CTF-RE-webassembly （2019强网杯）

原创

SuperGate 于 2019-06-06 12:37:55 发布 1283 收藏 2
分类专栏： CTF-RE

本文链接：https://blog.csdn.net/getsum/article/details/90598131
版权

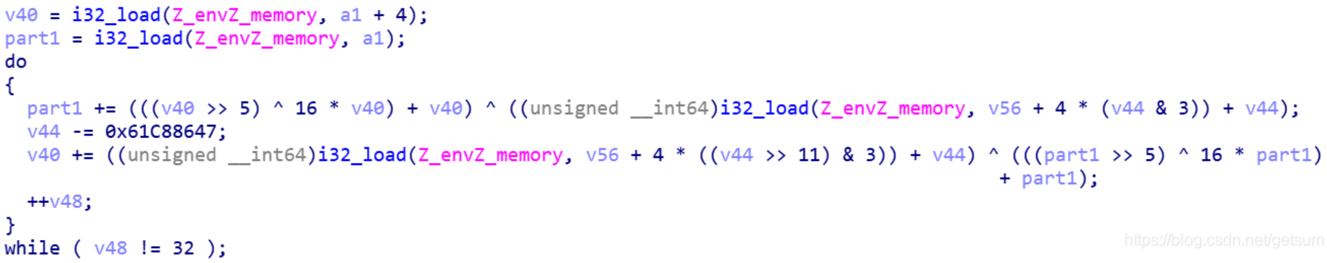
CTF-RE 专栏收录该内容
19 篇文章 0 订阅
订阅专栏

打开之后得到三个同名文件，分别是.js .wasm .html

.html用浏览器打开，发现让我们输入flag。由于.js在我电脑里好像打不开，没配置相关环境，所以选择通过.wasm进行逆向。

.wasm逆向的一般套路，用wasm2c将wasm转化成c伪代码，然后再用gcc -c命令编译但是不链接得到.o文件。注意编译时缺少什么文件就在wasm2c文件夹中复制过去即可。

.o文件用ida打开，然后f15函数得到加密函数

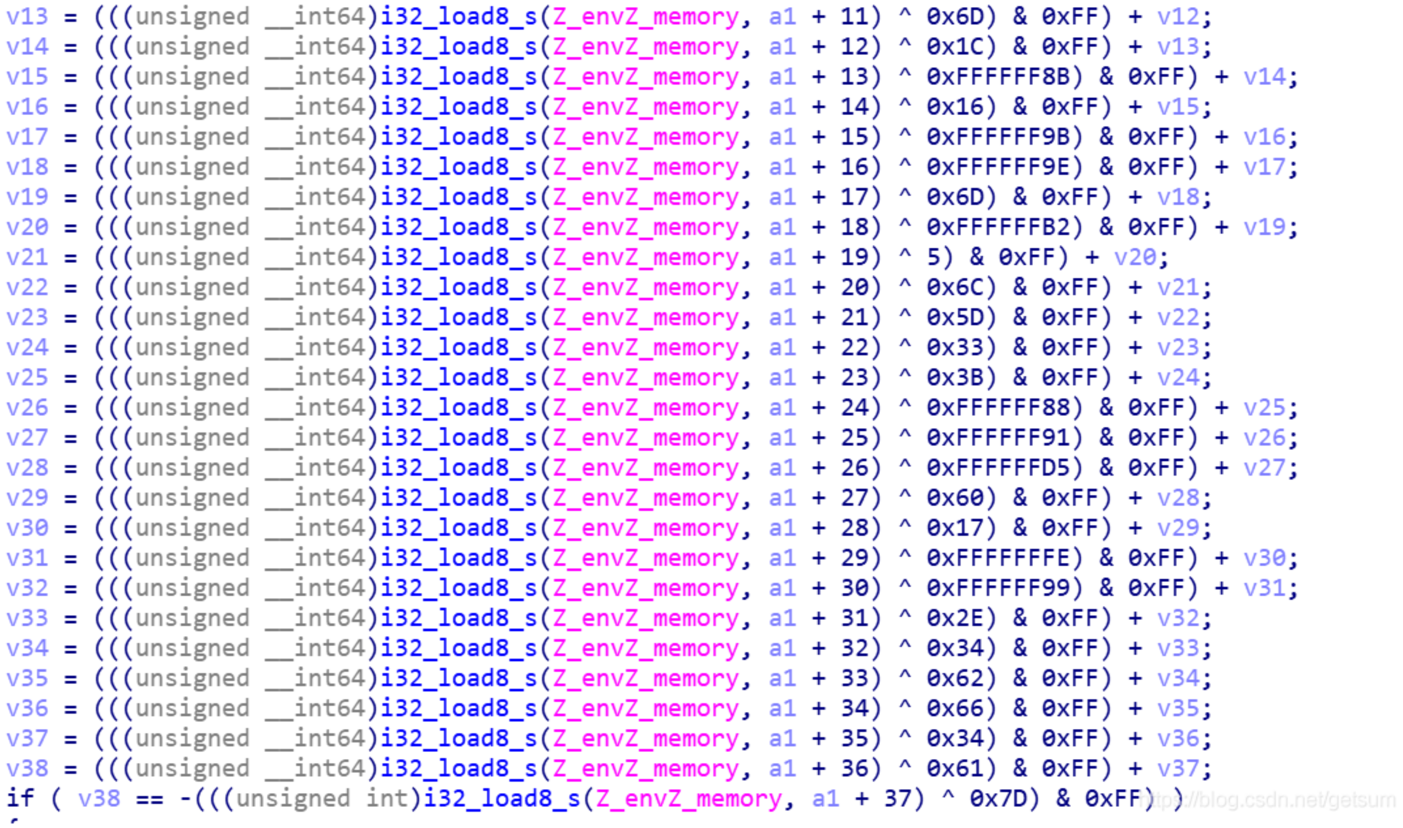


```
v40 = i32_load(Z_envZ_memory, a1 + 4);
part1 = i32_load(Z_envZ_memory, a1);
do
{
  part1 += (((v40 >> 5) ^ 16 * v40) + v40) ^ ((unsigned __int64)i32_load(Z_envZ_memory, v56 + 4 * (v44 & 3)) + v44);
  v44 -= 0x61C88647;
  v40 += ((unsigned __int64)i32_load(Z_envZ_memory, v56 + 4 * ((v44 >> 11) & 3)) + v44) ^ (((part1 >> 5) ^ 16 * part1)
                                                                                        + part1);
  ++v48;
}
while ( v48 != 32 );
```

