Project MILTIADES: **M**ult**I**-**L**ayer **T**echn**I**ques for **A**ttack **DE**tection **S**ystems

Deliverable 4.1: Experimental Evaluation and Real-world Deployment

This document presents the experimental evaluation of *nemu*, an network-level attack detector based on code emulation, as well as attack statistics from long-term deployments of nemu in production networks.

Due Delivery Date	31/03/2008
Actual Delivery Date	31/03/2008

Project Partners

FORTH-ICS	Project Leader	Greece
Columbia University	Project Leader	US
Virtual Trip	Co-operating organisation	Greece

1 Table of Contents

2 Introduction

As a response to the ever increasing number of automated Internet attacks, as well as to the increasing levels of attack sophistication, the project MILTIADES explores novel attack detection and defense approaches against modern cyber-attacks. Specifically, our contributions revolve around the following two research directions: i) IP address space randomization, and ii) emulation-based polymorphic attack detection.

2.1 IP address space randomization

To evade detection during their outbreak, and generate as little traffic as possible, sophisticated worms gather information about their targets several weeks before they launch their attack. During those weeks they probe a very large number of IP addresses, if possible all 4 billions of them available in the Internet today (IP version 4), in order to find those hosts which are vulnerable to the planned attack. All vulnerable hosts found, are included in a special list of targets, which is frequently referred to as the "hit-list". Instead of attacking computers at random during the outbreak, as naive worms do, sophisticated hit-list-based worms only attack computers included in the hitlist and therefore (1) they generate the minimum traffic possible, and (2) they do not generate any unsuccessful (TCP) Internet connections. Therefore, hit-list-based worms propagate at the maximum possible speed, evading their timely detection by worm detection systems.

To slow down the rate of their spread, and if possible neutralize those hit-list worms, we propose to conduct research towards mechanisms which will make the contents of the hit list stale. Armed with a stale hit list, a worm will not be able to successfully infect hosts, but it will also make several unsuccessful attempts to compromise non-vulnerable computers, resulting in several unsuccessful TCP connections. These unsuccessful connections will probably be more visible to firewalls and intrusion detection systems, which will quickly take notice of the spreading worm. To put it simply: a stale hit-list will slow down the spreading worm, and make it visible to firewalls and Intrusion Detection Systems.

2.2 Emulation-based polymorphic attack detection

To protect themselves against malicious intruders, organizations usually employ Network-level Intrusion Detection Systems (NIDSes) in their gateways to the Internet. NIDSes, inspect all incoming traffic against a preloaded set of known attack signatures (i.e., attack rules) in order to see whether any network packet(s) match any of the attack signatures. As soon as the NIDSes find network packets which match any one of the attack signatures, they log the offending packets and alert the system administrators of the intrusion attempt.

Although IDSes have been successfully used to identify and prevent traditional attacks, they are getting increasingly less effective when faced with the next generation of polymorphic and metamorphic worms for several reasons. First, traditional Intrusion Detection Systems operate using predefined attack signatures, which means that they cannot detect previously unknown (“zero day”) attacks for which no signature exists. Second, as organizations start deploying state-of-the-art detection technology, attackers are likely to react by employing advanced evasion techniques, such as polymorphism and metamorphism, to defeat these defenses.

In contrast to previous work, in this project we explore the approach of *emulation-based polymorphic attack detection*, a novel approach for the detection of previously unknown polymorphic attacks, which is based on the actual execution of attack data on a CPU emulator. Our prototype detector implementation, called *nemu*, does not rely on any exploit or vulnerability specific signatures, which allows the detection of previously unknown attacks. The main principle network-level emulation is the use of a generic heuristic that matches the runtime behavior of polymorphic shellcodes. At the same time, the actual execution of the attack code on a CPU emulator makes nemu robust to evasion techniques such as highly obfuscated or self-modifying code. Furthermore, each shellcode is detected separately, which gives nemu the ability to effectively detect targeted attacks.

3 Experimental evaluation of the Prototype Detector

In this section, we present the experimental evaluation of `nemu`, our prototype implementation of a network-level polymorphic attack detector based on binary code emulation [1, 2]. Specifically, we focus on the raw processing throughput of the detector.

