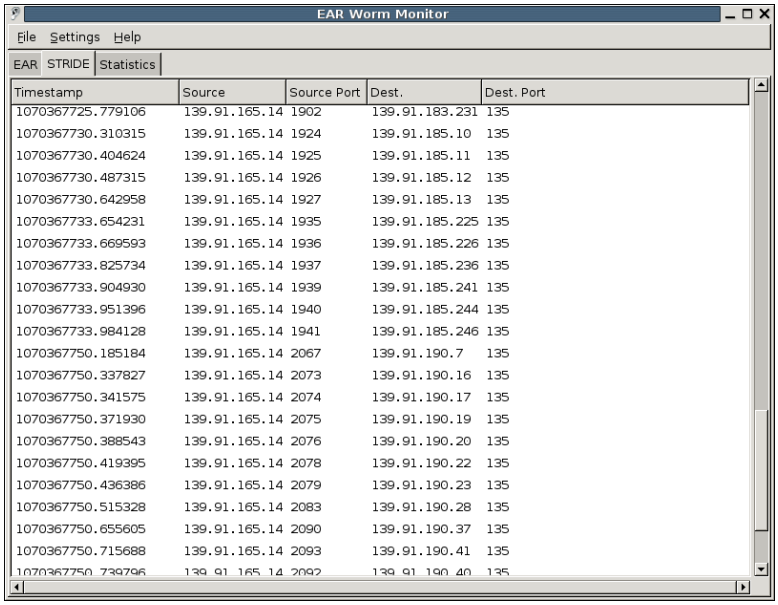


Figure 2.2: STRIDE-generated alerts.

STRIDE alerts show the offending flow, as shown in Figure 2.2. In the future we intend to add contagion and “number of targets” criteria for STRIDE alerts too, as well as support for more intelligent fusion of alerts and gracefull degradation of

the system's sensitivity. For example, an alert may need to reach either a distinct targets threshold of 10 together with a contagion threshold of 1, or, in the absence of contagion, a higher distinct targets threshold of, say, 15.

The monitor parameters can be configured by selecting the EAR Configuration menu entry from the Tools menu. Several parameters have been added in this version. It is possible to configure the following additional parameters, as shown in Figure 2.3:

Min. Sled Length STRIDE will be sensitive only to sleds longer than this value.

Max. Offset STRIDE will look for sleds only this deep into flows.

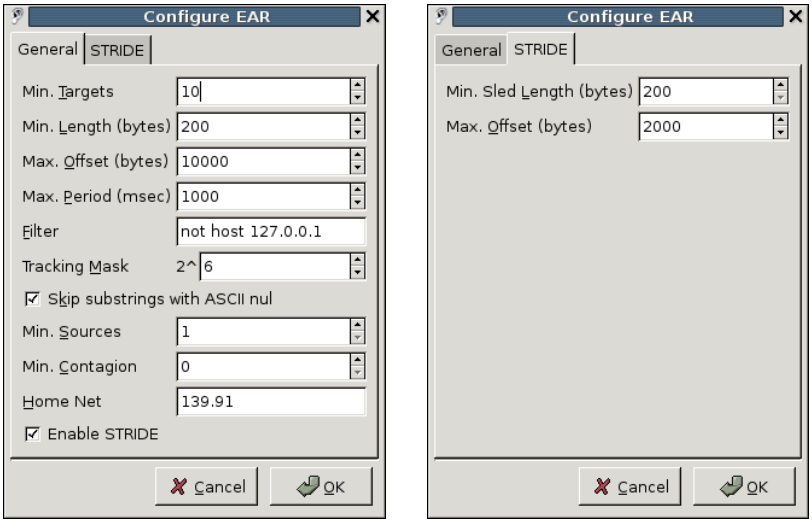


Figure 2.3: Configuration dialogs.

Chapter 3

Evaluation

In this chapter we evaluate inidividually the following detection mechanisms, that have been integrated into the worm detection system.

- **EAR** The EAR detection mechanism is able to create signatures matching non-polymorphic worms
- **STRIDE** The STRIDE detection mechanism is able to identify individual obfuscated buffer overflow attacks.

3.1 EAR Evaluation

In this section we evaluate the effectiveness and efficiency of our approach for identifying substrings that belong to Internet worms and investigate its parameter space using real network traffic traces that contain a real worm (Welchia). The traces were collected from the same network that we later deployed the worm detetion system, as described in Chapter 4.

3.1.1 Experimental Environment

Two sets of real network traffic traces have been used for our evaluation, gathered from FORTH's Local Area Network in early 2004, when the Welchia worm had been active. The monitored networks contain about 150 hosts. Each trace is about 5 Gbytes large and contains between 11.7 and 14.4 million network packets that represent traffic from web clients, peer-to-peer programs, SMB shares, IMAP, printers, etc. The traffic characteristics of the traces are shown in table 3.1.

3.1.2 Detection Delay and False Positives

In this section we explore the parameter space of the worm detector that we have described. The parameters of the experiments are:

- substring length,

Table 3.1: Characteristics of the traces used in the experiments.

Trace	TCP Packets	TCP (Payload) Bytes	Duration	Attacks
TRACE-I	11,746,790	5,108,857,877	2h	102
TRACE-II	14,429,618	5,333,977,518	2h30m	57

- distinct destinations threshold,
- substring cache size,
- flow offset limit, and
- sampling mask.

We measure the following quantities:

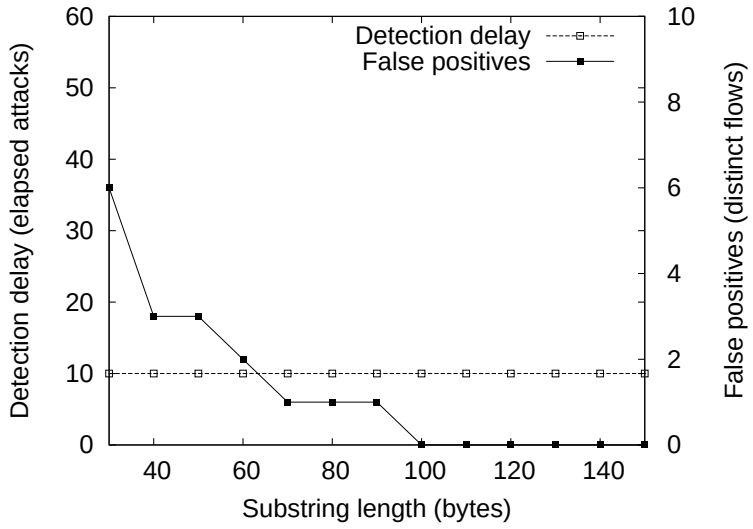
- **false positives**, which are defined to be the number of distinct flows that triggered an alert but did not correspond to any real worm,
- **detection delay**, which is defined as the elapsed worm attacks up to the detection of the worm.

