XCTF 逆向 re1-100简单wp

原创

菜鸟m号 于 2019-10-17 21:27:26 发布 603 收藏

文章标签： CTF XCTF 逆向

本文链接：https://blog.csdn.net/jentle8/article/details/102615301

题目下载链接：re1-100

下载之后拖到kali中运行一下发现，超级简单的demo，直接让输入密码：

```
root@kali:~/Desktop/xctfpwn# ./RE100
Input key : 12345
Wrong !!!
```

查看文件属性，发现是64位的elf:

```
root@kali:~/Desktop/xctfpwn# file RE100
RE100: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), dynamically lin
ked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1
]=94dd59e952c54304bd14f282695ae62c4f6cbe55, with debug_info, not stripped
root@kali:~/Desktop/xctfpwn# ./RE100
```

拖到ida 64进行分析，讲真，第一眼看到这个缩略图我以为进行了代码混淆，后来发现是我想多了。

```
if ( pipe(pParentWrite) == -1 )
  exit(1);
if ( pipe(pParentRead) == -1 )
  exit(1);
v3 = fork();
if ( v3 != -1 )
{
  if ( v3 )
  {
    close(pParentWrite[0]);
    close(pParentRead[1]);
    while ( 1 )
    {
      printf("Input key : ", argva);
      memset(bufWrite, 0, 0xC8uLL);
      gets(bufWrite, 0LL);                    // 保存输入的值
      v4 = strlen(bufWrite);
      v5 = write(pParentWrite[1], bufWrite, v4);
      if ( v5 != strlen(bufWrite) )
        printf("parent - partial/failed write", bufWrite);
      do
      {
        memset(bufParentRead, 0, 0xC8uLL);
        numReada = read(pParentRead[0], bufParentRead, 0xC8uLL);
```

第一个IF语句判断就是liunx正常创建管道读写我们输入的值（应该是我理解的），后面的才是有意思的部分：

```
{
  memset(bufParentRead, 0, 0xC8uLL);
  numRead = read(pParentWrite[0], bufParentRead, 0xC8uLL);
```

```
if ( numRead == -1 )
  break;
if ( numRead )
{
  if ( childCheckDebugResult() )
  {
    responseFalse();
  }
  else if ( bufParentRead[0] == '{' )
  {
    if ( strlen(bufParentRead) == 42 )
    {
      if ( !strncmp(&bufParentRead[1], "53fc275d81", 0xAuLL) )
      {
        if ( bufParentRead[strlen(bufParentRead) - 1] == '}' )
        {
          if ( !strncmp(&bufParentRead[31], "4938ae4efd", 0xAuLL) )
          {
            if ( !confuseKey(bufParentRead, 42) )
            {
              responseFalse();
```


直接了当的判断，然后前十个数字必须是53fc275d81，然后最后十个一定是4938ae4efd。最后的那个confusekey函数是处理的关键，就是交换输入的一串数字的顺序，进去康康：

```
strncpy(szPart1, szKey + 1, 0xAuLL);
strncpy(szPart2, szKey + 11, 0xAuLL);
strncpy(szPart3, szKey + 21, 0xAuLL);
strncpy(szPart4, szKey + 31, 0xAuLL);
memset(szKey, 0, iKeyLength);
*szKey = 123;
strcat(szKey, szPart3);
strcat(szKey, szPart4);
strcat(szKey, szPart1);
strcat(szKey, szPart2);
szKey[41] = '}';
```


就是把输入的字符串进行划分，四个长度为10的小块，然后进行3，4，1，2 的顺序进行拼接，拼接完之后和它进行比较：

```
    }
    else if ( !strncmp(bufParentRead, "{daf29f59034938ae4efd53fc275d81053ed5be8c}", 0x2AuLL)
    {
```

所以思路还是比较清晰的，最后得到flag：

```
Input key : {53fc275d81053ed5be8cdaf29f59034938ae4efd}
True
```