Binary EXPLOITATION in a Nutshell

Albert Mario

albertmario19@gmail.com

Cyber Security IPB

$ whoami

Albert Mario

a.k.a berdoezt

BackEnd Dev || CSI Member || CTF Enthusiast

CTF

COMPFEST 9

IDSECCONF
Indonesia IT Security Conference

ARKAVIDIA
INFORMATICS & IT FESTIVAL

Joints
2018

SLASHROOT
2.0

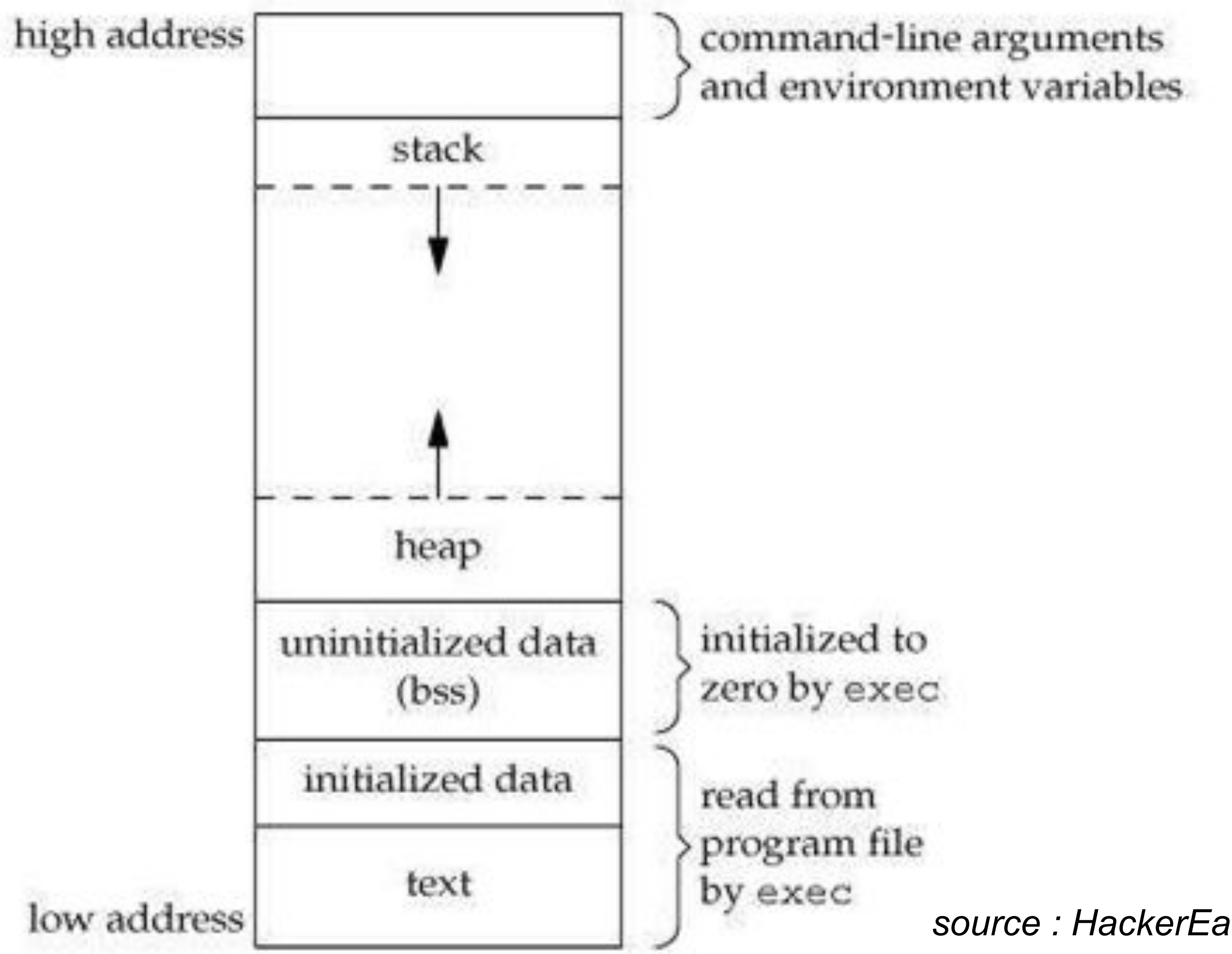
CYBER
JAWARA

GEMASTIK 10
2017

Binary Exploitation

- Binary exploitation is the process of subverting a compiled application such that it violates some trust boundary in a way that is advantageous to you, the attacker.
- Binary exploitation involves taking advantage of a bug or vulnerability in order to cause unintended or unanticipated behaviour in the problem.
- Low Level
- Memory Corruption
- Hijacking Control Flow
- Arbitrary Code Execution
- Get ROOT!!