Figure 3.1 shows the detection delay and the number of false positives incurred by our approach as a function of the substring length for TRACE-I and TRACE-II. We immediately notice that as the substring length increases, the number of false positives decreases. This is as expected. Indeed, it is quite possible for unrelated network packets to contain identical small substrings. Therefore, these identical small substrings that can be found in unrelated (non-worm) network packets, will generate a large number of false alarms. However, as the substring length is getting larger, it is rather unlikely for unrelated network packets to contain large identical substrings. The remaining false positives persist up to a substring length of 150 bytes and are caused by common strings such as protocol headers.

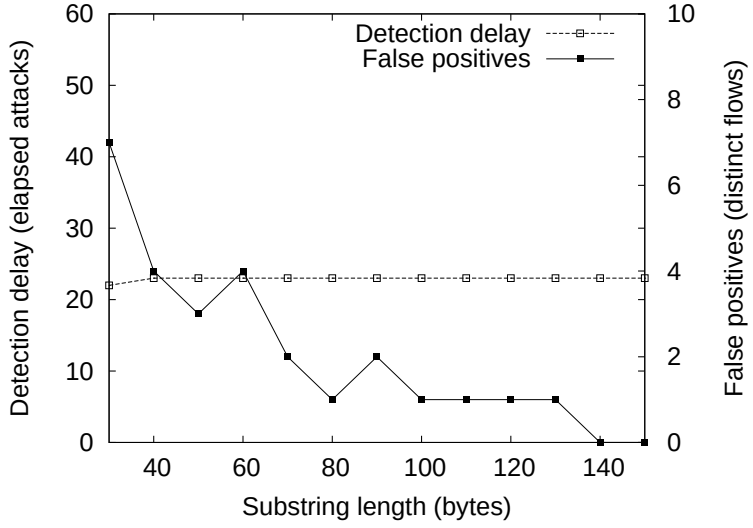
It is very encouraging to see in Figure 3.1 that as the string length increases beyond 150, the number of false positives reaches zero, which implies that no false alarms for worm outbreaks are generated by our approach. Given that most worms are longer than 150 bytes (as seen in Table **??**), operating our approach with substring length longer than 150 bytes, will probably be able to identify these known worms without generating any false positives.

We also see that the detection delay is independent of the substring length and therefore, is plotted as a line parallel to the x-axis. This is because all the true positive alerts were generated by strings belonging to the Welchia worm, which has size larger than 150 bytes.

However, the worm contains a few 150-byte strings that can also be found in normal RPC traffic. Filtering these strings would result in inadvertent denial-of-service attacks. This problem is solved by using a string length of 250 bytes or above.

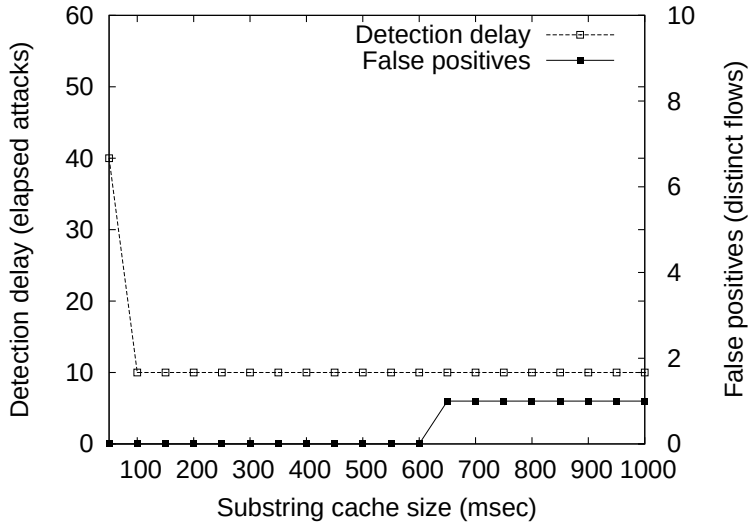


(a) TRACE-I

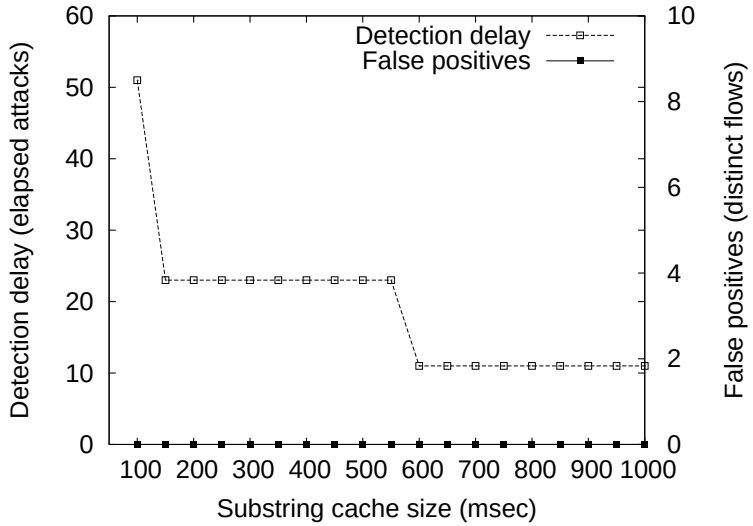


(b) TRACE-II

Figure 3.1: Detection delay and false positives as a function of substring length for a cache size worth of 500 msec, distinct destination threshold of 10, sampling mask value of 0xf, and an offset limit of 100K. We observe that we have zero false positives for substring lengths above 150 bytes, while at the same time we have detection after about 10–20 attacks.

