#### Binary Reverse Engineering And Analysis Course 6: ASLR and ROP

Caragea Radu

March 23, 2021

- Last time we studied basic stack exploits
- The main idea was to hijack execution
- The destination was still in the target binary
- Today: construct new pathways in a program

#### Linux address space (in the olden times)

[-----] 00001 0x7fffffffde60 --> 0x401230 (< libc csu init>: push r15) 88881 8x7ffffffffde68 ...> 8x7ffff7dea89h (< libc start main+235>; edi eav) 0016 0x7fffffffde70 --> 0x0 8024 0x7fffffffde78 --> 8x7fffffffdf48 --> 0x7fffffffe291 ("/ctf/unibuc/curs re/curs "...) 80321 0x7fffffffde80 --> 0x100040000 88481 8v7fffffffde88 ...> 8v4811ee (<main>: nush rhn) 00481 0x7fffffffde90 --> 0x0 80561 0x7fffffffde98 --> 8x26f429881fe236d8 Legend: code, data, rodata, value odb-nedas ymnan End Perm Name 0x00400000 0×00401000 F - - D /ctf/unibuc/curs re/curs 86/demo 81 linux memory/hello 0×00401000 0×00402000 r-xp /ctf/unibuc/curs re/curs 86/demo 81 linux memory/hello 8×88482888 0×00403000 /ctf/unibuc/curs re/curs 86/demo 81 linux memory/hello r - - n 0x00403000 0100404000 r - - p /ctf/unibuc/curs re/curs 86/demo 81 linux memory/hello 0x00404000 0×00405000 rw-p /ctf/unibuc/curs re/curs 86/demo 81 linux memory/hello 0×00405000 0×88426888 rw-p [heap] Ax00007ffff7dc6000 Ax00007ffff7de8000 r--p /lib/x86\_64-linux-gnu/libc-2.28.so 8v88887ffff7de8888 8v88887ffff7f38888 r.vn /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007ffff7f30000 0x00007ffff7f7c000 r--p /lib/x86\_64-linux-gnu/libc-2.28.so 8x88887ffff7f7c888 8x88887ffff7f7d888 ---p /lib/x86\_64-linux-gnu/libc-2.28.so 8x88887fffffffffffff81888 r... /lib/x86\_64-linux-anu/libc-2.28.so 0x00007ffff7f81000 0x00007ffff7f83000 rw-p /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007ffff7f83000 0x00007ffff7f87000 rw-p mapped 8x00007ffff7f87000 0x00007ffff7f89000 rw-p mapped Ax00007ffff7fd0000 Ax00007ffff7fd3000 r...p [vvar] 0x00007ffff7fd3000 0x00007ffff7fd5000 r-xp [vdso] 0x00007ffff7fd5000 0x00007ffff7fd6000 r--p /lib/x86 64-linux-gnu/ld-2.28.so 8x08807ffff7fd6088 0x88007ffff7ff4880 r-xp /lib/x86\_64-linux-gnu/ld-2.28.so AxAAAA7fffffffffffffaAAAA AxAAAAA7ffffffffffffffaAAA r...p /lib/x86\_64-linux-anu/ld-2.28.so Avanaatffffffffaffcann avanaa7fffffffdann r.... /lib/x86\_64-linux-gnu/ld-2.28.so /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007ffff7ffd000 0x00007ffff7ffe000 rw-p 8x88887ffff7ffe888 8x88887ffff7fff888 rw-p mapped 0x00007ffffffde000 0x00007fffffff000 rw-p [stack]

gdb-peda\$

Retaddr corruption is possible => anything in std lib can be called! DEMO

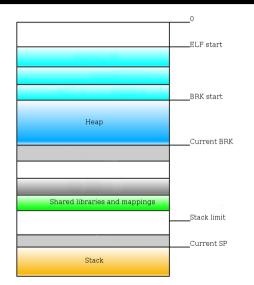
#### Linux address space (in the olden times)

