# REVERSE ENGINEERING – CLASS 0x06

## ASLR/PIE, RELRO AND ROP

Cristian Rusu

# LAST TIME

- **the memory layout**
- **the stack**
- **problems with the stack**
- **mitigations for stack issues**
  - Stack Smashing Protector (SSP)

SSP: -fstack-protector for gcc and /GS for cl

# TODAY

- short review of ASLR/PIE
- RELRO
- ROP

# RELRO: THE GOT AND PLT

- we have previously talked about this
- what happens when we call a function from an external library?

```
.text:0000000000401186 ; =============== S U B R O U T I N E =======================
.text:0000000000401186
.text:0000000000401186 ; Attributes: bp-based frame
.text:0000000000401186
.text:0000000000401186                 public hello_world
.text:0000000000401186 hello_world     proc near               ; CODE XREF: main+13↓p
.text:0000000000401186 ; __unwind {
.text:0000000000401186                 push    rbp
.text:0000000000401187                 mov     rbp, rsp
.text:000000000040118A                 lea     rdi, s          ; "Hello, world"
.text:0000000000401191                 call    _puts
.text:0000000000401196                 nop
.text:0000000000401197                 pop     rbp
.text:0000000000401198                 retn
.text:0000000000401198 ; } // starts at 401186
.text:0000000000401198 hello_world     endp
```

- this is famous puts(“Hello, world”) example

# RELRO: THE GOT AND PLT

- at runtime, the loader (ld.so) finds the function
- the call to puts from main is actually to a stub

```
.plt:0000000000401030 ; =============== S U B R O U T I N E =======================================
.plt:0000000000401030
.plt:0000000000401030 ; Attributes: thunk
.plt:0000000000401030
.plt:0000000000401030 ; int puts(const char *s)
.plt:0000000000401030 _puts           proc near               ; CODE XREF: hello_world+B↓p
.plt:0000000000401030                                         ; goodbye_world+B↓p
.plt:0000000000401030                 jmp     cs:off_404018
.plt:0000000000401030 _puts           endp
.plt:0000000000401030
```

# RELRO: THE GOT AND PLT

- all the addresses which are filled-in are placed in the GOT

```
.got.plt:0000000000404000 ; Segment type: Pure data
.got.plt:0000000000404000 ; Segment permissions: Read/Write
.got.plt:0000000000404000 ; Segment alignment 'qword' can not be represented in assembly
.got.plt:0000000000404000 _got_plt        segment para public 'DATA' use64
.got.plt:0000000000404000                 assume cs:_got_plt
.got.plt:0000000000404000                 ;org 404000h
.got.plt:0000000000404000 _GLOBAL_OFFSET_TABLE_ dq offset _DYNAMIC
.got.plt:0000000000404008 qword_404008    dq 0                  ; DATA XREF: sub_401020↑r
.got.plt:0000000000404010 qword_404010    dq 0                  ; DATA XREF: sub_401020+6↑r
.got.plt:0000000000404018 off_404018      dq offset puts        ; DATA XREF: _puts↑r
.got.plt:0000000000404020 off_404020      dq offset printf      ; DATA XREF: _printf↑r
.got.plt:0000000000404028 off_404028      dq offset malloc      ; DATA XREF: _malloc↑r
.got.plt:0000000000404030 off_404030      dq offset __isoc99_scanf
.got.plt:0000000000404030                                       ; DATA XREF: ___isoc99_scanf↑r
.got.plt:0000000000404038 off_404038      dq offset exit        ; DATA XREF: _exit↑r
.got.plt:0000000000404038 _got_plt        ends
.got.plt:0000000000404038
```