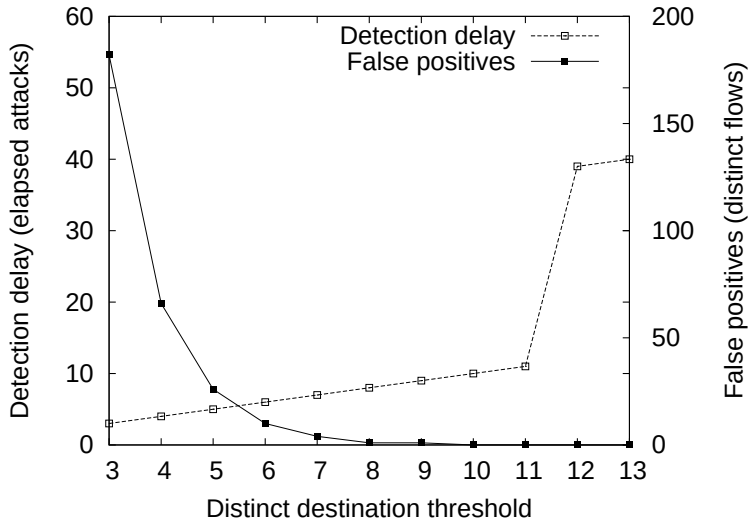


(a) TRACE-I

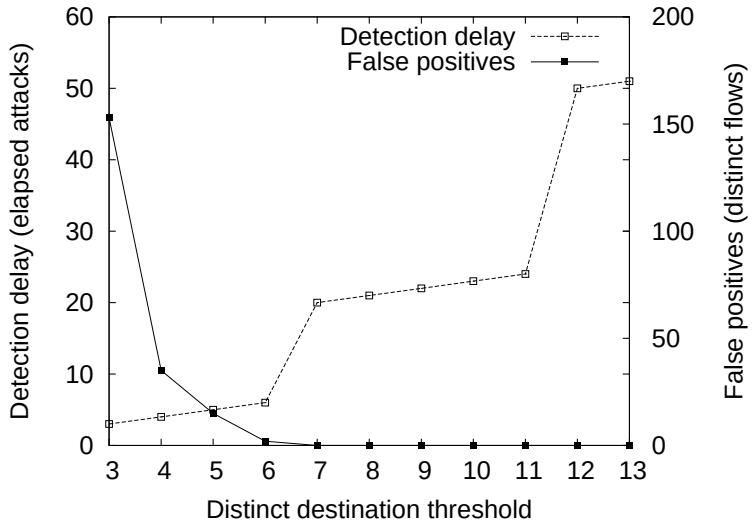


(b) TRACE-II

Figure 3.2: Detection delay and false positives as a function of cache size for a fixed substring length of 250 bytes, distinct destination threshold of 10, sampling mask value of 0xf, and an offset limit of 100K. We observe that we have zero false positives for cache sizes less than 600 msec, while at the same time we have detection after about 10–20 attacks. We also observe that smaller cache sizes increase detection delay, and eventually prevent detection.

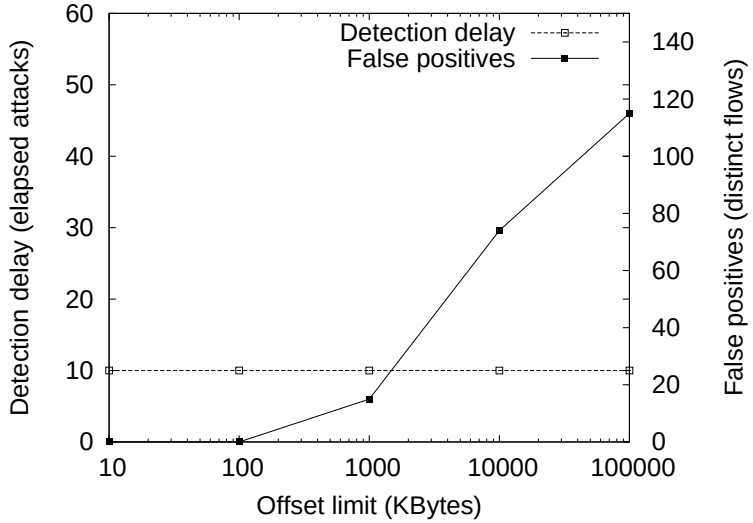


(a) TRACE-I

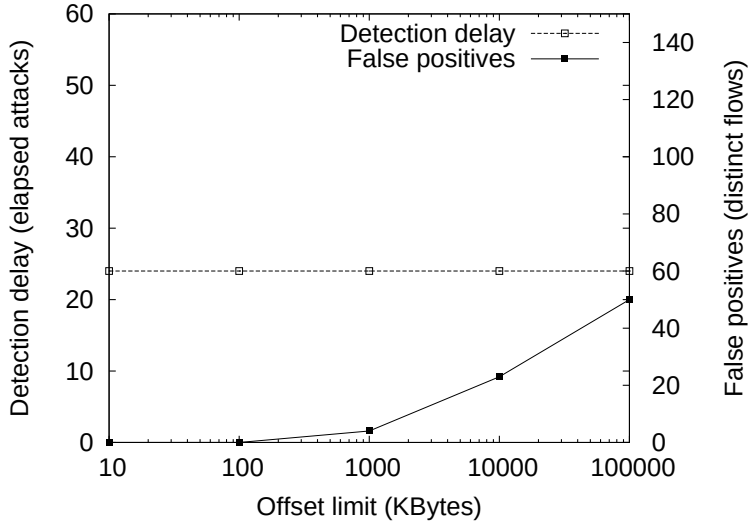


(b) TRACE-II

Figure 3.3: Detection delay and false positives as a function of distinct destination threshold for a fixed substring length of 250 bytes, cache size of 500 msec, sampling mask value of 0xf, and offset limit of 100K. We observer that we have zero false positives for a threshold of 10 or greater, while at the same time we have detection after about 10–20 attacks.



(a) TRACE-I



(b) TRACE-II

Figure 3.4: Detection delay and false positives as a function of offset limit for a fixed substring length of 250 bytes, cache size of 500 msec, distinct destination threshold value of 10, and sampling mask value of 0xf. We observer that for a limit of 100K or less, we have zero false positives, while at the same time we have detection after about 10–20 attacks.

Figure 3.2 shows the detection delay and the number of false positives as a function of the substring cache size measured in milliseconds. By definition our approach is incapable of detecting worms that are encountered less often than the period that substring fingerprints are cached. Indeed, we observe that the worm contained in the traces may evade detection for cache sizes less than 100 msec (Figure 3.2(b)). On the other hand, we observe that larger cache sizes result in more false positives, as expected, since strings with a lower rate of new destinations are retained in the cache, and can trigger false detection.

Figure 3.3 shows the detection delay and the number of false positives as a function of the distinct destination threshold. We observe that decreasing the threshold decreases detection delay. This is expected, since less worm attacks are required to trigger detection. However, decreasing the threshold may also result in false positives. This is expected too, since there exist legitimate strings that are sent to more than one destinations.

Finally, in Figure 3.4 we investigate the effects of various offset limit values. We observe that penalizing strings that appear deep in their flows significantly reduces the encountered false positives. Inspection of these false positives revealed that they are caused by peer-to-peer traffic and are encountered at random offsets in long-lived connections.

Summarizing, Figures 3.1–3.4 suggest that our approach is able to identify the worms contained in the studied traces within an acceptable time-frame without generating any false alarms.