[-----] 00001 0x7fffffffde60 --> 0x401230 (< libc csu init>: push r15) 88881 8x7ffffffffde68 ...> 8x7ffff7dea89h (< libc start main+235>; edi eav) 0016 0x7fffffffde70 --> 0x0 8024 0x7fffffffde78 --> 8x7fffffffdf48 --> 0x7fffffffe291 ("/ctf/unibuc/curs re/curs "...) 80321 0x7fffffffde80 --> 0x100040000 88481 8v7fffffffde88 ...> 8v4811ee (<main>: nush rhn) 00481 0x7fffffffde90 --> 0x0 80561 0x7fffffffde98 --> 8x26f429881fe236d8 Legend: code, data, rodata, value odb-nedas ymnan End Perm Name 0x00400000 0×00401000 F - - D /ctf/unibuc/curs re/curs 86/demo 81 linux memory/hello 0×00401000 0×00402000 r-xp /ctf/unibuc/curs re/curs 86/demo 81 linux memory/hello 8×88482888 0×00403000 /ctf/unibuc/curs re/curs 86/demo 81 linux memory/bello r - - n 0x00403000 0100404000 r - - p /ctf/unibuc/curs re/curs 86/demo 81 linux memory/hello 0x00404000 0×00405000 rw-p /ctf/unibuc/curs re/curs 86/demo 81 linux memory/hello 0×00405000 0×88426888 rw-p [heap] Ax00007ffff7dc6000 0x00007ffff7de8000 r.-.n /lib/x86\_64-linux-gnu/libc-2.28.so 8v88887ffff7de8888 8v88887ffff7f38888 r.vn /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007ffff7f30000 0x00007ffff7f7c000 r--p /lib/x86\_64-linux-gnu/libc-2.28.so 8x88887ffff7f7c888 8x88887ffff7f7d888 ---p /lib/x86\_64-linux-gnu/libc-2.28.so 8x88887fffffffffffff81888 r... /lib/x86\_64-linux-anu/libc-2.28.so 0x00007ffff7f81000 0x00007ffff7f83000 rw-p /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007ffff7f83000 0x00007ffff7f87000 rw-p mapped 8x00007ffff7f87000 0x00007ffff7f89000 rw-p mapped Ax00007ffff7fd0000 Ax00007ffff7fd3000 r...p [vvar] 0x00007ffff7fd3000 0x00007ffff7fd5000 r-xp [vdso] 0x00007ffff7fd5000 0x00007ffff7fd6000 r--p /lib/x86 64-linux-gnu/ld-2.28.so 8x08807ffff7fd6088 0x88007ffff7ff4880 r-xp /lib/x86\_64-linux-gnu/ld-2.28.so AxAAAA7fffffffffffffaAAAA AxAAAAA7ffffffffffffffaAAA r...p /lib/x86\_64-linux-anu/ld-2.28.so Avanaatffffffffaffcann avanaa7fffffffdann r.... /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007ffff7ffd000 0x00007ffff7ffe000 rw-p /lib/x86\_64-linux-gnu/ld-2.28.so 8x08807ffff7ffe088 0x80007ffff7fff880 rw-p mapped 0x00007ffffffde000 0x00007fffffff000 rw-p [stack]

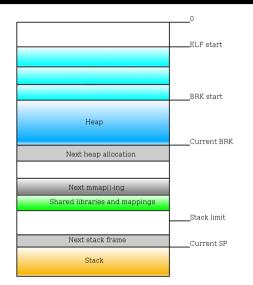
gdb-peda\$

Retaddr corruption is possible => anything in std lib can be called! DEMO How can we mitigate this?

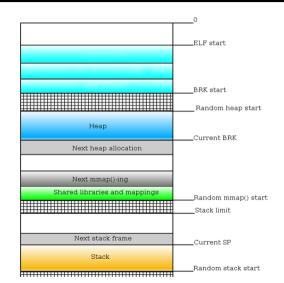
# ELF memory space (1/3)



