hide—writeup

原创

[0Paw0](https://blog.csdn.net/qq_38756764) 于 2019-05-31 10:23:06 发布 130 收藏

分类专栏： reverse


reverse 专栏收录该内容

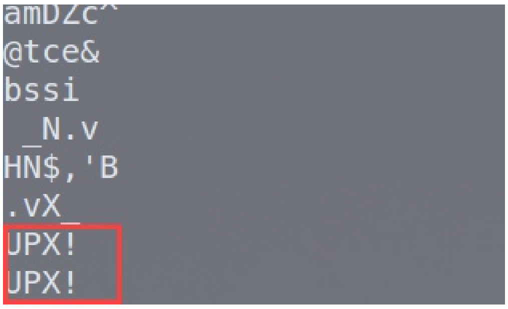
2 篇文章 0 订阅

订阅专栏

> 思路：
>
> 1. 文件加了UPX壳，upx -d脱壳不成功，进行手动脱壳（dd命令将加载进内存的文件dump出来）
> 2. dump出大量函数，利用strings <文件名> 寻找敏感字符串->定位函数->分析加密逻辑
> 3. 编写逆向代码

> 参考文章：
>
> 1. TEA、XTEA、XXTEA加密解密算法 https://blog.csdn.net/gsls200808/article/details/48243019

1. strings hide 加入了UPX壳



2. 手动脱壳，基本原理是程序在运行时会将代码释放到内存中，我们只需在运行时将内存中的可执行代码段dump出来即可

```
./hide
sudo dd if=/proc/$(pidof hidebak)/mem of=hide_dump1 skip=0x400000  bs=1c count=827392
sudo dd if=/proc/$(pidof hide)/mem of=hide_dump2 skip=7110656  bs=1c count=20480
cat hide_dump1 hide_dump2 > hide_dump
#$(pidof hidebak)：进程号（ps -ef查看）
# if=文件名：输入文件名，缺省为标准输入。即指定源文件。< if=input file >
# of=文件名：输出文件名，缺省为标准输出。即指定目的文件。< of=output file >
# ibs=bytes：一次读入bytes个字节，即指定一个块大小为bytes个字节。
    obs=bytes：一次输出bytes个字节，即指定一个块大小为bytes个字节。
    bs=bytes：同时设置读入/输出的块大小为bytes个字节。
# cbs=bytes：一次转换bytes个字节，即指定转换缓冲区大小。
# skip=blocks：从输入文件开头跳过blocks个块后再开始复制。
注意：通常只用当输出文件是磁盘或磁带时才有效，即备份到磁盘或磁带时才有效。
# count=blocks：仅拷贝blocks个块，块大小等于ibs指定的字节数。
```

```
syk@syk-virtual-machine:~/test/reverse$ sudo dd if=/proc/116081/mem of=hide_dump1 skip=4194304  bs=1c count=827392
dd: /proc/116081/mem: cannot skip to specified offset
827392+0 records in
827392+0 records out
827392 bytes (827 kB, 808 KiB) copied, 1.78653 s, 463 kB/s
syk@syk-virtual-machine:~/test/reverse$ sudo dd if=/proc/116081/mem of=hide_dump2 skip=7110656  bs=1c count=1277952
dd: /proc/116081/mem: cannot skip to specified offset
dd: error reading '/proc/116081/mem': Input/output error
20480+0 records in
20480+0 records out
20480 bytes (20 kB, 20 KiB) copied, 0.0713378 s, 287 kB/s
syk@syk-virtual-machine:~/test/reverse$ cat hide_dump1 hide_dump2 > hide_dump
syk@syk-virtual-machine:~/test/reverse$ ls
core  hide_dump  hide_dump2            shm_re
```


3. shift+fn+f12 查看字符串列表

Address	Length	Type	String
LOAD:000···	00000018	C	qwb{this_is_wrong_flag}
LOAD:000···	00000011	C	Enter the flag:\n
LOAD:000···	0000000F	C	You are right\n
LOAD:000···	0000000F	C	You are wrong\n
LOAD:000···	00000014	C	../csu/libc-start.c
LOAD:000···	00000017	C	FATAL: kernel too old\n
LOAD:000···	00000030	C	__ehdr_start.e_phentsize == sizeof *GL(dl_phdr)