3.1.3 Performance

In our next set of experiments we evaluate the performance of our approach and measure the effects of fingerprint selection. In our experiments we have used deterministic sampling using the `0xf` mask therefore the processed substrings have been reduced by a factor of 16. We carried out these experiments without an offset limit, to take into account the case where the smoother variation would be used as described in the design document.

Table 3.2 shows the CPU time spent for the optimized and the unoptimized version of the worm detection system to process TRACE-I: a trace of network packets 5 Gbytes large. We see that the un-optimized version takes close to 13 minutes which corresponds to a processing rate of 57 Mbits/s, while the optimized version takes close to 145 seconds, which corresponds to a processing rate of 307 Mbits/s.

Figure 3.5 demonstrates the savings in CPU-time that result from applying different sampling masks. We observer that performance increases exponentially with the number of bits in the sampling mask, as expected.

Table 3.2: CPU time consumed by experiments with TRACE-I and extrapolated throughput.

Sampling Mask	CPU Time	Throughput
0x0	13m	57 Mbits/s
0xf	145s	307 Mbits/s

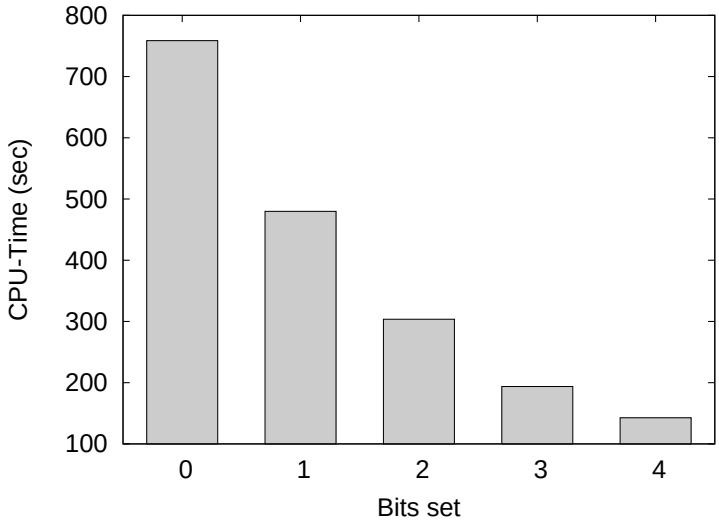


Figure 3.5: CPU-time as a function of the number of bits set in the sampling mask.

3.2 STRIDE Evaluation

In this section we evaluate STRIDE, a buffer overflow detection mechanism that has been designed to counter future polymorphic worms.

As discussed in the "System Design" document, many existing detection mechanisms have also focused on detecting the sled component in order to detect buffer overflow attacks. For example, signatures to match simple sleds have been included in the shellcode rule set of the Snort NIDS [3]. In addition, Snort has been extended with the Fnord plugin [4] that searches for obfuscated sleds. Finally, Toth and Krügel proposed the Abstract Payload Execution (APE) method [5] which further improves the sensitivity of obfuscated sled detection. However, existing detection mechanisms cannot detect all obfuscated sleds presented in the "System Design" document.

Since no polymorphic worms exist to this date, we evaluate STRIDE by comparing it to alternative buffer overflow detection techniques.

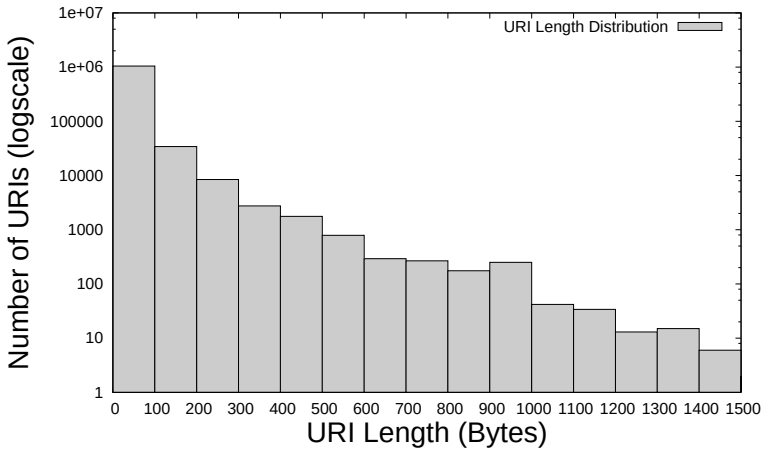


Figure 3.6: Distribution of URI lengths in the real trace used for the evaluation.

3.2.1 Experimental Environment

We evaluate the accuracy of the detection rate of our proposed algorithm STRIDE, Snort's shellcode signatures [3], Fnord mutated sled detection plugin for Snort [4], and APE [5], by generating 10,000 different sleds of each type using the Metasploit Framework v2.2 [2], modified to generate sleds ranging from type-1 (simple NOP sleds) up to type-6 (obfuscated trampoline).

We also evaluate the false positives rate of the four methods as in [5], by applying them to HTTP URIs. The URIs were captured from our institution's LAN, which contains about 150 hosts. Figure 3.6 shows the distribution of URI lengths in our traces.

For STRIDE and APE the attacks were fed as HTTP requests, while for Snort NIDS signatures and the Fnord plugin we created packets containing the attacks. In all cases, we ensured that the detection method was applied to the unquoted URI.

Sled detection methods which are based on instruction decoding, employ the decoder used in [6].

3.2.2 Detection Rate

The results of applying all four detection methods on the generated sleds are shown in Table 3.3. We observe that Snort's shellcode signatures detect simple NOP sleds with 100% success, but fail to detect more elaborate sleds. Fnord is able to detect simple NOP sleds with 100% success too, and in addition is able to detect sleds with one-byte NOP-equivalent instructions with a 55.4% rate. Although it could have achieved a 100% rate for one-byte NOP-equivalent sleds, it achieves a lower-rate due to an incomplete NOP-equivalent instruction list. Fnord also fails to detect sleds with multi-byte NOP-equivalent instructions, but it should be possible to update its list of NOP-equivalents to include them as well. However, this is as far

Sled Type in Trace	Scheme			
	Snort	Fnord	APE	STRIDE
NOP instructions	100%	100%	100%	100%
One-byte NOP-equivalents	0%	55.4%	100%	100%
Multi-byte NOP-equivalents	0%	0%	100%	100%
Four-byte Aligned	0%	0%	100%	100%
Trampoline-sled	0%	0%	0%	100%
Obfuscated Trampoline-sled	0%	0%	Fig. 3.7	100%

Table 3.3: Detection rates of the various detection schemes for traces containing 10,000 different generated sleds of a single type.

