Binary Reverse Engineering And Analysis

Course 6: ASLR and ROP

Caragea Radu

March 23, 2021

Recap

- 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)

```
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffde60 --> 0x401230 (<__libc_csu_init>:   push   r15)
0008| 0x7fffffffde68 --> 0x7ffff7dea09b (<__libc_start_main+235>:       mov    edi,eax)
0016| 0x7fffffffde70 --> 0x0
0024| 0x7fffffffde78 --> 0x7fffffffdf48 --> 0x7fffffffe291 ("/ctf/unibuc/curs_re/curs_"...)
0032| 0x7fffffffde80 --> 0x100040000
0040| 0x7fffffffde88 --> 0x4011ee (<main>:      push   rbp)
0048| 0x7fffffffde90 --> 0x0
0056| 0x7fffffffde98 --> 0x26f429881fe236d0

[-----------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x00000000004011fc in main ()
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x00401000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00401000         0x00402000         r-xp      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00402000         0x00403000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00403000         0x00404000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00404000         0x00405000         rw-p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00405000         0x00426000         rw-p      [heap]
0x00007ffff7dc6000 0x00007ffff7de8000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7de8000 0x00007ffff7f30000 r-xp      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7f30000 0x00007ffff7f7c000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
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
0x00007ffff7f81000 0x00007ffff7f83000 rw-p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7f83000 0x00007ffff7f87000 rw-p      mapped
0x00007ffff7f87000 0x00007ffff7f89000 rw-p      mapped
0x00007ffff7fd0000 0x00007ffff7fd3000 r--p      [vvar]
0x00007ffff7fd3000 0x00007ffff7fd5000 r-xp      [vdso]
0x00007ffff7fd5000 0x00007ffff7ff6000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007ffff7ff6000 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
0x00007ffff7ffd000 0x00007ffff7ffe000 rw-p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007ffff7ffe000 0x00007ffff7fff000 rw-p      mapped
0x00007fffffffde000 0x00007fffffffff000 rw-p    [stack]
gdb-peda$
```

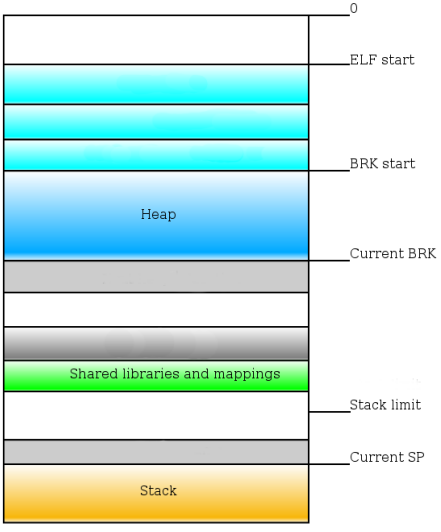
Retaddr corruption is possible => anything in std lib can be called! DEMO

Linux address space (in the olden times)

