Real-world Polymorphic Attack Detection using Network-level Emulation

Michalis Polychronakis
FORTH-ICS, Greece
mikepo@ics.forth.gr

Kostas G. Anagnostakis
I^2R, Singapore
kostas@i2r.a-star.edu.sg

Evangelos P. Markatos
FORTH-ICS, Greece
markatos@ics.forth.gr

ABSTRACT

As state-of-the-art attack detection technology becomes more prevalent, attackers have started to employ techniques such as code obfuscation and polymorphism to defeat these defenses. We have recently proposed *network-level emulation*, a heuristic detection method that scans network traffic to detect polymorphic attacks. Our approach uses a CPU emulator to dynamically analyze every potential instruction sequence in the inspected traffic, aiming to identify the execution behavior of certain malicious code classes, such as self-decrypting polymorphic shellcode.

Network-level emulation does not rely on any exploit or vulnerability specific signatures, which allows the detection of previously unknown attacks, while the actual execution of the attack code makes the detector robust to evasion techniques such as self-modifying code. After more than a year of continuous operation in production networks, our prototype implementation has captured more than a million attacks against real systems, employing a highly diverse set of exploits, often against less widely used vulnerable services, while so far has not resulted to any false positives.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network Monitoring*

General Terms

Measurement, Security

Keywords

Polymorphism, shellcode, emulation, intrusion detection

1. INTRODUCTION

Along with the phenomenal growth of the Internet, the number of attacks against Internet-connected systems continues to grow at alarming rates [5, 6]. Besides the constantly increasing number of security incidents, we have also been witnessing a steady increase in attack sophistication and diversity. During the last few years, there has been a decline in the number of massive, easy-to-spot global worm epidemics, and a shift towards more stealthy and localized attacks against selected targets.

The constant increase in the amount, sophistication, and diversity of remote system compromise attacks, and the consequent increase in the deployment and effectiveness of defenses, have resulted in an arms race between attack detection and evasion techniques. As detection mechanisms improve, attackers employ increasingly sophisticated methods to evade them. Techniques such as code obfuscation and polymorphism pose significant challenges to existing network-level detectors. Using polymorphism, the code in the attack vector—which is usually referred to as *shellcode*—is mutated so that each instance of the same attack acquires a unique byte pattern, thereby making fingerprinting of the whole breed very difficult. At the same time, accurate attack fingerprinting is getting increasingly important for the already inherently hard problem of identifying previously unknown attacks, also known as *zero-day* attacks, while trying to minimize the rate of false positives.

A major outstanding question in security research and engineering is thus whether we can proactively develop the tools needed to contain advanced polymorphic attacks. While recent results have been promising, most of the existing proposals can be defeated using only minor enhancements to the attack vector. In fact, some publicly-available polymorphic shellcode engines are currently one step ahead of the most advanced publicly-documented network-level detectors [3].

Along with the several research efforts towards this goal, we have recently proposed network-level emulation [3, 4], a passive network monitoring approach for the detection of zero-day polymorphic attacks. In contrast to previous work, network-level emulation uses a CPU emulator to dynamically analyze every potential instruction sequence in the inspected traffic, aiming to identify the execution behavior of certain malicious code classes, such as self-decrypting polymorphic shellcode. Network-level emulation does not rely on any exploit or vulnerability specific signatures, which allows the detection of previously unknown attacks, while the actual execution of the attack code on the emulator makes the detector robust to evasion techniques such as self-modifying code. Furthermore, each input is inspected autonomously, making the approach effective against targeted attacks.

We have deployed our prototype implementation, called

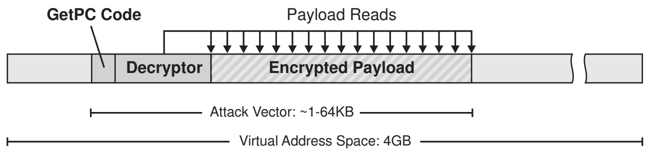

Figure 1: A typical execution of a polymorphic shellcode using network-level emulation.

`nemu`, as part of LOBSTER [2, 1], a large-scale distributed passive network monitoring infrastructure. LOBSTER has installed more than 50 passive monitoring sensors across Europe, providing an advanced pilot platform for network performance and security applications. In an effort to foster research on network monitoring, LOBSTER also makes publicly available network statistics and captured data, after being properly anonymized using the data anonymization tools provided by the platform.