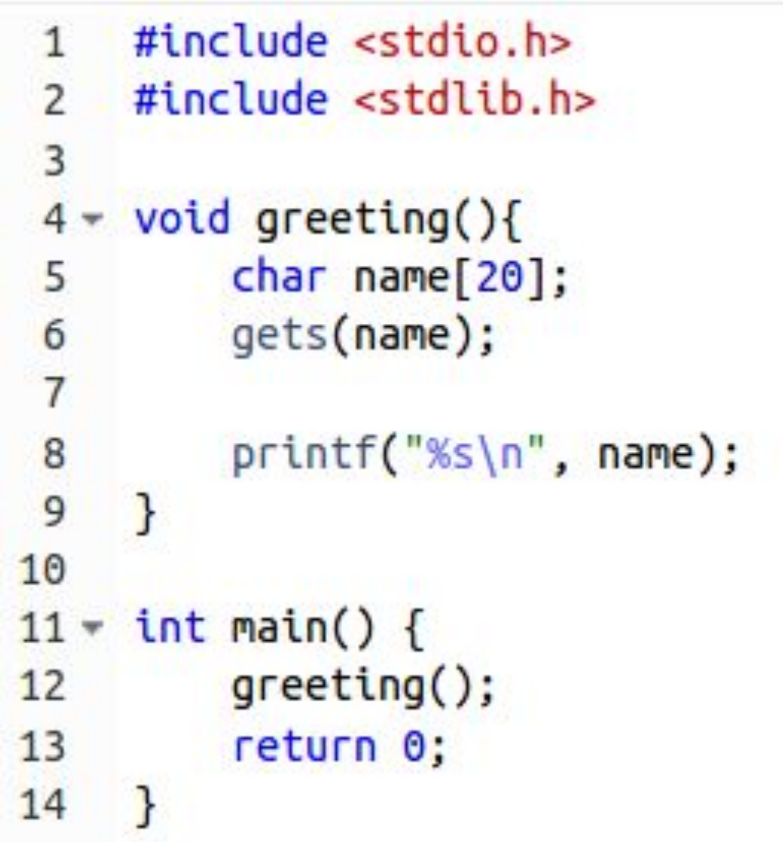
How does it work ?



source : HackerEarth

- Text : code segment, executable instruction.
- Data : global and static variables.
- BSS : uninitialized data, set to arithmetic 0 by kernel.
- stack : data created while program run, grow to lower address.
- heap : dynamic data ordered by malloc(), grow to high address.

How does it work ?