# RELRO: THE GOT AND PLT

- this tabled can be filled in at start of process or at runtime whenever we actually need a function

```
gdb-peda$ telescope 0x404000 30
0000| 0x404000 --> 0x403e20 --> 0x1
0008| 0x404008 --> 0x7ffff7ffe190 --> 0x0
0016| 0x404010 --> 0x7ffff7fea440 (<_dl_runtime_resolve_xsave>: push   rbx)
0024| 0x404018 --> 0x401036 (<free@plt+6>:      push   0x0)
0032| 0x404020 --> 0x401046 (<unlink@plt+6>:    push   0x1)
0040| 0x404028 --> 0x401056 (<_exit@plt+6>:     push   0x2)
0048| 0x404030 --> 0x401066 (<fread@plt+6>:     push   0x3)
0056| 0x404038 --> 0x401076 (<fclose@plt+6>:    push   0x4)
0064| 0x404040 --> 0x401086 (<opendir@plt+6>:   push   0x5)
0072| 0x404048 --> 0x401096 (<strlen@plt+6>:    push   0x6)
0080| 0x404050 --> 0x4010a6 (<closedir@plt+6>:  push   0x7)
0088| 0x404058 --> 0x4010b6 (<srand@plt+6>:     push   0x8)
0096| 0x404060 --> 0x4010c6 (<strcmp@plt+6>:    push   0x9)
0104| 0x404068 --> 0x4010d6 (<time@plt+6>:      push   0xa)
0112| 0x404070 --> 0x4010e6 (<__xstat@plt+6>:   push   0xb)
0120| 0x404078 --> 0x4010f6 (<readdir@plt+6>:   push   0xc)
0128| 0x404080 --> 0x401106 (<fseek@plt+6>:     push   0xd)
0136| 0x404088 --> 0x401116 (<ptrace@plt+6>:    push   0xe)
0144| 0x404090 --> 0x401126 (<asprintf@plt+6>:  push   0xf)
0152| 0x404098 --> 0x401136 (<mprotect@plt+6>:  push   0x10)
0160| 0x4040a0 --> 0x401146 (<fopen@plt+6>:     push   0x11)
0168| 0x4040a8 --> 0x401156 (<rename@plt+6>:    push   0x12)
0176| 0x4040b0 --> 0x401166 (<sprintf@plt+6>:   push   0x13)
0184| 0x4040b8 --> 0x401176 (<fwrite@plt+6>:    push   0x14)
0192| 0x4040c0 --> 0x401186 (<sleep@plt+6>:     push   0x15)
0200| 0x4040c8 --> 0x401196 (<rand@plt+6>:      push   0x16)
0208| 0x4040d0 --> 0x0
0216| 0x4040d8 --> 0x0
0224| 0x4040e0 --> 0x0
0232| 0x4040e8 --> 0x0
```

# RELRO: THE GOT AND PLT

- **as functions are needed, table is filled**

```
gdb-peda$ telescope 0x404000 30
0000| 0x404000 --> 0x403e20 --> 0x1
0008| 0x404008 --> 0x7ffff7ffe190 --> 0x0
0016| 0x404010 --> 0x7ffff7fea440 (<_dl_runtime_resolve_xsave>: push   rbx)
0024| 0x404018 --> 0x401036 (<free@plt+6>:      push   0x0)
0032| 0x404020 --> 0x401046 (<unlink@plt+6>:    push   0x1)
0040| 0x404028 --> 0x401056 (<_exit@plt+6>:     push   0x2)
0048| 0x404030 --> 0x401066 (<fread@plt+6>:     push   0x3)
0056| 0x404038 --> 0x401076 (<fclose@plt+6>:    push   0x4)
0064| 0x404040 --> 0x7ffff7e87f60 (<__opendir>: cmp    BYTE PTR [rdi],0x0)
0072| 0x404048 --> 0x7ffff7f22560 (<__strlen_avx2>:     mov    ecx,edi)
0080| 0x404050 --> 0x7ffff7e87fa0 (<__closedir>:        test   rdi,rdi)
0088| 0x404058 --> 0x4010b6 (<srand@plt+6>:     push   0x8)
0096| 0x404060 --> 0x7ffff7f1daa0 (<__strcmp_avx2>:     mov    eax,edi)
0104| 0x404068 --> 0x4010d6 (<time@plt+6>:      push   0xa)
0112| 0x404070 --> 0x4010e6 (<__xstat@plt+6>:   push   0xb)
0120| 0x404078 --> 0x7ffff7e88160 (<__GI___readdir64>:  push   r13)
0128| 0x404080 --> 0x401106 (<fseek@plt+6>:     push   0xd)
0136| 0x404088 --> 0x401116 (<ptrace@plt+6>:    push   0xe)
0144| 0x404090 --> 0x401126 (<asprintf@plt+6>:  push   0xf)
0152| 0x404098 --> 0x401136 (<mprotect@plt+6>:  push   0x10)
0160| 0x4040a0 --> 0x401146 (<fopen@plt+6>:     push   0x11)
0168| 0x4040a8 --> 0x401156 (<rename@plt+6>:    push   0x12)
0176| 0x4040b0 --> 0x401166 (<sprintf@plt+6>:   push   0x13)
0184| 0x4040b8 --> 0x401176 (<fwrite@plt+6>:    push   0x14)
0192| 0x4040c0 --> 0x401186 (<sleep@plt+6>:     push   0x15)
0200| 0x4040c8 --> 0x401196 (<rand@plt+6>:      push   0x16)
0208| 0x4040d0 --> 0x0
0216| 0x4040d8 --> 0x0
0224| 0x4040e0 --> 0x1
0232| 0x4040e8 --> 0x0
```

