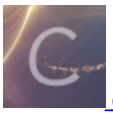
攻防世界Reverse进阶区-SignIn-writeup

原创

y4ung 于 2020-12-04 20:36:17 发布 459 收藏

分类专栏： ctf 文章标签： ctf

本文链接：https://blog.csdn.net/qq_35056292/article/details/110672456



1. 介绍

本题是xctf攻防世界中Reverse的进阶区的题SignIn

题目来源： 2019_SUCTF

2. 分析

2.1 静态分析

```
$ file signin
signin: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=b98a398a7beb2d05a64c2679ad5b937c80ce533f, stripped
$ chmod +x signin
$ ./signin
[sign in]
[input your flag]: 123
GG!
```

扔到IDA里看一下。字符串 `[input your flag]:` 在main函数中使用。用户输入保存在v8中。然后调用了 `sub_96A(&v8, &v9)` ，参数是v8和v9。通过gdb调试可知，该函数将用户输入的十六进制保存在v9中。

再往下，调用了很多次 `__gmpz_init_set_str` 函数，通过网上的资料gmp.h 可以知道，这个函数其实是gmp.h里的 `mpz_init_set_str` 函数。该函数的作用是把字符串初始化为gmp大整数， `mpz_init_set_str` 函数的三个参数分别是多精度整数变量，字符串，进制。

> gmp又叫GNU多精度算术库，是一个提供了很多操作高精度的大整数，浮点数的运算的算术库，几乎没有什么精度方面的限制，功能丰富。GMP的主要目标应用领域是密码学的应用和研究、 互联网安全应用、 代数系统、 计算代数研究等。

```
#define mpz_init_set_str __gmpz_init_set_str
__GMP_DECLSPEC int mpz_init_set_str (mpz_ptr, const char *, int);
```

说白了， `mpz_init_set_str(mpz_ptr, const char *, int)` 函数，就是根据第三个参数指定的进制，对第二个参数字符串进行转换，并保存到第一个参数中。

比如， `mpz_init_set_str(z_i, "1", 10);` 就是将1看成十进制的数，保存到z_i中，运行完该函数以后，z_i的值为1。

再比如，题目中的

```
__gmpz_init_set_str((__int64)&v7, (__int64)"ad939ff59f6e70bcbfad406f2494993757eee98b91bc244184a377520d06fc35", 16LL);
```

就是将字符串`"ad939ff59f6e70bcbfad406f2494993757eee98b91bc244184a377520d06fc35"`看做是一个十六进制数，将其保存到v7中，v7中的值为数字类型的十六进制数`ad939ff59f6e70bcbfad406f2494993757eee98b91bc244184a377520d06fc35`。

这里，可以写个脚本测试一下：

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

int main(void){
        mpz_t v7;
        mpz_init_set_str(v7, "ad939ff59f6e70bcbfad406f2494993757eee98b91bc244184a377520d06fc35", 16);
        gmp_printf("%Zx\n", v7);

        mpz_t pie;
        mpz_init_set_str (pie,"3141592653589793238462643383279502884", 10);
        gmp_printf("%Zd\n", pie);

        return 0;
}
```

编译：

```
gcc -o test test.c -lgmp -lm
```

运行，输出：

```
$ ./test
ad939ff59f6e70bcbfad406f2494993757eee98b91bc244184a377520d06fc35
3141592653589793238462643383279502884
```

用python验证一下：

```
v7_raw = "ad939ff59f6e70bcbfad406f2494993757eee98b91bc244184a377520d06fc35"
v7 = "78510953323073667749065685964447569045476327122134491251061064910992472210485"
str(int(v7_raw, 16)) == v7  # True
```

因此，main函数中的这段代码，就是以不同的进制，将字符串转为数字。
其中，用户输入的字符串先转成16进制字符串v9，再通过__gmpz_init_set_str函数变成十六进制数v6。

