Offensive Software Exploitation

SEC-300-01/CSI-301-02

Ali Hadi

@binaryz0ne

Shellcode

```
/* the Aleph One shellcode */
"\x31\xc0\x31\xdb\xb0\x17\xcd\x80\xeb\x1f\x5e\x89"
"\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb"
"\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh";
```

Shellcode?

- Small piece of code used as the payload in the exploitation of a software vulnerability.

- Problems of writing shellcodes:
  - Not easy to write
  - Architecture and OS dependent
  - Must remove all string-delimiting characters

System Calls

- Kernel trap calls used by user-space programs to access kernel-space functions.

- Linux:
  - INT \x80, Sysenter, etc
- Windows
  - INT 0x2e, Sysenter, DLL(s), API(s), etc

- System Call # stored in EAX.
  - 1st ARG in EBX, 2nd in ECX, and so on.

Shellcode Basics

- Spawning the process
  - Linux/Unix: execve
  - Windows: CreateProcess
- How child process deals with input and output is very important
- File descriptors (regardless of OS):
  - 0 for Standard Input (stdin)
  - 1 for Standard Output (stdout)
  - 2 for Standard Error (stderr)

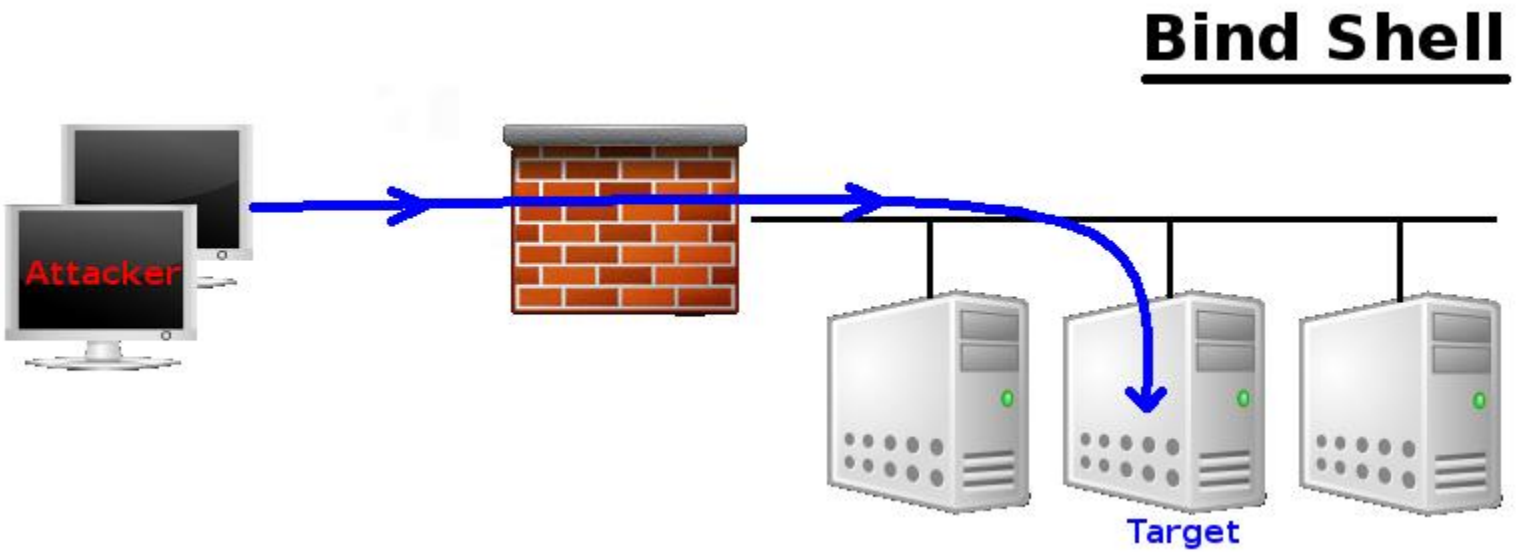
Shellcode Types

- Port Binding
- Reverse
- Find Socket
- Command Execution Code
- File Transfer
- Multistage
- System Call Proxy
- Process Injection
- Kernel Space

Port Binding Shellcode

- AKA "bind shell"
- Why/When to use this type of S.C.?
- What it does:
  - Create TCP socket
  - Bind socket to port (hardcoded and specified by the attacker)
  - Make socket Listen
  - Dup listening socket onto stdin, stdout, and stderr
  - Spawn command shell (bash, cmd.exe, etc)
- Attacker connects to that port to get control
- Problems:
  - Firewalls
  - Not Invisible
  - Can't distinguish between connections made to it

