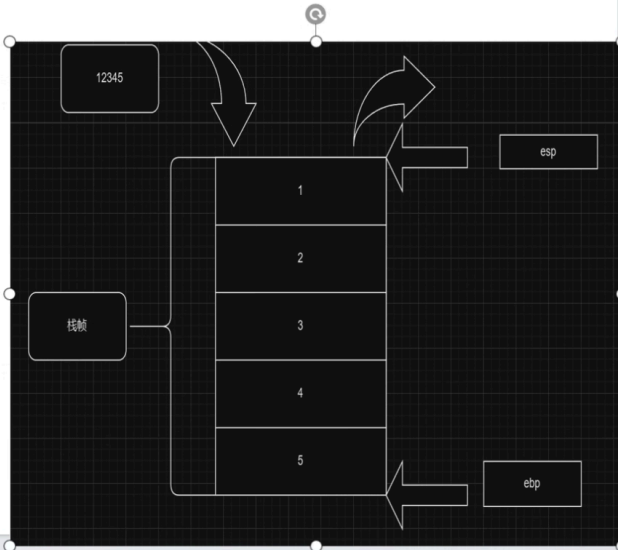
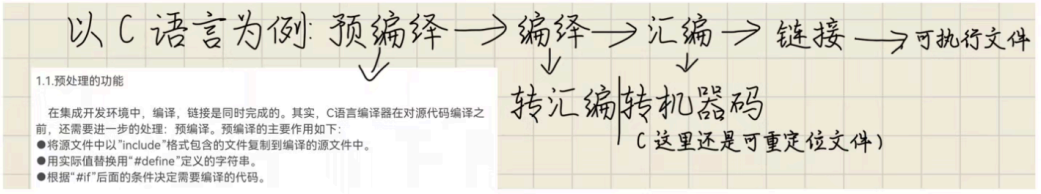
栈是一种数据结构

- 1 栈是一种先进后出的结构
- 2 栈在内存中由高地址向低地址生长
- 3 栈在内存中的入栈顺序是从右到左
- 4 栈一般会绑定两个指针，栈顶（esp）栈底（ebp），通常查询栈内的数据用到栈底



操作系统和计组

- 要学pwn，基础一定要扎实，想理解程序如何运行，这两门课的一定要学好。
- 了解一点汇编基础，可以去看看王爽的《汇编语言》
- 程序归根结底是运行在cpu上的，而计算机是只能识别机器码的，那么程序是如何从高级语言转为机器码的这一个过程也要理解。



了解pwndbg用法

- 首先这里简单来说就是要会调试这个程序，理解函数调用过程中栈的变化。这里的调试需要慢慢去练习，通过调试去理解程序如何运行的。
- 学会gdb调试，掌握gdb的基本用法
- pwntools安装好后，可以利用它提供给我们的工具去分析一个程序。就比如checksec，它能够帮助我们分析程序开启了哪些保护。当然还有其他的如ropgadget，one_gadget等等。
- ROP（Return-Oriented Programming，返回导向编程），就是我们控制程序的返回地址，让它反复横跳，从而执行类似system等getshell的操作。