```
16  __gmpz_init_set_str((__int64)&v7, (__int64)"ad939ff59f6e70bcbfad406f2494993757eee98b91bc244184a377520d06fc35", 16LL);
17  __gmpz_init_set_str((__int64)&v6, (__int64)&v9, 16LL);
18  __gmpz_init_set_str(
19    (__int64)&v4,
20    (__int64)"10346103590081691412139010129904904441395040517371217043416168653987816098454 9",
21    10LL);
22  __gmpz_init_set_str((__int64)&v5, (__int64)"65537", 10LL);
```


然后，调用了`__gmpz_powm`函数。根据文档gmp_lib.mpz_powm Method，函数的原型为：

C# VB C++ F#

Copy

```
public:
static void mpz_powm(
        mpz_t^ rop,
        mpz_t^ base,
        mpz_t^ exp,
        mpz_t^ mod
)
```

Parameters

rop
Type: Math.Gmp.Native::mpz_t
The result integer.

base
Type: Math.Gmp.Native::mpz_t
The base integer.

exp
Type: Math.Gmp.Native::mpz_t
The exponent integer.

mod
Type: Math.Gmp.Native::mpz_t
The modulo integer.

也就是：rop = (base ^ exp) % mod，求base的exp次方，再对mod取模以后的结果保存到rop中。对应到本题中，就是`v6 = (v6 ^ v5) % v4`。最终，v6必须与v7相等才行，即 v7 = (v6 ^ v5) % v4，求v6。

本来想写个脚本跑一下，但是好像不是很好写嘛…

2.2 寻找网上的writeup

实在想不出来咋写，去网上找了下writeup：复盘 SUCTF 2019 Reverse 精解。

文章里提到本题其实是一个已知密文，求解RSA明文的题，需要用到yafu 这个工具。

关于RSA的介绍，可以参考阮一峰老师的博客：RSA算法原理（二）

> 看来还是自己的知识面太窄了…

```
D:\sectools\yafu-1.34>yafu-x64.exe
factor(10346103590081691412139010129904904441395040517371217043416168653987816098454 9)


fac: factoring 10346103590081691412139010129904904441395040517371217043416168653987816098454 9
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
fmt: 1000000 iterations
rho: x^2 + 3, starting 1000 iterations on C78
rho: x^2 + 2, starting 1000 iterations on C78
rho: x^2 + 1, starting 1000 iterations on C78
pm1: starting B1 = 150K, B2 = gmp-ecm default on C78
ecm: 30/30 curves on C78, B1=2K, B2=gmp-ecm default
ecm: 74/74 curves on C78, B1=11K, B2=gmp-ecm default
ecm: 161/161 curves on C78, B1=50K, B2=gmp-ecm default, ETA: 0 sec

starting SIQS on c78: 10346103590081691412139010129904904441395040517371217043416168653987816098454 9

==== sieving in progress (1 thread):   36224 relations needed ====
====           Press ctrl-c to abort and save state           ====
36211 rels found: 18754 full + 17457 from 185306 partial, (2471.78 rels/sec)

SIQS elapsed time = 84.1098 seconds.
Total factoring time = 100.7290 seconds


***factors found***

P39 = 366669102002966856876605669837014229419
P39 = 282164587459512124844245113950593348271

ans = 1
```

因此，根据阮一峰老师的博客，

```
p = 366669102002966856876605669837014229419
q = 282164587459512124844245113950593348271
```

e = 65537

接下来就可以求出私钥 d，并通过私钥 d，求出明文 m，再将其转化成 ASCII 即可得到 flag：`suctf{Pwn_@_hundred_years}`

gmpy2的安装可以参考Ubuntu 安装gmpy2模块

```
import gmpy2
p = 366669102002966856876605669837014229419
q = 282164587459512124844245113950593348271
N = 10346103590081691412139010129904904441395040517371217043416168653987816098454 9
c = 0xad939ff59f6e70bcbfad406f2494993757eee98b91bc244184a377520d06fc35
e = 65537
d = gmpy2.invert(e,(p-1)*(q-1))
m = gmpy2.powmod(c,d,p*q)

print(ex(m)[2:].decode('hex'))
```

3. 总结

平时应该对一些通信的过程、加密算法有一些了解。还有对一些解密的工具也有些了解。可以适当做点Crypto的题。