4. 定位到sub_4009EF

```
aEnterTheFlag   db 'Enter the flag:',0Ah,0
                                        ; DATA XREF: sub_4009EF+4F↑o
                                        ; LOAD:00000000004C8EC2↓o
; char aYouAreRight[]
aYouAreRight    db 'You are right',0Ah,0
                                        ; DATA XREF: sub_4009EF+8E↑o
                                        ; sub_4C8EF4:loc_4C8FB2↓o
aYouAreWrong    db 'You are wrong',0Ah,0
                                        ; DATA XREF: sub_4009EF+A4↑o
                                        ; sub_4C8EF4:loc_4C8FA9↓o
```


5. 加密函数逻辑

```
signed __int64 sub_4C8EF4()
{
  _BYTE *v0; // rdi
  __int64 *v1; // rsi
  unsigned __int64 v2; // rdx
  signed __int64 result; // rax

  if ( strlen(&unk_6CCDB0) == 21
    && *(&unk_6CCDB0 + 1) == 'w'
    && *(&unk_6CCDB0 + 2) == 'b'
    && *(&unk_6CCDB0 + 3) == '{'
    && *(&unk_6CCDB0 + 20) == '}' )
  {
    sub_4C8CC0(&unk_6CCDB4);           #共进行三轮加密
    sub_4C8E50(&unk_6CCDB4);
    sub_4C8CC0(&unk_6CCDB4);
    sub_4C8E50(&unk_6CCDB4);
    sub_4C8CC0(&unk_6CCDB4);
    v0 = &unk_6CCDB4;
    sub_4C8E50(&unk_6CCDB4);
    v1 = qword_4C8CB0;
    v2 = 0LL;
    while ( v2 < 0x10 && *v0 == *v1 )
    {
      ++v2;
      ++v0;
      v1 = (v1 + 1);
    }
  }
  __asm { syscall; LINUX - sys_write }
  result = 60LL;
  __asm { syscall; LINUX - sys_exit }
  return result;
}
```

6. sub_4C8CC0和sub_4C8E50为一轮加密算法，经查阅，此算法是xtea算法

```
__int64 __fastcall sub_4C8CC0(__int64 a1)
{
  __int64 result; // rax
  unsigned __int64 v2; // rt1
  unsigned int v3; // [rsp+18h] [rbp-48h]
  __int64 v4; // [rsp+1Ch] [rbp-44h]
  signed int i; // [rsp+24h] [rbp-3Ch]
  signed int j; // [rsp+28h] [rbp-38h]
  int v7; // [rsp+40h] [rbp-20h]
  int v8; // [rsp+44h] [rbp-1Ch]
  int v9; // [rsp+48h] [rbp-18h]
  int v10; // [rsp+4Ch] [rbp-14h]
  unsigned __int64 v11; // [rsp+58h] [rbp-8h]

  v11 = __readfsqword(0x28u);
  v7 = 1883844979;
  v8 = 1165112144;
  v9 = 2035430262;
  v10 = 861484132;
  for ( i = 0; i <= 1; ++i )
  {
    v3 = *(8 * i + a1);
    v4 = *(a1 + 4 + 8 * i);
    for ( j = 0; j <= 7; ++j )
    {
      v3 += (*(&v7 + (BYTE4(v4) & 3)) + HIDWORD(v4)) ^ (((v4 >> 5) ^ 16 * v4) + v4);
      HIDWORD(v4) += 1735289196;
      LODWORD(v4) = ((*(&v7 + ((HIDWORD(v4) >> 11) & 3)) + HIDWORD(v4)) ^ (((v3 >> 5) ^ 16 * v3) + v3)) + v4;
    }
    *(a1 + 8 * i) = v3;
    *(a1 + 4 + 8 * i) = v4;
  }
  v2 = __readfsqword(0x28u);
  result = v2 ^ v11;
  if ( v2 != v11 )
    result = (loc_4C8B9A)();
  return result;
}
```

```
_BYTE *__fastcall sub_4C8E50(__int64 a1)
{
  _BYTE *result; // rax
  signed int i; // [rsp+14h] [rbp-4h]

  for ( i = 0; i <= 15; ++i )
  {
    result = (i + a1);
    *result ^= i;
  }
  return result;
}
```

