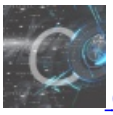
攻防世界 pseudorandom Writeup

原创

Code Segment 于 2021-02-17 10:05:54 发布 188 收藏

分类专栏： CTF Reverse python

本文链接：https://blog.csdn.net/qq_43547885/article/details/113831507


CTF Reverse 同时被 2 个专栏收录

6 篇文章 0 订阅

订阅专栏


python

5 篇文章 0 订阅

订阅专栏

攻防世界 pseudorandom Writeup

初步分析

- 在 IDA 中分析二进制文件

```
printf("I will generate some random numbers.\n");
printf("If you can give me those numbers, you will be $$rewarded$$\n");
printf("hmm..thinking...");
fflush(stdout);
rand();
v18 = rand();
v17 = 0;
printf("OK. I am Ready. Enter Numbers.\n");
v14 = sub_400C30(v18);
v15 = (1 << sub_400B40(v18)) - 1;
MD5_Init(v12);
SHA1_Init(v11);
while ( v17 != v18 )
{
  __isoc99_scanf("%d", &v16);
  if ( !(unsigned int)sub_400EA0(v15, v16) )
    sub_400AE0();
  v15 += v16;
  sprintf(s, "%d", v16);
  v3 = strlen(s);
  MD5_Update(v12, s, v3);
  v4 = strlen(s);
  SHA1_Update(v11, s, v4);
  while ( (~v14 | ~v15) != -1 )
  {
    v17 = v14 ^ v17 | v14 & v17;
    v15 &= v14 ^ v15;
    v14 = sub_400C30(v18 & (v17 ^ v18));
  }
}
MD5_Final(v9, v12);
for ( i = 0; i < 16; ++i )
  sprintf(&s1[2 * i], "%02x", (unsigned __int8)v9[i]);
if ( strcmp(s1, (const char *)(unsigned int)"15b74b4db57d0afdfe98eb5dbc3b542b") )
  sub_400AE0();
printf("Good Job!!\n");
printf("Wait till I fetch your reward...");
fflush(stdout);
rand();
SHA1_Final(v8, v11);
for ( i = 0; i < 20; ++i )
  sprintf(&v6[2 * i], "%02x", (unsigned __int8)v8[i]);
printf("OK. Here it is\n");
for ( i = 0; i < 40; ++i )
  v6[i] = (dword_6020D0[i] & 0x4A | ~LOBYTE(dword_6020D0[i]) & 0xB5) ^ (v6[i] & 0x4A | ~v6[i] & 0xB5);
```

- 程序的主要逻辑还是比较好理解的，先是获得一个伪随机数 x，然后根据 x 初始化一些数，通过这些数去约束用户的输入
- 用户输入的数被用来计算 MD5，计算后与内置的 MD5 值进行比对，比对通过即可拿到 flag
- 可以看到主要是通过这个 `verify` 函数对输入进行的约束

```
__isoc99_scanf( "%d", &v16);
if ( !(unsigned int)veryfy(v15, v16) )
  sub_400AE0();
v15 += v16;
```

分析 `verify` 函数

- `verify` 的第一个参数最初由程序内部运算后给出，第二个参数是我们的输入。当第一个输入的数通过验证后，`verify` 的第一个参数会加上输入的数。因此自然想到可以逐次地对输入的数进行爆破
- 通过动态调试，知道第一次 `verify` 的第一个参数为 `0xffff`
- 这时有多种不同的方式可以对输入的数进行爆破
    1. IDAPython 调用 Appcall
    2. gdb 脚本
    3. **angr**
- 在网上看大佬的 Writeup 的时候看到了用 angr 进行符号执行进而得到输入的解法，这里就用 angr 了

使用 `angr` 分析 `verify` 函数

angr 进行符号执行的大致流程如下

1. 初始化一个 `state`，`state` 中包含了符号执行的起始位置，各个寄存器和内存单元的值等
2. 指定 `find`, `avoid` 等参数。对于当前题目，`find` 就是 `verify` 函数即将结束，并将返回值放入 `eax` 处指令的地址
3. 调用 `explore` 函数进行符号执行
4. 在 `found` 成员中可以找到到达 `find` 处的一个 `state`
5. 为 `found` 处的 `state` 增加约束条件，进而可以反求得之前待确定的数

写 angr 脚本时建议使用 `Jupyter Notebook`，因为 `import angr` 真的太费时间了

附上脚本

```
# To add a new cell, type '# %%'
# To add a new markdown cell, type '# %% [markdown]'
# %%
import angr
# %%
# 导入项目
proj=angr.Project("./pseudorandom")
# 初始 state 为 verify 函数的地址
state=proj.factory.blank_state(addr=0x400EA0)
# BVS 可以理解为符号的意思，也就是为输入->输出的映射指定一个自变量。一般来说这就是我们要求解的值
arg2=state.solver.BVS('arg2',32)
# 通过动调知道 verify 的第一个参数为 0xffff
state.regs.edi=0xffff
state.regs.esi=arg2
# 初始化 simulation_manager
simgr=proj.factory.simulation_manager(state)
# 找到一条 0x400ea0 -> 0x401039 的路径
simgr.explore(find=0x401039)
# %%
# 获得 found 的 state
found=simgr.found[0]
# 增加限制条件，即存放返回值的内存单元为 True
found.add_constraints(found.memory.load(found.regs.rbp-8,4)!=0)
# 对输入进行求解
value=found.solver.eval(arg2)
print(hex(value))
```

此时得到了第一个合法的输入（唯一），`verify` 的第一个参数产生了变化，然后提示用户输入第二个数并进行验证。因此可以得到下一个数（重复调用上面的脚本呢即可），得到的第二个数为 0x10000

发现输入序列是部分有规律的，对于有规律的子序列，第 n+1 个数为 2 倍的第 n 个数，因此没必要逐次调用上面的脚本了，只在上述规律不好用的时候用脚本算一下下一个值就行了

得到输入的序列

```
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
268435456
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
2048
4096
```

```
8192
16384
32768
65536
131072
262144
524288
1048576
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
256
512
1024
2048
4096
8192
16384
32768
65536
128
256
512
1024
2048
4096
8192
16384
32768
64
128
256
512
1024
2048
4096
32
64
128
256
16
32
```

```
64
128
8
16
32
64
4
8
16
32
2
1
```

输入上述序列即可得到 flag