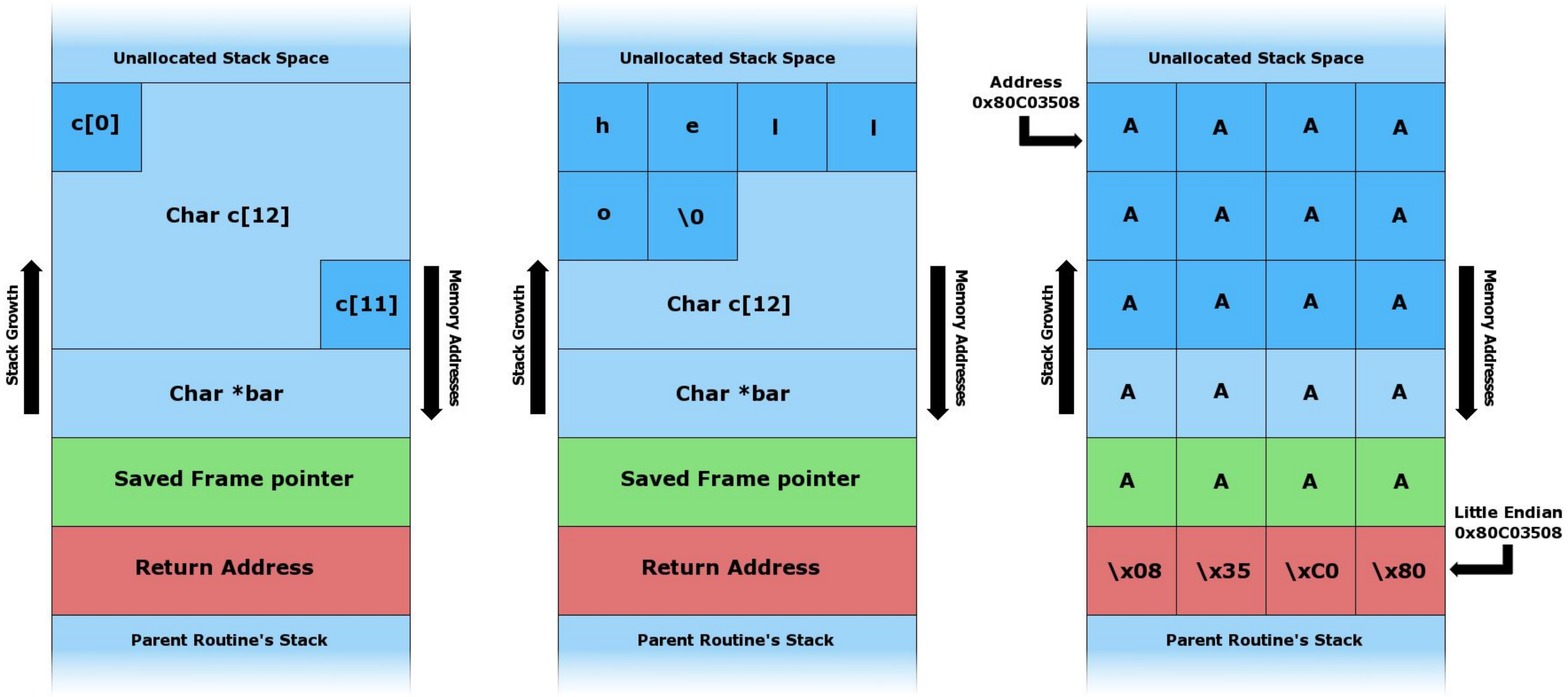


```c
#include <stdio.h>
#include <stdlib.h>

void greeting(){
    char name[20];
    gets(name);

    printf("%s\n", name);
}

int main() {
    greeting();
    return 0;
}
```

source : berdoezt's box

- Simple C program.
- Only take user's name and print it out.
- Wait, did you realize that we can change the flow, like got a shell??

How does it work ?



Unallocated Stack Space

c[0]

Char c[12]

c[11]

Char *bar

Saved Frame pointer

Return Address

Parent Routine's Stack

Stack Growth

Memory Addresses

Unallocated Stack Space

h e l l

o \0

Char c[12]

