Real-world Detection of Polymorphic Attacks

M. Polychronakis[1], K.G. Anagnostakis[2] and E.P. Markatos[1]

[1]Institute of Computer Science, Foundation for Research & Technology – Hellas,
[2]Institute for Infocomm Research, Singapore
e-mail: {mikepo, markatos}@ics.forth.gr, kostas@i2r.a-star.edu.sg

Abstract

As state-of-the-art attack detection technology becomes more prevalent, attackers have started to employ evasion techniques such as code obfuscation and polymorphism to defeat existing 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. In this paper, we present results and experiences from deployments of network-level emulation in production networks. After more than a year of continuous operation, our prototype implementation has captured more than a million attacks against real systems, while so far has not resulted to any false positives. The observed attacks employ a highly diverse set of exploits, often against less widely used vulnerable services, and in some cases, sophisticated obfuscation schemes.

Keywords

Polymorphism, intrusion detection, code emulation

1. Introduction

The number of attacks against Internet-connected systems continues to grow at alarming rates (Provos et al., 2007, Yegneswaran et al., 2003). Along with the more recently popularized client-side attacks that exploit vulnerabilities in users' software such as browsers and media players (Provos et al., 2008), remote code execution vulnerabilities continue to plague even the latest versions of popular OSes and server applications (Microsoft Corp. web site, 2008) and are effectively being exploited by malware, resulting in millions of infected hosts (F-Secure web site, 2009).

Besides the constantly increasing number of security incidents, we have also been witnessing a steady increase in attack sophistication and diversity. Motivated by the illicit financial gain against their victims, cyber-criminals constantly try to improve the effectiveness and evasiveness of their attacks, with the aim to compromise as many systems as possible and keep them under control for as long as possible. As detection mechanisms improve, attackers employ increasingly sophisticated methods to evade them. Techniques such as code obfuscation and polymorphism (Szor, 2005) 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. Polymorphic shellcode engines (Metasploit Web Site, 2009, Bania 2005, Detristan et al., 2003, K2, 2001, Rix, 2001, Wever, 2004) create different mutations of the same initial shellcode—which is also known as the payload—by encrypting it with a different random key, and prepending to it a decryption routine that on runtime decrypts and executes the encrypted payload. Since the decryption code itself cannot be encrypted, advanced polymorphic encoders also mutate the exposed part of the shellcode using metamorphism (Szor, 2005).

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.

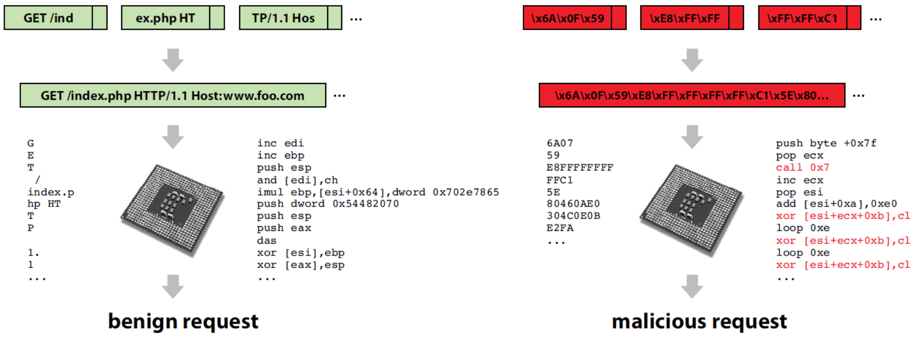
Along with the several research efforts towards this goal, we have recently proposed network-level emulation (Polychronakis et al., 2006, Polychronakis et al., 2007), a passive network monitoring approach for the detection of zero-day polymorphic attacks. In contrast to previous work, network-level emulation 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 `nemu`, in research and educational networks across Europe. After more than a year of continuous operation, `nemu` has detected more than a million attacks against real systems in the monitored networks, while so far has not resulted to any false positives. In this work, we present an analysis of more than 1.2 million polymorphic code injection attacks against real Internet hosts—not honeypots—detected over the course of more than 20 months in the above deployments. Besides common exploits against popular OS services associated with multiple well known vulnerabilities, we 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, we observed a few attacks employing more sophisticated obfuscation schemes.