# ELF memory space (2/3) + demo



## ELF memory space (3/3)



#### ASLR disabled (1/3)

[-----] 0000| 0x7fffffffde60 --> 0x401230 (< libc csu init>: push r15) 0008| 0x7ffffffde68 --> 0x7ffff7dea09b (< libc start main+235>: mov edi,eax) 0016| 0x7ffffffde70 --> 0x00024| 0x7fffffffde78 --> 0x7fffffffdf48 --> 0x7fffffffe291 ("/ctf/unibuc/curs re/curs "...) AA32| Ax7fffffffde80 --> Ax1000400000040 0x7fffffffde88 --> 0x4011ee (<main>: nush rbn)  $00481 0 \times 7 fffffffde 90 --> 0 \times 0$ 0056 0x7ffffffde98 --> 0x26f429881fe236d0 f-----1 Legend: code, data, rodata, value 0x00000000004011fc in main () adb-peda\$ vmmap Start End Perm Name 0×00400000 0×00401000 /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello r - - p /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello 0x00401000 0x00402000 r-xp 0×00402000 0x00403000 r - - p /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello AxAA4A3AAA AxAA4A4AAA r - - p /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello 0×00404000 0×00405000 rw-p 0×00405000 0×00426000 rw-n [hean] /lib/x86 64-linux-gnu/libc-2.28.so 0x00007ffff7dc6000 0x00007ffff7de8000 r--p 0x00007ffff7de8000 0x00007ffff7f30000 r-xp /lib/x86\_64-linux-gnu/libc-2.28.so /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007ffff7f30000 0x00007ffff7f7c000 r--p 0x00007ffff7f7c000 0x00007ffff7f7d000 ---p /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007ffff7f7d000 0x00007ffff7f81000 r--p /lib/x86\_64-linux-gnu/libc-2.28.so /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007ffff7f81000 0x00007ffff7f83000 rw-p 0x00007ffff7f83000 0x00007ffff7f87000 rw-p mapped 0x00007ffff7f87000 0x00007ffff7f89000 rw-n manned 0x00007ffff7fd0000 0x00007ffff7fd3000 r--p [vvar] 0x00007ffff7fd3000 0x00007ffff7fd5000 r-xp [vdso] 0x00007ffff7fd5000 0x00007ffff7fd6000 r--p /lib/x86 64-linux-anu/ld-2.28.so 0x00007ffff7fd6000 0x00007ffff7ff4000 r-xp /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007ffff7ff4000 0x00007ffff7ffc000 r--p /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007ffff7ffc000 0x00007ffff7ffd000 r--p /lib/x86\_64-linux-gnu/ld-2.28.so AxAAAA7ffffffffffdAAA AxAAAA7ffffffffffeAAA rw-n /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007ffff7ffe000 0x00007ffff7fff000 rw-p mapped 0x00007ffffffde000 0x00007ffffffff000 rw-n [stack] ddb-peda\$

#### ASLR enabled (2/3)

[-----] 00001 0x7ffd896dd730 --> 0x401230 (< libc csu init>: push r15) 0008| 0x7ffd896dd738 --> 0x7f51f040109b (< libc start main+235>: mov edi,eax) 0016| 0x7ffd896dd740 --> 0x00024| 0x7ffd896dd748 --> 0x7ffd896dd818 --> 0x7ffd896de291 ("/ctf/unibuc/curs re/curs "...) 00321 0x7ffd896dd750 --> 0x100040000 0040 0x7ffd896dd758 --> 0x4011ee (<main>: nush rbn)  $00481 0 \times 7 ff d 896 d d 760 - -> 0 \times 0$ 0056 0x7ffd896dd768 --> 0x611985a4db582665 f-----1 Legend: code, data, rodata, value 0x00000000004011fc in main () adb-peda\$ vmmap Start End Perm Name 0×00400000 0×00401000 /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello r - - p 0x00401000 0x00402000 /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello r-xp 0×00402000 0x00403000 r - - p /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello AxAA4A3AAA AxAA4A4AAA r - - n /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello 0×00404000 0x00405000 rw-p 0x0124f000 0×01270000 rw-n [hean] /lib/x86 64-linux-gnu/libc-2.28.so 0x00007f51f03dd000 0x00007f51f03ff000 r--p 0x00007f51f03ff000 0x00007f51f0547000 r-xp /lib/x86\_64-linux-gnu/libc-2.28.so /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007f51f0547000 0x00007f51f0593000 r--p 0x00007f51f0593000 0x00007f51f0594000 ---p /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007f51f0594000 0x00007f51f0598000 r--p /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007f51f0598000 0x00007f51f059a000 rw-p /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007f51f059a000 0x00007f51f059e000 rw-p mapped 0x00007f51f059e000 0x00007f51f05a0000 rw-n manned 0x00007f51f05e7000 0x00007f51f05e8000 r--n /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007f51f05e8000 0x00007f51f0606000 r-xp /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007f51f0606000 0x00007f51f060e000 r--p /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007f51f060e000 0x00007f51f060f000 r--p /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007f51f060f000 0x00007f51f0610000 rw-p /lib/x86 64-linux-gnu/ld-2.28.so 0x00007f51f0610000 0x00007f51f0611000 rw-p mapped AxAAAA7ffd896beAAA AxAAAA7ffd896dfAAA rw-n [stack] 0x00007ffd89779000 0x00007ffd8977c000 r--p [vvar] 0x00007ffd8977c000 0x00007ffd8977e000 r-xp [vdso] odb-neda\$