We accumulated full payload packet traces of frequently attacked ports captured at FORTH-ICS and the University of Crete across several different periods. We also captured a two hour long trace of all the TCP traffic of the access link that connects FORTH-ICS to the Internet. Since we are interested in client-initiated traffic, which contains requests to network services, we keep only the packets that correspond to the client-side stream of each TCP flow. For large flows, which for example may correspond to file uploads, we keep the packets of the first 64KB of the stream. Trace details are summarized in Table 1. Note that the initial size of the FORTH-ICS trace, before extracting the client-initiated only traffic, was 106GB.

Table 1: Details of the client-initiated network traffic traces used in the experimental evaluation.

Name	**Port Number**	**Number of Streams**	**Total Size**
HTTP	80	6511815	5.6GB
NetBIOS	137-139	1392679	1.5GB
Microsoft-DS	445	2585308	3.8GB
FORTH-ICS	*all*	668754	821MB

`Nemu` is based on a custom IA-32 CPU emulator that uses interpretive emulation. We measured the user time required for processing the network traces presented in Table 1, and computed the processing throughput for different values of the CPU execution threshold. The detector was running on a PC equipped with a 2.53GHz Pentium 4 processor and 1GB RAM, running Debian Linux (kernel v2.6.18). Figure 1 presents the results for the four different network traces.

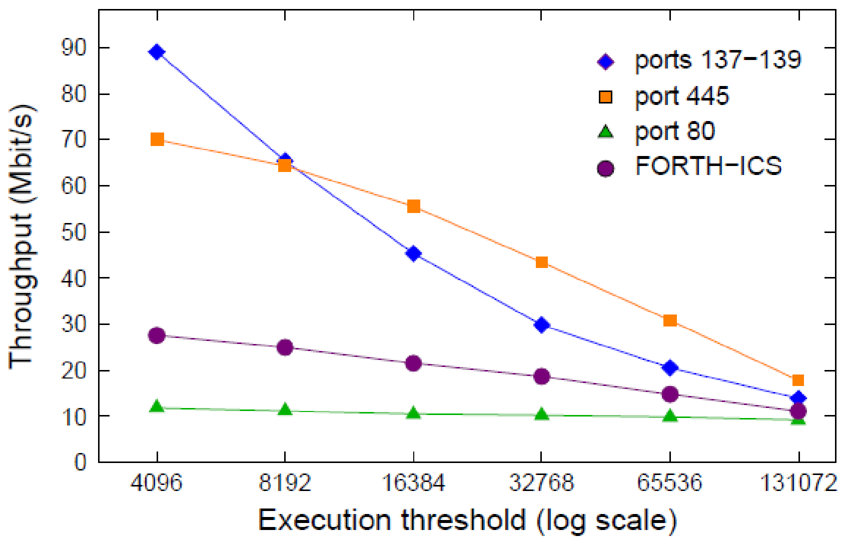


Figure 1: Raw processing throughput for different execution thresholds.

As expected, the processing throughput decreases as the CPU execution threshold increases, since more cycles are spent on streams with very long execution chains or seemingly endless loops. We measured that in the worst case, for port 445 traffic, 3.2% of the streams reach the CPU execution threshold due to some loop when using a threshold higher than 8192. This percentage remains almost the same even when using a threshold as high as 131072 instructions, which means that these loops would require a prohibitively large number of iterations until completion.

Port 80 traffic exhibits the worst performance among all traces, with an almost constant throughput that drops from 12 to 10 Mbit/s. The throughput is not affected by the CPU execution threshold because i) the zero-delimited chunk optimization is not effective because HTTP traffic rarely contains any null bytes, and ii) the execution chains of port 80 traffic have a negligible amount of endless loops, so a higher CPU execution threshold does not result to the execution of more instructions due to extra loop iterations. However, ASCII data usually result to very long and dense execution chains with many one or two byte instructions, which consume a lot of CPU cycles.