Char *bar

Saved Frame pointer

Return Address

Parent Routine's Stack

Stack Growth

Memory Addresses

Address
0x80C03508

Unallocated Stack Space

A A A A

A A A A

A A A A

A A A A

A A A A

\x08 \x35 \xC0 \x80

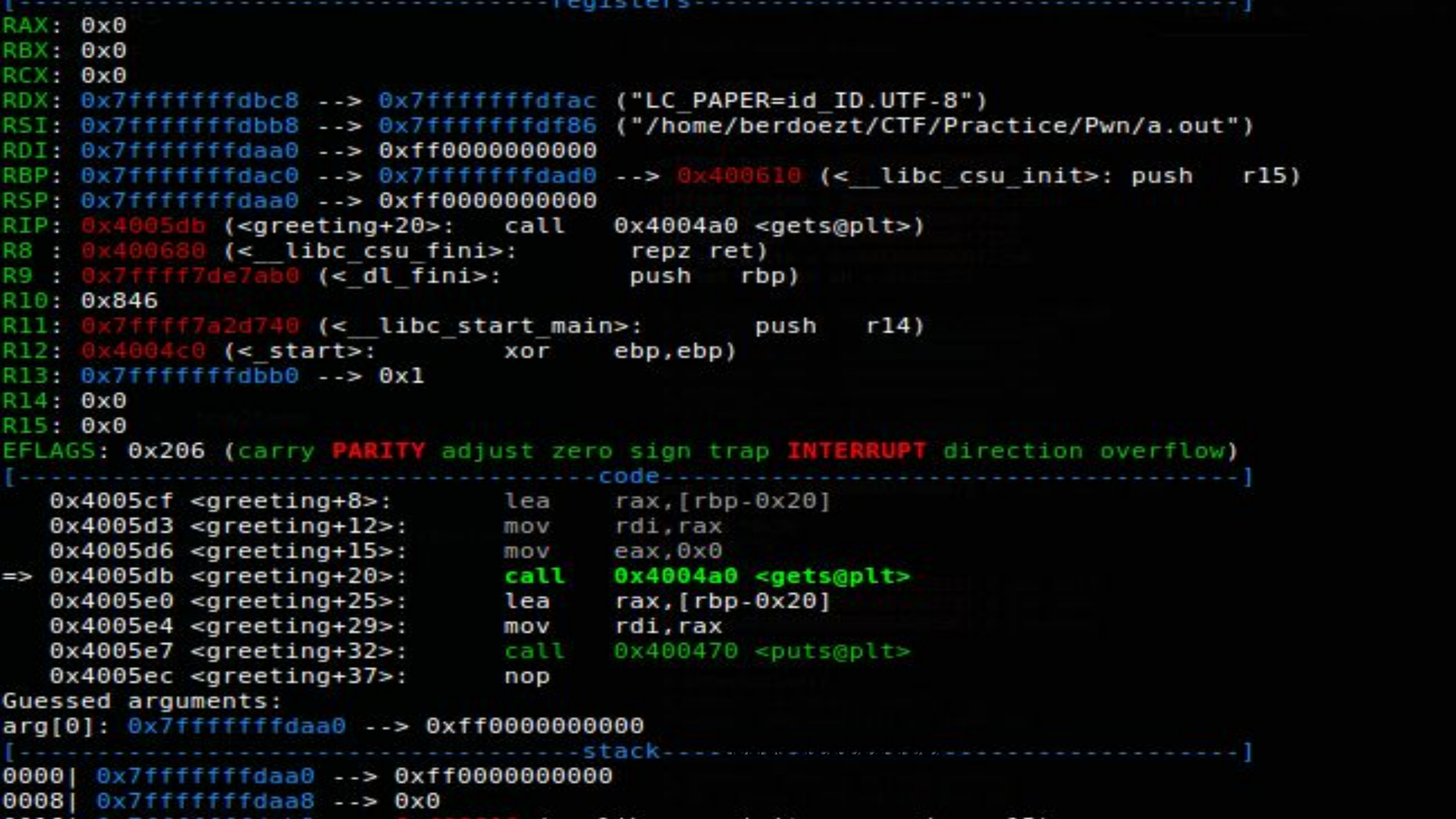
Parent Routine's Stack

Stack Growth

Memory Addresses

Little Endian
0x80C03508

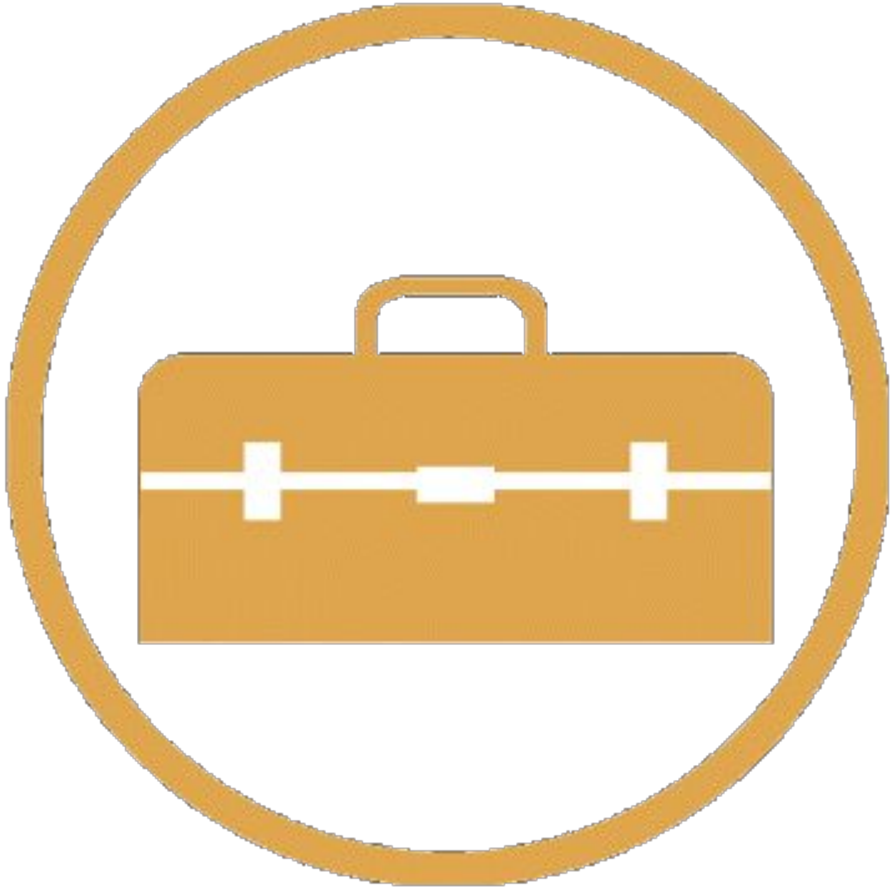
source : Wikipedia



```
[----------------------------------registers-----------------------------------]
RAX: 0x0
RBX: 0x0
RCX: 0x0
RDX: 0x7fffffffdbc8 --> 0x7fffffffdfac ("LC_PAPER=id_ID.UTF-8")
RSI: 0x7fffffffdbb8 --> 0x7fffffffdf86 ("/home/berdoezt/CTF/Practice/Pwn/a.out")
RDI: 0x7fffffffdaa0 --> 0xff0000000000
RBP: 0x7fffffffdac0 --> 0x7fffffffdad0 --> 0x400610 (<__libc_csu_init>: push   r15)
RSP: 0x7fffffffdaa0 --> 0xff0000000000
RIP: 0x4005db (<greeting+20>:   call   0x4004a0 <gets@plt>)
R8 : 0x400680 (<__libc_csu_fini>:       repz ret)
R9 : 0x7ffff7de7ab0 (<_dl_fini>:        push   rbp)
R10: 0x846
R11: 0x7ffff7a2d740 (<__libc_start_main>:       push   r14)
R12: 0x4004c0 (<_start>:        xor    ebp,ebp)
R13: 0x7fffffffdbb0 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x4005cf <greeting+8>:       lea    rax,[rbp-0x20]
   0x4005d3 <greeting+12>:      mov    rdi,rax
   0x4005d6 <greeting+15>:      mov    eax,0x0