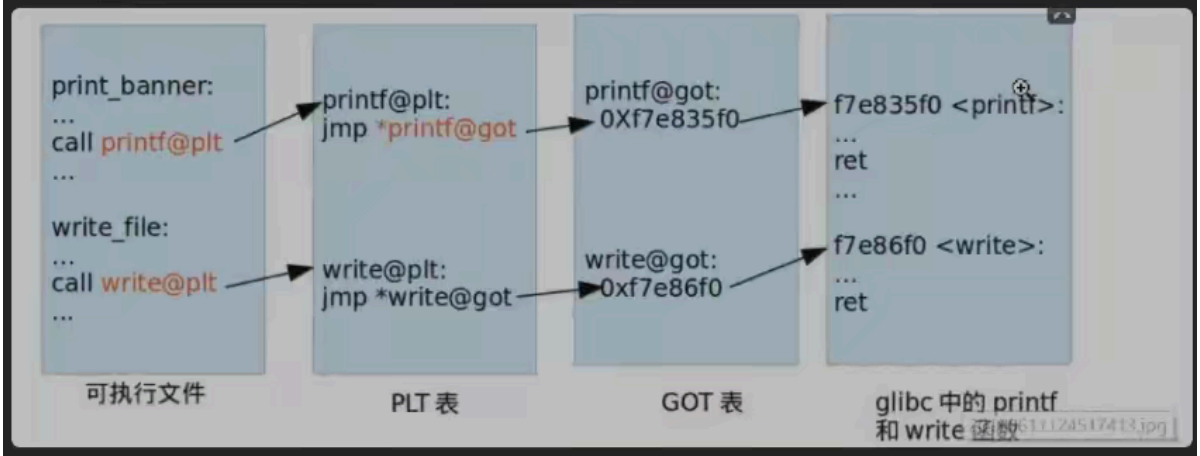
plt表和got表的联系

1 首先，plt表里面存放的是获取外部链接的那串代码，而这串代码的地址存放在got表里面，所以plt表里面会有 `jmp *fun@got` 这样的代码，这里表示的是跳转到存放fun函数地址的got表。