After almost a year of continuous operation, `nemu` has detected more than a million attacks against real systems (not honeypots) in the monitored networks, while so far has not resulted to any false positives. Besides common exploits against popular OS services associated with multiple well known vulnerabilities, we have witnessed sporadic attacks against less widely used services and third-party applications. At the same time, although the bulk of the attacks use naive encryption or polymorphism techniques, there is an increase in the number of attacks that employ more sophisticated obfuscation schemes. These observations are indicative of the change in attackers' tactics and goals.

2. NETWORK-LEVEL EMULATION

The principle behind network-level emulation is that the machine code interpretation of arbitrary data results to random code, which, when it is attempted to run on an actual CPU, usually crashes soon, e.g., due to the execution of an illegal instruction. In contrast, if some network request actually contains polymorphic shellcode, then the shellcode runs normally, exhibiting a certain detectable behavior.

`Nemu` inspects the client-initiated data of each network flow, which may contain malicious requests towards vulnerable services. Any server-initiated data, such as the content served by a web server, are ignored. For TCP packets, the application-level stream is reconstructed using TCP stream reassembly. In case of large client-initiated streams, e.g., due to file uploads, only the first 64KB of the stream are inspected. Each input is mapped to a random memory location in the virtual address space of an IA-32 emulator, as shown in Fig. 1. Since the exact location of the shellcode in the input stream is not known in advance, the emulator repeats the execution multiple times, starting from each and every position of the stream, although in certain cases some the execution of some code paths can be skipped to optimize runtime performance [4].

Before the beginning of a new execution, the state of the CPU is randomized, while any accidental memory modifications in the addresses where the attack vector has been mapped to are rolled back after the end of each execution. Since the execution of random code sometimes may not stop soon, e.g., due to the accidental formation of loop structures that may execute for a very large number of iterations, if the number of executed instructions in some execution chain

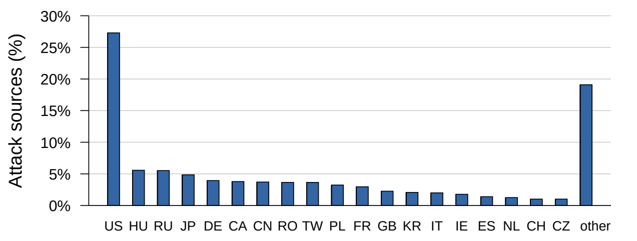

Figure 2: Distribution of attack sources according to country of origin. Category "other" includes 98 countries with less than 1% of the total sources each.

reaches a certain *execution threshold*, then the execution is terminated.

The execution of polymorphic shellcode is identified by two key runtime behavioral characteristics: the execution of some form of *GetPC* code, and the occurrence of several read operations from the memory addresses of the input stream itself, as illustrated in Fig 1. The GetPC code is used by the shellcode for finding the absolute address of the injected code, which is mandatory for subsequently decrypting the encrypted payload, and involves the execution of an instruction from the `call` or `fstenv` instruction groups [3].

3. 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 four deployments in three National Research Networks and one Educational Network across Europe. The sensors have 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 21 different port numbers. 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. The distribution of external attack source addresses according to country of origin is presented in Fig. 2. In the remaining, we focus only on these external attacks that were targeting hosts within the protected networks.

An overall view of the external attack activity is presented in Fig. 3. The upper part of the figure shows the attack activity according to the targeted port. The bottom part shows the number of external attacks per hour. There are occasions with several hundreds of attacks in one hour, mostly due to bursts from a single source which attacks all active hosts in local neighboring subnets.

As expected, the port numbers 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, 5000, 41523, and so on. 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,