=> 0x4005db <greeting+20>:      call   0x4004a0 <gets@plt>
   0x4005e0 <greeting+25>:      lea    rax,[rbp-0x20]
   0x4005e4 <greeting+29>:      mov    rdi,rax
   0x4005e7 <greeting+32>:      call   0x400470 <puts@plt>
   0x4005ec <greeting+37>:      nop
Guessed arguments:
arg[0]: 0x7fffffffdaa0 --> 0xff0000000000
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffdaa0 --> 0xff0000000000
0008| 0x7fffffffdaa8 --> 0x0
```

Various Techniques

- Stack buffer overflow
- Return to libc
- Return oriented programming
- Stack pivoting
- Format string
- Heap buffer overflow
- House of force
- House of prime
- House of mind
- Etc.



When it comes to the world

1. CVE-2018-7445 Mikrotik RouterOS Buffer overflow
    - When processing NetBIOS session request message.
    - Gain arbitrary code execution on the system to take control.
    - Occur before authentication.
    - Affected version :
      < 6.41.3/6.42rc27
    - Severity : critical (CVSS 3.0)

When it comes to the world

2. CVE-2017-8717 Microsoft Jet Database Engine Buffer Overflow

- Database engine used by Microsoft Access and Microsoft Visual Basic.
- Fails to adequately bounds-check user supplied.
- Gain arbitrary code execution on the system to take control.
- Mitigate by modify how Jet handles objects in memory.
- Severity : critical

