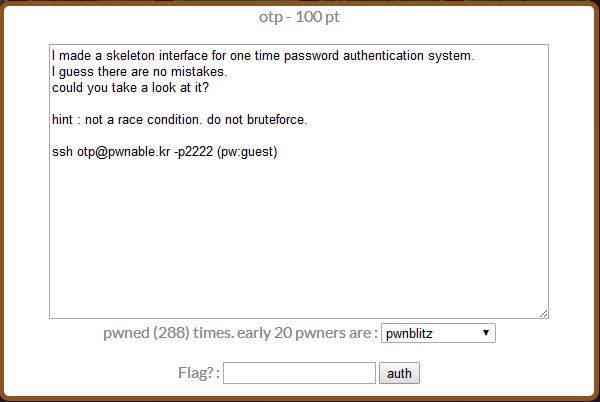
解题思路

1. 题目要求



1. 题目解读

有关OTP的题目。下面复习下OTP的概念：

**动态密码**（英语：One Time Password，简称OTP），又称一次性密码或单次有效密码，是指计算机系统或其他数位装置上只能使用一次的密码，有效期为只有一次登录会话或交易。

对这道题目来说，就是单次登录密码，每次登陆的密码都不一样。猜测随机密码是否与时间有关。

1. 解题过程

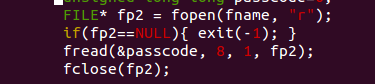
ssh登录后可以看到OTP的源码,仔细看下otp.c源文件。



分析下流程：

1. 从/dev/urandom中读取随机数（猜测与时间有关的随机数错误）。
2. 将随机数的前8个字节作为文件名，用于创建临时文件。
3. 将随机数的后8个字节则写入创建的临时文件当中。
4. 从临时文件中读取数据，然后与输入的参数进行比较。一致的话就说明认证成功。

大概看了一下代码，没有明显的问题，也不存在缓冲区溢出的情况。一句一句对应看，发现在代码32行处可能存在问题：



调用fread函数时，没有判断fread是否成功。也就是说不管fread是否执行成功，都会来校验passcode是否与输入参数一致。因此可以设法让fread函数执行失败，则passcode就为0，只要设置参数为0，即可绕过判断。

**补充说明：**

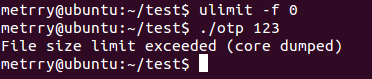
**ulimit命令**

ulimit 是一种 linux 系统的内键功能，它具有一套参数集，用于为由它生成的 shell 进程及其子进程的资源使用设置限制。

|  |  |  |
| --- | --- | --- |
| 选项 | 含义 | 例子 |
| -H | 设置硬资源限制，一旦设置不能增加。 | ulimit -Hs 64；限制硬资源，线程栈大小为 64K。 |
| -S | 设置软资源限制，设置后可以增加，但是不能超过硬资源设置。 | ulimit -Sn 32；限制软资源，32个文件描述符。 |
| -a | 显示当前所有的 limit 信息。 | ulimit -a；显示当前所有的 limit 信息。 |
| -c | 最大的core 文件的大小，以 blocks 为单位。 | ulimit -c unlimited不限制core文件的大小。 |
| -d | 进程最大的数据段的大小，以 Kbytes 为单位。 | ulimit -d unlimited；对进程的数据段大小不进行限制。 |
| -f | 进程可以创建文件的最大值，以 blocks 为单位。 | ulimit -f 2048；限制进程创建最大文件为2048 blocks。 |
| -l | 最大可加锁内存大小，以 Kbytes 为单位。 | ulimit -l 32；限制最大可加锁内存大小为 32 Kbytes。 |
| -m | 最大内存大小，以 Kbytes 为单位。 | ulimit -m unlimited；对最大内存不进行限制。 |
| -n | 可以打开最大文件描述符的数量。 | ulimit -n 128；限制最大可以使用128 个文件描述符。 |
| -p | 管道缓冲区的大小，以 Kbytes 为单位。 | ulimit -p 512；限制管道缓冲区的大小为 512 Kbytes。 |
| -s | 线程栈大小，以 Kbytes 为单位。 | ulimit -s 512；限制线程栈的大小为 512 Kbytes。 |
| -t | 最大的 CPU 占用时间，以秒为单位。 | ulimit -t unlimited;最大的CPU占用时间不进行限制 |
| -u | 用户最大可用的进程数。 | ulimit -u 64；限制用户最多可以使用 64 个进程。 |
| -v | 进程最大可用的虚拟内存，以 Kbytes 为单位。 | ulimit -v 200000；限制最大可用的虚拟内存为 200000Kbytes. |

在这个例子中我们可以用-f参数，来设置进程可以创建文件的最大值为0，即：

ulimit –f 0



如上图所示，执行otp程序时又出现了一个错误：**File size limit exceeded (core dumped)**。

也就是说由于设置了可以创建文件的最大值为0，所以在创建临时文件后，往里面写入数据时就会产生SIGXFSZ信号。

当某个进程写文件，单个文件大小超出限制，这时操作系统会向它发送SIGXFSZ。而SIGXFSZ信号的默认动作是：终止进程+生成coredump文件。

于是新的问题就出现了，在调用fwrite函数时由于超出限制，会终止进程，并生成coredump文件(生成coredump文件默认是在当前目录，另外要设置ulimit –c xxxx才可以)。

怎么让操作系统发送SIGXFSZ信号时，不终止进程呢？可以使用python的subprocess和signal模块。编写python脚本如下所示：



signal包的核心是使用signal.signal()函数来预设(register)信号处理函数，如下所示：

***singnal.signal(signalnum, handler)***

signalnum为某个信号，handler为该信号的处理函数。进程可以无视信号，可以采取默认操作，还可以自定义操作。当handler为signal.SIG\_IGN时，信号被无视(ignore)。当handler为singal.SIG\_DFL，进程采取默认操作(default)。当handler为一个函数名时，进程采取函数中定义的操作。

也就是说这里handler参数设置为signal.SIG\_IGN，信号就会被无视。

最终答案为：

***Darn... I always forget to check the return value of fclose() :(***