any security issue you might see?

# RELRO: THE GOT AND PLT

- as functions are needed, table is filled

```
gdb-peda$ telescope 0x404000 30
0000| 0x404000 --> 0x403e20 --> 0x1
0008| 0x404008 --> 0x7ffff7ffe190 --> 0x0
0016| 0x404010 --> 0x7ffff7fea440 (<_dl_runtime_resolve_xsave>: push   rbx)
0024| 0x404018 --> 0x401036 (<free@plt+6>:      push   0x0)
0032| 0x404020 --> 0x401046 (<unlink@plt+6>:    push   0x1)
0040| 0x404028 --> 0x401056 (<_exit@plt+6>:     push   0x2)
0048| 0x404030 --> 0x401066 (<fread@plt+6>:     push   0x3)
0056| 0x404038 --> 0x401076 (<fclose@plt+6>:    push   0x4)
0064| 0x404040 --> 0x7ffff7e87f60 (<__opendir>: cmp    BYTE PTR [rdi],0x0)
0072| 0x404048 --> 0x7ffff7f22560 (<__strlen_avx2>:     mov    ecx,edi)
0080| 0x404050 --> 0x7ffff7e87fa0 (<__closedir>:        test   rdi,rdi)
0088| 0x404058 --> 0x4010b6 (<srand@plt+6>:     push   0x8)
0096| 0x404060 --> 0x7ffff7f1daa0 (<__strcmp_avx2>:     mov    eax,edi)
0104| 0x404068 --> 0x4010d6 (<time@plt+6>:      push   0xa)
0112| 0x404070 --> 0x4010e6 (<__xstat@plt+6>:   push   0xb)
0120| 0x404078 --> 0x7ffff7e88160 (<__GI___readdir64>:  push   r13)
0128| 0x404080 --> 0x401106 (<fseek@plt+6>:     push   0xd)
0136| 0x404088 --> 0x401116 (<ptrace@plt+6>:    push   0xe)
0144| 0x404090 --> 0x401126 (<asprintf@plt+6>:  push   0xf)
0152| 0x404098 --> 0x401136 (<mprotect@plt+6>:  push   0x10)
0160| 0x4040a0 --> 0x401146 (<fopen@plt+6>:     push   0x11)
0168| 0x4040a8 --> 0x401156 (<rename@plt+6>:    push   0x12)
0176| 0x4040b0 --> 0x401166 (<sprintf@plt+6>:   push   0x13)
0184| 0x4040b8 --> 0x401176 (<fwrite@plt+6>:    push   0x14)
0192| 0x4040c0 --> 0x401186 (<sleep@plt+6>:     push   0x15)
0200| 0x4040c8 --> 0x401196 (<rand@plt+6>:      push   0x16)
0208| 0x4040d0 --> 0x0
0216| 0x4040d8 --> 0x0
0224| 0x4040e0 --> 0x1
0232| 0x4040e8 --> 0x0
```

any security issue you might see?
puts(“/bin/sh”) becomes system(“/bin/sh”)

