# The Maya Society Reverse Writeup

原创

tuck3r 于 2019-08-08 21:19:35 发布 195 收藏
分类专栏： CTF Reverse 文章标签： ctf reverse




实验内容：

题目：The_Maya_Society

二进制文件：src.zip

配置环境：

VM Win7 SP1 IDA pro 7.0

Kali Linux

解题步骤：

1、首先我们解压src.zip，发现里面有一个launcher和一个网页及相关文件，我们先打开网页，如下：

![The Maya Society HTML](https://img-blog.csdnimg.cn/20190806223049973.png?x-oss-process=image/watermark,type_ZmFuZ3poZW5naGVpdGk,shadow_10,text_aHR0cHM6Ly9ibG9nLmNzZG4ub

从中我们暂时没发现什么信息，其中的download貌似是下载launcher的，但是现在好像不能用了，因此我们使用zip中的launcher就行。

2、首先我们先判断launcher的类型，在Linux中file一下：

```
root@kali:~/桌面/reverse/src# file launcher
launcher: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=464269c36073a8820e69146b00cc8e68d1f50718, stripped
```

发现是64位的ELF文件。（这里我要补充一点，这个ELF文件启用了PIE（https://hardenedlinux.github.io/system-security/2016/07/01/PIC_PIE_and_ASLR_analysis.html）Position Independent Code，因此在使用gdb调试的时候无法确定具体的代码地址，不能再main函数的地方下断点，暂时不知道有什么办法解决。。。。）

（附）gdb调式运行如下：

```
root@kali:~/桌面/reverse/src# gdb -q launcher
Reading symbols from launcher...(no debugging symbols found)...done.
gdb-peda$ startNo unwaited-for children left.
段错误
```

3、我们接下来运行一下：

```
root@kali:~/桌面/reverse/src# ./launcher
root@kali:~/桌面/reverse/src#
```

发现什么都没有。我们先用IDA来看一看代码：

```
signed __int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  size_t src_len; // rbx
  size_t f_net_len; // rax
  unsigned __int64 dest_2_len; // rax
  unsigned __int64 v7; // rax
  unsigned __int64 v8; // rsi
  char *v9; // rdi
  time_t timer; // [rsp+18h] [rbp-128h]
  char fluxfingers_net[8]; // [rsp+20h] [rbp-120h]
  char src; // [rsp+40h] [rbp-100h]
  char s; // [rsp+60h] [rbp-E0h]
  unsigned __int64 v14; // [rsp+C8h] [rbp-78h]
  char v15; // [rsp+D4h] [rbp-6Ch]
  char v16; // [rsp+DDh] [rbp-63h]
  char v17; // [rsp+E6h] [rbp-5Ah]
  char v18; // [rsp+EFh] [rbp-51h]
  char *v19; // [rsp+F8h] [rbp-48h]
  char *dest_3; // [rsp+100h] [rbp-40h]
  char *dest_2; // [rsp+108h] [rbp-38h]
  char *dest; // [rsp+110h] [rbp-30h]
  int *v23; // [rsp+118h] [rbp-28h]
  size_t len_s; // [rsp+120h] [rbp-20h]
  struct tm *tp; // [rsp+128h] [rbp-18h]

  strcpy(fluxfingers_net, ".fluxfingers.net");
  timer = time(0LL);
  tp = localtime(&timer);
  strftime(&s, 0x63uLL, "%Y-%m-%d", tp);        // 格式化日期并以char形式存放在s中
  len_s = strlen(&s);
  sub_5576392A4B5A(&s, len_s);
  v23 = &dword_5576394A70B8;                    // 在函数sub_5606726BEB5A进行初始化
  snprintf(
    &v18,                                       // v18的空间至多为9bytes,能存放最多8bytes
    9uLL,
    "%02x%02x%02x%02x",
    (unsigned __int8)dword_5576394A70B8,
    BYTE1(dword_5576394A70B8),
    BYTE2(dword_5576394A70B8),
    HIBYTE(dword_5576394A70B8));
  v23 = &dword_5576394A70C0;
  snprintf(
    &v17,
```

```
    9uLL,
    "%02x%02x%02x%02x",
    (unsigned __int8)dword_5576394A70C0,
    BYTE1(dword_5576394A70C0),
    BYTE2(dword_5576394A70C0),
    HIBYTE(dword_5576394A70C0));
  v23 = &dword_5576394A70B4;
  snprintf(
    &v16,
    9uLL,
    "%02x%02x%02x%02x",
    (unsigned __int8)dword_5576394A70B4,
    BYTE1(dword_5576394A70B4),
    BYTE2(dword_5576394A70B4),
    HIBYTE(dword_5576394A70B4));
  v23 = &dword_5576394A70BC;
  snprintf(
    &v15,
    9uLL,
    "%02x%02x%02x%02x",
    (unsigned __int8)dword_5576394A70BC,
    BYTE1(dword_5576394A70BC),
    BYTE2(dword_5576394A70BC),
    HIBYTE(dword_5576394A70BC));
  snprintf(&src, 0x21uLL, "%s%s%s%s", &v18, &v17, &v16, &v15);
  src_len = strlen(&src);
  f_net_len = strlen(fluxfingers_net);
  dest = (char *)malloc(src_len + f_net_len + 1);
  if ( !dest )
    return 1LL;
  *dest = 0;
  strcat(dest, &src);
  strcat(dest, fluxfingers_net);
  dest_2 = sub_5576392A58A4(dest);              // 如果该函数返回零，则会跳出
  if ( !dest_2 )
    return 1LL;
  dest_2_len = strlen(dest_2);
  dest_3 = sub_5576392A55E0((__int64)dest_2, dest_2_len, &v14);
  v7 = strlen(dest_2);
  v19 = sub_5576392A55E0((__int64)dest_2, v7, &v14);
  if ( !dest_3 )
    return 1LL;
  v8 = v14;
  v9 = dest_3;
  sub_5576392A5858((__int64)dest_3, v14, (__int64)v19);
  ((void (__fastcall *)(char *, unsigned __int64))v19)(v9, v8);
  return 0LL;
}
```

