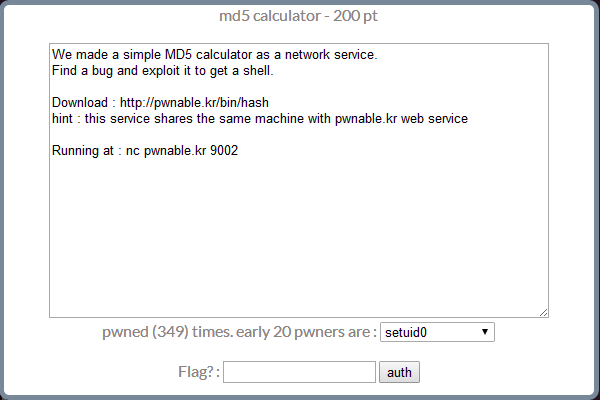
解题思路

# 题目要求



# 题目解读

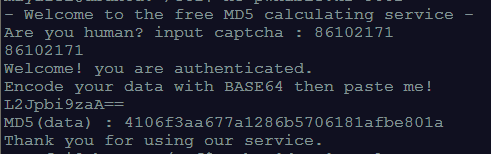
找到一个bug,并利用它来获取shell,并给出了对应的可执行文件hash的下载地址；同时还给了一个提示：MD5的计算服务与pwnable.kr的web服务共享同一台机器。

看完提示，不是很理解这句提示的作用。

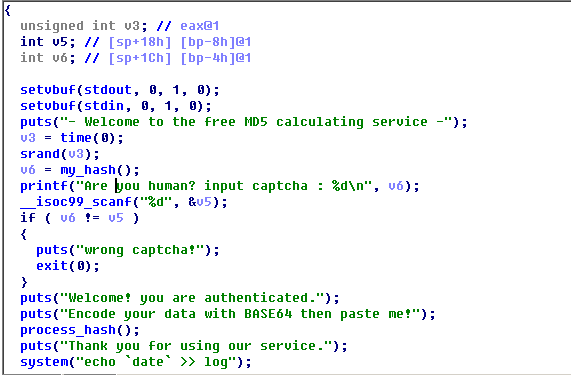
把hash下载下来，执行一遍看看效果：



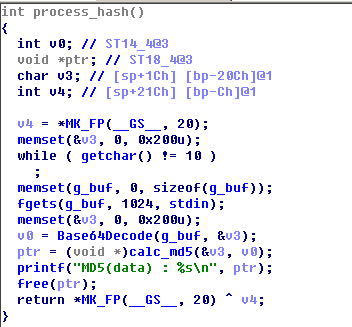
报错了，提示没有libcrypto.so动态库，看来我机器上跑不了这个程序,直接nc连上看看效果，输出如下：



有需要输入的地方,猜测是溢出漏洞利用,IDA载入文件进行分析,F5查看伪代码如下:



大概看了下比较验证码这块应该没有可以利用的溢出漏洞,于是直接把重点放在了process\_hash函数中，如下所示：

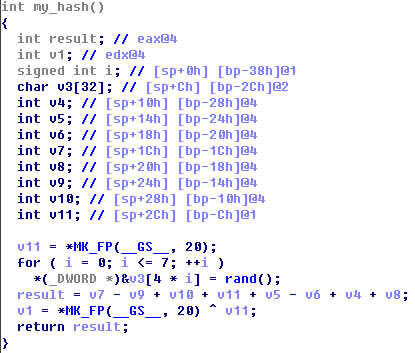


从stdin标准输入中读取1024个字节，然后解base64，解码的值放在v3中。这里看到v3只有512个字节大小，而1024字节解base64之后应该有1024\*3/4大小，它要比512大，确定这里存在缓冲区溢出漏洞。