# RELRO: THE GOT AND PLT

- **solution: Read Only RELocations (RELRO)**

```
gdb-peda$ telescope 0x403f20 30
0000| 0x403f20 --> 0x403d30 --> 0x1
0008| 0x403f28 --> 0x0
0016| 0x403f30 --> 0x0
0024| 0x403f38 --> 0x7ffff7e4abc0 (<__GI___libc_free>:  push   rbx)
0032| 0x403f40 --> 0x7ffff7eb2290 (<unlink>:    mov    eax,0x57)
0040| 0x403f48 --> 0x7ffff7e8cca0 (<__GI__exit>:        mov    edx,edi)
0048| 0x403f50 --> 0x7ffff7e367f0 (<fread>:     push   r14)
0056| 0x403f58 --> 0x7ffff7e359e0 (<fclose>:    push   r12)
0064| 0x403f60 --> 0x7ffff7e87f60 (<__opendir>: cmp    BYTE PTR [rdi],0x0)
0072| 0x403f68 --> 0x7ffff7f22560 (<__strlen_avx2>:     mov    ecx,edi)
0080| 0x403f70 --> 0x7ffff7e87fa0 (<__closedir>:        test   rdi,rdi)
0088| 0x403f78 --> 0x7ffff7e008f0 (<__srandom>: sub    rsp,0x8)
0096| 0x403f80 --> 0x7ffff7f1daa0 (<__strcmp_avx2>:     mov    eax,edi)
0104| 0x403f88 --> 0x7ffff7fd3f00 (<time>:      mov    rax,QWORD PTR [rip+0xffffffffffffc1a1]
0112| 0x403f90 --> 0x7ffff7eafd60 (<__GI___xstat>:      mov    rax,rsi)
0120| 0x403f98 --> 0x7ffff7e88160 (<__GI___readdir64>:  push   r13)
0128| 0x403fa0 --> 0x7ffff7e3deb0 (<fseek>:     push   rbx)
0136| 0x403fa8 --> 0x7ffff7eb7bf0 (<ptrace>:    sub    rsp,0x68)
0144| 0x403fb0 --> 0x7ffff7e1e950 (<___asprintf>:       sub    rsp,0xd8)
0152| 0x403fb8 --> 0x7ffff7eba510 (<mprotect>:  mov    eax,0xa)
0160| 0x403fc0 --> 0x7ffff7e363e0 (<_IO_new_fopen>:     mov    edx,0x1)
0168| 0x403fc8 --> 0x7ffff7e338d0 (<rename>:    mov    eax,0x52)
0176| 0x403fd0 --> 0x7ffff7e1e890 (<__sprintf>: sub    rsp,0xd8)
0184| 0x403fd8 --> 0x7ffff7e36c10 (<fwrite>:    push   r15)
0192| 0x403fe0 --> 0x7ffff7e8c910 (<__sleep>:   push   rbp)
0200| 0x403fe8 --> 0x7ffff7e00fc0 (<rand>:      sub    rsp,0x8)
0208| 0x403ff0 --> 0x7ffff7de9fb0 (<__libc_start_main>: push   r14)
0216| 0x403ff8 --> 0x0
0224| 0x404000 --> 0x0
0232| 0x404008 --> 0x0
```

security-wise this is OK, but any drawback?

# SUMMARY OF MITIGATIONS

- **Position Independent Execution (PIE)**
  - on by default on both Windows and Linux
- **Stack Smashing Protection (SSP)**
  - on by default on Windows, off by default on Linux
- **Read Only RELocations (RELRO)**
  - on by default on Windows, off by default on Linux

**all these are done at the compiler**

# SUMMARY OF MITIGATIONS

- **Position Independent Execution (PIE)**
  - on by default on both Windows and Linux
- **Stack Smashing Protection (SSP)**
  - on by default on Windows, off by default on Linux
- **Read Only RELocations (RELRO)**
  - on by default on Windows, off by default on Linux

**all these techniques come for free?**

# SUMMARY OF MITIGATIONS

- **Position Independent Execution (PIE)**
  - on by default on both Windows and Linux
- **Stack Smashing Protection (SSP)**
  - on by default on Windows, off by default on Linux
- **Read Only RELocations (RELRO)**
  - on by default on Windows, off by default on Linux