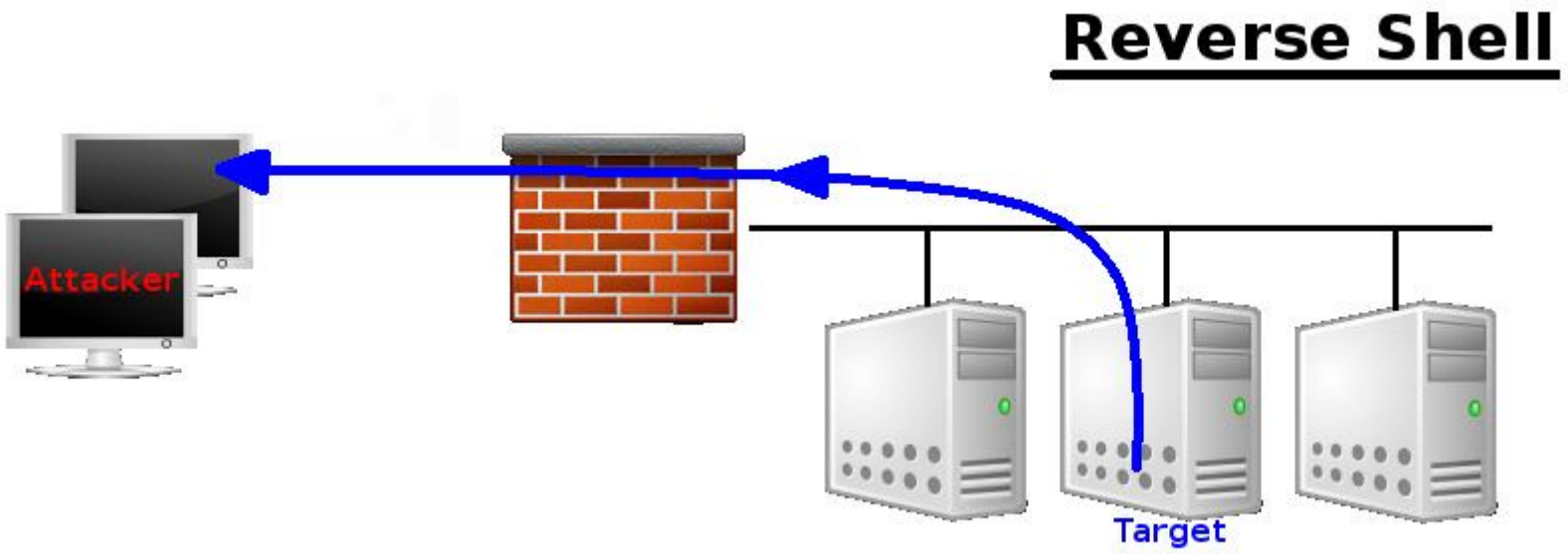
Port Binding Shellcode – Cont.



Reverse Shellcode

- AKA 'callback shellcode", solves bind shell problems
- Why connect to the target, were we can make the target connect to us?
- What it does:
  - Create TCP socket
  - Make socket connect back to the attacker on IP+Port (hardcoded and specified by the attacker)
  - Connect to the IP and port
  - Dup the socket onto stdin, stdout, and stderr
  - Spawn command shell (bash, cmd.exe, etc)
- Problems
  - Outbound Filtering
  - Attacker must be listening on the specified port
  - Attacker behind NAT
  - Target behind some proxy
  - Not invisible too

Reverse Shellcode – Cont.



Find Socket Shellcode

- Search for the file descriptor that represents attackers connection
  - POSIX (File descriptors)
  - Windows (File Handlers)
- Query each descriptor to find which is remotely connected to the attackers computer
- Hardcode the outbound port into the shellcode, makes find much easier on target
- No new network connection (*hard to detect*)!

Find Socket Shellcode – Cont.

- Steps:
  - Find file descriptor for the network connection.
  - Duplicate the socket onto stdin, stdout, and stderr.
  - Spawn a new command shell process (will use original socket for I/O).

- Problem:
  - Attacker behind NAT device, can't control the outbound port from which his connection originated (P.S. won't know what file descriptor is used for his connection!)