We should stress that our home-grown CPU emulator is highly unoptimized, and the use of interpretive emulation results to orders of magnitude slowdown compared to native execution. It is expected that an optimized CPU emulator like QEMU [3] would boost performance, and we plan in our future work to proceed with such a change. Nevertheless, the low processing throughput of the current implementation does not prevent it from being practically usable. In the contrary, since the vast majority of the traffic is server-initiated, the detector inspects only a small subset of the total traffic of the monitored link. For example, web requests are usually considerably smaller than the served content. Note that all client-initiated streams are inspected, in both directions.

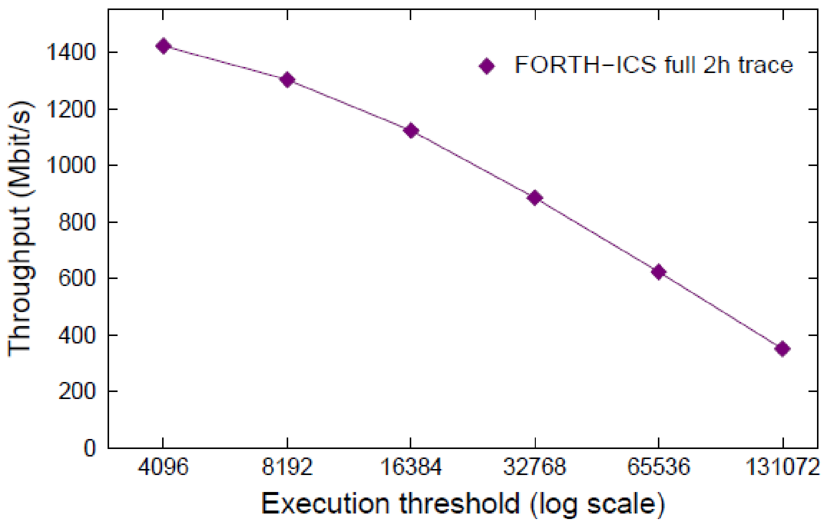


Figure 2: Raw processing throughput for the complete two-hour trace.

Furthermore, even in case of large client-initiated flows, e.g., due to file uploads, the detector inspects only the first 64KB of the client stream, so again the vast amount of the traffic will not be inspected. Indeed, as shown in Fig. 2, when processing the complete 106GB long trace captured at FORTH-ICS, the processing throughput is orders of magnitude higher. Thus, the detector can easily sustain the traffic rate of the monitored link, which for this 2-hour long trace was on average around 120 Mbit/s.

4 Real-world Deployment

In this section, we present attack activity results from real-world deployments of our prototype detector implementation. In each installation, `nemu` runs on a passive monitoring sensor that inspects all the traffic of the access link that connects the protected network to the Internet. Here, we collectively report statistics from two deployments in a National Research Network and an Educational Network in Europe.

4.1 Educational network

`Nemu` has been installed on a passive monitoring sensor that inspects the traffic of the access link that connects part of an educational network with hundreds of hosts to the Internet. The detector has been continuously operational since 7 November 2006, except a two-day downtime on January.