**they cause an increase of 15–25% in running time**

# ROP

- **Return Oriented Programming (ROP)**

# ROP: DATA EXECUTION

- **the good-old times (shellcode.c)**

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

int main()
{
  int e;
  char *argv[] = { "/bin/ls", "-l", NULL };

  e = execve("/bin/ls", argv, NULL);
  if (e == -1)
      fprintf(stderr, "Error: %s\n", strerror(errno));
  return 0;
}
```

# ROP: DATA EXECUTION

- **same program in Assembly**

```
.text
.globl _start

_start:
        xor %eax,%eax
        push %eax
        push $0x68732f2f
        push $0x6e69622f
        mov %esp,%ebx
        push %eax
        push %ebx
        mov %esp,%ecx
        mov $0xb,%al
        int $0x80

        movl $1, %eax
        movl $0, %ebx
        int $0x80
```

```
root@kali:~# objdump -d shellcode

shellcode:     file format elf32-i386


Disassembly of section .text:

08048054 <_start>:
 8048054:       31 c0                   xor    %eax,%eax
 8048056:       50                      push   %eax
 8048057:       68 2f 2f 73 68          push   $0x68732f2f
 804805c:       68 2f 62 69 6e          push   $0x6e69622f
 8048061:       89 e3                   mov    %esp,%ebx
 8048063:       50                      push   %eax
 8048064:       53                      push   %ebx
 8048065:       89 e1                   mov    %esp,%ecx
 8048067:       b0 0b                   mov    $0xb,%al
 8048069:       cd 80                   int    $0x80
 804806b:       b8 01 00 00 00          mov    $0x1,%eax
 8048070:       bb 00 00 00 00          mov    $0x0,%ebx
 8048075:       cd 80                   int    $0x80
```

# ROP: DATA EXECUTION

- **the same program back in C**

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

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
        fprintf(stdout,"Length: %d\n",strlen(shellcode));
        (*(void(*)()) shellcode)();
        return 0;
}
```

what is going on here?

