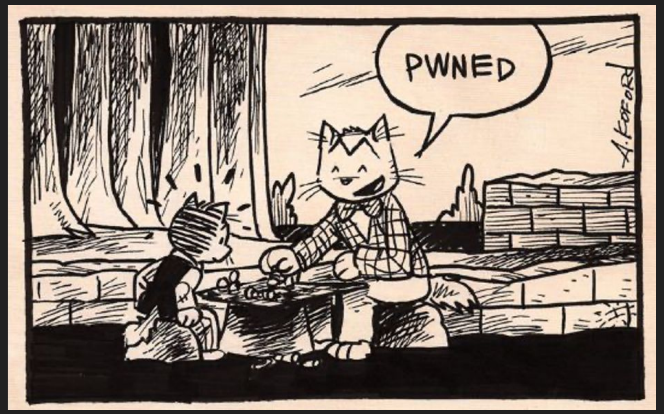
PWN 基础实践



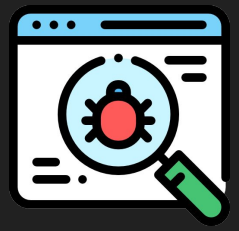
Outline

- OS and Binary Basics
- What is PWN Challenge
- Code Injection Bug (Part I)

Talk is cheap and boring,
let's learn this from exploiting

So what is PWN

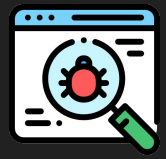
Find the Bug and Exploit it



OK, you already found some bugs in lab-0

- link

But what is bug actually ? really boring one



- (Generally) A Software Bug is **a failure or flaw** in a program that produces **undesired or incorrect results**. It's an error that prevents the application from functioning as it should.

太抽象了吧

Digression: it's just so hard to define "BUG"

beliefs are facts about the system implied by the code, can flag **belief contradictions** as errors

- Bugs as deviant behavior: A general approach to inferring errors in systems code [SIGOPS2021]

Fortunately

Most ***CTF pwn*** bugs are

- well defined → memory corruption
- easy to find (are you sure? → patternized
- obvious effects → control flow hijacking

So, let’s see a hello-world pwn challenge

- **3 minutes**, please **REVERSE** the given binary (hello)
- and 3 minutes **READ** the source code (hello.c)

Note 1

1. C Programming in Linux Platform
    - unfamiliar headers
    - what is Makefile

2. Challenge structure
    - with libc and loader with libc only
        i. do not expect source code
        ii. some time with bug introducing diff
    - local / remote
    - good challenge should issue everything you needed to run and test it
        i. is this true in realworld exploit?

Knowledg 1

- debug symbols (dwarf)
- **environment** variables
  - `PATH`
- **dynamically linked** program
  - `__libc_start_main`
- program memory layout
  - this is **soooooo** important
- `prepare()` function, see you next time

Note 2

please use **GDB Plugins**

- I prefer gef, or you can choose pwndbg, peda
  - up your choice

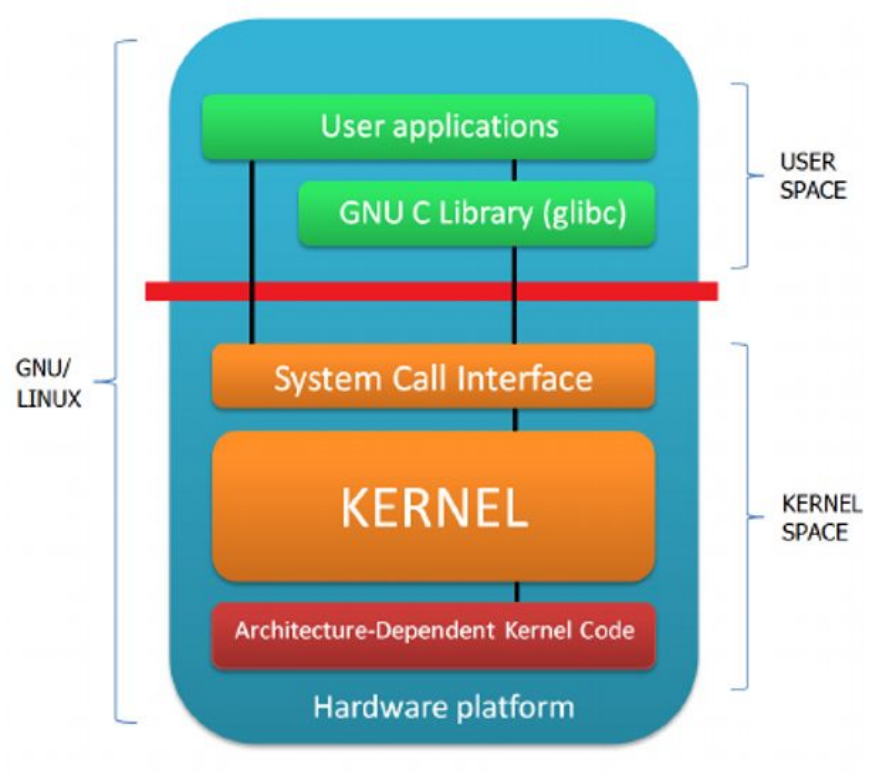
- installation is simple

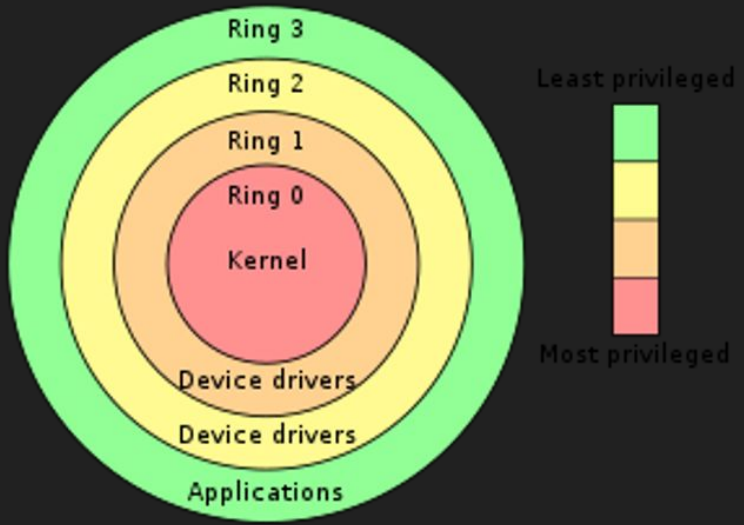
- **IDA-based debugging** is even more powerful