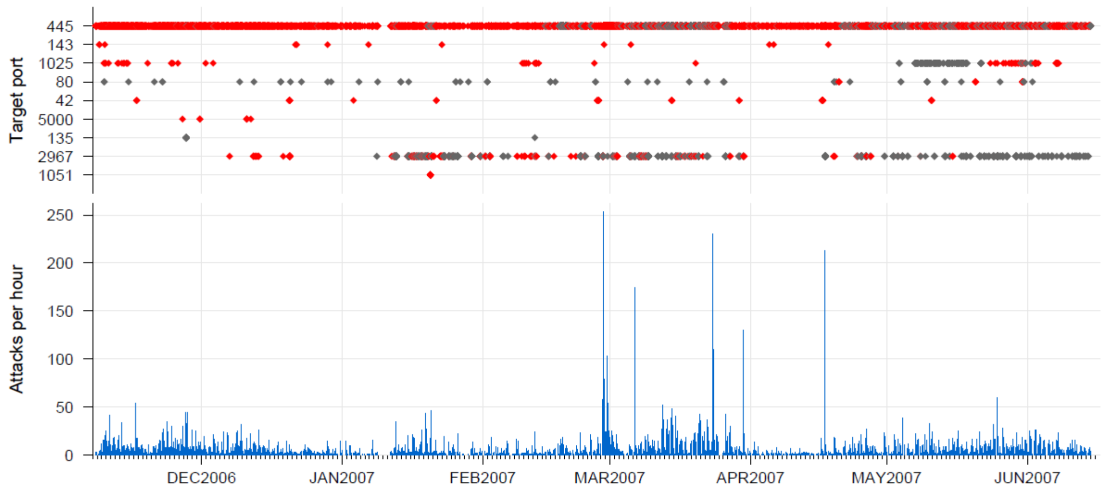


Figure 3: Overall attack activity from a real-world deployment of `nemu` in an Educational Network.

As of 14 June 2007, the detector has captured 21795 attacks targeting nine different ports. An overall view of the attack activity during these seven months is presented in Fig. 3. The upper part of the figure shows the attack activity according to the targeted port. From the 21795 attacks, 14956 (68.62%) were launched from 5747 external IP addresses (red dots), while the rest 6839 (31.38%) originated from 269 infected hosts in the monitored network (gray dots). Almost one third of the internal attacks came from a single IP address, using the same exploit against port 445. The bottom part of the figure shows the number of attacks per hour of day. There are occasions with hundreds of attacks in one hour, mostly due to bursts from a single source that horizontally attacks all active hosts in local neighboring subnets. The vast majority of the attacks (88%) target port 445. Interestingly, however, there also exist attacks to less commonly attacked ports like 1025, 1051, and 5000.We should note that for all captured attacks the emulator was able to successfully decrypt the payload, while so far has zero false positives.

For each identified attack, our prototype detector generates:

i) an alert file with generic attack information and the execution trace of the shellcode
ii) a raw dump of the reassembled TCP stream
iii) a full payload trace of all attack traffic (both directions) in libpcap format
iv) the raw contents of the modified addresses in the virtual memory of the emulator, i.e., the decrypted shellcode.

Although we have not thoroughly analyzed all captured attacks, we can get a rough estimate on the diversity of the different exploitation tools, worms, or bots that launched these attacks, based on a simple analysis of the decrypted payloads of the captured polymorphic shellcodes. Computing the MD5 hash of the decrypted payload for all above attacks resulted to 1021 unique payloads. However, grouping further these 1021 payloads according to their size, resulted to 64 different payload size groups. By manually inspecting some of the shellcodes with same or similar lengths, but different MD5 hashes, we observed that in most cases the actual payload code was the same, but the seeding URL or IP address from where the "download and execute" shellcode would retrieve the actual malware was different. Our results are in accordance with previous studies [4] and clearly show that polymorphic shellcodes are extensively used in the wild, although in most cases they employ naive encryption methods, mostly for concealing restricted payload bytes.

4.2 National Research Network

In this section we report statistics from a deployment of `nemu` in a National Research Network in Europe. The sensor has been continuously operational since 9 March 2007, except some occasional daily downtimes.

As of 13 February 2008, `nemu` has captured 1,052,332 attacks targeting 20 different ports. From these attacks, 31.35% were launched from 8981 different external IP addresses against internal hosts, while the rest 68.65% originated from 204 infected hosts in the monitored networks that were massively attempting to propagate malware. In the remaining, we focus only on the external attacks that were targeting hosts within the protected networks.

An overall view of the external attack activity is presented in Fig. 4. The upper part of the figure shows the attack activity according to the targeted port. The bottom part of the figure shows the number of external attacks per hour. Again, as in the case of the Educational Network, the ports of popular OS services associated with well-known vulnerabilities, e.g., 135, 139, and 445, receive the highest number of attacks. However, it is interesting to note that there also exist sporadic attacks to less commonly attacked ports like 1051, 3050, 30708, 41523, and so on. Table 2 presents the total number of internal and external attacks according to the targeted destination port number. With firewalls and OS-level protections now being widely deployed, attackers have turned their attention to third-party services and applications, such as corporate virus scanners, mail servers, backup servers, and DBMSes. Although such services are not very popular among typical home users, they are commonly found in corporate environments, and most importantly, they usually do not get the proper attention regarding patching, maintenance, and security hardening.