When it comes to the world

3. CVE-2018-1000117 Buffer overflow vulnerability in os.symlink on Windows

- Affected version : >= 3.2 && <= 3.6.4
- Exploitable via python script that creates symlink.
- Attacker control the name or location of symlink created.
- Gain privilege escalation.
- Already patched for next release 3.4, 3.5, 3.6, 3.7.
- Severity : medium

(dis)advantages

- Data leaked
- Data loss
- Company suffered financial lossess

All cool things you can do when you have everything in your hand

Prevention - Linux Builtin

- Address Space Layout Randomization (Kernel)
  - Random stack offset memory whenever a program starts.
  - Other segment still static.
- Canary (Compiler)
  - Not a bird :p
  - Known value placed between buffer and control data on the stack to monitor buffer overflow.
  - If verification failed, will alert an overflow.

Prevention - Linux Builtin

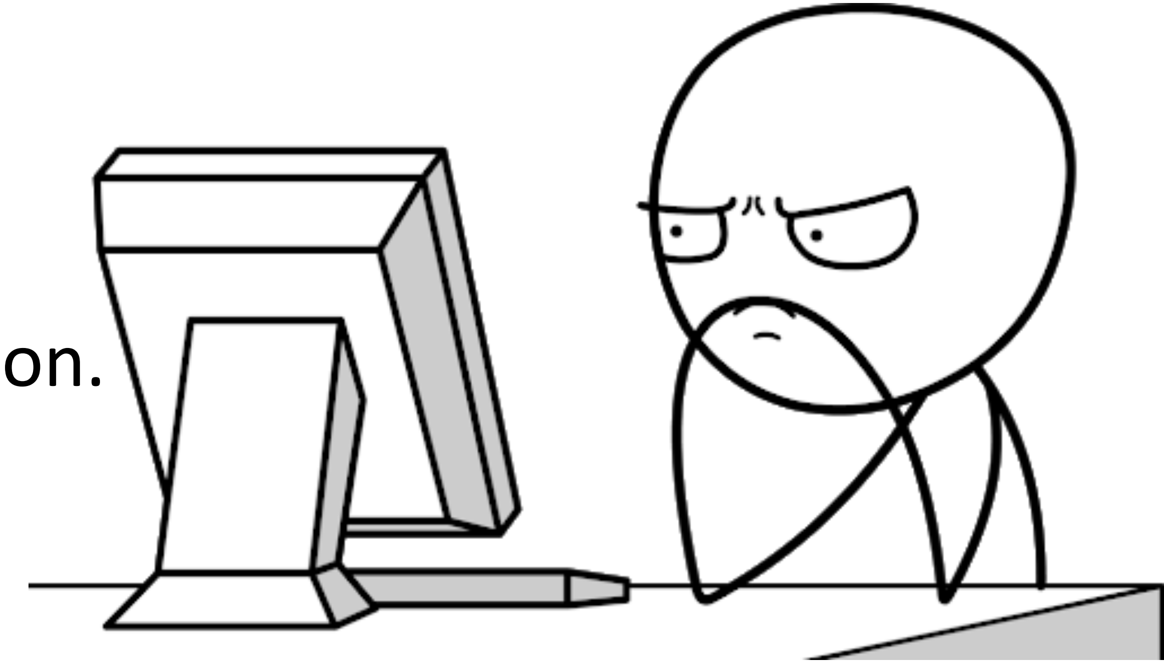
- Data Execution Prevention (Compiler)
  - Disable execution permission on stack.
  - Preventing attacker from executing shellcode.
- Position Independent Executables (Compiler)
  - Once again, not a cake :p
  - Randomize code, data, and bss segment offset whenever a program starts.
- Fortify Source (Compiler)
  - Detect overflow potential on several function that perform access on memory and string.

Prevention - Programmer Side

Insecure Function	Secure Function
strcpy()	strlcpy(), strcpy_s()
strcat()	strlcat(), strcat_s()
printf() / snprintf()	snprintf(), snprintf_s()
gets()	fgets()

Prevention - Programmer Side

- Never trust user's input.
- Don't use vulnerable function.
- Always validate buffer length.
- Upgrade system to the latest version.
- Stay up-to-date.



Where to start

- Programming.
- Assembly Language.
- Memory Layout.
- Reverse Engineering.
- How program works in low level.

Thank You

Cyber Security IPB

facebook.com/cysecipb