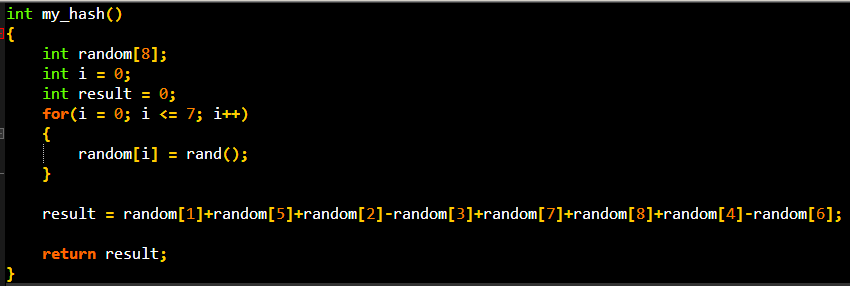
然而实验半天仍没有解决方法，遂上网搜了一下别人的解决方法，发现自己的问题出在哪里，原来这里有栈保护机制来保护栈溢出利用。鉴于对栈保护机制的不熟悉，现补上一份栈保护机制的学习文档。参考的解决方案为：<https://gist.github.com/ihciah/948b37ab7587c3f70268>

canary的值放在gs:14h处，栈保护代码由gcc编译器在编译的时候添加，这里应该是一个随机数，通过检查栈中的canary值是否与gs:14h处的值一致与否来判断有没有栈溢出问题。需要注意的是整个canary的值是在程序运行后赋予的，并且在程序运行过程中对所有使用栈保护机制的函数来说都是一样的，不会发生变化。当然重新运行程序则与上一次程序运行时canary的值是不一样的。

分析一下my\_hash函数，伪代码如下所示：



其中v11就是canary的值。查看汇编代码，对具体实现进行了解。得出对应的C代码应为：



发现其中result的计算有点问题，random的大小只有8，因此理论上不存在random[8]这一元素，应该是[0到7]，而实际在计算result值的时候用了这个元素，从canary的原理上了解到，random紧挨着canary，也就是说这里的random[8]其实就是canary的值，result则是作为验证码输出在屏幕上的，这样就可以得到canary值的计算公式为：



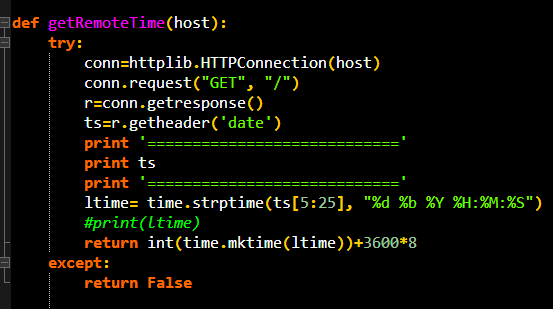
其中random是通过rand()随机函数来生成的，种子则是当前时间，我们知道这两个函数其实生成的是伪随机数，在随机种子一样的情况下，生成的随机序列是一致的。因此只要时间能与服务器的时间保持一致，就完全可以构造random的值，这也验证了题目中的提示信息：MD5的计算服务与pwnable.kr的web服务共享同一台机器。

# 解题过程

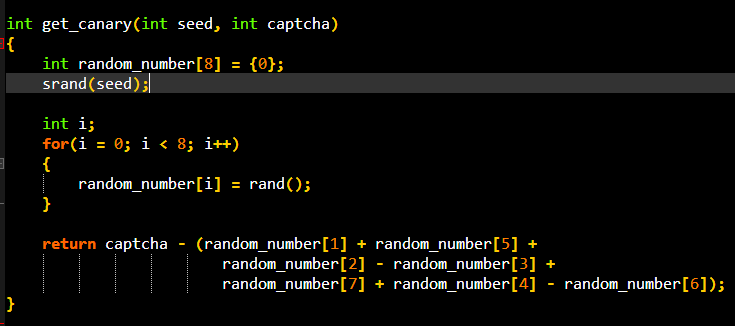
通过步骤2的解读,我们知道生成随机数的种子是当前服务器时间，而如何获取服务器时间呢？这里使用http头的date域来获取。