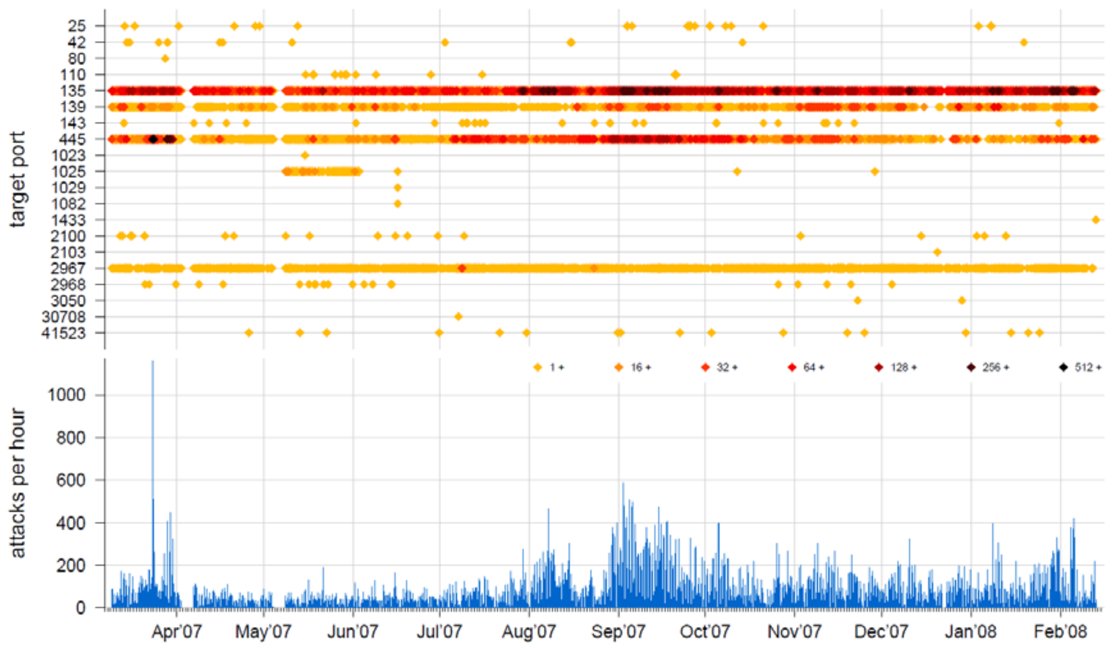


Figure 4: Overall attack activity from a deployment of nemu in a National Research Network. The graph shows only the attacks that were launched from external hosts against hosts in the protected network.

Table 2: Number of attacks per destination port number.

Internal Attacks		External Attacks	
Destination Port	**Number of Attacks**	**Destination Port**	**Number of Attacks**
135	313342	135	272134
2967	291458	445	61233
139	89651	139	19434
445	22235	2967	3142
2968	5	1025	107
		143	42
		42	36
		2100	25