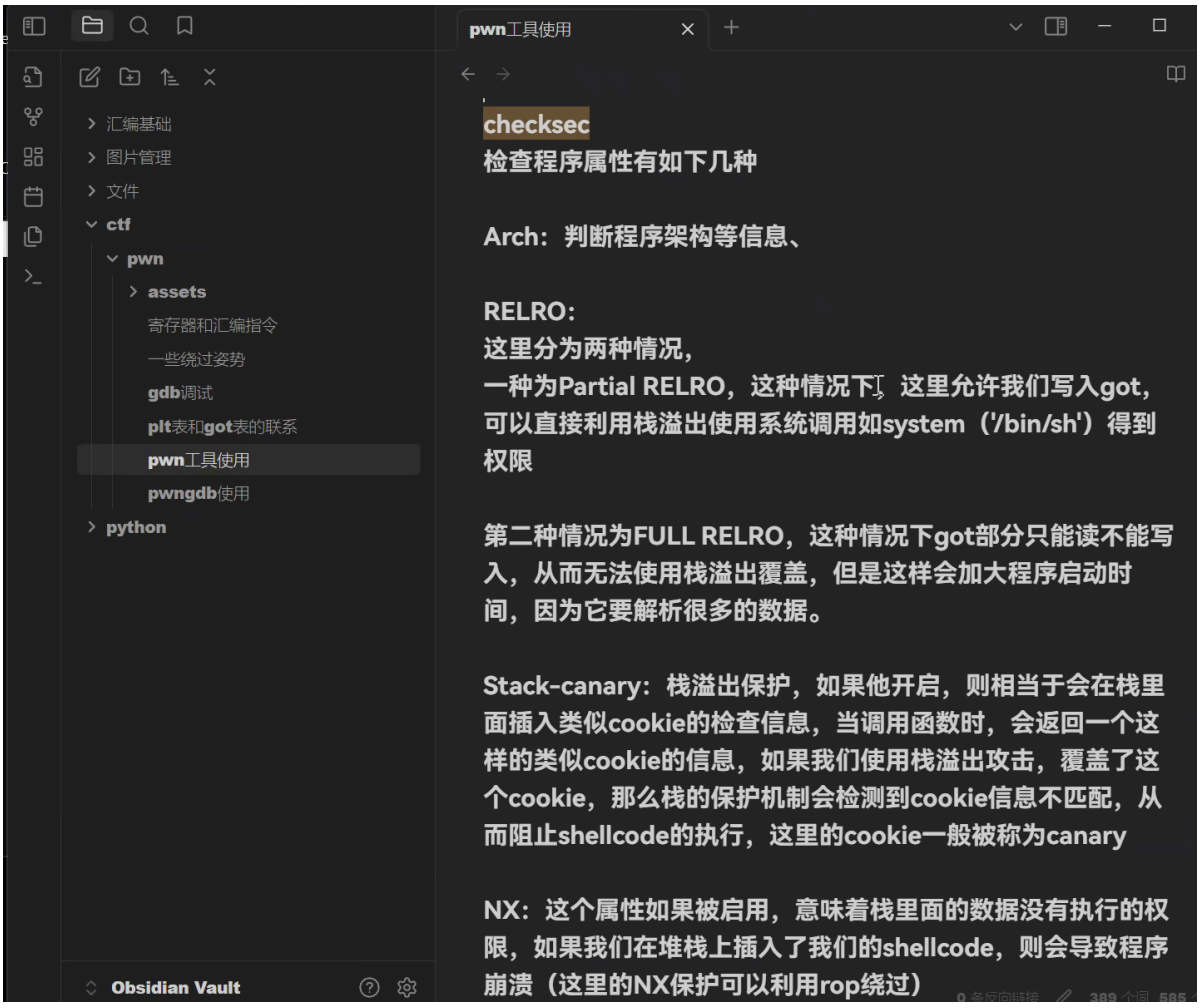
2 当到了got表时，就会从这个地方找到函数真正的位置，从而调用函数。

所以这样的过程一般是这样的：

`call fun@plt -> jmp *fun@got -> 函数的真正的地址`



可以结合这个图片理解。


pwn工具使用
汇编基础
图片管理
文件
ctf
pwn
assets
寄存器和汇编指令
一些绕过姿势
gdb调试
plt表和got表的联系
pwn工具使用
pwngdb使用
python
Obsidian Vault
checksec
检查程序属性有如下几种
Arch：判断程序架构等信息、
RELRO:
这里分为两种情况，
一种为Partial RELRO，这种情况下，这里允许我们写入got，可以直接利用栈溢出使用系统调用如system（'/bin/sh'）得到权限
第二种情况为FULL RELRO，这种情况下got部分只能读不能写入，从而无法使用栈溢出覆盖，但是这样会加大程序启动时间，因为它要解析很多的数据。
Stack-canary：栈溢出保护，如果他开启，则相当于会在栈里面插入类似cookie的检查信息，当调用函数时，会返回一个这样的类似cookie的信息，如果我们使用栈溢出攻击，覆盖了这个cookie，那么栈的保护机制会检测到cookie信息不匹配，从而阻止shellcode的执行，这里的cookie一般被称为canary
NX：这个属性如果被启用，意味着栈里面的数据没有执行的权限，如果我们在堆栈上插入了我们的shellcode，则会导致程序崩溃（这里的NX保护可以利用rop绕过）

第二种情况为FULL RELRO，这种情况下got部分只能读不能写入，从而无法使用栈溢出覆盖，但是这样会加大程序启动时间，因为它要解析很多的数据。

Stack-canary：栈溢出保护，如果他开启，则相当于会在栈里面插入类似cookie的检查信息，当调用函数时，会返回一个这样的类似cookie的信息，如果我们使用栈溢出攻击，覆盖了这个cookie，那么栈的保护机制会检测到cookie信息不匹配，从而阻止shellcode的执行，这里的cookie一般被称为canary

NX：这个属性如果被启用，意味着栈里面的数据没有执行的权限，如果我们在堆栈上插入了我们的shellcode，则会导致程序崩溃（这里的NX保护可以利用rop绕过）

PIE：这里如果启用，则代表程序运行时的堆栈地址将随机，我们无从得知，我们将对内存布局一无所知。（这里我没有了解到如何绕过）

ropgadget

就是在代码里搜索指令，字符串的一个小工具，通过这个我们可以构造exp，利用rop技术getshell。

覆盖返回地址

- 一般来说就是输入超过定义的数据，覆盖返回地址执行特定函数。

一般步骤：

1 检查文件属性

2 查看保护

3 找寻漏洞点（函数，字符串等）

4 构造exp得到flag