Binary Reverse Engineering And Analysis

Course 7: Mitigations and Bypasses

Caragea Radu

March 30, 2021

Recap

- Last time we studied ROP and ASLR
- Some information regarding the GOT
- Stack buffer overflows are pretty dangerous.
- What mitigations are available?

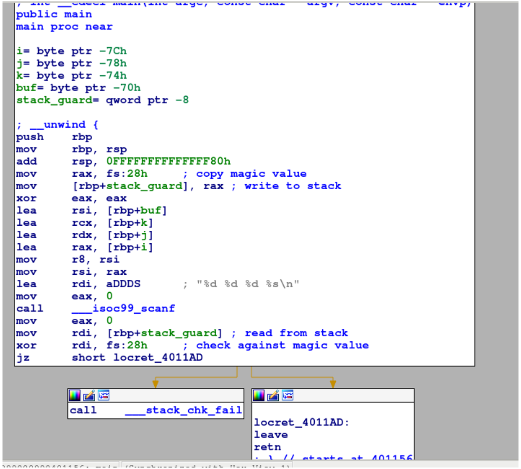
Today

- Preventing return address overflows (SSP)
- How dynamic linking works at runtime
- More advanced mitigations

Preventing stack buffer overflows

- Linux (gcc) and Windows (cl) adopt similar strategies
- Buffers are moved to the bottom of the stack frame
- A magic value is placed after all allocated variables and buffers
- Before returning, the magic value is checked
- Called: cookie or canary or guard

Stack smashing protector (Linux)



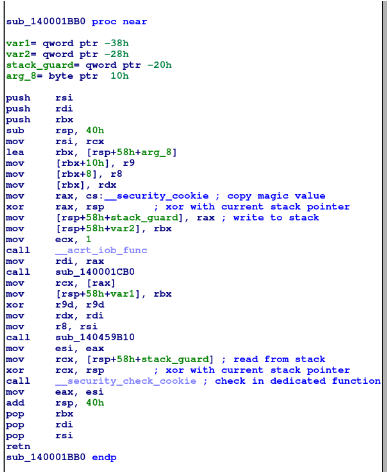
```
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

i= byte ptr -7Ch
j= byte ptr -78h
k= byte ptr -74h
buf= byte ptr -70h
stack_guard= qword ptr -8

; __unwind {
push    rbp
mov     rbp, rsp
add     rsp, 0FFFFFFFFFFFFFF80h
mov     rax, fs:28h     ; copy magic value
mov     [rbp+stack_guard], rax ; write to stack
xor     eax, eax
lea     rsi, [rbp+buf]
lea     rcx, [rbp+k]
lea     rdx, [rbp+j]
lea     rax, [rbp+i]
mov     r8, rsi
mov     rsi, rax
lea     rdi, aDDDS      ; "%d %d %d %s\n"
mov     eax, 0
call    ___isoc99_scanf
mov     eax, 0
mov     rdi, [rbp+stack_guard] ; read from stack
xor     rdi, fs:28h     ; check against magic value
jz      short locret_4011AD
```

```
call    ___stack_chk_fail
```

```
locret_4011AD:
leave
retn
; } // starts at 401156
```

- On Linux: compile with ‘-fstack-protector‘ (off by default)

Stack smashing protector (Windows)



```
sub_140001BB0 proc near

var1= qword ptr -38h
var2= qword ptr -28h
stack_guard= qword ptr -20h
arg_8= byte ptr  10h

push    rsi
push    rdi
push    rbx
sub     rsp, 40h
mov     rsi, rcx
lea     rbx, [rsp+58h+arg_8]
mov     [rbx+10h], r9
mov     [rbx+8], r8
mov     [rbx], rdx
mov     rax, cs:__security_cookie ; copy magic value
xor     rax, rsp          ; xor with current stack pointer
mov     [rsp+58h+stack_guard], rax ; write to stack
mov     [rsp+58h+var2], rbx
mov     ecx, 1
call    __acrt_iob_func
mov     rdi, rax
call    sub_140001CB0
mov     rcx, [rax]
mov     [rsp+58h+var1], rbx
xor     r9d, r9d
mov     rdx, rdi
mov     r8, rsi
call    sub_140459B10
mov     esi, eax
mov     rcx, [rsp+58h+stack_guard] ; read from stack
xor     rcx, rsp          ; xor with current stack pointer
call    __security_check_cookie ; check in dedicated function
mov     eax, esi
add     rsp, 40h
pop     rbx
pop     rdi
pop     rsi
retn
sub_140001BB0 endp
```

- On Windows: compile with '/GS' (on by default)

SSP pros and cons

- On Linux, the original value is at a hard-to-determine address

SSP pros and cons

- On Linux, the original value is at a hard-to-determine address
- On Windows, the original value is in the .data section
- However, it is xored with rsp for added security

SSP pros and cons

- On Linux, the original value is at a hard-to-determine address
- On Windows, the original value is in the .data section
- However, it is xored with rsp for added security
- In both cases, there are scenarios where it does not protect from overflows. Which?