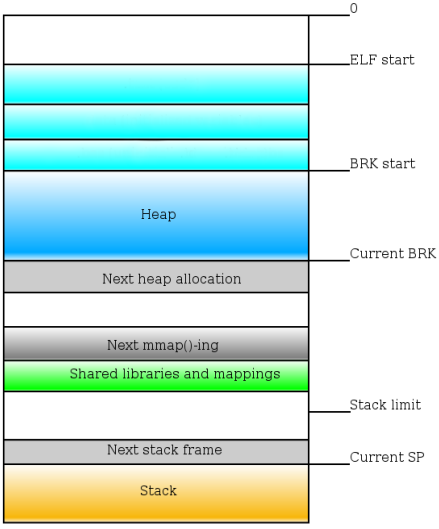
```
[-------------------------------------stack--------------------------------------]
0000| 0x7fffffffde60 --> 0x401230 (<__libc_csu_init>:   push   r15)
0008| 0x7fffffffde68 --> 0x7ffff7dea09b (<__libc_start_main+235>:       mov    edi,eax)
0016| 0x7fffffffde70 --> 0x0
0024| 0x7fffffffde78 --> 0x7fffffffdf48 --> 0x7fffffffe291 ("/ctf/unibuc/curs_re/curs_"...)
0032| 0x7fffffffde80 --> 0x100040000
0040| 0x7fffffffde88 --> 0x4011ee (<main>:      push   rbp)
0048| 0x7fffffffde90 --> 0x0
0056| 0x7fffffffde98 --> 0x26f429881fe236d0

[-----------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x00000000004011fc in main ()
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x00401000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00401000         0x00402000         r-xp      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00402000         0x00403000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00403000         0x00404000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00404000         0x00405000         rw-p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00405000         0x00426000         rw-p      [heap]
0x00007ffff7dc6000 0x00007ffff7de8000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7de8000 0x00007ffff7f30000 r-xp      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7f30000 0x00007ffff7f7c000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
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
0x00007ffff7f81000 0x00007ffff7f83000 rw-p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7f83000 0x00007ffff7f87000 rw-p      mapped
0x00007ffff7f87000 0x00007ffff7f89000 rw-p      mapped
0x00007ffff7fd0000 0x00007ffff7fd3000 r--p      [vvar]
0x00007ffff7fd3000 0x00007ffff7fd5000 r-xp      [vdso]
0x00007ffff7fd5000 0x00007ffff7ff6000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007ffff7ff6000 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
0x00007ffff7ffd000 0x00007ffff7ffe000 rw-p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007ffff7ffe000 0x00007ffff7fff000 rw-p      mapped
0x00007fffffffde000 0x00007fffffffff000 rw-p    [stack]
gdb-peda$
```

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

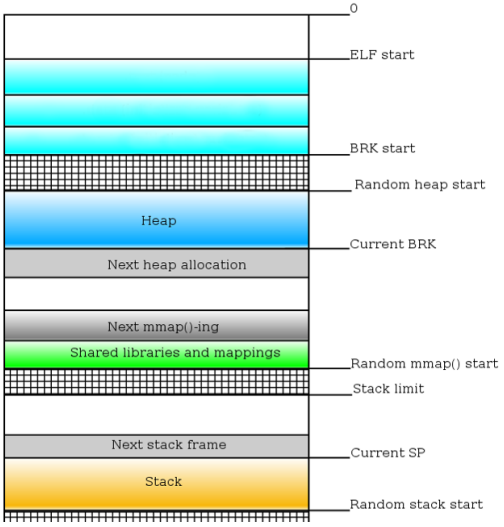
ELF memory space (1/3)



ELF memory space (2/3) + demo



ELF memory space (3/3)



ASLR disabled (1/3)

```
[----------------------------------stack-------------------------------------]
0000| 0x7fffffffde60 --> 0x401230 (<__libc_csu_init>:   push   r15)
0008| 0x7fffffffde68 --> 0x7ffff7dea09b (<__libc_start_main+235>:       mov    edi,eax)
0016| 0x7fffffffde70 --> 0x0
0024| 0x7fffffffde78 --> 0x7fffffffdf48 --> 0x7fffffffe291 ("/ctf/unibuc/curs_re/curs_"...)
0032| 0x7fffffffde80 --> 0x100040000
0040| 0x7fffffffde88 --> 0x4011ee (<main>:      push   rbp)
0048| 0x7fffffffde90 --> 0x0
0056| 0x7fffffffde98 --> 0x26f429881fe236d0

[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x00000000004011fc in main ()
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x00401000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00401000         0x00402000         r-xp      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00402000         0x00403000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00403000         0x00404000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00404000         0x00405000         rw-p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00405000         0x00426000         rw-p      [heap]
0x00007ffff7dc6000 0x00007ffff7de8000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7de8000 0x00007ffff7f30000 r-xp      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7f30000 0x00007ffff7f7c000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
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
0x00007ffff7f81000 0x00007ffff7f83000 rw-p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7f83000 0x00007ffff7f87000 rw-p      mapped
0x00007ffff7f87000 0x00007ffff7f89000 rw-p      mapped
0x00007ffff7fd0000 0x00007ffff7fd3000 r--p      [vvar]
0x00007ffff7fd3000 0x00007ffff7fd5000 r-xp      [vdso]
0x00007ffff7fd5000 0x00007ffff7fd6000 r--p      /lib/x86_64-linux-gnu/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
0x00007ffff7ffd000 0x00007ffff7ffe000 rw-p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007ffff7ffe000 0x00007ffff7fff000 rw-p      mapped
0x00007ffffffde000 0x00007ffffffff000 rw-p      [stack]
gdb-peda$
```

