corporate_slave

发表于 2021-03-05 更新于 2021-03-15 分类于 Challenge， 2020， SWPU CTF 2020， Pwn
Challenge | 2020 | SWPU CTF 2020 | Pwn | corporate_slave

点击此处获得更好的阅读体验

WriteUp来源

官方WP

解题思路

本题为glibc2.27保护全开

题目逻辑比较简单，只有一个add功能，并且漏洞点很明显off by X(null)，通过readSize的索引来往地址中写一个0字节

3）难点在于如何利用，我们知道当我们malloc一个大的堆块（0x200000）时就会使用mmap来分配堆块，此时堆地址紧挨libc,因此我们可以利用这一点来往libc范围内写0字节，那该往哪里写呢

4）我们首先需要泄露libc，通过源码我们可以知道要想泄露数据需要完成的流程是：

puts->_IO_file_xsputn->_IO_file_overflow->_IO_new_do_write->_IO_SYSWRITE (fp, data, to_do)

通过条件判定我们需要满足以下条件

fp->_flags &~ 0x1000 && fp->_IO_read_end = fp->_IO_write_base

由于0xfbad2887 & 0x1000已经满足了条件

因此只需要让 fp->_IO_read_end = fp->_IO_write_base

即可进入_IO_SYSWRITE (fp, fp->_IO_write_base,fp->_IO_write_base-fp->_IO_write_ptr),从而泄露libc

那泄露libc之后呢？

我们第三次写0字节考虑往stdin里面写，将fp->_IO_buf_base的最后一字节写为0,此时io_buf_base恰好指向io_file的头部，因此我们可以伪造整个io_file，那如何getshell呢？

我们采用二重写入的方法来将另一部分的io_file写入到stdout，从而当puts的时候通过io_str_overflow来获取shell

那如何二重写入，我们研究一下stdin的源码可以知道，程序在写入之后会回调到_IO_getline

而_IO_getline里面判断了fp->_IO_read_end - fp->_IO_read_ptr是否小于0

若小于0则调用__uflow,因此我们需要让fp->_IO_read_end - fp->_IO_read_ptr小于0

在__uflow里面又调用了_IO_default_uflow而此函数调用了_IO_file_underflow这里就达成了我们二次写入的条件

我们知道第一次只能读0x84的数据因此剩下我们发送的就会通过_IO_SYSREAD (fp, fp->_IO_buf_base, fp->_IO_buf_end - fp->_IO_buf_base);向fp->_IO_buf_base写入

此时fp->_IO_buf_base已经被我们修改为了stdout的头部，因此可以实现二次写入，我们布置好stdout结构体的值使其走向io_str_overflow通过以下源码getshell

最后获取flag

- ***本文作者：*** *CTFHub*
- ***本文链接：*** *https://writeup.ctfhub.com/Challenge/2020/SWPU-CTF-2020/Pwn/khFW2uRBdzLB1R2z747s9Q.html*
- ***版权声明：*** *本博客所有文章除特别声明外，均采用 BY-NC-SA 许可协议。转载请注明出处！*

 # Challenge # 2020 # Pwn # SWPU CTF 2020

tnote
jailbreak