		25	25
		2968	22
		41523	18
		2103	3
		1433	3
		1029	3
		1082	2
		110	2
		30708	2

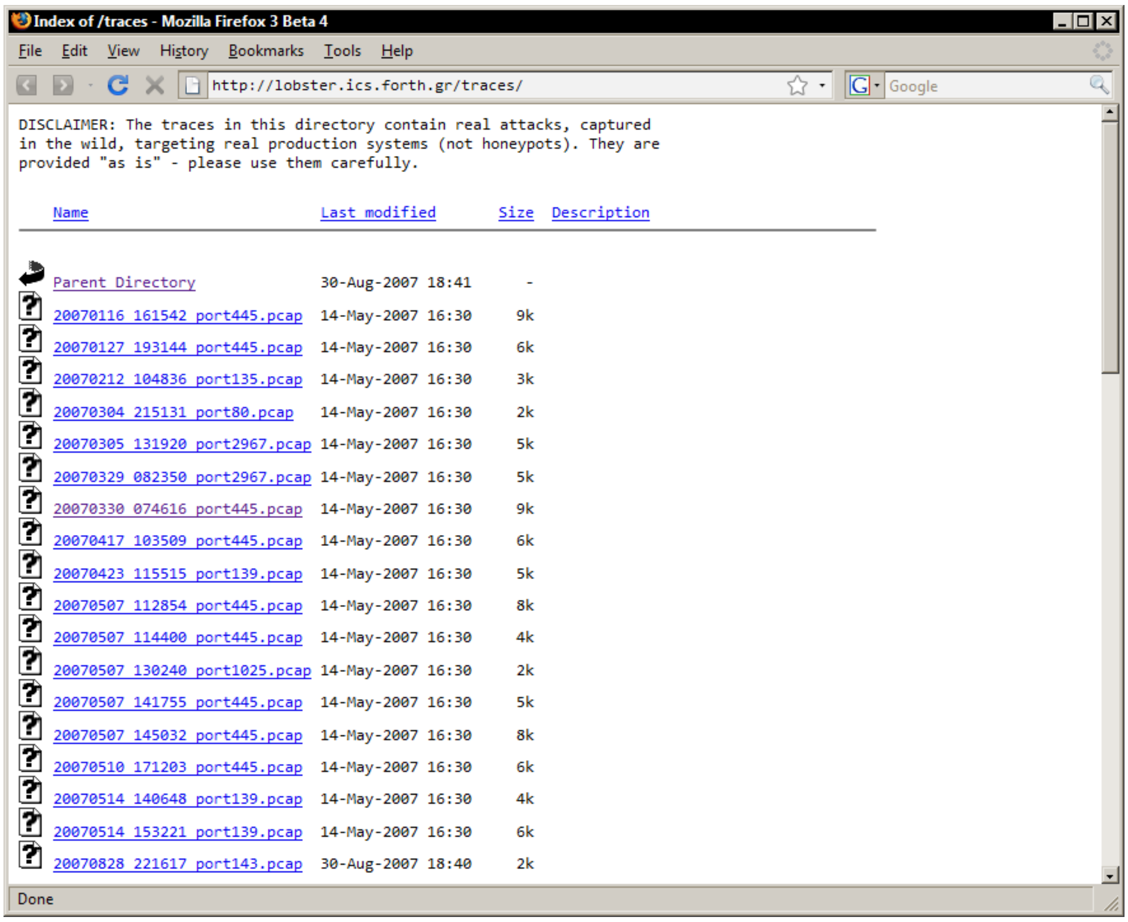
The above results clearly show that polymorphic shellcode is extensively used in the wild, although in most cases it employs naive encryption methods, mostly for concealing restricted payload bytes. However, as shown in Fig. 2, in the past few months we have observed a slight increase in the overall number of detected incidents, while there has been an increased use of more sophisticated engines and obfuscation techniques.

5 Attack Network Trace Repository

In an effort to provide useful real-world attack data to the security research community, we have created an attack network trace repository with publicly available traces of attacks captured by various installations of nemu in production networks. The attack trace repository (shown in Fig. 5) is publicly accessible from:

http://lobster.ics.forth.gr/traces/

We have focused on providing a few diverse traces of attacks against different services and using different exploits or shellcodes, rather than providing a bulk of almost identical attack instances.