The end of buffer overflows?

- Maybe... but not really.

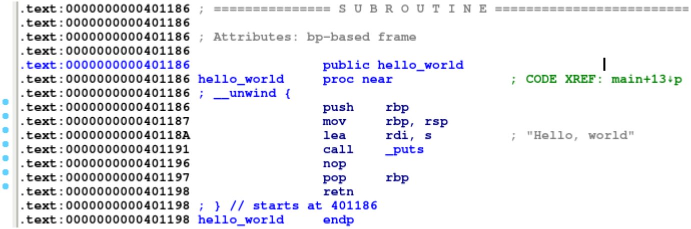
The end of buffer overflows?

- Maybe... but not really.
- Information leaks (very common)
- Buffer underflows can also occur
- Out-of-bounds access (very common)
  - Relative read/write (jump over the cookie)
  - Absolute read/write
- Heap abuse (dynamic allocation)

RELRO mitigation intro

- Protects the GOT table
- To understand why, let's dig into dynamic linking
- Through this mitigation we'll learn a new exploitation avenue

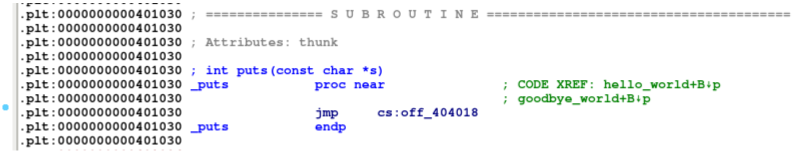
Dynamic linking (1/3)



```
.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
```

A program function calls puts("Hello, world")

Dynamic linking (2/3)



```
.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
```

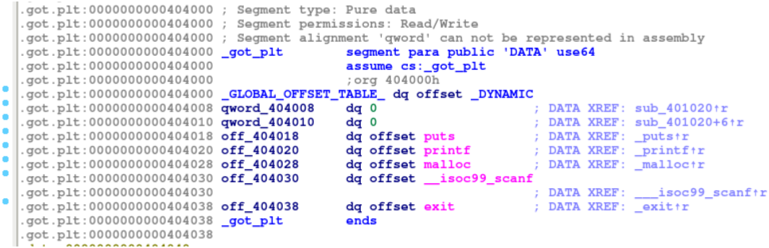
Puts() is actually a stub that uses a pointer from another table

Dynamic linking (2/3)

THANK YOU MARIO!

BUT OUR PRINCESS IS IN
ANOTHER CASTLE!

Dynamic linking (3/3)



```
.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
```

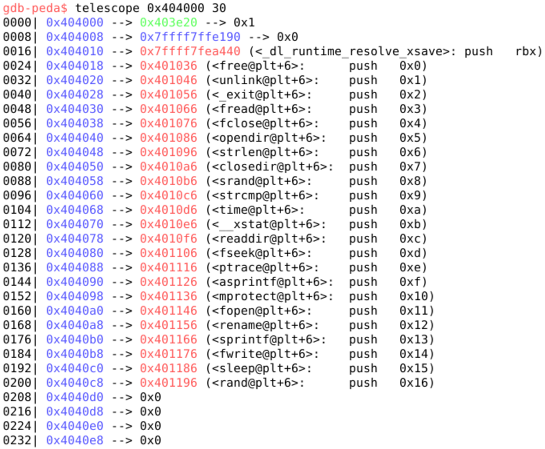
Global Offset Table entries (filled in at runtime)

Symbol resolution algorithm 1/3

- The GOT is initially almost empty (lazy loading)
- Only the entry at index 0 is filled in
- Index 0: generic resolver function in ld-linux