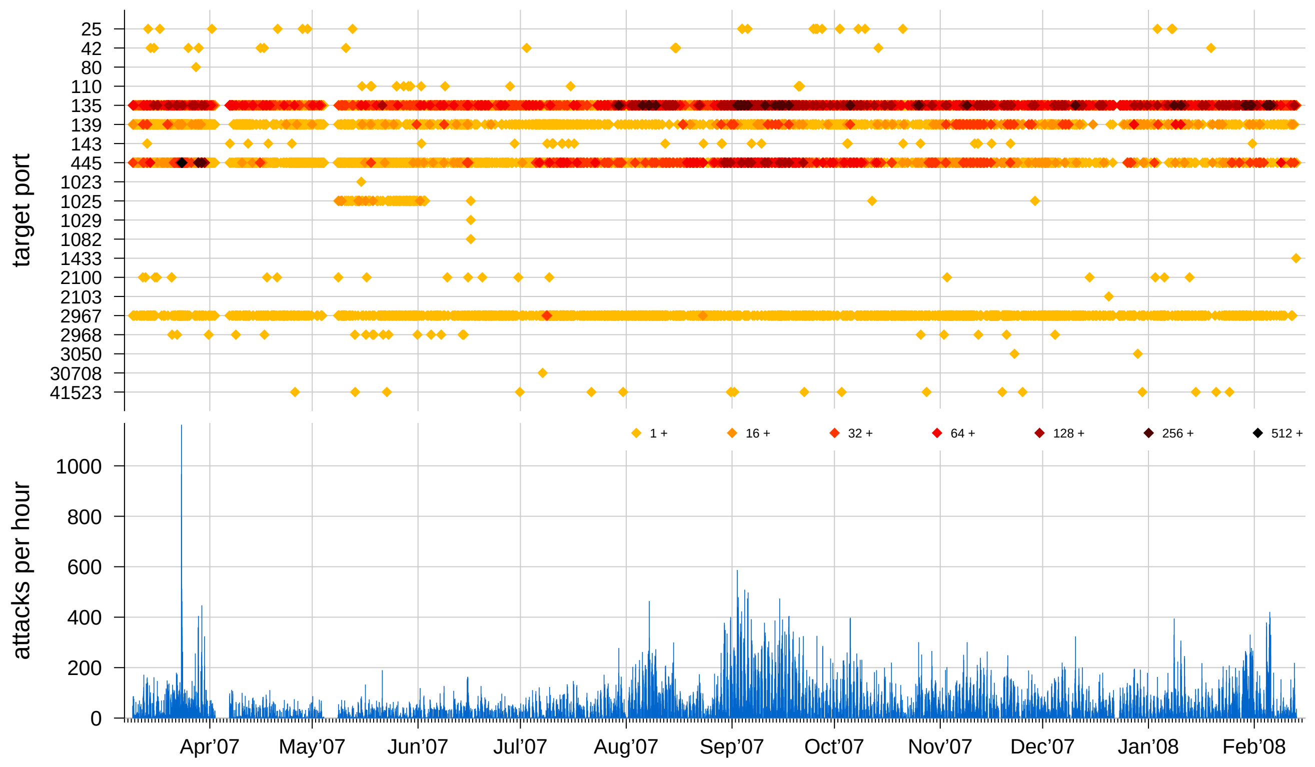

Figure 3: Overall attack activity from four deployments of nemu. The graph shows only the attacks that were launched from external hosts against hosts in the protected networks. Although the bulk of the attacks target well known vulnerable services, there are also sporadic attacks against less widely used services.

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.

`Nemu` scans the traffic towards any service and does not rely on exploit or vulnerability specific signatures, thus it is capable to detect polymorphic attacks destined to even less widely used or "forgotten" services. We should note that for all captured attacks, `nemu` was able to successfully decrypt the attack payload, while so far has zero false positives. For each identified attack, `nemu` generates i) an alert 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,[1] and iv) the raw contents of the modified locations in the virtual memory of the emulator, i.e., the decrypted shellcode.

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. 3, 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.

[1]Anonymized full payload traces of some of the captured attacks are available from `http://lobster.ics.forth.gr/traces/`

4. REFERENCES

[1] LOBSTER Project. `http://www.ist-lobster.org/`.

[2] D. Antoniades, M. Polychronakis, A. Papadogiannakis, P. Trimintzios, S. Ubik, V. Smotlacha, A. Øslebø, and E. P. Markatos. LOBSTER: A european platform for passive network traffic monitoring. In *Proceedings of the 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TRIDENTCOM)*, March 2008.

[3] M. Polychronakis, E. P. Markatos, and K. G. Anagnostakis. Network-level polymorphic shellcode detection using emulation. In *Proceedings of the Third Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, July 2006.

[4] M. Polychronakis, E. P. Markatos, and K. 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.

[5] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadug. The Ghost In The Browser: Analysis of Web-based Malware. In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets (HotBots)*, 2007.

[6] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: global characteristics and prevalence. In *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2003.