8 / 29

#### ASLR enabled (3/3)

[-----] 0000| 0x7ffflacb6c40 --> 0x401230 (< libc csu init>: push r15) 0008| 0x7ffflacb6c48 --> 0x7f5559bad09b (< libc start main+235>: mov edi,eax) 0016| 0x7ffflacb6c50 --> 0x00024| 0x7ffflacb6c58 --> 0x7ffflacb6d28 --> 0x7ffflacb7291 ("/ctf/unibuc/curs re/curs "...) 00321 0x7ffflach6c60 --> 0x100040000 0040 0x7ffflach6c68 --> 0x4011ee (<main>: nush rbn)  $00481 0 \times 7 ffflacb6c70 --> 0 \times 0$ 0056 0x7ffflacb6c78 --> 0xed442d94cd1c4c04 f-----1 Legend: code, data, rodata, value 0x00000000004011fc in main () adb-peda\$ vmmap Start End Perm Name 0×00400000 0×00401000 /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello r - - p 0x00401000 0x00402000 /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello r-xp 0×00402000 0x00403000 r - - p /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello AxAA4A3AAA AxAA4A4AAA r - - n /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello /ctf/unibuc/curs re/curs 06/demo 01 linux memory/hello 0×00404000 0x00405000 rw-p [hean] Аходаесоод 0x00b0d000 rw-n /lib/x86 64-linux-gnu/libc-2.28.so 0x00007f5559b89000 0x00007f5559bab000 r--p 0x00007f5559bab000 0x00007f5559cf3000 r-xp /lib/x86\_64-linux-gnu/libc-2.28.so /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007f5559cf3000 0x00007f5559d3f000 r--p 0x00007f5559d3f000 0x00007f5559d40000 ---p /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007f5559d40000 0x00007f5559d44000 r--p /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007f5559d44000 0x00007f5559d46000 rw-p /lib/x86\_64-linux-gnu/libc-2.28.so 0x00007f5559d46000 0x00007f5559d4a000 rw-p mapped 0x00007f5559d4a000 0x00007f5559d4c000 rw-n manned /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007f5559d93000 0x00007f5559d94000 r--p 0x00007f5559d94000 0x00007f5559db2000 r-xp /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007f5559db2000 0x00007f5559dba000 r--p /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007f5559dba000 0x00007f5559dbb000 r--p /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007f5559dbb000 0x00007f5559dbc000 rw-p /lib/x86\_64-linux-gnu/ld-2.28.so 0x00007f5559dbc000 0x00007f5559dbd000 rw-p mapped 0x00007ffflac97000 0x00007ffflacb8000 rw-n [stack] 0x00007fff1ad0c000 0x00007fff1ad0f000 r--p [vvar] 0x00007fff1ad0f000 0x00007fff1ad11000 r-xp [vdso] ddb-peda\$

## ASLR info

- All maps randomized (except main exe)
- Recently, main exe also randomized
- Linux: ASLR system-wide
- Windows: ASLR system-wide (with a catch)

#### Implementation by the compiler

How does an executable know where the libraries reside?

- How does an executable know where the libraries reside?
- On Linux, loader copies pointers to imported functions in a section called .GOT

```
gdb-peda$ telescope 0x404008
0000| 0x404008 --> 0x7ffff7ffe190 --> 0x0
0008 0x404010 --> 0x7ffff7fea440 (< dl runtime resolve xsave>: push
                                                                         rbx)
0016 0x404018 --> 0x7ffff7e37b10 (<puts>:
                                                 push
                                                        r13)
0024| 0x404020 --> 0x7ffff7ele710 (< printf>: sub
                                                        rsp.0xd8)
0032| 0x404028 --> 0x7ffff7e4a570 (< GI
                                          libc malloc>:
                                                                  push
                                                                         r13)
0040 0x404030 --> 0x7ffff7e33ac0 (< isoc99 scanf>:
                                                                rbx)
                                                         push
0048 0x404038 --> 0x401076 (<exit@plt+6>:
                                                 push
                                                        0x4)
00561 0 \times 404040 - -> 0 \times 0
```

On Windows, the loader simply overwrites wherever a function is called (fixup)

## The end of (usefulness for) Buffer overflows?

- In full RCE exploits we want to call system("/bin/sh") to open a remote shell
- Most programs do not call "system"
- Address of libc (and thus "system" ) is randomized
- We are "contained" within the program functionality
- Can this protection be bypassed?