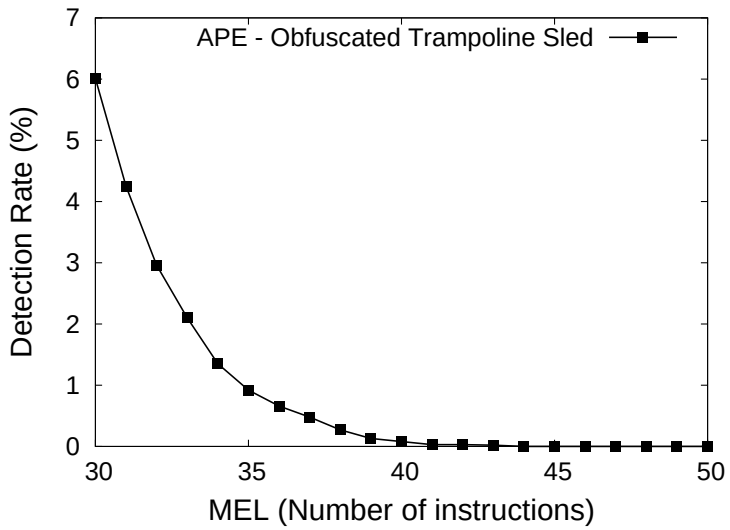


Figure 3.7: Detection rate for APE when applied to obfuscated trampoline sleds as a function of MEL.

as Fnord can get. Indeed, Table 3.3 shows that Fnord fails to detect sophisticated sleds, such as 4-byte aligned and trampoline sleds.

Table 3.3 suggests that the APE method is able to detect simple NOP sleds, sleds with one-byte and multi-byte NOP-equivalent instructions, as well as four-byte aligned sleds with a 100% success rate. However, APE cannot detect trampoline sleds. This was expected, because trampoline sleds reach the core attack code by executing only a small number of jump instructions, while APE bases its detection method on the sequential execution of a long sequence of instructions.

It is interesting, however, to point out that although APE can not detect trampoline sleds, it is able to detect some of the more difficult obfuscated trampoline sleds. Indeed, as Figure 3.7 shows, APE is able to detect as many as 6% of the obfuscated trampoline sleds for small MEL. This is because the NOP-equivalent instructions that are used for the obfuscation cause an increase of the overall execution steps of the sled, which can now reach a low MEL threshold. Nevertheless, the detection rate of APE is still very low, at 6%, even for the minimum suggested MEL value.

Finally, STRIDE is able to detect simple NOP sleds, sleds with one-byte or multi-byte NOP-equivalent instructions, as well as four-byte aligned sleds and plain or obfuscated trampoline sleds with 100% success, as expected.

3.2.3 False Positives

The results of the false positives rate evaluation for the four methods with real traces are shown in Figure 3.8. In this experiment STRIDE has a sled length parameter of 130 bytes and APE has a MEL value of 35 instructions with 100 samples per kilobyte. With these parameters APE is sensitive, like STRIDE, to sled lengths of about 130 bytes and above.

The Snort shellcode signatures have zero false positives, because there was no sufficiently long NOP-sequence in our traces. Fnord also has almost 0% false positives, because there were very few sequences of bytes in the traces which corresponded to sequences of NOP-equivalent instructions. Although both Snort and Fnord have an attractive practically 0% of false positives rate, they are severely limited in their ability to detect elaborate sleds, such as trampoline sleds. Figure 3.8 shows that APE has a false positive rate of 0.006%. Finally, STRIDE has a false positive rate of 0.00027%, close to an order of magnitude smaller than APE. Overall, we see that STRIDE seems to strike a good balance between true positives and false positives. That is, it is able to find more true positives than any other method, while keeping the false positives as low as those of Fnord and Snort.

The interested reader should notice that the exact value of false positives for APE and STRIDE depends heavily on their parameters. To explore the influence of the parameters to the false positive rate of APE and STRIDE, we investigate the false positives rate for both methods as a function of MEL and sled length, and display the results in Figure 3.9. We see that as the size of MEL increases, the percentage of false positives for APE decreases. However, we should point out

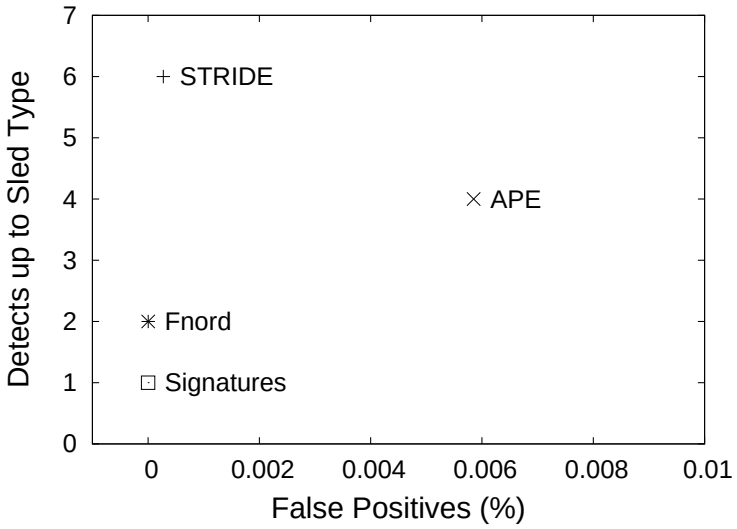


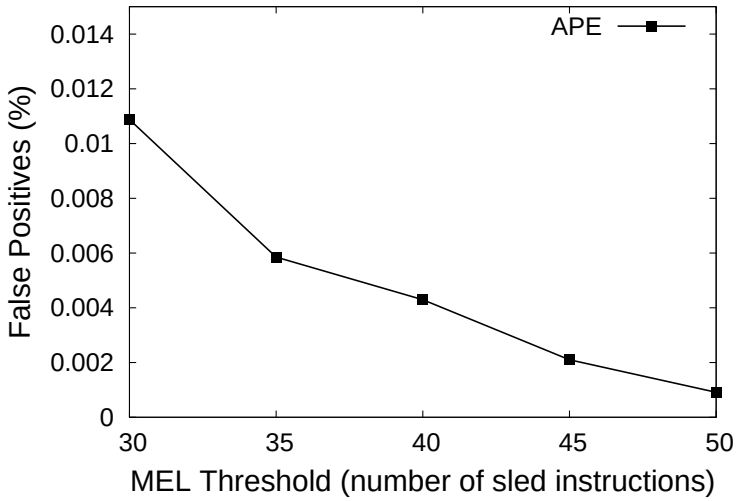
Figure 3.8: Comparative effectiveness of the various detection schemes. The results for APE are for a MEL value of 35 with 100 samples per kilobyte and for STRIDE for sled length 130 bytes.

that larger MEL values also decrease the number of detected true positives, as can be seen in Figure 3.7. Figure 3.9 also shows that the percentage of false positives for STRIDE decreases with the sled length, and reaches zero for sled length larger than 230 bytes. This is an encouraging result, since typical sleds are usually longer than 250 bytes. Overall, our results suggest that STRIDE is able to have a true positive rate of 100% (as shown in table 3.3), while having a false positive rate of (close to) 0%.