ASLR enabled (2/3)

```
[----------------------------------stack-------------------------------------]
0000| 0x7ffd896dd730 --> 0x401230 (<__libc_csu_init>:   push   r15)
0008| 0x7ffd896dd738 --> 0x7f51f040109b (<__libc_start_main+235>:       mov    edi,eax)
0016| 0x7ffd896dd740 --> 0x0
0024| 0x7ffd896dd748 --> 0x7ffd896dd818 --> 0x7ffd896de291 ("/ctf/unibuc/curs_re/curs_"...)
0032| 0x7ffd896dd750 --> 0x100040000
0040| 0x7ffd896dd758 --> 0x4011ee (<main>:      push   rbp)
0048| 0x7ffd896dd760 --> 0x0
0056| 0x7ffd896dd768 --> 0x611985a4db582665

[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x00000000004011fc in main ()
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x00401000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00401000         0x00402000         r-xp      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00402000         0x00403000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00403000         0x00404000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00404000         0x00405000         rw-p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x0124f000         0x01270000         rw-p      [heap]
0x00007f51f03dd000 0x00007f51f03ff000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f51f03ff000 0x00007f51f0547000 r-xp      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f51f0547000 0x00007f51f0593000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
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-p      mapped
0x00007f51f05e7000 0x00007f51f05e8000 r--p      /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
0x00007ffd896be000 0x00007ffd896df000 rw-p      [stack]
0x00007ffd89779000 0x00007ffd8977c000 r--p      [vvar]
0x00007ffd8977c000 0x00007ffd8977e000 r-xp      [vdso]
gdb-peda$
```

ASLR enabled (3/3)

```
[-------------------------------------stack-------------------------------------]
0000| 0x7fff1acb6c40 --> 0x401230 (<__libc_csu_init>:   push   r15)
0008| 0x7fff1acb6c48 --> 0x7f5559bad09b (<__libc_start_main+235>:       mov    edi,eax)
0016| 0x7fff1acb6c50 --> 0x0
0024| 0x7fff1acb6c58 --> 0x7fff1acb6d28 --> 0x7fff1acb7291 ("/ctf/unibuc/curs_re/curs_"...)
0032| 0x7fff1acb6c60 --> 0x100040000
0040| 0x7fff1acb6c68 --> 0x4011ee (<main>:      push   rbp)
0048| 0x7fff1acb6c70 --> 0x0
0056| 0x7fff1acb6c78 --> 0xed442d94cd1c4c04

[-------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x00000000004011fc in main ()
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x00401000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00401000         0x00402000         r-xp      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00402000         0x00403000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00403000         0x00404000         r--p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00404000         0x00405000         rw-p      /ctf/unibuc/curs_re/curs_06/demo_01_linux_memory/hello
0x00aec000         0x00b0d000         rw-p      [heap]
0x00007f5559b89000 0x00007f5559bab000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f5559bab000 0x00007f5559cf3000 r-xp      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f5559cf3000 0x00007f5559d3f000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
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-p      mapped
0x00007f5559d93000 0x00007f5559d94000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
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
0x00007fff1ac97000 0x00007fff1acb8000 rw-p      [stack]
0x00007fff1ad0c000 0x00007fff1ad0f000 r--p      [vvar]
0x00007fff1ad0f000 0x00007fff1ad11000 r-xp      [vdso]
gdb-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?

Implementation by the compiler

- 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 --> 0x7ffff7e1e710 (<__printf>:  sub    rsp,0xd8)
0032| 0x404028 --> 0x7ffff7e4a570 (<__GI__libc_malloc>:        push   r13)
0040| 0x404030 --> 0x7ffff7e33ac0 (<__isoc99_scanf>:    push   rbx)
0048| 0x404038 --> 0x401076 (<exit@plt+6>:      push   0x4)
0056| 0x404040 --> 0x0
```

- 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?