# ROP: DATA EXECUTION

- **the same program back in C**

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

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
        fprintf(stdout,"Length: %d\n",strlen(shellcode));
        (*(void(*)()) shellcode)();
        return 0;
}
```

programs like these can no longer run on modern operating systems

- Data Execution Prevention (DEP)
- No eXecute (NX)

# ROP: THE IDEA

- **we are no longer in a golden age for attackers**
- **but there are some new ideas**

- **goal: we would still like to execute arbitrary code**
  - not be confined in the code space of the binary

- **problem: we cannot place code into data segments anymore**
  - so, where can we place code?
  - can we use something that exists already?

# ROP: THE IDEA

- **we are no longer in a golden age for attackers**
- **but there are some new ideas**

- **goal: we would still like to execute arbitrary code**
  - not be confined in the code space of the binary

- **problem: we cannot place code into data segments anymore**
  - so, where can we place code?
  - can we use something that exists already?

- **one solution: use pieces of code that already exist but stitch them together in a different order than the original one to perform overall the task that you want (like building a puzzle)**

# ROP: THE IDEA

- **we cannot just stitch different pieces of code in general**
- **so how do we do this?**

- **what do we want?**
  - jump to some instructions
  - execute starting from that point
  - then jump to other instructions

- **what can we use to perform the wishlist above?**

# ROP: THE IDEA

- **we cannot just stitch different pieces of code in general**
- **so how do we do this?**

- **what do we want?**
  - jump to some instructions
  - execute starting from that point
  - then jump to other instructions

- **what can we use to perform the wishlist above?**
  - CALL
  - RET

# ROP: THE IDEA

- **what does CALL *destination* do?**

- **what does RET do?**

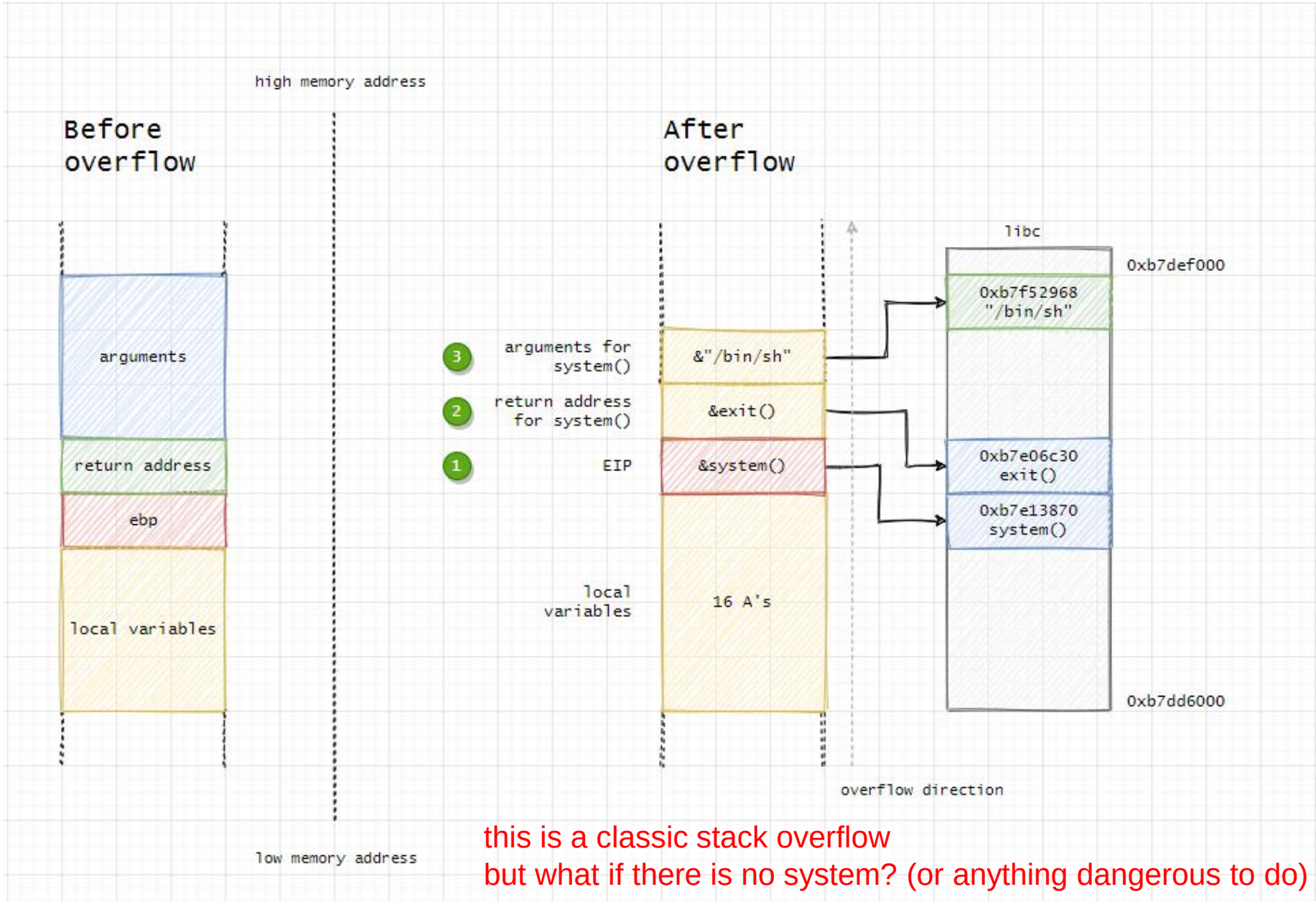
# ROP: THE IDEA

- **what does CALL *destination* do?**
  - pushes the return address on the stack (instruction after the CALL)
  - changes the Instruction Pointer to *destination*

- **what does RET do?**
  - pops the return address from the stack
    - go to where the Stack Pointer points to
    - take the value from there (it is an address)
    - increment Stack Pointer (i.e., remove address from the stack)