Command Execution Shellcode

- Why create a network session when all needed to do is run a command?
  - ssh-copy-id to target
  - Adding/modifying a user account
  - Modify configuration file

- Steps:
  - Assemble command name
  - Assemble arguments required (if any!)
  - Invoke system call to execute the command
- *Often very small*

File Transfer Shellcode

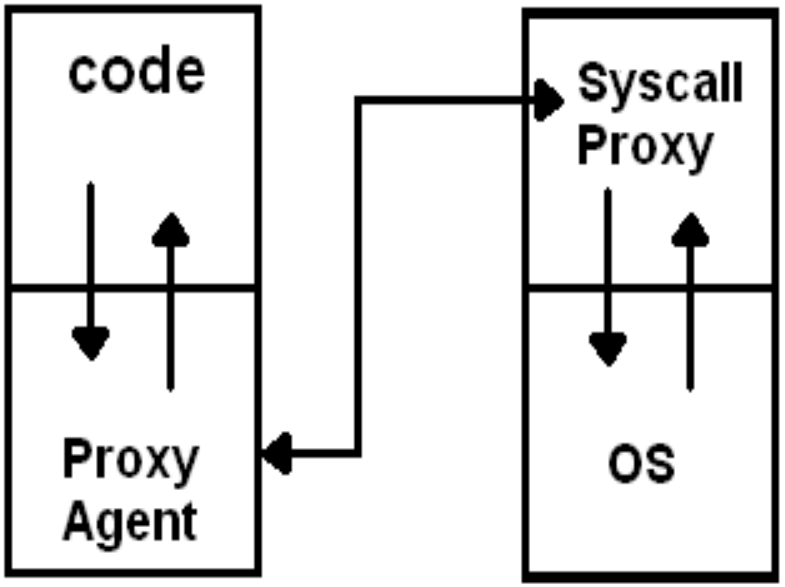
- Very simple, all needed is to upload a file to the target
- Steps:
  - Open new file on target
  - Read data from the network connection, and write it to the opened file (Note: connection obtained using previous discussed network shellcodes)
  - Repeat RW until file successfully transferred.
  - Close the open file
- Can be combined with a CmdExec Shellcode

Multistage Shellcode

- Vulnerability contains un-sufficient space for injecting shellcode
- Consist of 2 or more shellcode stages
- Steps:
  - Stage1:
    - read more shellcode,
    - pass control to Stage2 shellcode
  - Stage2: accomplish the functionality required

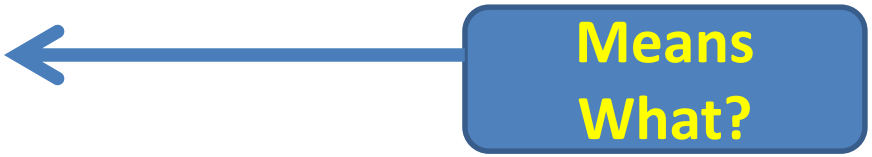
System Call Proxy Shellcode

- AKA Syscall Proxy
- Technique first introduced by Maximiliano Caceres (CORE Impact creators) which can provide a real remote interface to the target's kernel
- Local process running has no idea it is running remotely!
- Syscall proxy payload can continue to run in the context of the exploited process.



System Call Proxy – Cont.

- Use many tools without installing anything on the target machine
- Memory resident
- Kernel Interface
- Request Local, Execute Remote
- Remote Debugging
- Others? use your own imagination!



Process Injection Shellcode

- Loading libraries of code running under a separate thread of execution within the context of an existing process on the target.
- Host process can be:
  - Process exploited.
  - Migrate to a complete different process.