Index of /traces - Mozilla Firefox 3 Beta 4
File
Edit
View
History
Bookmarks
Tools
Help
http://lobster.ics.forth.gr/traces/
Google
Done

```
DISCLAIMER: The traces in this directory contain real attacks, captured
in the wild, targeting real production systems (not honeypots). They are
provided "as is" - please use them carefully.

    Name                          Last modified       Size  Description
  -----------------------------------------------------------------------

    Parent Directory              30-Aug-2007 18:41     -
    20070116_161542_port445.pcap  14-May-2007 16:30    9k
    20070127_193144_port445.pcap  14-May-2007 16:30    6k
    20070212_104836_port135.pcap  14-May-2007 16:30    3k
    20070304_215131_port80.pcap   14-May-2007 16:30    2k
    20070305_131920_port2967.pcap 14-May-2007 16:30    5k
    20070329_082350_port2967.pcap 14-May-2007 16:30    5k
    20070330_074616_port445.pcap  14-May-2007 16:30    9k
    20070417_103509_port445.pcap  14-May-2007 16:30    6k
    20070423_115515_port139.pcap  14-May-2007 16:30    5k
    20070507_112854_port445.pcap  14-May-2007 16:30    8k
    20070507_114400_port445.pcap  14-May-2007 16:30    4k
    20070507_130240_port1025.pcap 14-May-2007 16:30    2k
    20070507_141755_port445.pcap  14-May-2007 16:30    5k
    20070507_145032_port445.pcap  14-May-2007 16:30    8k
    20070510_171203_port445.pcap  14-May-2007 16:30    6k
    20070514_140648_port139.pcap  14-May-2007 16:30    4k
    20070514_153221_port139.pcap  14-May-2007 16:30    6k
    20070828_221617_port143.pcap  30-Aug-2007 18:40    2k
```

Εικόνα 5: The attack network trace repository.

5.1 *Trace details*

All available files are full payload traces in libpcap format. Each trace corresponds to a single attack attempt and contains all packets of the network flow (5-tuple) of the particular attack instance, including the initial TCP 3-way handshake. Traces are named in the form `[date]_[time]_[dstport].pcap`, where `[dstport]` is the port number of the attacked service.

5.2 *Anonymization*

Every effort has been made to anonymize the traces and remove any sensitive personal or professional information. All traces have been anonymized using anontool [5] and netdude [6] as follows: MAC addresses have been zeroed and IP addresses have been mapped to fake addresses (usually 1.0.0.1 for the attacking host and 1.0.0.2 for the victim host). Any other payload data that could reveal the attacking or victim hosts have also been anonymized - e.g., the HTTP 'Host' filed is changed to a fake address:

```
Host: 10.123.12.123\r\n
```

while various SMB or DCERPC fields that contain IP addresses, host names, or other identifiers, are sanitized - e.g.:

```
principal: xxxxxx$@XXXXXX.XXX
Server NetBIOS Name: XXXXXX
Domain DNS Name: xxxxxx.xxx
Path: \\10.123.12.12\IPC$
```

The checksums of all modified packets have been fixed accordingly. Note that in most cases, the encrypted shellcode (which is exposed only at runtime) may contain the IP address or URL of a "seeding" host from which the actual malware executable is downloaded. We have avoided including attack traces in which the encrypted shellcode contains information about a real host. Such information cannot be easily anonymized, since it is not exposed on the wire. Thus, here you will find only attacks that use either a bindshell or similar "listening" shellcodes, or that do contain some “download and execute” shellcode, but only of instances where it (mistakenly) tried to connect to a non-existent or private address (e.g., http://0.0.0.0/foo.exe). Since most of the attacks in the wild do contain a download and execute shellcode, this severely limits the number of traces we can make available. For thir reason, we have also included a few traces in which we have manually sanitized the encrypted seeding URL (e.g., http://xxxxxx.xxx/1.exe) by reverse engineering the encryption algorithm.

6 References

1. Michalis Polychronakis, Evangelos P. Markatos, and Kostas G. Anagnostakis. Network-level polymorphic shellcode detection using emulation. *Journal in Computer Virology*, 2(4):257-274, February 2007.

2. Michalis Polychronakis, Evangelos P. Markatos, and Kostas G. Anagnostakis. Emulation-based detection of non-self-contained polymorphic shellcode. In *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2007.

3. F. Bellard. QEMU, a fast and portable dynamic translator. In *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

4. J. Ma, J. Dunagan, H. J. Wang, S. Savage, and G. M. Voelker. Finding diversity in remote code injection exploits. In *Proceedings of the 6th ACM SIGCOMM on Internet measurement (IMC)*, pages 53–64, 2006.

5. Anontool. http://www.ics.forth.gr/dcs/Activities/Projects/anontool.html

6. NetDude. http://netdude.sourceforge.net/