## The end of (usefulness for) Buffer overflows?

- In full RCE exploits we want to call system("/bin/sh") to open a remote shell
- Most programs do not call "system"
- Address of libc (and thus "system" ) is randomized
- We are "contained" within the program functionality
- Can this protection be bypassed?
- Of course! But it's pretty tricky.

#### Buffer overflow: no corruption

[------code------] 0x4011b2 <vuln+24>: mov OWORD PTR [rbp-0x10].0x0 0x4011ba <vuln+32>: mov OWORD PTR [rbp-0x8].0x0 0x4011c2 <vuln+40>: lea rax,[rbp-0x20] 0x4011c6 <vuln+44>: mov rsi.rax 0x4011c9 <vuln+47>: lea rdi,[rip+0xe34] # 0x402004 0x4011d0 <vuln+54>: eax,0x0 mov 0x401040 < isoc99 scanf@plt> 0x4011d5 <vuln+59>: call => 0x4011da <vuln+64>: 0x4011db <vuln+65>: leave 0x4011dc <vuln+66>: ret 0x4011dd <main>: push rbp 0x4011de <main+1>: mov rbp,rsp 0x4011e1 <main+4>: mov eax.0x0 0x4011e6 <main+9>: call 0x40116b <setup> 0x4011eb <main+14>: mov eax,0x0 0x4011f0 <main+19>: call 0x40119a <vuln> [-----] 0000| 0x7ffc06c241e0 ("AAAAAAAA") 00081 0x7ffc06c241e8 --> 0x0 0016| 0x7ffc06c241f0 --> 0x0 0024 0x7ffc06c241f8 --> 0x0 0032| 0x7ffc06c24200 --> 0x7ffc06c24210 --> 0x401200 (< libc csu init>: DUS 0040| 0x7ffc06c24208 --> 0x4011f5 (<main+24>: mov eax.0x0) 0048 0x7ffc06c24210 --> 0x401200 (< libc csu init>: push r15) 0056| 0x7ffc06c24218 --> 0x7fd582bfe09b (< libc start main+235>: edi. mov 1------1

#### Buffer overflow: calling one function

[-----] 0x4011b2 <vuln+24>: mov OWORD PTR [rbp-0x10].0x0 0x4011ba <vuln+32>: mov OWORD PTR [rbp-0x8].0x0 0x4011c2 <vuln+40>: lea rax,[rbp-0x20] 0x4011c6 <vuln+44>: rsi.rax mov 0x4011c9 <vuln+47>: lea rdi,[rip+0xe34] # 0x402004 0x4011d0 <vuln+54>: eax,0x0 mov 0x4011d5 <vuln+59>: call 0x401040 < isoc99 scanf@plt> => 0x4011da <vuln+64>: 0x4011db <vuln+65>: leave 0x4011dc <vuln+66>: ret 0x4011dd <main>: push rbp 0x4011de <main+1>: mov rbp,rsp 0x4011e1 <main+4>: mov eax.0x0 0x4011e6 <main+9>: call 0x40116b <setup> 0x4011eb <main+14>: mov eax,0x0 0x4011f0 <main+19>: call 0x40119a <vuln> [-----stack-----1 00001 0x7fff87097b70 ("AAAAAAABBBBBBBBBCCCCCCCCD"...) 0008 0x7fff87097b78 ("BBBBBBBBBCCCCCCCDDDDDDDE"...) 0016 0x7fff87097b80 ("CCCCCCCDDDDDDDDEEEEEEEV"...) 0024 0x7fff87097b88 ("DDDDDDDDDEEEEEEEEV\021@") 00321 0x7fff87097b90 ("EEEEEEEEV\021@") 0040| 0x7fff87097b98 --> 0x401156 (<f1>: push rbp)0048 0x7fff87097ba0 --> 0x401200 (< libc csu init>: push r15) 0056| 0x7fff87097ba8 --> 0x7f521209709b (< libc start main+235>: edi. mov 