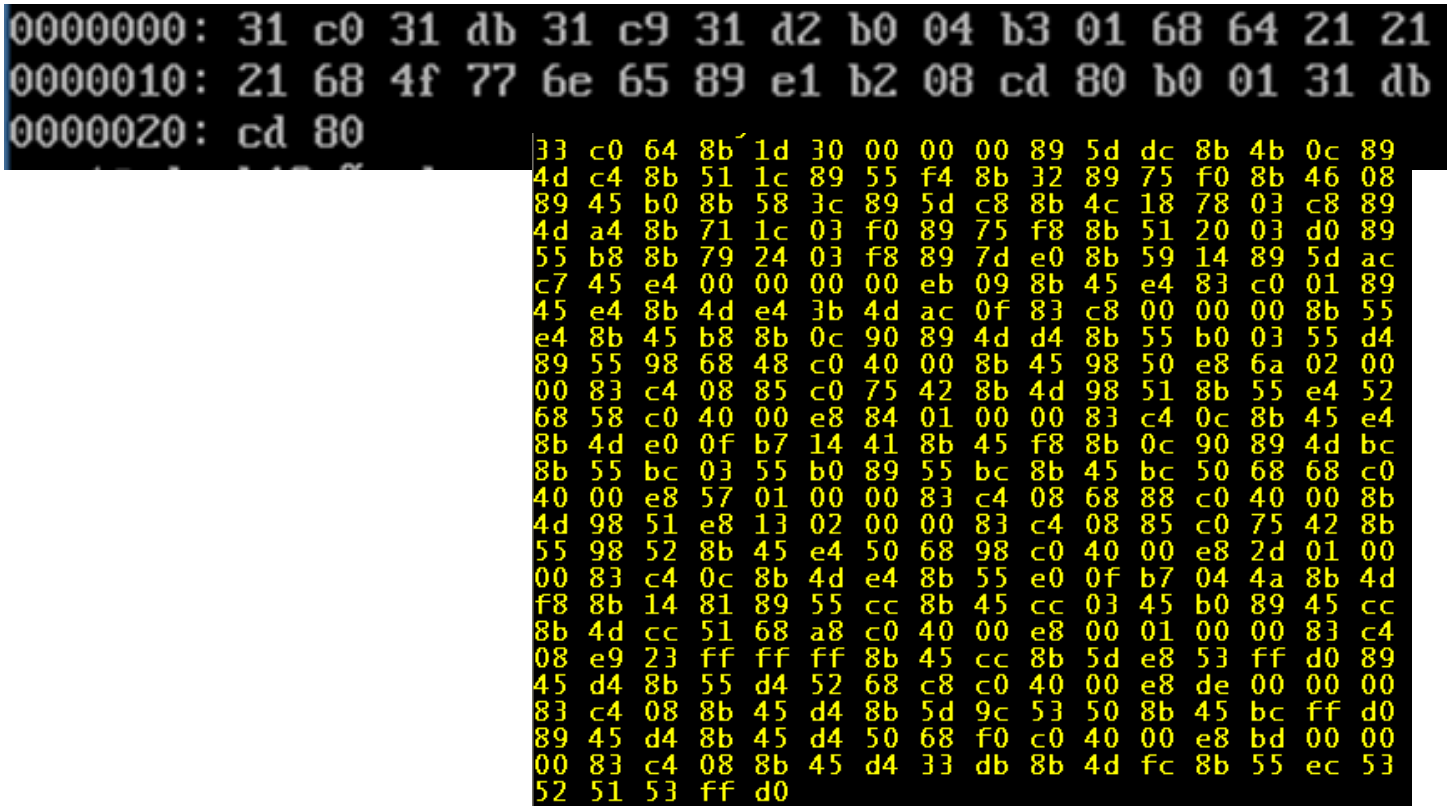
- Injected library might never get written to the hard drive and harness in memory (*hard even for forensics to discover*)
  - Ex: Metasploit's Meterpreter (later).

Ultimate Goal

- Our goal in exploit development is always ***arbitrary code execution***, so its time to get familiar with Windows shellcode architecture.
- Windows shellcode is brutally complicated compared to Linux shellcode, so prepare for battle.