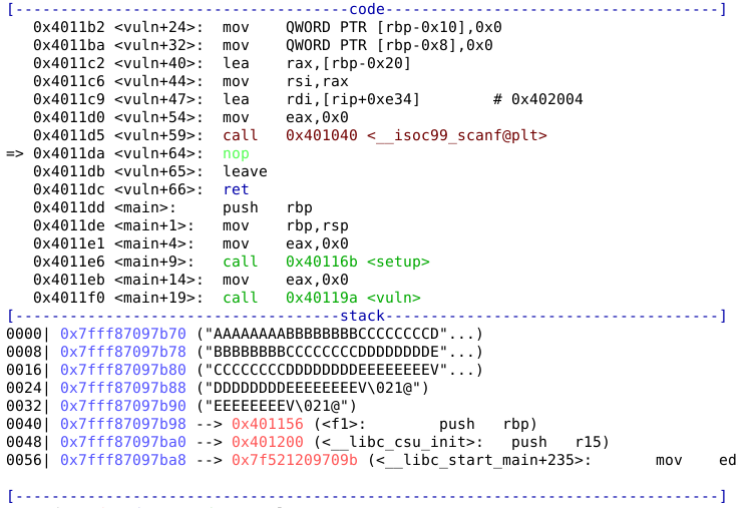
The end of (usefulness for) Buffer overflows?

- 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    QWORD PTR [rbp-0x10],0x0
   0x4011ba <vuln+32>:  mov    QWORD 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>:  mov    eax,0x0
   0x4011d5 <vuln+59>:  call   0x401040 <__isoc99_scanf@plt>
=> 0x4011da <vuln+64>:  nop
   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-----------------------------------]
0000| 0x7ffc06c241e0 ("AAAAAAAA")
0008| 0x7ffc06c241e8 --> 0x0
0016| 0x7ffc06c241f0 --> 0x0
0024| 0x7ffc06c241f8 --> 0x0
0032| 0x7ffc06c24200 --> 0x7ffc06c24210 --> 0x401200 (<__libc_csu_init>:        pus
0040| 0x7ffc06c24208 --> 0x4011f5 (<main+24>:   mov    eax,0x0)
0048| 0x7ffc06c24210 --> 0x401200 (<__libc_csu_init>:   push   r15)
0056| 0x7ffc06c24218 --> 0x7fd582bfe09b (<__libc_start_main+235>:       mov    edi,

[--------------------------------------------------------------------------]
```

Buffer overflow: calling one function



Buffer overflow: calling more functions

```
[-------------------------------------code-------------------------------------]
   0x4011b2 <vuln+24>:  mov    QWORD PTR [rbp-0x10],0x0
   0x4011ba <vuln+32>:  mov    QWORD 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>:  mov    eax,0x0
   0x4011d5 <vuln+59>:  call   0x401040 <__isoc99_scanf@plt>
=> 0x4011da <vuln+64>:  nop
   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-------------------------------------]
0000| 0x7ffc850b7f80 ("AAAAAAAABBBBBBBBCCCCCCCCD"...)
0008| 0x7ffc850b7f88 ("BBBBBBBBCCCCCCCCDDDDDDDDE"...)
0016| 0x7ffc850b7f90 ("CCCCCCCCDDDDDDDDEEEEEEEEV"...)
0024| 0x7ffc850b7f98 ("DDDDDDDDEEEEEEEEV\021@")
0032| 0x7ffc850b7fa0 ("EEEEEEEEV\021@")
0040| 0x7ffc850b7fa8 --> 0x401156 (<f1>:        push   rbp)
0048| 0x7ffc850b7fb0 --> 0x40115d (<f2>:        push   rbp)
0056| 0x7ffc850b7fb8 --> 0x401164 (<f3>:        push   rbp)

[------------------------------------------------------------------------------]
```

Buffer overflow: calling... nothing