2. Network-level Emulation

We briefly describe some aspects of the network-level emulation detection technique. The interested reader is referred to our previous work (Polychronakis et al., 2006, Polychronakis et al., 2007) for a thorough description of the approach and its implementation details.

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, as illustrated in Figure 1.



```
G
E
T
 /
index.p
hp HT
T
P

1.
1
...
```

```
inc edi
inc ebp
push esp
and [edi],ch
imul ebp,[esi+0x64],dword 0x702e7865
push dword 0x54482070
push esp
push eax
das
xor [esi],ebp
xor [eax],esp
...
```

```
6A07
59
E8FFFFFFFF
FFC1
5E
80460AE0
304C0E0B
E2FA
...
```

```
push byte +0x7f
pop ecx
call 0x7
inc ecx
pop esi
add [esi+0xa],0xe0
xor [esi+ecx+0xb],cl
loop 0xe
xor [esi+ecx+0xb],cl
loop 0xe
xor [esi+ecx+0xb],cl
...
```

Figure 1: Overview of network-level emulation.

After TCP stream reassembly, each network request is interpreted as machine code and is loaded on a CPU emulator. The execution of the random code corresponding to a benign request usually ends abruptly after a few instructions, while the execution of an actual polymorphic shellcode exhibits 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 Figure 2. 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 (Polychronakis et al., 2007).

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

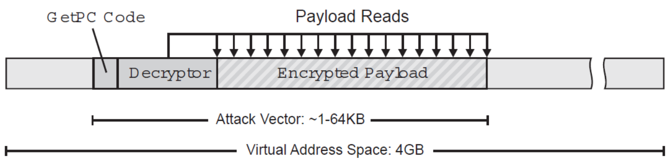


Figure 2: A typical execution of a polymorphic shellcode using network-level emulation.

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 Figure 2. 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 (Polychronakis et al., 2006).

Network	**Total # attacks**	**External**			**Internal**		
		#attacks	**#srcIP**	**#dstIP**	**#attacks**	**#srcIP**	**#dstIP**
NRN1	1240716	396899 (32.0%)	10014	769	843817 (68.0%)	143	331572
NRN2	12390	2617 (21.1%)	1043	82	9773 (78.9%)	66	4070
NRN3	1961	441 (22.5%)	113	49	1520 (77.5%)	8	1518
EDU	20516	13579 (66.2%)	3275	410	6937 (33.8%)	351	2253

Table 1: Number of captured attacks from four deployments of `nemu`.

3. Data Set

Our analysis is based on the attacks captured by `nemu` in three deployments in European National Research Networks (referred to as NRN1-3) and one deployment in a public Educational Network in Greece (referred to as EDU). In each installation, `nemu` runs on a passive monitoring sensor that inspects all the traffic of the access link that connects the organization to the Internet. The sensors were continuously operational for more than a year, except some occasional downtimes. The exact duration of each deployment was March 2007 – October 08 for NRN1 and EDU, and March 2007 – February 2008 for NRN2 and NRN3. Details about the exact number of detected attacks in each deployment, along with the number of attack sources and destinations, is shown in Table 1. In these four deployments, `nemu` collectively captured more than 1.2 million attacks targeting real production systems in the monitored networks.

We differentiate between external attacks, which originate from external IP addresses and target hosts within the monitored network, and internal attacks, which originate from hosts within the monitored networks. Internal attacks usually come from infected PCs that massively attempt to propagate malware in the local network and the Internet. We should note that due to NAT, DHCP, and the long duration of the data collection, a single IP may correspond to more than one physical computer.

4. Overall Attack Activity

As shown in Table 1, from the 1.240.716 attacks detected in NRN1, about one third of them were launched from 10.014 external IP addresses and targeted 769 hosts within the organization. The bulk of the attacks originated from143 different internal hosts, targeting 331.572 different active hosts across the Internet. Interestingly, 116 of the 143 internal hosts that launched attacks are also among the 769 victim hosts, indicating that possibly some of the detected attacks were successful.