首先我们看得出来是个XTEA加密，并且key为0（即v44的值，在加密过程之前会初始化，可以看到明显的置零语句）。

然后这样的加密函数会进行4次，每次从输入的flag中依次取8位进行XTEA加密。

于是我们就开始寻找加密后的数据，继续分析，在f15下面可以找到一片赋值操作



```
v13 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 11) ^ 0x6D) & 0xFF) + v12;
v14 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 12) ^ 0x1C) & 0xFF) + v13;
v15 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 13) ^ 0xFFFFFF8B) & 0xFF) + v14;
v16 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 14) ^ 0x16) & 0xFF) + v15;
v17 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 15) ^ 0xFFFFFF9B) & 0xFF) + v16;
v18 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 16) ^ 0xFFFFFF9E) & 0xFF) + v17;
v19 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 17) ^ 0x6D) & 0xFF) + v18;
v20 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 18) ^ 0xFFFFFFB2) & 0xFF) + v19;
v21 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 19) ^ 5) & 0xFF) + v20;
v22 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 20) ^ 0x6C) & 0xFF) + v21;
v23 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 21) ^ 0x5D) & 0xFF) + v22;
v24 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 22) ^ 0x33) & 0xFF) + v23;
v25 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 23) ^ 0x3B) & 0xFF) + v24;
v26 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 24) ^ 0xFFFFFF88) & 0xFF) + v25;
v27 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 25) ^ 0xFFFFFF91) & 0xFF) + v26;
v28 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 26) ^ 0xFFFFFFD5) & 0xFF) + v27;
v29 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 27) ^ 0x60) & 0xFF) + v28;
v30 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 28) ^ 0x17) & 0xFF) + v29;
v31 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 29) ^ 0xFFFFFFFE) & 0xFF) + v30;
v32 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 30) ^ 0xFFFFFF99) & 0xFF) + v31;
v33 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 31) ^ 0x2E) & 0xFF) + v32;
v34 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 32) ^ 0x34) & 0xFF) + v33;
v35 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 33) ^ 0x62) & 0xFF) + v34;
v36 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 34) ^ 0x66) & 0xFF) + v35;
v37 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 35) ^ 0x34) & 0xFF) + v36;
v38 = (((unsigned __int64)i32_load8_s(Z_envZ_memory, a1 + 36) ^ 0x61) & 0xFF) + v37;
if ( v38 == -(((unsigned int)i32_load8_s(Z_envZ_memory, a1 + 37) ^ 0x7D) & 0xFF) )
```

是用加密后的每一位分别异或一个值，然后取低8位，最后和前面一次算出来的值迭代相加。

我们可以假设所有的值最后算出来都等于0，然后就可以反推出加密得到的数据即为异或的数据

```
0x21,0x04,0x24,0x9a,
0xe1,0x41,0xc1,0xa4,//part1

0x2d,0x00,0x63,0x6d,
0x1c,0x8b,0x16,0x9b,//part2

0x9e,0x6d,0xb2,0x05,
0x6c,0x5d,0x33,0x3B,//part3

0x88,0x91,0xd5,0x60,
0x17,0xfe,0x99,0x2e,//part4

0x34,0x62,0x66,0x34,0x61,0x7d//未加密的部分
```

找到了一个比较靠谱的XTEA加密的轮子，然后跑一下即可

```
#include <stdint.h>
#include <stdio.h>

/* take 64 bits of data in v[0] and v[1] and 128 bits of key[0] - key[3] */

void encipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], sum=0, delta=0x9E3779B9;
    for (i=0; i < num_rounds; i++) {
        v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
        sum += delta;
        v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) & 3]);
    }
    v[0]=v0; v[1]=v1;
}

void decipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], delta=0x9E3779B9, sum=delta*num_rounds;
    for (i=0; i < num_rounds; i++) {
        v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) & 3]);
        sum -= delta;
        v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
    }
    v[0]=v0; v[1]=v1;
}

int main(){
 unsigned int key[4]={0,0,0,0};
 unsigned char data[]={
  0x21,0x04,0x24,0x9a,
  0xe1,0x41,0xc1,0xa4,

  0x2d,0x00,0x63,0x6d,
  0x1c,0x8b,0x16,0x9b,

  0x9e,0x6d,0xb2,0x05,
  0x6c,0x5d,0x33,0x3B,

  0x88,0x91,0xd5,0x60,
  0x17,0xfe,0x99,0x2e,
  0x34,0x62,0x66,0x34,0x61,0x7d
 };
 for(int i=0;i<4;i++)
  decipher(32,(unsigned int*)(data+i*8),key);
 printf("%s",data);
}
```