从代码中我们可以看到调用time函数，获取当前的时间，并使用当前的时间来对内存进行操作，但此处我们并不知道具体正确的时间是什么时候，这个时候就需要脑洞啦。。。我们在之前的网页中发现有一个2012，因此这可能就是提示时间是2012年的某一天，因此我们可以采用爆破的办法。

3、接下来就是重点了。先介绍几个重要结构：

```
struct tm
{
    int tm_sec;  /*秒，正常范围0-59， 但允许至61*/
    int tm_min;  /*分钟，0-59*/
    int tm_hour; /*小时， 0-23*/
    int tm_mday; /*日，即一个月中的第几天，1-31*/
    int tm_mon;  /*月， 从一月算起，0-11*/  1+p->tm_mon;
    int tm_year;  /*年， 从1900至今已经多少年*/  1900＋ p->tm_year;
    int tm_wday; /*星期，一周中的第几天， 从星期日算起，0-6*/
    int tm_yday; /*从今年1月1日到目前的天数，范围0-365*/
    int tm_isdst; /*日光节约时间的旗标*/
};


time_t
Although not defined by the C standard, this is almost always an integral value holding the number of secon


time_t mktime ( struct tm * timeptr );
You can use the mktime() function to convert a struct tm into a time_t, which is an integer value.


size_t strftime (char* ptr, size_t maxsize, const char* format,
                 const struct tm* timeptr );
Copies into ptr the content of format, expanding its format specifiers into the corresponding values that r
```

此处我们需要在2012-01-01--2012-12-31之间进行爆破，首先需要算出当时距离1970年有多少秒，此处我们可以借助Windows自带的计算器，得到值为1325376000---1356912000之间，我们可以写一个动态链接库，里面包含一个伪time函数，让launcher运行时调用我们的伪time函数，获取我们指定的时间，另一方面，我们可以再使用一个bash脚本，来进行循环。代码如下：

```
//faketime.c
#include <stdio.h>
#include <stdlib.h>

int time(int * second )
{
    return atoi(getenv("CURR_TIME"));
}
```

（说明：getenv - get value of an environment variable

可参考http://pubs.opengroup.org/onlinepubs/007904975/functions/getenv.html）

在Linux中编译为.so动态链接库（https://stackoverflow.com/questions/14884126/build-so-file-from-c-file-using-gcc-command-line）

具体如下：

```
gcc -c -fPIC faketime.c -o faketime.o
gcc faketime.o -shared -o faketime.so
```

或者：

```
gcc -shared -o faketime.so -fPIC faketime.c
```

然后编写一个简单地bash script：

```
# time.sh
#!/bin/bash
for (( c=1325376000; c<=1356912000; c++ ))
do
 # echo "c=" $c "\r"
    set CURR_TIME = c  LD_PRELOAD=$(pwd)/faketime.so  ./launcher
done
```

（说明：set设置环境变量的值，

If you set `LD_PRELOAD` to the path of a shared object, that file will be loaded **before** any other library (including the C runtime, `libc.so`). So to run `ls` with your special `malloc()` implementation, do this:

```
$ LD_PRELOAD=/path/to/my/malloc.so /bin/ls
```

可参考https://stackoverflow.com/questions/426230/what-is-the-ld-preload-trick）

然后运行：

```
root@kali:~/桌面/reverse/src# ./time.sh
```

耐心等待即可。。