Symbol resolution algorithm 2/3

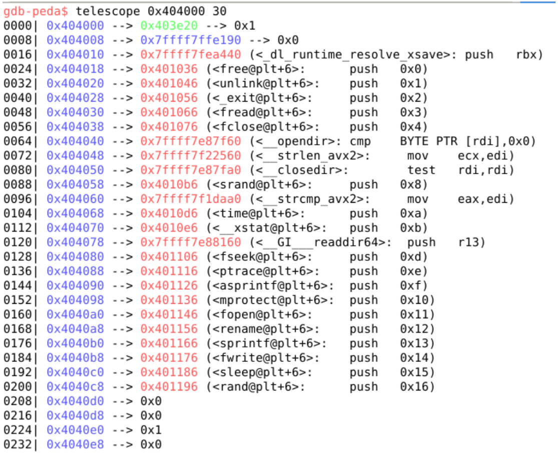
- All other entries are stubs that call the resolver



```
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
```

Symbol resolution algorithm 3/3

- Stub only called once (the first time)
- Resolver replaces the stub with a direct pointer



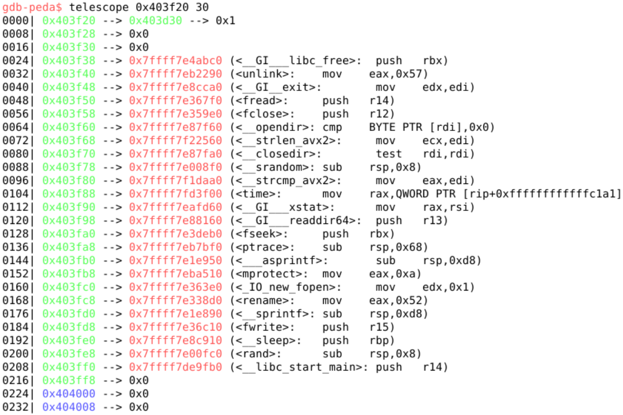
```
gdb-peda$ telescope 0x404000 30
0000| 0x404000 --> 0x403e20 --> 0x1
0008| 0x404008 --> 0x7ffff7ffe190 --> 0x0
0016| 0x404010 --> 0x7ffff7fea440 (<_dl_runtime_resolve_xsave>: push   rbx)
0024| 0x404018 --> 0x401036 (<free@plt+6>:      push   0x0)
0032| 0x404020 --> 0x401046 (<unlink@plt+6>:    push   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
```

But why do we care?

- The GOT is modifiable by the loader
- The GOT is also a potential target for overwrite
- Exploit a relative arbitrary write from a global buffer
- Exploit an absolute arbitrary write

RELRO mitigation mechanism

- Resolve everything at the start
- Set memory permissions to read only
- Read Only RELocations



```
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
```

RELRO Tradeoff

- The loader needs to do extra work at program startup
- But the loader needs to do less work afterwards
- And the program is more secure without much effort

RELRO Tradeoff

- The loader needs to do extra work at program startup
- But the loader needs to do less work afterwards
- And the program is more secure without much effort
- In practice, GOT tables are still overwritten (but in libraries)

Write What Where

- `https://cwe.mitre.org/data/definitions/123.html`
- Very Powerful! Overwrite anything
- Using the GOT table:
    - Overwrite free.got with system.got
    - Overwrite puts.got with printf.got
    - Overwrite stack-modifying functions with gets.got
- Endless possibilities

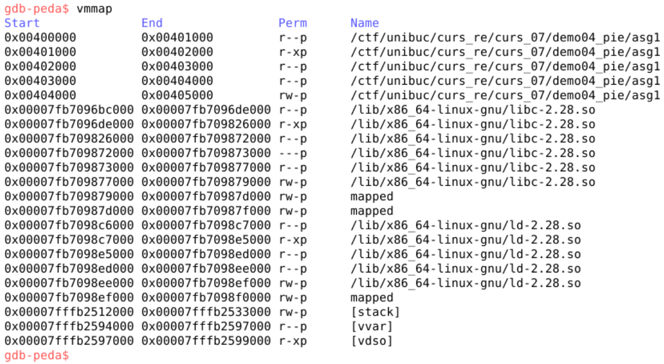
PIE mitigation

- The final ‘nail’ in the coffin
- The main executable is compiled as a library
- Position Independent Executable
- Kills off many vulnerability classes
- Cost: 20-25% performance penalty