  - changes the Instruction Pointer to that address

.

# ROP: THE IDEA

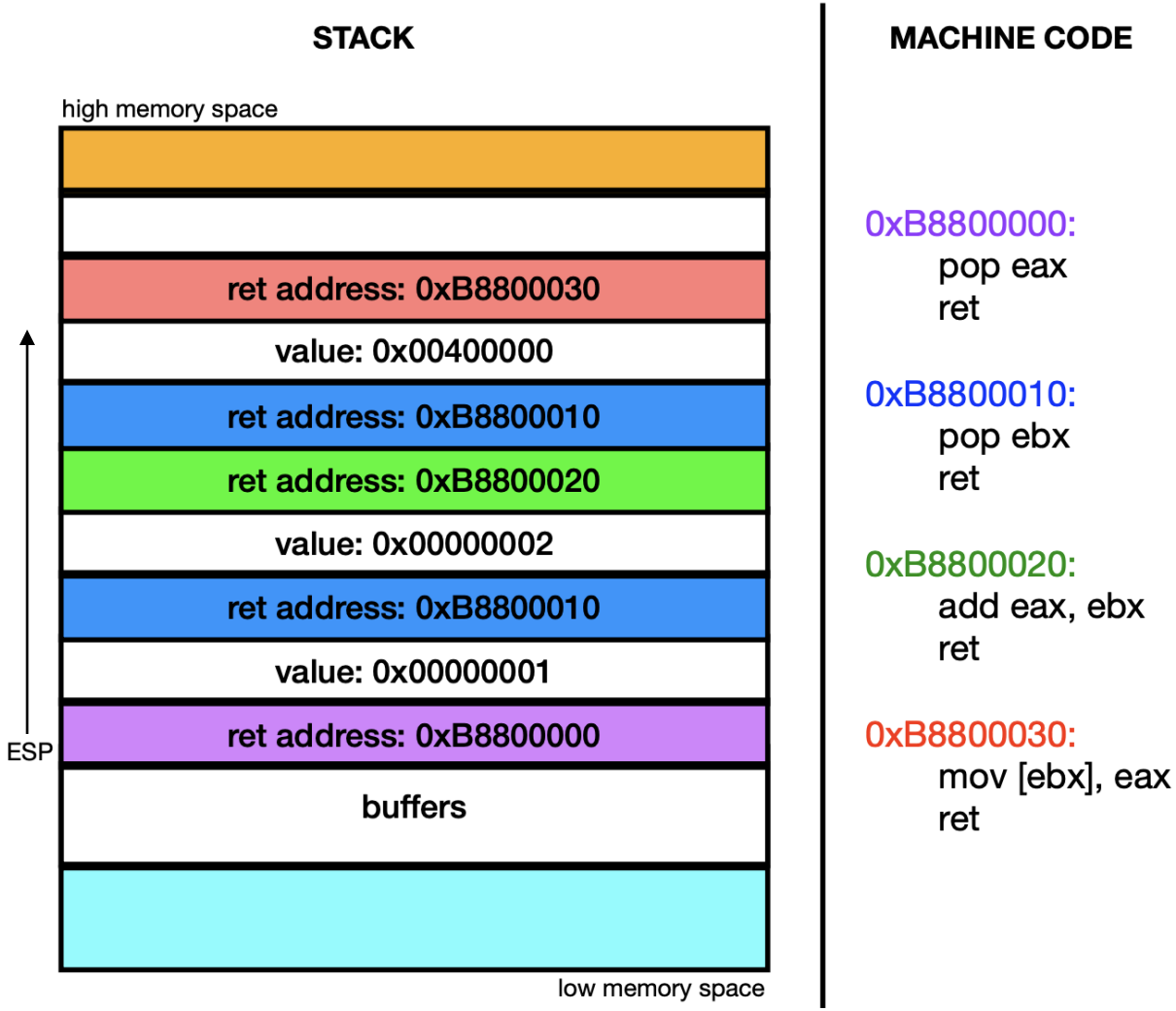



this is a classic stack overflow
but what if there is no system? (or anything dangerous to do)

https://www.ired.team/offensive-security/code-injection-process-injection/binary-exploitation/return-to-libc-ret2libc

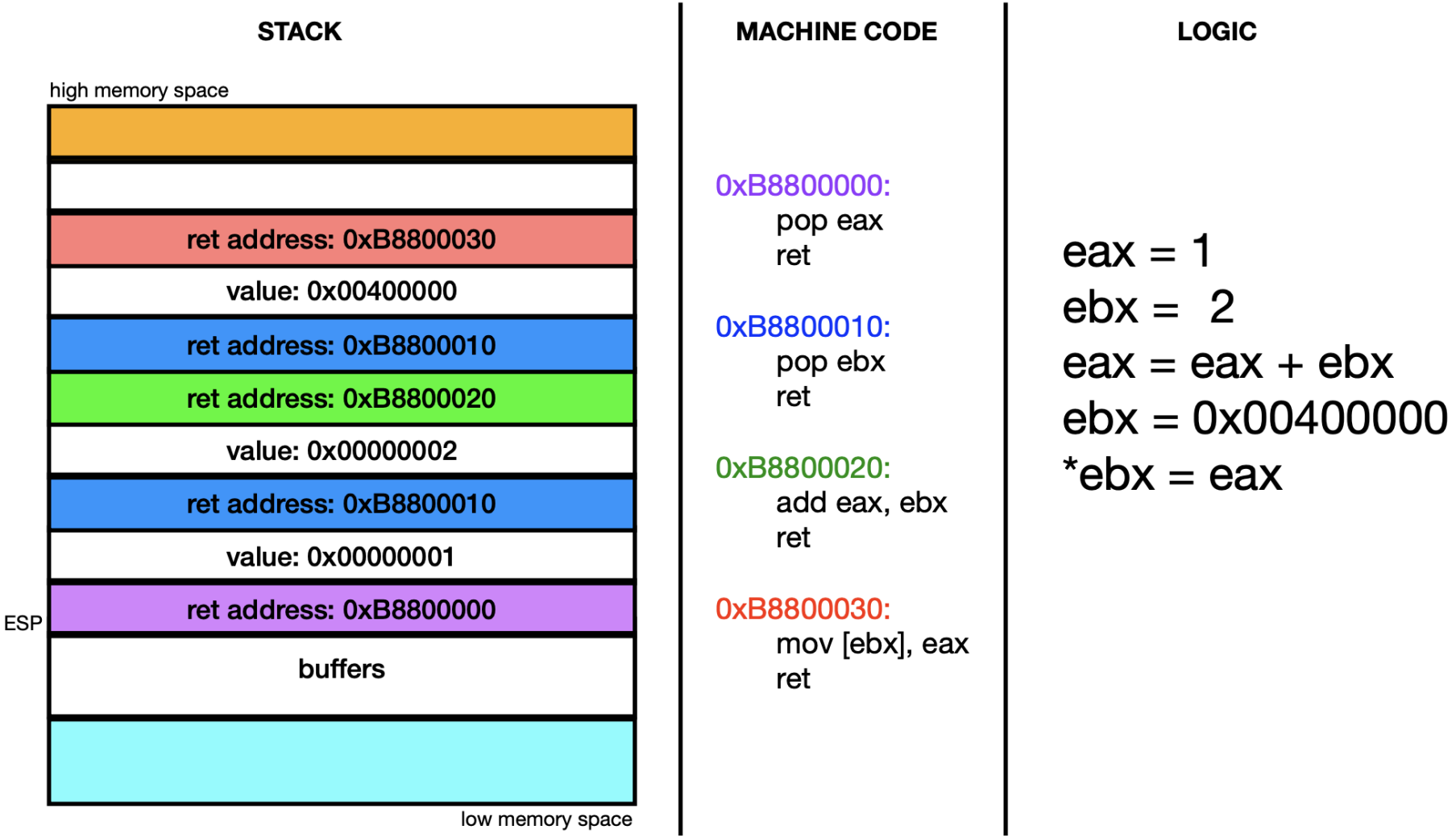
# ROP: THE IDEA

- we overflow a lot more than just the return address



# ROP: THE IDEA

- we overflow a lot more than just the return address




these are called gadgets

# WHAT WE DID TODAY

- short review of ASLR/PIE
- SSP
- RELRO
- ROP

# NEXT TIME …

- **RE for bytecode**

.

# REFERENCES


- Stack Binary Exploitation, https://ir0nstone.gitbook.io/notes/types/stack
- pwntools-tutorial, https://github.com/Gallopsled/pwntools-tutorial/blob/master/rop.md
- Return Oriented Programming (ROP) attacks, https://resources.infosecinstitute.com/topic/return-oriented-programming-rop-attacks/
- Binary exploitation, https://www.ired.team/offensive-security/code-injection-process-injection/binary-exploitation
- Weird Return-Oriented Programming Tutorial, https://www.youtube.com/watch?v=zaQVNM3or7k

.