ar_u_ok

发表于 2021-01-21 分类于 [Challenge](#) , [2017](#) , [HCTF](#) , [Bin](#)
Challenge | 2017 | HCTF | Bin | ar_u_ok

[点击此处](#)获得更好的阅读体验

WriteUp 来源

<https://xz.aliyun.com/t/1589>

题目考点

- 对ptrace的认识
- rc6识别
- rc4识别

解题思路

加密解密

真正的加密和解密过程很简单，就是一个标准的rc6，只要把函数中的那个int常量放到google里搜索一下就知道是rc6加密（这个函数的代码被rc4加密了，不解密是看不到的）。

rc6加密和解密的代码见源码

程序流程

程序首先判断启动参数，如果argc为1，则以debugger身份启动，利用fork分出parent和child。parent作为真正的debugger，child利用execve来启动自身并以父进程的pid作为启动参数。

如果argc为2，说明是debuggee。程序利用puts打印plz_input_flag，但是write的syscall被ptrace hook了。puts的原始内容是乱码，需要debugger对其进行解密。

然后是利用scanf来接收flag。默认是允许输入%48s但是这里ptrace hook了read syscall，检测read syscall触发的次数（在程序开头利用setbuf将stdin和stdout的缓冲调整为0），从而使flag的真实最大长度为32。

接着是一段判断是否调试者为父进程的代码，没问题的话会调用fclose来关闭之前打开的文件。此处用ptrace hook了close syscall。但是在程序运行前也会调用close syscall。这里利用设置变量的方式，使得在第二次close的时候触发。

触发时执行的代码是利用rc4将两个函数解密，然后patch代码为0xcc使程序停在检测trace代码的下一行，在将其patch成jmp到data段的那段唯一可视的雷军ascii字符处，并将flag传递给rdx，接着继续执行。雷军那段ascii其实是代码。前面的52Mi!是xor eax, 0x21694d32，从而使后面的jne全部成立，R_是push rdx;pop rdi，从而将之前在rdx中的flag传递到rdi中。利用u_这个jne跳转跳过中间的非代码区，最后jmp到encrypt函数中。

encrypt函数就是调用rc6加密，将32位的flag分16位两次加密，最后和enc结果比较。

由于调用了很多的ptrace来实现smc和hook，纯动态分析应该不太可能实现，需要静态分析后patch程序才能使用动态分析。

完整程序见github，由于有smc部分，可能在不同机子上编译结果不正确，所以提供了一个测试用的binary。

Flag

1 无

- 本文作者：CTFHub
- 本文链接：<https://writeup.ctfhub.com/Challenge/2017/HCTF/Bin/32niRibc9DiPTLbWEtFSuw.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](#) [#2017](#) [#HCTF](#) [#Bin](#)
[babyprintf](#)
[babystack](#)