#### Buffer overflow: calling more functions

[------code------1 0x4011b2 <vuln+24>: mov OWORD PTR [rbp-0x10].0x0 0x4011ba <vuln+32>: mov OWORD PTR [rbp-0x8].0x0 0x4011c2 <vuln+40>: lea rax,[rbp-0x20] 0x4011c6 <vuln+44>: rsi.rax mov rdi,[rip+0xe34] # 0x402004 0x4011c9 <vuln+47>: lea 0x4011d0 <vuln+54>: eax,0x0 mov 0x401040 < isoc99 scanf@plt> 0x4011d5 <vuln+59>: call => 0x4011da <vuln+64>: 0x4011db <vuln+65>: leave 0x4011dc <vuln+66>: ret 0x4011dd <main>: push rbp 0x4011de <main+1>: mov rbp,rsp 0x4011e1 <main+4>: mov eax.0x0 0x4011e6 <main+9>: call 0x40116b <setup> 0x4011eb <main+14>: mov eax,0x0 0x4011f0 <main+19>: call 0x40119a <vuln> [-----stack-----] 00001 0x7ffc850b7f80 ("AAAAAAABBBBBBBBBBCCCCCCCCD"...) 0008 0x7ffc850b7f88 ("BBBBBBBBBCCCCCCCDDDDDDDE"...) 0016 0x7ffc850b7f90 ("CCCCCCCDDDDDDDDEEEEEEEEV"...) 0024 0x7ffc850b7f98 ("DDDDDDDDDEEEEEEEEV\021@") 00321 0x7ffc850b7fa0 ("EEEEEEEEV\021@") 0040| 0x7ffc850b7fa8 --> 0x401156 (<f1>: push rbp) 0048| 0x7ffc850b7fb0 --> 0x40115d (<f2>: push rbp) 0056| 0x7ffc850b7fb8 --> 0x40115d (<f2>: push rbp) [-----]

#### Buffer overflow: calling... nothing

```
[-----]
  0x4011b2 <vuln+24>: mov
                        OWORD PTR [rbp-0x10].0x0
  0x4011ba <vuln+32>: mov
                        OWORD PTR [rbp-0x8].0x0
  0x4011c2 <vuln+40>: lea
                        rax,[rbp-0x20]
  0x4011c6 <vuln+44>:
                        rsi.rax
                   mov
  0x4011c9 <vuln+47>:
                   lea
                        rdi,[rip+0xe34] # 0x402004
  0x4011d0 <vuln+54>:
                        eax,0x0
                   mov
  0x4011d5 <vuln+59>:
                   call
                         0x401040 < isoc99 scanf@plt>
=> 0x4011da <vuln+64>:
  0x4011db <vuln+65>:
                  leave
  0x4011dc <vuln+66>:
                   ret
  0x4011dd <main>:
                   push
                         rbp
  0x4011de <main+1>:
                   mov
                        rbp,rsp
  0x4011e1 <main+4>:
                   mov
                        eax.0x0
  0x4011e6 <main+9>:
                  call 0x40116b <setup>
  0x4011eb <main+14>: mov
                        eax,0x0
  0x4011f0 <main+19>: call 0x40119a <vuln>
[-----stack-----1
00001 0x7ffed41f87f0 ("AAAAAAABBBBBBBBBBCCCCCCCCD"...)
0008 0x7ffed41f87f8 ("BBBBBBBBBCCCCCCCDDDDDDDE"...)
0016 0x7ffed41f8800 ("CCCCCCCDDDDDDDEEEEEEE", <incomplete sequence \334>...)
0024 0x7ffed41f8808 ("DDDDDDDDEEEEEEEE\334\021@")
0032 0x7ffed41f8810 ("EEEEEEEEL\334\021@")
0040| 0x7ffed41f8818 --> 0x4011dc (<vuln+66>:
                                      ret)
0048| 0x7ffed41f8820 --> 0x4011dc (<vuln+66>: ret)
0056 0x7ffed41f8828 --> 0x4011dc (<vuln+66>:
                                      ret)
[-----]
```

- The CPU basically executes: "ret; ret; ret"
- Similar to a program with only NOPs

- The CPU basically executes: "ret; ret; ret"
- Similar to a program with only NOPs
- How would the following piece of code be useful?