Knowledg 2

- `open / read / write`
  - syscalls (next page

- `system("/bin/sh")`
  - shell launcher backdoor
  - `execve()`
  - process


User applications
GNU C Library (glibc)
USER SPACE
GNU/ LINUX
System Call Interface
KERNEL
KERNEL SPACE
Architecture-Dependent Kernel Code
Hardware platform


Ring 3
Ring 2
Ring 1
Ring 0
Kernel
Device drivers
Device drivers
Applications
Least privileged
Most privileged

Okay, let’s hack this pwn challenge

- Anyhow, let’s get the `FLAG1` first
  - TEA cryptography
- And what about the other part?
  - What’s the bug/vulnerablity here?
- How we talk to the program
  - directly run / gdb (run, attach) / nc / **pwntools wrapper**

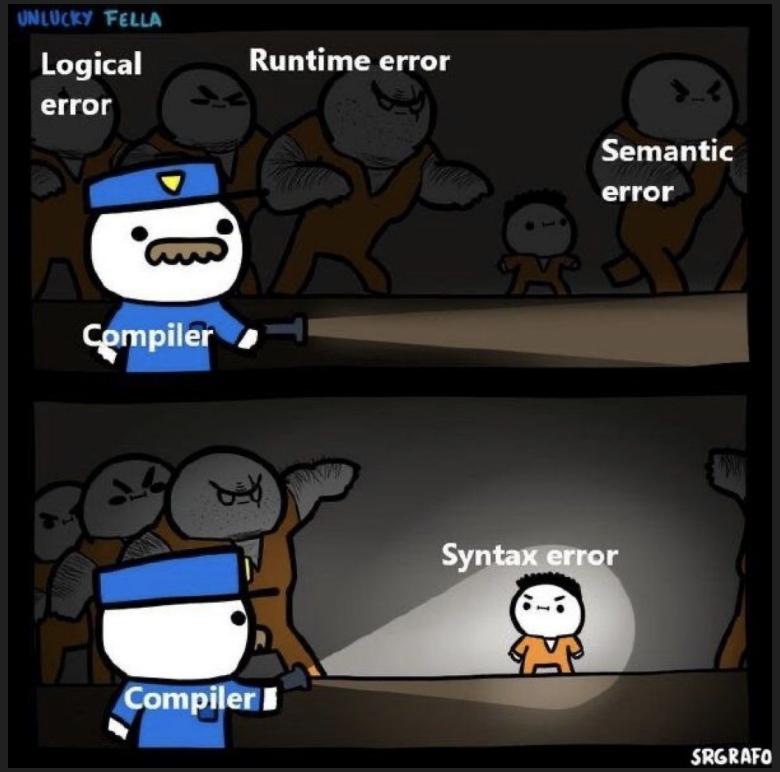
```
sudo apt install -y gdbserver
sudo pip3 install pwntools
```

Knowledg 3

- malicious end
  - dataflow Hijacking
  - controlflow Hijacking

- what is **primitive**

- **bug finding** requires
  - domain knowledge
  - sensitivity



Some Quick **Review**

- what does PWN challenge looks like
  - local / remote
- binary basics
  - dynamic linked / environment variables / memory layout
- OS basics
  - syscalls / process (task)

Let’s have a break

Code Injection Bug

or **代码注入攻击**

- **primitive:** (arbitrary) code execution
- classical, powerful, always easy to find
- easy to defense (really?

Let’s see this example: injection1

- **5 minutes**, please **REVERSE** the given binary
- no source code this time :(

Note 3

- proof-of-work
  - helpers

- **statically linked** binary
  - pros v.s. cons
  - 静态去符号

Okay, let’s hack this injection1 challenge

Wooow, so easy, we can learn that

- `system()` is just so strong
  - what does it do in syscall perspective

- how to defend such code injection?

- Web challenges?
  - path traversal

Let’s see this example: injection2

- **3 minutes**, please **READ** the given source code

Knowledg 4

- `mmap()` and `munmap()` syscall
  - low-level heap primitive
  - protection and flag
  - memory map from file descriptor

Note 4

- pwntools assembly helpers
  - `asm`
  - `disasm`
  - don’t forget to set architecture

- cross-compile target architecture

Let’s settle down some of these delegates

- Please fix the remaining part and get the first flag
- Wait? there is another flag here?

Execute user given code is **dangerous** !!!

calculator? just inject shellcode

- see `shellcode_test.c`
  - let's print something here

where to get great shellcode ?

1. shellstorm
2. pwntools shellcraft
3. write by yourself :)

back to injection2

Some Quick **Review**

- Code Injection Bug is quite dangerous
- Proof-of-Work (PoW)
  - 挖矿
- Additional Knowleges
  - statically linked binary
  - `map` / `munmap`

Homework

see course site for details

FYI

对于有意向选择 pwn 专题的同学，请

- 复习一下 x86_64 汇编，确保可以读懂
- 预习一下程序的栈结构，推荐资料 1 2 or just Google it