Week 10 – More ROP

Introduction to Offensive Security

But First…

- Midterm grades are on NYUClasses
  - Let us know if you see any discrepencies
  - (especially people who have gotten extensions)
- There's a new "CTF Writeup" challenge
  - Let us know if you've submitted a writeup and haven't gotten credit
- We'll be ending at 7:50 today
- No office hours Friday due to CSAW

Homework review

syscall

- Some confusion about what syscall is, what args it takes, etc.
- `syscall` is an actual x86-64 instruction that takes us in to kernel mode
  - Read, write, fork, execve, etc.
- It is not a function like libc's `system`!
- What the syscall does is determined by the value in rax
  - 0 = sys_read
  - 1 = sys_write
  - 2 = sys_open
  - …
  - 59 = sys_execve
  - …
- Arguments are then passed normally (arg1 in rdi, arg2 in rsi, …)

64-bit syscall reference

http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

%rax	System call	%rdi	%rsi	%rdx	%r10	%r8	%r9
0	sys_read	unsigned int fd	char *buf	size_t count			
1	sys_write	unsigned int fd	const char *buf	size_t count			
2	sys_open	const char *filename	int flags	int mode			
3	sys_close	unsigned int fd					

syscall

- Those syscall names may be familiar though…
  - libc has a read(…), write(…), execve(…)
- One of the things libc has is C definitions for all common syscalls
  - … and eventually pass though to the `syscall` instruction
- That said, libc does a lot more than just syscall wrapping
  - `system(const char *command)`
  - There is no `system` syscall!
  - `system` is a helper around `execve`

Homework Walkthrough

Getting Creative with ROP

ROP Creativity

- "You can't always get what you want"
- Most challenge binaries won't have easy gadgets to do everything

- Example: need to do a 3 argument syscall
- Have:
  - `pop rdi; ret`
  - `pop rsi; ret`
  - `syscall; ret`
- How can we get stuff into rdx?

ROP Creativity

- Setting rdx (in order of simplicity)
  - Is there a `mov rdx, ____` and a way to set ___
    - E.g. `mov rdx, rsi` since we already have a `pop rsi`
  - What is rdx at the point the chain starts?
    - Can we influence it?
  - Can we do arithmetic in the ROP?
    - Last resort, but possible for "nice" numbers or numbers near rdx

Taking a step back…

- Common issue: writing a non-standard constant into memory and getting an address to it
  - E.g. don't know libc base, need /bin/sh string
- If we have a stack leak, we can use that
- But what if we don't
- Example: inspector but without the "useful_string"

Reading into .bss

- In all binaries, there's a "bss" segment where globals live
  - It's RW
  - And at a known address!
- Let's go through inspector, but without useful_string