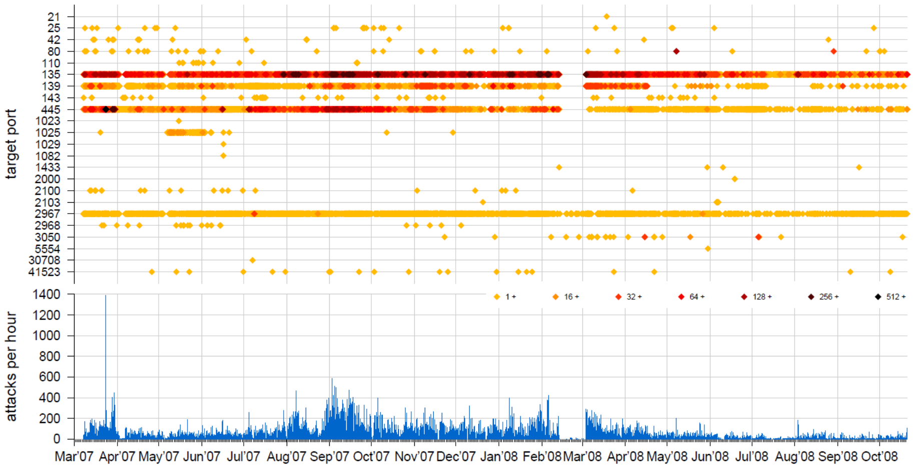


Figure 3: Overall external attack activity. Although the bulk of the attacks target well known vulnerable services, there are also sporadic attacks against less widely used services.

The overall attack statistics for NRN2 and NRN3 are similar to NRN1, but the number of detected attacks is orders of magnitude smaller, due to the smaller attack surface of infected or potentially vulnerable internal hosts that launched or received attacks. In contrast to the three NRNs, about two thirds of the attacks captured in the EDU deployment originated from external hosts.

An overall view of the external and internal attack activity for all deployments is presented in Figure 3 and Figure 4, respectively. In both figures, the upper part shows the attack activity according to the targeted port, while the bottom part shows the number of attacks per hour. For the targeted ports, the darker the color of the dot, the larger the number of attacks targeting this port in that hour. There are occasions with hundreds of attacks in one hour, mostly due to attack bursts from a single source that target all active hosts in neighboring subnets.

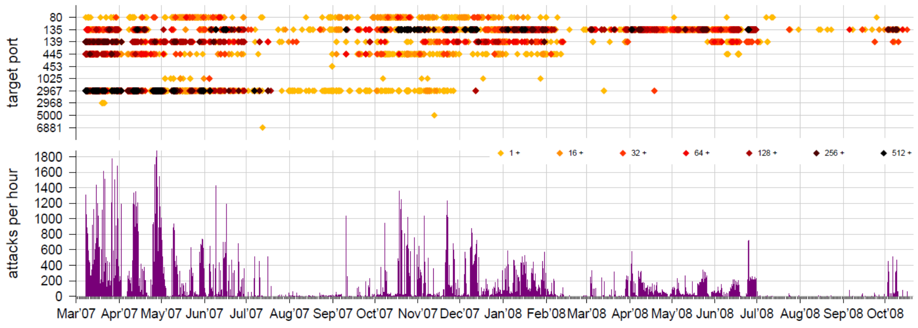


Figure 4: Overall internal attack activity.

5. Attack Analysis

As expected, the most highly attacked ports for both internal and external attacks include ports 135, 139, and 445, which correspond to Windows services that have been associated with multiple vulnerabilities and are still being highly exploited in the wild. Actually, the second most attacked port is port 2967, which is related to an exploit against a popular corporate virus scanner that happened to be installed in many hosts of the monitored networks. As shown in Figure 4, several of these hosts got infected before the patch was released and were constantly propagating the attack for a long period. Other commonly attacked services include web servers (port 80) and mail servers (port 25).