0x401c55: POP RBX 0x401c56: RET ROP



- Code reuse at a finer level
- Benign pieces of code reused: gadgets
- Does not assume code modification (shellcode)

## Function call convention (Linux)

- Params: RDI, RSI, RDX, RCX, R8, R9
- Ret: RAX

## Function call convention (Linux)

- Params: RDI, RSI, RDX, RCX, R8, R9
- Ret: RAX
- Reconstruct parameter passing using ROP gadgets

```
0x401aef: POP RDI
0x401af0: RET
...
0x40231c: POP RSI
0x40231d: RET
```

#### Example ROP chain

- ADDR + 0x00 => 0x401aef (POP RDI ; RET)
- ADDR + 0x08 => 0x406020 "%s"
- ADDR + 0x10 => 0x40145a (POP RSI; RET)
- **ADDR** + 0x18 => 0x12345
- ADDR + 0x20 => 0x401508 (\_\_isoc99\_scanf@plt)

#### Example ROP chain

ADDR + 0x00 => 0x401aef (POP RDI ; RET) ADDR + 0x08 => 0x406020 "%s" ADDR + 0x10 => 0x40145a (POP RSI; RET) ADDR + 0x18 => 0x12345 ADDR + 0x20 => 0x401508 (\_\_isoc99\_scanf@plt) A hit or a set in the set of a set of

Arbitrary read into address 0x12345

#### ROP usefulness

- Extra degrees of freedom
- Reuse functions and bits of functions in a more clever way
- Execute multi-stage exploits

- Now we can "program" in terms of reusing the code
- In a previous slide we show how to arbitrarily read into an address
- Consider the following pseudo-program written in ROP:

call puts to leak memory (obtain the loader pointers) call main again

- Now we can "program" in terms of reusing the code
- In a previous slide we show how to arbitrarily read into an address
- Consider the following pseudo-program written in ROP:

call puts to leak memory (obtain the loader pointers) call main again

- We have information regarding the randomness
- ASLR is defeated!

#### Information leak alternatives

- The larger the systems/programs the more vulnerabilities
- Find and use another weaker vulnerability to gain info
- Here's an example
- Source: https://twitter.com/0xRaindrop/status/864704956116254720

#### FSB step 1: change phone name



#### FSB step 2: pair BMW with phone



#### FSB step 3: profit



## FSB example 2

	Safa	ri	File	Edit	View	Histor	/ Bookn	narks	Develop	Window	Help					
	0									rui	nic.pl/te	sty/sandb	ox/forma	t.html		
۹		2	+ 💽	runic.p	l/testy	/sandbox	/format.htm	nl								
	III Ap	ople	iCloud	Face	book	Twitter	Wikipedia	Yahoo	News 🔻	Popular ▼						
								?		0 ffff8076c49 7f893b53d 7f893b53e 7f893b53e 7f893b53e 7f893b625 7ff58a685 7fff58a673 0 7f893b625 7ff93b626 7f893b626 7f893b534 7f893b706	0x0 0-2795f 6a0 d10 1d0 280 7035322 8f0 60 c0 0x23 f10 78 f60 ff0 470 tore for a on on you	0 0. 0x7 0x7f8 0x7f8 0x7f8 0x7f89 0x7f89 0x7f89 0x7f89 0x7f89 0x7ff9	x <sup>7</sup> f893b6 7f893b81 93b40bc 0x0 22000000 93b62ebl 0x7f893b 93b6386 i3b534ff0 7197b88 0x7f893b 0x1 3b7064at 0x0 758a6856 in that can can	19340 4290 40 0x7 000 56258ft a0 441841 0x7f 0 0x7ft 0. open this	0x7f893b622ff 0x0 f893b608cf0 0x7fff78a444d0 0 0x20 0x20 0x7fff8e6e1080 0x64	

#### FSB example 3

Generation Prompt
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\mark>sort AAAAAAABBBBBBBB%x%x%xx%x~%x-%x%x%x%x%x%x%x%x%x%x AAAAAAAABBBBBBBBa2d43bbe5dc0064a5c14-a2c617e0-a2d426902ea2c5dcfe4141414142424242 782578252d78252d78257825782578257825The system cannot find the file specified.

C:\Users\mark>sort AAAAAAAABBBBBBBB%x%x%x%x-%s-%x%x%x%x%x%x%x%x%x AAAAAAAABBBBBBBBd3453bbe5e5c064a5c14-SsHd,-d34526902ed341d94e4141414142424242782 578252d73252d7825782578257825The system cannot find the file specified.

C:\Users\mark>

#### Practice

- Any Questions?
- http://pwnthybytes.ro/unibuc\_re/06-lab.html