**框架：**

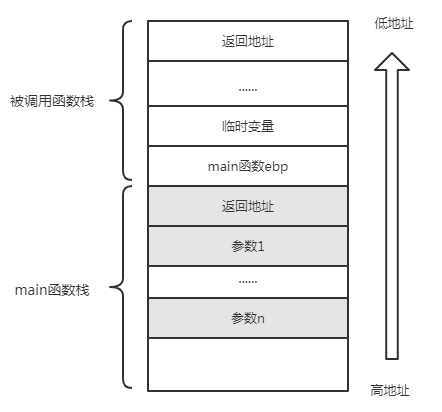
常见的程序(以32位为例，64位传参方式不一样)漏洞以及如何利用：

1. 栈溢出

原理不用多说。来说下怎么利用：

低地址

|  |
| --- |
| **…………** |
| **Var2** |
| **Var1** |
| **EBP** |
| **EIP** |
| **Arg1** |
| **Arg2** |

**Esp-->**

**Ebp-->**

高地址

1.1函数的栈分布

覆盖eip即可（用shellcode的首地址）

1. 常见的编译器的保护措施：
2. canary（栈保护）

栈溢出保护是一种缓冲区溢出攻击缓解手段，当函数存在缓冲区溢出攻击漏洞时，攻击者可以覆盖栈上的返回地址来让shellcode能够得到执行。当启用栈保护后，函数开始执行的时候会先往栈里插入cookie信息，当函数真正返回的时候会验证cookie信息是否合法，如果不合法就停止程序运行。攻击者在覆盖返回地址的时候往往也会将cookie信息给覆盖掉，导致栈保护检查失败而阻止shellcode的执行。在Linux中我们将cookie信息称为canary。

1. NX（DEP）

NX即No-eXecute（不可执行）的意思，NX（DEP）的基本原理是将数据所在内存页标识为不可执行，当程序溢出成功转入shellcode时，程序会尝试在数据页面上执行指令，此时CPU就会抛出异常，而不是去执行恶意指令。

1. PIE（ASLR）

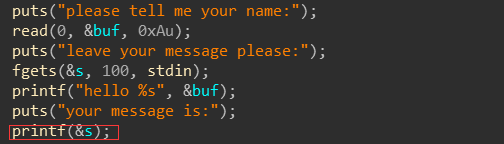
PIE(position-independent executable) 是一个针对代码段.text, 数据段.\*data，.bss等固定地址的一个防护技术。同ASLR一样，应用了PIE的程序会在每次加载时都变换加载基址

(4) RELRO（read only relocation）

设置符号重定向表格为只读或在程序启动时就解析并绑定所有动态符号，从而减少对GOT（Global Offset Table）攻击。RELRO为” Partial RELRO”，说明我们对GOT表具有写权限。

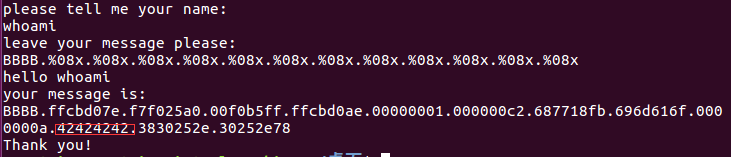
1. 格式化字符串漏洞

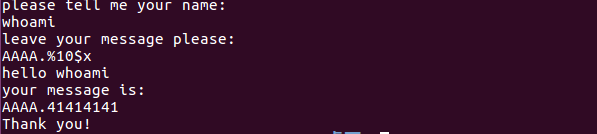
看如下写法：



这是一种非常危险的写法。由于printf函数族的设计缺陷，当其第一个参数可被控制时，攻击者将有机会对任意内存地址进行读写操作。

当我们输入printf可识别的格式化字符串时，printf会将其作为格式化字符串进行解析并输出,printf(“%x.%x.%x”)那么参数就默认为后面的地址

比如造成内存泄露：

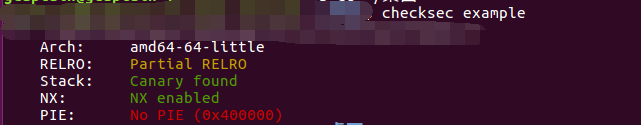


可以对内存进行更改（利用%n）:

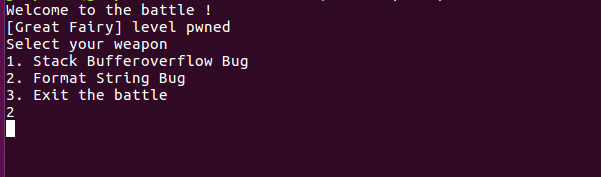
printf(“AAAA%n”,&a),那么a的值就是4 (%c$n : 对第c个参数进行更改)

1. 利用格式化字符串泄露canary的值从而绕过

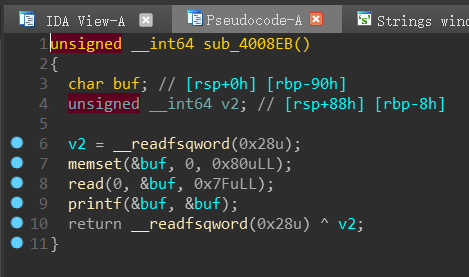
以一道题为例：

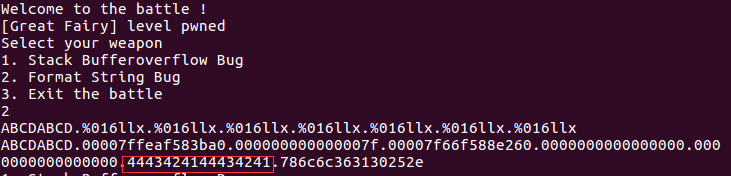


Canary found有栈保护

运行一哈：

选择2的代码：



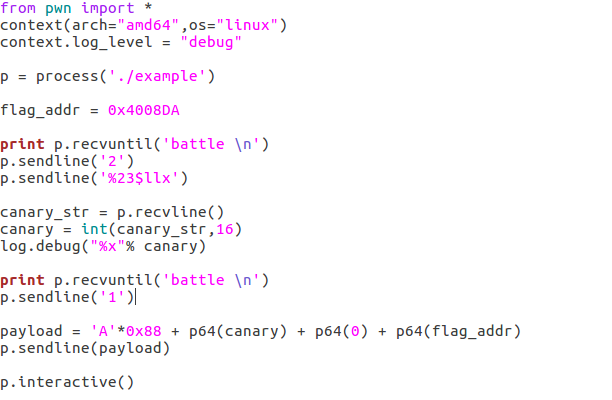


又 char buf; // [rsp+0h] [**rbp-90h**]

0x90 / 0x8 = 0x12 = 18

所以canary位于18 + 6 -1 = 23 (18+6可读取ebp,-1读canary)：

构造脚本：



跑一下脚本

泄露成功：