It is interesting to note that there also exist sporadic attacks to many less commonly attacked ports like 3050, 5000, and 41523. With firewalls and OS-level protections now being widely deployed, attackers have started turning their attention to third-party services and applications, such as virus scanners, mail servers, backup servers, and DBMSes. Although such services are not very popular among typical home users, they are often found in corporate environments, and most importantly, they usually do not get the proper attention regarding patching, maintenance, and security hardening. Thus, these services have become attackers' next target option for remote system compromise, and as the above results show, many such exploits have been actively used in the wild. `Nemu` scans the traffic towards any port 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.

Overall, the captured attacks targeted 26 different ports. The number of attacks per port is shown in Figure 5 (dark bars). A large number of attacks targeted port 1025, attempting to exploit the Microsoft Windows RPC malformed message buffer overflow vulnerability. Less commonly attacked services include POP3 and IMAP servers (ports 110 and 143), Oracle XDB FTP servers (port 2100), the Windows Internet Naming Service (WINS) (port 42), Microsoft SQL servers (port 1433), Creative Server (port 453), ShixxNOTE 6.net messenger (port 2000), Microsoft Message Queuing service (2103), Borland InterBase database server (port 3050), and the CA BrightStor Agent for Microsoft SQL Server (port 41523). The attack against port 5000 was related to a vulnerability in the Windows XP Universal Plug and Play implementation, while the attack against port 6881 attempted to exploit a vulnerable P2P file sharing program. The single attack against port 5554 was launched by W32.dabber, a worm that propagates by exploiting a vulnerability in the FTP server started by W32.Sasser and its variants. Other incidents involving vulnerable malware include attacks against W32.Sasser's FTP server (port 1023) and the WinHole trojan (port 1082).

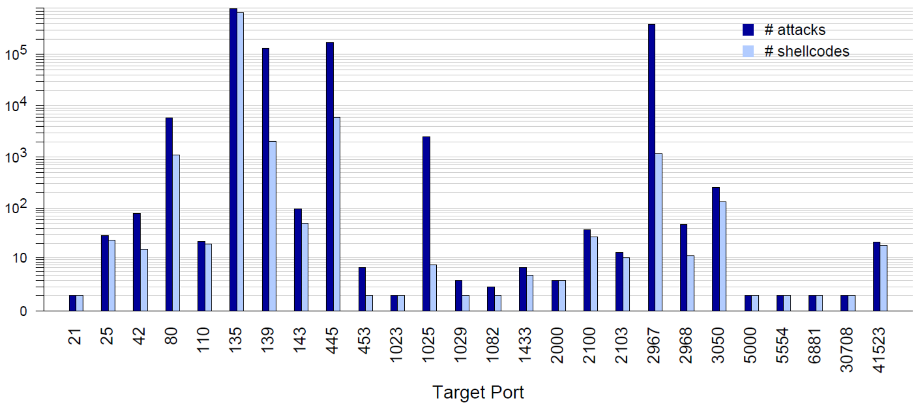


Figure 5: Number of attacks and unique shellcodes for different ports.

For each attack, we computed theMD5 hash of the initial shellcode as seen on the wire and plotted the number of unique shellcodes per port in Figure 5 (light bars). Comparing the dark and light bars, we see that in some cases the number of unique shellcodes is quite smaller than the number of attacks. If truly polymorphic shellcode were used, we would expect the number of shellcodes to be equal to the number of attacks, since each instance of a polymorphic shellcode is different than any other instance. However, in most attacks the encryption scheme is very simple, and for the same malware family, the generated shellcodes usually have been encrypted using the same key and carry the same decryption routine. Besides code obfuscation, even such naively applied encryption is convenient for the avoidance of NULL, CR, LF, and depending on the exploit, other restricted bytes that should not be present in the attack vector, since this can be taken care of by the encryption engine (Metasploit Web Site, 2009).