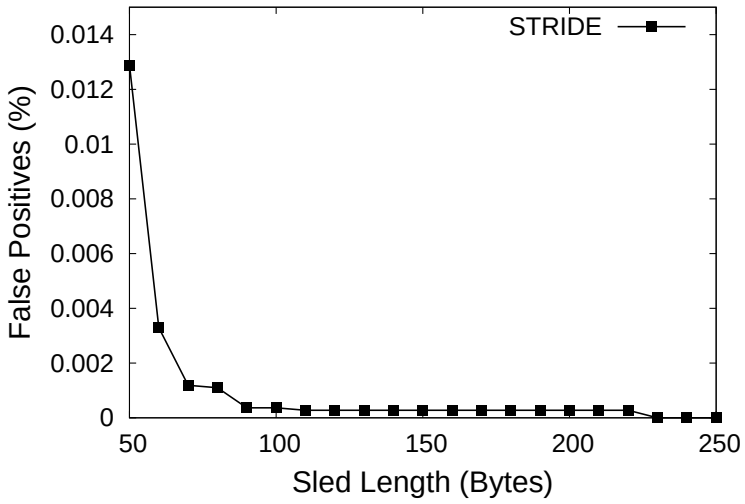
However, MEL is not directly related to the sled length in bytes. First, an instruction can be multiple bytes long, and, second, APE uses sampling. In Figure 3.10 we attempt to determine the detectable sled length for APE with the default parameters of the APE implementation by applying it on traces of type-4 sleds with varying lengths. The results show that the actual detectable sled length is about 150 bytes, so the comparison was fair.

3.2.4 Performance

Besides being accurate, a worm detection method should also be fast, so as to be able to detect worms in real-time. To evaluate the speed of STRIDE we measured the CPU time consumed by STRIDE with a sled length value of 200 bytes, and compared it to the execution time of APE with a MEL count of 35 on a Pentium 4 machine (2.6GHz clock speed, 512KB cache size) for a trace with 1,093,249 requests, and show the results in Table 3.4. We see that STRIDE outperforms APE by a factor of 5. This is mostly due to the different handling of branch instructions by the two algorithms. Indeed, when APE encounters a branch instruction, whose



(a) APE



(b) STRIDE

Figure 3.9: False positives rate for APE and STRIDE with varying parameters.

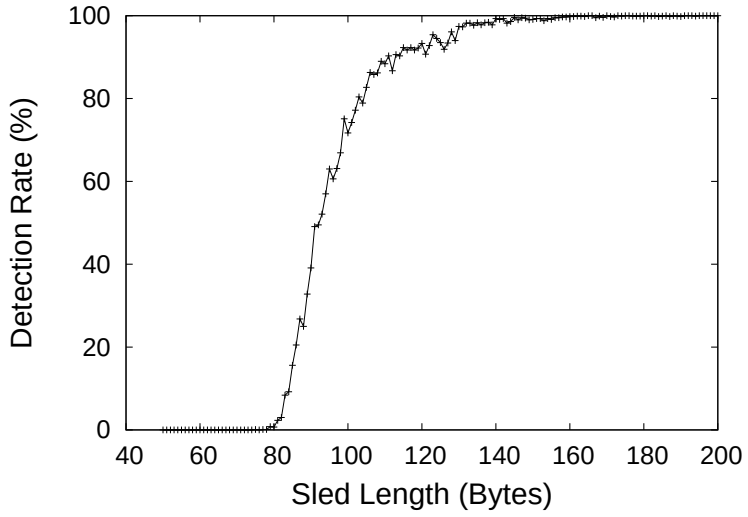


Figure 3.10: Sensitivity of the APE method to various sled lengths of type-4 sleds. For the recommended MEL parameter value of 35, the detectable sled length is about 150 bytes.

target can be determined statically it follows both branches, a decision, which may potentially lead to the exploration of an exponential number of execution paths. Unlike APE, when STRIDE encounters a branch instruction, it assumes that it found a valid sequence, without making any attempt to follow the branch. By being conservative, STRIDE avoids the exponential explosion and significantly reduces the associated run-time cost.

Table 3.4: Comparison of the processing cost of APE with a MEL count of 35 and STRIDE with a sled length of 200 bytes, for a trace with 1,093,249 requests, using the same instruction decoder on the same machine. STRIDE outperforms APE by a factor of 5.

	Total CPU Time	CPU Time Per Request
APE	$25sec$	$22.9\mu sec$
STRIDE	$4.85sec$	$4.4\mu sec$

Chapter 4

Deployment

We have already deployed the worm detection system on FORTH's network and several other deployments are under way, including FORTHNET and the University of Crete network.

4.1 Monitored Network

ICS-FORTH's network consists of several LANs. The deployment covers more than 200 machines in total. About half of them run the Windows OS, which is the typical worm target. Table 4.1 shows the break down of the monitored population to LANs.

4.2 Deployment Setup

The deployment involves a router, which has been instructed to mirror the traffic from several of the institution's LANs to a dedicated monitoring system running on commodity hardware. The bulk of the traffic is processed by the dedicated monitor. The results of worm detection are transered over a secure (SSH) connection to an operator's computer and are processed by the worm detection system user interface which is running on that computer. This setup is illustrated in Figure 4.2.

The parameters used to run the system were determined empirically during the evalution in Chapter 3 and are as follows:

- **Min. Targets:** 10 (A worm must have target at least this many targets.)
- **Min. Length:** 200 bytes (A worm must have at least this length.)
- **Max. Offset:** 2000 bytes (A worm must occur at most at this offset within its flow.)
- **Max. Period:** 500 μ sec (A worm must occur with a period at most equal to this value.)