Linux vs Windows Shellcode

Cited [1]



```
0000000: 31 c0 31 db 31 c9 31 d2 b0 04 b3 01 68 64 21 21
0000010: 21 68 4f 77 6e 65 89 e1 b2 08 cd 80 b0 01 31 db
0000020: cd 80
```

```
33 c0 64 8b 1d 30 00 00 00 89 5d dc 8b 4b 0c 89
4d c4 8b 51 1c 89 55 f4 8b 32 89 75 f0 8b 46 08
89 45 b0 8b 58 3c 89 5d c8 8b 4c 18 78 03 c8 89
4d a4 8b 71 1c 03 f0 89 75 f8 8b 51 20 03 d0 89
55 b8 8b 79 24 03 f8 89 7d e0 8b 59 14 89 5d ac
c7 45 e4 00 00 00 00 eb 09 8b 45 e4 83 c0 01 89
45 e4 8b 4d e4 3b 4d ac 0f 83 c8 00 00 00 8b 55
e4 8b 45 b8 8b 0c 90 89 4d d4 8b 55 b0 03 55 d4
89 55 98 68 48 c0 40 00 8b 45 98 50 e8 6a 02 00
00 83 c4 08 85 c0 75 42 8b 4d 98 51 8b 55 e4 52
68 58 c0 40 00 e8 84 01 00 00 83 c4 0c 8b 45 e4
8b 4d e0 0f b7 14 41 8b 45 f8 8b 0c 90 89 4d bc
8b 55 bc 03 55 b0 89 55 bc 8b 45 bc 50 68 68 c0
40 00 e8 57 01 00 00 83 c4 08 68 88 c0 40 00 8b
4d 98 51 e8 13 02 00 00 83 c4 08 85 c0 75 42 8b
55 98 52 8b 45 e4 50 68 98 c0 40 00 e8 2d 01 00
00 83 c4 0c 8b 4d e4 8b 55 e0 0f b7 04 4a 8b 4d
f8 8b 14 81 89 55 cc 8b 45 cc 03 45 b0 89 45 cc
8b 4d cc 51 68 a8 c0 40 00 e8 00 01 00 00 83 c4
08 e9 23 ff ff ff 8b 45 cc 8b 5d e8 53 ff d0 89
45 d4 8b 55 d4 52 68 c8 c0 40 00 e8 de 00 00 00
83 c4 08 8b 45 d4 8b 5d 9c 53 50 8b 45 bc ff d0
89 45 d4 8b 45 d4 50 68 f0 c0 40 00 e8 bd 00 00
00 83 c4 08 8b 45 d4 33 db 8b 4d fc 8b 55 ec 53
52 51 53 ff d0
```

The top image is an example of Linux hello world style shellcode, the lower image is an equivalent example in Win32. Ouch!!!

I'm not finished yet !

- Never run shellcode from unknown sources!
- Test the code you're running before using it!
  - Who knows that the code won't exploit your own system?!?!?!
- So always Disassemble
  - Maybe running a backdoor !
- Encoding (you're gona need this for sure ☺)
  - Bad char(s) is chasing you!

?

- How can we debug a shellcode?

Summary

- What Shellcodes are, and problems that face shellcode developers
- Types of Shellcodes
- Why it’s important to disassemble a shellcode you didn’t write
- Why sometimes you need to encode your shellcode
- List of useful tools related to shellcode development

References #1

1. Software Exploitation by Open Security Training
2. Stack Based Overflow, https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/
3. MEMORY CORRUPTION 101 , NYU by Dino Dai Zovi , @dinodaizovi
4. ShellCode, http://www.blackhatlibrary.net/Shellcode
5. Introduction to win32 shellcoding, Corelan, http://www.corelan.be/index.php/2010/02/25/exploit-writing-tutorial-part-9-introduction-to-win32-shellcodeing/
6. Hacking/Shellcode/Alphanumeric/x64 printable opcodes, http://skypher.com/wiki/index.php/Hacking/Shellcode/Alphanumeric/x64_printable_opcodes
7. Learning Assembly Through Writing Shellcode, http://www.patternsinthevoid.net/blog/2011/09/learning-assembly-through-writing-shellcode/
8. Shellcoding for Linux and Windows Tutorial, http://www.vividmachines.com/shellcode/shellcode.html
9. Unix Assembly Codes Development, http://pentest.cryptocity.net/files/exploitation/asmcodes-1.0.2.pdf
10. Win32 Assembly Components, http://pentest.cryptocity.net/files/exploitation/winasm-1.0.1.pdf

References #2

11. 64-bit Linux Shellcode, http://blog.markloiseau.com/2012/06/64-bit-linux-shellcode/
12. Writing shellcode for Linux and *BSD, http://www.kernel-panic.it/security/shellcode/index.html
13. Understanding Windows's Shellcode (Matt Miller's, aka skape)
14. Metasploit's Meterpreter (Matt Miller, aka skape)
15. Syscall Proxying fun and applications, csk @ uberwall.org
16. X86 Opcode and Instruction Reference, http://ref.x86asm.net/
17. Shellcode: the assembly cocktail, by Samy Bahra, http://www.infosecwriters.com/hhworld/shellcode.txt
18. Grayhat Hacking: The Ethical Hacker's Handbook, 3rd Edition
19. The Shellcoders Handbook,
20. The Art of Exploitation, 2nd Edition,
21. Exploit-DB: http://www.exploit-db.com/shellcodes/
22. Shell Storm: http://www.shell-storm.org/shellcode/
23. BETA3 - Multi-format shellcode encoding tool, http://code.google.com/p/beta3/
24. X86 Opcode and Instruction Reference, http://ref.x86asm.net/
25. bin2shell, http://blog.markloiseau.com/wp-content/uploads/2012/06/bin2shell.tar.gz