Three of the attacks against port 3050 are particularly interesting due to the increased sophistication of the encryption scheme used. The shellcode is encrypted with a variant of the `alpha mixed` encoder from Metasploit, which produces alphanumeric mixed-case shellcode, with some differences in the GetPC code (the decryption loops are identical). The interesting aspect of these particular attacks is that the decrypted payload produced by the alphanumeric shellcode is again an instance of a self-decrypting shellcode, this time generated by a variant of the popular countdown encoder from Metasploit. That is, the initial payload was first encoded using a countdown-like encoder, and the resulting shellcode was then encoded using an alpha mixed-like encoder. The overall decryption process of the shellcode is illustrated in Figure 6. Although such layered encryption using multiple executable packers is commonly found in malware binaries, we are not aware of any previous report of in-the-wild attacks employing doubly encrypted shellcode.

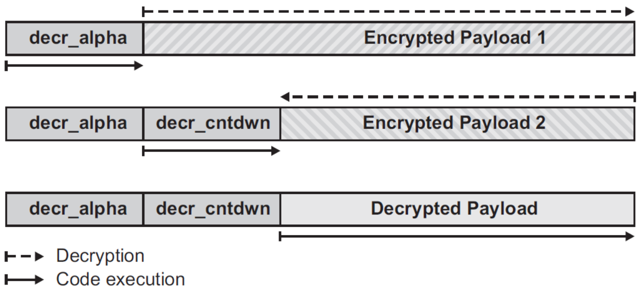


Figure 6: An illustration of the execution of the doubly encrypted shellcode found in three of the attacks.

6. Conclusion

In this paper, we have presented our experiences from the deployment of network-level emulation in research and educational networks. `Nemu`, our prototype implementation, uses a CPU emulator to dynamically analyze every potential instruction sequence in the inspected traffic and identify the execution behavior of self-decrypting shellcode. After more than a year of continuous operation, `nemu` has captured more than a million attacks that employed a highly diverse set of exploits, while so far has not resulted to any false positives.

The attack activity observed in these three networks clearly shows that polymorphic attacks are extensively used in the wild, although polymorphism is mostly employed in its simplest for just for concealing restricted payload bytes. Considering the wide availability of sophisticated polymorphic shellcode engines, this probably indicates that attackers are satisfied with the effectiveness of current shellcode, and they do not need to bother with more complex encryption schemes for evading existing network-level defenses. However, attackers have also turned to the exploitation of less widely used services and third-party applications, while we observed attacks employing more sophisticated encryption schemes, such as doubly-encrypted shellcode. It is thus not unlikely that in the future the use of advanced polymorphic shellcode engines will be commonplace, as has already happened with executable packers, which are nowadays widely used by malware.

7. Acknowledgments

Michalis Polychronakis and Evangelos Markatos are also with the University of Crete. Kostas Anagnostakis is also with FORTH-ICS.

8. References

Bania, P. TAPiON, 2005. http://pb.specialised.info/all/tapion/.

Detristan, T., Ulenspiegel, T., Malcom, Y. and Underduk, M. Polymorphic shellcode engine using spectrum analysis. *Phrack*, 11(61), Aug. 2003.

F-Secure. Calculating the Size of the Downadup Outbreak, Jan. 2009. http://www.f-secure.com/weblog/archives/00001584.html.

K2. ADMmutate, 2001. http://www.ktwo.ca/ADMmutate-0.8.4.tar.gz.

The Metasploit Project. http://www.metasploit.com/.

Microsoft Corp. Microsoft Security Bulletin MS08-067 – Critical, Oct. 2008. http://www.microsoft.com/technet/security/Bulletin/MS08-067.mspx.

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

Polychronakis,M., Markatos, E. and Anagnostakis, K. 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.

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

Provos, N., Mavrommatis,P., Rajab, M. and Monrose, F. All your iFRAMEs point to us. In *Proceedings of the 17th USENIX Security Symposium*, pages 1–16, 2008.

Rix. Writing ia32 alphanumeric shellcodes. *Phrack*, 11(57), Aug. 2001.

Szor, P. *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, February 2005.

Wever, B. Alpha 2, 2004. http://www.edup.tudelft.nl/˜bjwever/src/ alpha2.c.

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