Table 4.1: Number of hosts in FORTH deployment

LAN	Number of Hosts
Total	218
139.91.190	67
139.91.70	35
139.91.183	34
139.91.185	23
139.91.68	17
139.91.182	15
139.91.157	7
139.91.197	3
139.91.76	2
139.91.6	2
139.91.200	2
139.91.165	2
139.91.151	2
139.91.88	1
139.91.72	1
139.91.195	1
139.91.189	1
139.91.187	1
139.91.184	1
139.91.1	1

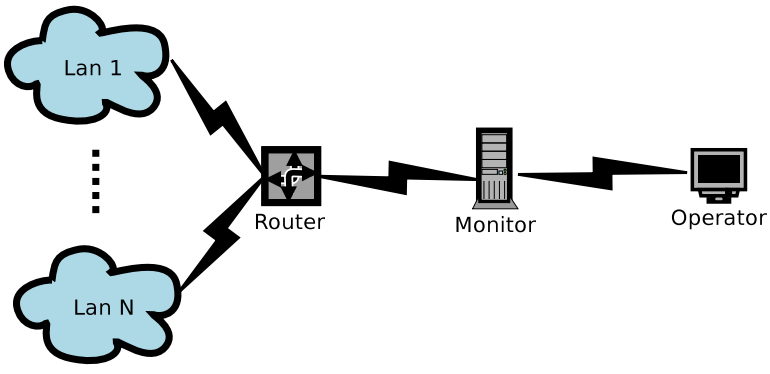


Figure 4.1: Deployment diagram.

- **Filter:** dst port not 6346 (A filter that is applied on network traffic before it is processed. We use it to exclude peer-to-peer traffic on the standard port.)
- **Tracking Mask:** 1 out of 32 (This value control the deterministic sampling rate.)
- **Skip substrings with ASCII nul = TRUE** (Whether to process substring with the ASCII nul character.)
- **Min. Sources:** 1 (A worm must be spread by at least this many sources.)
- **Min. Contagion:** 0 (The value of the contagion threshold.)
- **Home Net:** Unspecified (The network under protection. We did not limit the monitoring to FORTH's LAN in order to stress the system more.)

4.3 Deployment Exprerience

We have operated the system for several weeks trying to verify the parameter values found in the evaluation with traces for longer time intervals with live traffic. Our experience was successfull, with the only problem being false positives caused by Gnutella traffic on the standard Gnutella port. Several adjustments of the parameters supressed these false positives:

- Increasing the number of sources to a value greater than one
- Increasing the contagion threshold to a value greater than zero
- Excluding the Gnutella port using a filter expression

We deemed the third option as the more appropriate for our setup because with the sacrifice of detecting worms targetting the standard Gnutella port we did not have to lower our sensitivity for worms in general.

Apparently, the best of both worlds would be possible if the thresholds could be adjusted per port. For example, traffic on every port other than the standard Gnutella protocol port could require low thresholds, but traffic on the Gnutella port would require a larger source or contagion threshold. We plan to add support for this feature.

Chapter 5

Conformance to the Requirements

In this chapter, we discuss how the implementation conforms to the requirements laid out in the Specifications document. We list each requirement and discuss the relevant implementation features.

5.1 Functional Requirements

5.1.1 Detected Attacks

F 1.1 *The system should detect attacks that use unfragmented packets, as well as fragmented packets to spread their payload over multiple packets in order to obfuscate their signatures.*

By employing flow reassembly mechanisms, the design ensures that detection cannot be evaded by breaking signatures across packet boundaries. Specifically, in the implementation we have used the libnids [1] for flow reassembly.

F 1.2 *The system should detect worms with an attack that contains at least 300 consecutive bytes.*

The relevant parameter of the system is substring size, the evaluation shows that a substring size of 300 bytes is sufficient to detect worms.

F 1.3 *The system must detect worms that spread over the TCP protocol. However, it is highly desirable, but not initially required, to include support for UDP worms as well.*

There is support for the TCP protocol, including session tracking and stream reassembly. We are considering UDP support as a future extension.

F 1.4 *The system is not expected to detect stealth worms.*

The system is capable of detecting stealth worms as long as they are detected by the STRIDE heuristic. However, some logic needs to be added for managing the information generated from STRIDE in order to detect and report stealth worms.

F 1.5 *Strings of small length are often popular without belonging to a worm. For example, many unrelated HTTP requests contain the string* GET /, *or* HTTP/1.1. *Therefore, a minimum detectable string length must be established.*

The system has a substring length threshold that controls the minimum detectable signature length. By adjusting it to more than about 200 bytes the number of false positives due to such string is greatly reduced. In addition, the option of white-listing some popular strings should help. Finally, the contagion heuristic almost completely eliminates such false positives.

5.1.2 Detection Delay

F 2.1 *The detection delay of the system, measured in elapsed attacks before an alert has been triggered, should be evaluated theoretically and experimentally for worms with different levels of aggressiveness.*

We have performed an evaluation of the detection delay as a function of various detection parameters.

5.1.3 False Positives

F 3.1 *Timely detection is worthless in the presence of false positives, therefore the system is required to have a zero false positives rate.*

The evaluation has shown that it is possible to detect worms without false positives. In addition, we have considered a number of mechanisms to further reduce false positives, including buffer overflow detection and the contagion heuristic.

F 3.2 *As a last means of preventing false positives, the system should support a white-list that allows handling persistent false positives. Strings listed in the white-list should not be considered as worm signatures.*

The implementation allows the tagging of certain substrings with an “ignore” flag. The user interface does not yet provide access to this functionality.

5.1.4 Configuration - Customization

F 4.1 *It should be possible to adjust the sensitivity of the system using a threshold. The system could be adjusted for faster detection with the cost of the false positives rate going up.*

This tradeoff has been demonstrated in the evaluation and the requirement is supported by allowing the tuning of parameters.

F 4.2 *It must be possible to adjust the amount of information that will be recorded, so as to cater for those that worry about their privacy being revealed.*

The minimal amount of recorded information consists of the offending signature. We are currently not considering of providing more information.

F 4.3 *It should be possible to configure whether log-files, result packets, or both will be used to report alerts.*

In the implementation the results are communicated from the monitor application to the graphical user interface through its standard output. It is trivial for an operator to add log-file support, and any other mechanisms by piping the output of the monitor through any UNIX utility (for example the "tee" utility allows for saving the data to a file).

F 4.4 *It should be possible to configure the log-file location and the destination to which result packets are sent.*

This has been left to the operator. Sample configurations will be provided in the future.

5.1.5 Security Constrains

F 5.1 *The network where the system is hosted must not be exposed to vulnerabilities because of the presence of the early warning system.*

The monitoring application is written with security in mind and the "privilege separation" principle has been applied by implementing as much functionality as possible in an unprivileged process (the alert post processing and gui modules). In the future, the monitoring application will be modified to drop its privileges soon after openning the network interface.