PIE mitigation

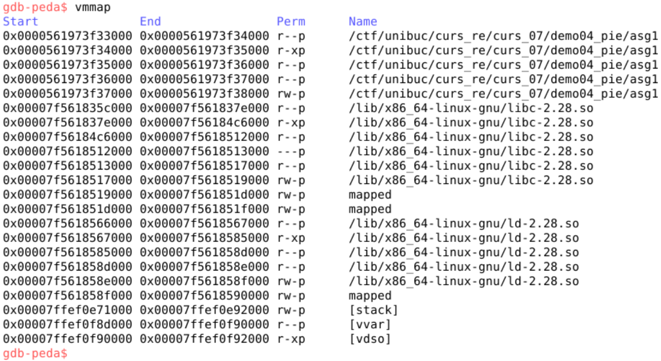
- The final ‘nail’ in the coffin
- The main executable is compiled as a library
- Position Independent Executable
- Kills off many vulnerability classes
- Cost: 20-25% performance penalty
- Believe it or not, it can be bypassed in many situations

Without PIE



```
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x00401000         r--p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x00401000         0x00402000         r-xp      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x00402000         0x00403000         r--p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x00403000         0x00404000         r--p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x00404000         0x00405000         rw-p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x00007fb7096bc000 0x00007fb7096de000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007fb7096de000 0x00007fb709826000 r-xp      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007fb709826000 0x00007fb709872000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007fb709872000 0x00007fb709873000 ---p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007fb709873000 0x00007fb709877000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007fb709877000 0x00007fb709879000 rw-p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007fb709879000 0x00007fb70987d000 rw-p      mapped
0x00007fb70987d000 0x00007fb70987f000 rw-p      mapped
0x00007fb7098c6000 0x00007fb7098c7000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007fb7098c7000 0x00007fb7098e5000 r-xp      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007fb7098e5000 0x00007fb7098ed000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007fb7098ed000 0x00007fb7098ee000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007fb7098ee000 0x00007fb7098ef000 rw-p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007fb7098ef000 0x00007fb7098f0000 rw-p      mapped
0x00007fffb2512000 0x00007fffb2533000 rw-p      [stack]
0x00007fffb2594000 0x00007fffb2597000 r--p      [vvar]
0x00007fffb2597000 0x00007fffb2599000 r-xp      [vdso]
gdb-peda$
```

With PIE



```
gdb-peda$ vmmap
Start              End                Perm      Name
0x0000561973f33000 0x0000561973f34000 r--p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x0000561973f34000 0x0000561973f35000 r-xp      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x0000561973f35000 0x0000561973f36000 r--p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x0000561973f36000 0x0000561973f37000 r--p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x0000561973f37000 0x0000561973f38000 rw-p      /ctf/unibuc/curs_re/curs_07/demo04_pie/asg1
0x00007f561835c000 0x00007f561837e000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f561837e000 0x00007f56184c6000 r-xp      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f56184c6000 0x00007f5618512000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f5618512000 0x00007f5618513000 ---p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f5618513000 0x00007f5618517000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f5618517000 0x00007f5618519000 rw-p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007f5618519000 0x00007f561851d000 rw-p      mapped
0x00007f561851d000 0x00007f561851f000 rw-p      mapped
0x00007f5618566000 0x00007f5618567000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007f5618567000 0x00007f5618585000 r-xp      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007f5618585000 0x00007f561858d000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007f561858d000 0x00007f561858e000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007f561858e000 0x00007f561858f000 rw-p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007f561858f000 0x00007f5618590000 rw-p      mapped
0x00007ffef0e71000 0x00007ffef0e92000 rw-p      [stack]
0x00007ffef0f8d000 0x00007ffef0f90000 r--p      [vvar]
0x00007ffef0f90000 0x00007ffef0f92000 r-xp      [vdso]
gdb-peda$
```

Compiler defaults

- PIE is on by default (Linux, Windows)

Compiler defaults

- PIE is on by default (Linux, Windows)
- RELRO is off by default on Linux
- RELRO is on by default on Windows

Compiler defaults

- PIE is on by default (Linux, Windows)
- RELRO is off by default on Linux
- RELRO is on by default on Windows
- SSP is off by default on Linux
- SSP (as GS) is on by default on Windows

Final remarks

- Some ASLR bypass is still needed (and usually found)
- All libraries/dependencies become the new attack surface
- Only works if you have the EXACT binaries at hand
- GOT tables are just a particular case of function pointers
- There are no libraries without read/write function pointers

Practice

- Any Questions?
- `http://pwnthybytes.ro/unibuc_re/07-lab.html`