```
[-------------------------------------code-------------------------------------]
   0x4011b2 <vuln+24>:  mov    QWORD PTR [rbp-0x10],0x0
   0x4011ba <vuln+32>:  mov    QWORD 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>:  mov    eax,0x0
   0x4011d5 <vuln+59>:  call   0x401040 <__isoc99_scanf@plt>
=> 0x4011da <vuln+64>:  nop
   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------------------------------------]
0000| 0x7ffed41f87f0 ("AAAAAAAABBBBBBBBCCCCCCCCD"...)
0008| 0x7ffed41f87f8 ("BBBBBBBBCCCCCCCCDDDDDDDDE"...)
0016| 0x7ffed41f8800 ("CCCCCCCCDDDDDDDDEEEEEEEE", <incomplete sequence \334>...)
0024| 0x7ffed41f8808 ("DDDDDDDDEEEEEEEE\334\021@")
0032| 0x7ffed41f8810 ("EEEEEEEE\334\021@")
0040| 0x7ffed41f8818 --> 0x4011dc (<vuln+66>:   ret)
0048| 0x7ffed41f8820 --> 0x4011dc (<vuln+66>:   ret)
0056| 0x7ffed41f8828 --> 0x4011dc (<vuln+66>:   ret)

[------------------------------------------------------------------------------]
```

Why?

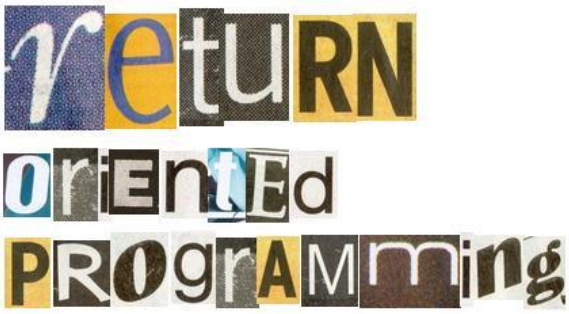
- The CPU basically executes: "ret; ret; ret"
- Similar to a program with only NOPs

Why?

- 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)
```

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

So how does the ASLR bypass work?

- 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
```

So how does the ASLR bypass work?

- 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:

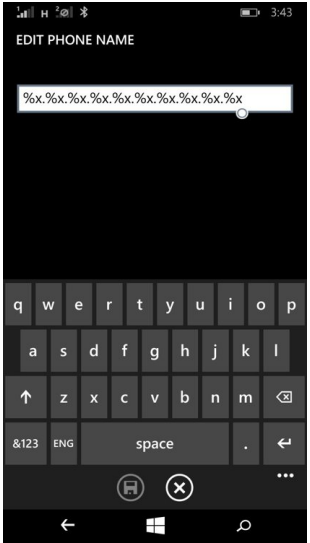
```
call puts to leak memory (obtain the loader pointers)
call main again
```

- 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`

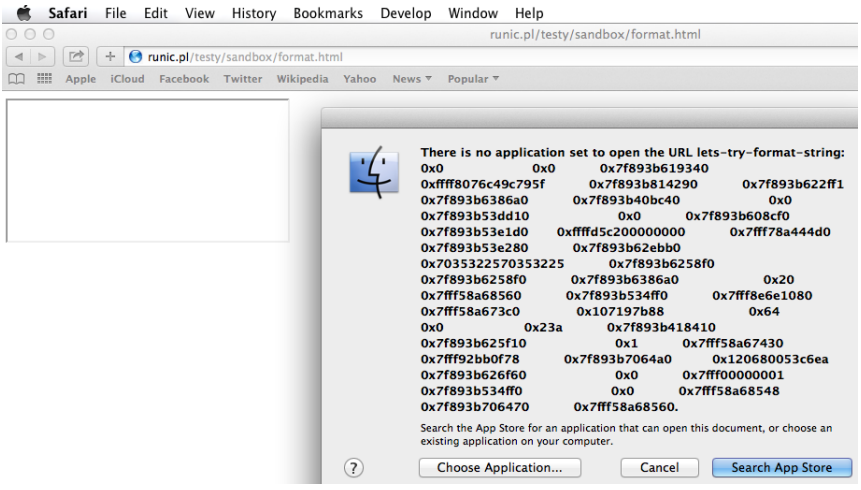
FSB step 1: change phone name



FSB step 2: pair BMW with phone

FSB step 3: profit

FSB example 2



FSB example 3

Practice

- Any Questions?
- `http://pwnthybytes.ro/unibuc_re/06-lab.html`