F 5.2 *The system must also be resilient to malicious traffic targeted to attack the system itself. Finally it must be carefully engineered so that it will be immune to any kind of attack. This is particular important as a poorly configured/engineered system can pose a huge security risk for the whole network. Thus is essential that the system is programmed with security practices in mind.*

The chief security design concern is the abuse of the system by causing false positives that match legitimate traffic. Acting on those false positives would effectively mount a denial-of-service attack on the legitimate traffic. The main mechanism of defence against such an attack is sampling, which forces the attacker to generate considerable traffic. Additional filters such as buffer overflow detection reduce significantly the possibility of matching legitimate traffic.

5.1.6 Privacy

F 6.1 *The data gathered should be strictly used for analyzing, identifying a possible attack, implementing ways to protect and for absolutely no other reason. The EAR framework will operate in a way that the data do not have to be shared among various individuals and/or companies thus minimizing the risk of revealing important information about the end-user or corporate data.*

The processing of the data that takes place inside the monitor component provides protection of sensitive data.

F 6.2 *The data analysis must take place within the organization data center which is considered to be a trusted body.*

The system has been designed as a centralized facility and the only information that might be publicized are the detected signatures.

F 6.3 *The issued alerts and reports must not include any information that will reveal the identity of the user (eg. the IP addresses belonging to the infected hosts). Furthermore, only content that belongs to traffic that has been identified as traffic initiated by a worm is going to be included by the system in alerts.*

F 6.4 *The system should be able to integrate with an external application that will take over the task of transmitting the alerts to the administrators.*

Such an application has been implemented by the GUI component. In addition, it can be trivially extended with additional notification mechanisms.

F 6.5 *The necessary information should be recorded in log files. The format of log files should be specified in detail when the system has been developed completely, it shall contain enough information to create a filtering signature.*

The monitor application emits records in a specific format.

5.2 Performance Constrains

5.2.1 Monitoring Capacity

P 1.1 *The system must be capable of operating at 100 Mbits/sec. However, it is highly desirable to achieve operation speeds up to 1 Gbit/sec.*

Depending on the detection parameters, the system can achieve a performance of 300-400 Mbits/sec.

P 1.2 *The system does not operate in-line and therefore does not impose any limitations or delay on the traffic capacity of the monitored network.*

This requirement is supported by design.

P 1.3 *It should be possible to operate the system by processing only part of the traffic.*

Several filtering mechanism are employed to reduce the load of the system. The main provision is the processing of client to server flows only, and in addition only the start of the flows is processed. Ignoring high-traffic hosts is also supported by specifying appropriate filter expressions.

5.3 Deployment

5.3.1 Software and hardware platform

D 1.1 *The system will be developed for the GNU/Linux operating system and the x86 hardware platform.*

The system was developed and deployed on Linux running on x86.

D 1.2 *It should be easily portable to other platforms and Unix-class operating systems.*

The software does not depend on non-portable libraries and is itself written in standard C and the high-level python language, which is ported to all major platforms.

D 1.3 *The system will require a dedicated machine for production operation.*

We recommend using a dedicated machine.

5.3.2 Placement

D 2.1 *The system will not be placed in-line, but will operate as a network tap, processing all traffic visible to its network interface. Therefore, it will not be able to interrupt the operations of other production systems.*

The system conforms to this requirement.

5.3.3 Monitored area

D 3.1 *The networks to be monitored will be selected by mirroring the appropriate traffic to the monitoring system's network interface.*

The system has been deployed this way.

D 3.2 *The number of the hosts that the system shall be capable of monitoring will be in the order of hundreds.*

The system has been deployed in a network of 150 hosts.

D 3.3 *The system must be able to see symmetric traffic. Sometimes routers only see one direction of the traffic. This is called assymetric routing. Our system will not have to tackle with this.*

This is required for flow reassembly.

5.3.4 Software Distribution

D 4.1 *The system relies on the Snort NIDS [3] and is distributed as a Snort plug-in in the form of a non-intrusive patch against the standard Snort distribution. This means that there are no compatibility issues with existing systems, thus it will be easy to deploy.*

The dependency on Snort NIDS has been dropped during the development of the system. The flow reassembly is performed using libnids [1].

5.3.5 Initial Setup and Periodic Updates

D 5.1 *Initial installation will require building the software from source. New versions or configuration file modifications may require stopping and starting the system on software level.*

This is the case, although the distribution of precompiled versions is also considered.

5.3.6 Hardware performance requirements

D 6.1 *The system should be designed to operate on commodity PC-class hardware. Sample specifications: 3GHz Processor, 1GByte RAM.*

We have successfully deployed the system on a 2.6GHz processor with 1GBytes of RAM.

Chapter 6

Conclusions

In the context of this project we have developed a method to detect current generation Internet worms by finding strings with a high rate of transfer to different targets and looking for them in the start of client-to-server traffic, and we have provided protection against future polymorphic worms by developing STRIDE, a new mechanism for network-level detection of buffer overflow attacks that relies on the identification of the sled component that is usually part of such attacks. Because it operates at the network-level, STRIDE can be used for detecting worms that replicate through buffer overflow exploits, even if they involve elaborate obfuscation.

During the evaluation described in this document, we have shown that on the studied scale, of about 150 hosts, detection without false positives of non-polymorphic worms is possible with a detection delay that suggests a 7-14% infection. The detection was sensitive to worms generating collectively at least one attack about every 500 msec. These results are very encouraging. Presumably, higher aggregation (larger deployment) would further reduce detection time, and also enable detection of less aggressive worms.

As for future polymorphic worms, the high accuracy, low false positive rate, and low processing cost achieved by STRIDE make it highly useful as part of an automated network-level defense mechanism against large-scale zero-day worm outbreaks, especially as worms become more aggressive and more sophisticated. STRIDE may be useful as a defense against target attacks as well.

In conclusion, we have presented an efficient method to detect current generation Internet worms with practically zero false positives, and have provided protection against future polymorphic worms.

References

[1] Libnids. http://libnids.sourceforge.net/.

[2] Metasploit project, 2004. http://www.metasploit.com/.

[3] Martin Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of USENIX LISA '99*, November 1999. (software available from http://www.snort.org/).

[4] Dragos Ruiu. Fnord: Multi-architecture mutated NOP sled detector, February 2002. http://www.cansecwest.com/spp_fnord.c.

[5] T. Toth and C. Kruegel. Accurate Buffer Overflow Detection via Abstract Payload Execution. In *Proceedings of the* 5^{th} *Symposium on Recent Advances in Intrusion Detection (RAID)*, October 2002.

[6] Thomas Toth. Apache buffer overflow detector module, March 2002. http://www.infosys.tuwien.ac.at/Staff/tt/abstract_execution/.