Date头域表示消息发送的时间，时间的描述格式由rfc822定义，它描述的时间是世界标准时，若要换算成本地时间，需要知道用户所在的时区。

编写python脚本来获取pwnable.kr服务器的时间。需要转换成本地时间，北京时区，因此需要加上3600\*8秒来表示，并转换为时间戳。



然后按照步骤2分析道的canary计算方法进行实现，通过C的方式进行实现，并在python中进行调用，如下所示：



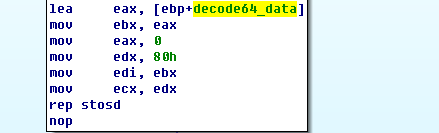
这里seed就是上一步获取到的时间戳，而captcha则是服务返回的验证码，这样就可以计算出canary的值是多少。

接下来就是构造shellcode了，在上面的分析中我们知道在解base64编码的时候存在溢出漏洞，而canary的值我们也能够获取到，这样就能够绕过栈溢出的保护机制，从而进行漏洞利用。

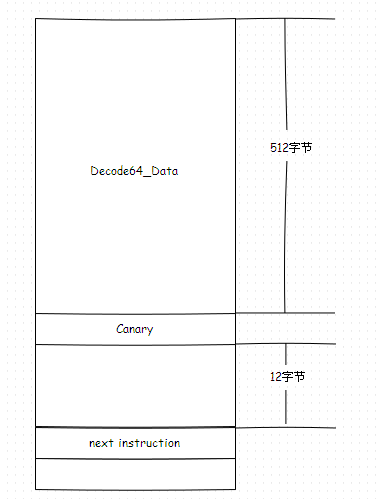
首先知道g\_buf是的位置、大小如下：



g\_buf是用来存放我们输入的经过base64编码过后的字符串，而解码后的数据是存放在栈中的。1024字节的base64编码数据解码后的大小应为：1024\*3/4=768字节，而代码中开辟的解码数据内存的大小为：512字节。



下面来看一下栈的内存分布：

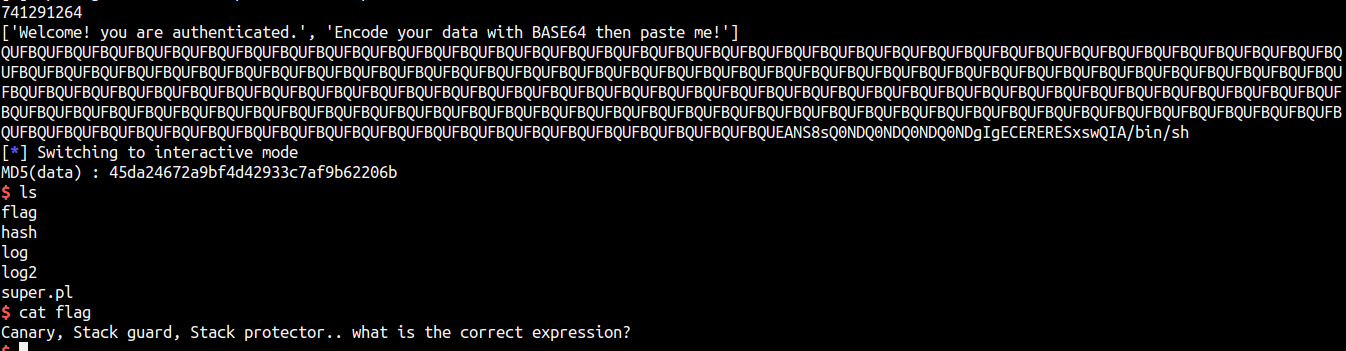


执行完base64 decode之后，如果解码后的数据长度大于512字节，则canary就会被覆盖，这样就会触发栈保护机制，因此我们要做的就是保证解base64之后canary不会改变，这就用到了前面计算到的canary的值。同时我们要想拿到一个shell，可以利用程序中原有的函数system，用它来替换next instruction就可以执行system调用了。这里还会有一个system参数的问题，如何执行system(“/bin/sh”)，可以利用g\_buf来做，将”/bin/sh”字符串放到g\_buf中，而next\_instruction栈区中设置”/bin/sh”字符串的指针即可。最终构造的输入字符串为：



需要注意的是p32(g\_buf+0x2d1)中的0x2d1的由来，它表示/bin/sh字符串在g\_buf中的偏移，因此长度为：（512+4+12+4+8）\*4/3+1=721，即0x2d1。

# 解题结果



Canary, Stack guard, Stack protector.. what is the correct expression?