7. 逆向破解代码

```
/*sunyukun 31/5/2019*/
#include <stdio.h>
#include <stdint.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]) {
```

```
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=0X676E696C;
    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=0X676E696C, 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;
}
void xor1(uint32_t v[2]){
    uint32_t t[2];
    unsigned int i=0;
    t[0]=0;
    t[1]=0;
    for(i=0;i<4;i++){
        t[0]+=(((unsigned int)v[0]>>(8*i) & 0xff)^i)<<(8*i);
    }
    for(i=4;i<8;i++){
        t[1]+=(((unsigned int)v[1]>>(8*i) & 0xff)^i)<<(8*i);
    }
    v[0]=t[0];
    v[1]=t[1];
}
void xor2(uint32_t v[2]){
    uint32_t t[2];
    unsigned int i=8;
    t[0]=0;
    t[1]=0;
    for(;i<12;i++){
        t[0]+=(((unsigned int)v[0]>>(8*i) & 0xff)^i)<<(8*i);
    }
    for(;i<16;i++){
        t[1]+=(((unsigned int)v[1]>>(8*i) & 0xff)^i)<<(8*i);
    }
    v[0]=t[0];
    v[1]=t[1];
}
void getflag(uint32_t v[2],uint32_t v2[2]){
    unsigned int i=0;
    printf("qwb{");
    for(i=0;i<4;i++){
        printf("%c", (v[0]>>(8*i)&0xff));
    }
    for(i=0;i<4;i++){
        printf("%c", (v[1]>>(8*i)&0xff));
    }
    for(i=0;i<4;i++){
        printf("%c", (v2[0]>>(8*i)&0xff));
```

```
    }
    for(i=0;i<4;i++){
        printf("%c", (v2[1]>>(8*i)&0xff));
    }
    printf("}");
    printf("\n");
}
//c=[0x52 ,0xB8,0x13 ,0x7F ,0x35 ,0x8C ,0xF2 ,0x1B ,0xF4 ,0x63 ,0x86 ,0xD2 ,0x73 ,0x4F ,0x1E ,0x31]
int main()
{
    // uint32_t v[2]={0x7c11b952,0x1cf48931};
    uint32_t const k[4]={0x70493173,0x45723350,0x79523376,0x33593464};
    unsigned int r=8;//num_rounds建议取值为32
    uint32_t v[2]={0x7f13b852,0x1bf28c35};
    uint32_t v2[2]={0xd28663f4,0x311e4f73};
    //part one
    xor1(v);
    decipher(r, v, k);
    xor1(v);
    decipher(r, v, k);
    xor1(v);
    decipher(r, v, k);
    printf("解密后的低64位数据: 0x%.8x 0x%.8x\n",v[0],v[1]);
    //part two
    xor2(v2);
    decipher(r, v2, k);
    xor2(v2);
    decipher(r, v2, k);
    xor2(v2);
    decipher(r, v2, k);
    printf("解密后的高64位数据: 0x%.8x 0x%.8x\n",v2[0],v2[1]);
    getflag(v,v2);
    return 0;
}
```

运行结果

```
[Running] cd "e:\vscode\candc++\" && g++ hide.cpp -o hide && "e:\vscode\candc++\"hide
解密后的低64位数据: 0x644e3166 0x3348545f
解密后的高64位数据: 0x65646c48 0x45643043
qwb{f1Nd_TH3HldeC0dE}

[Done] exited with code=0 in 1.77 seconds
```