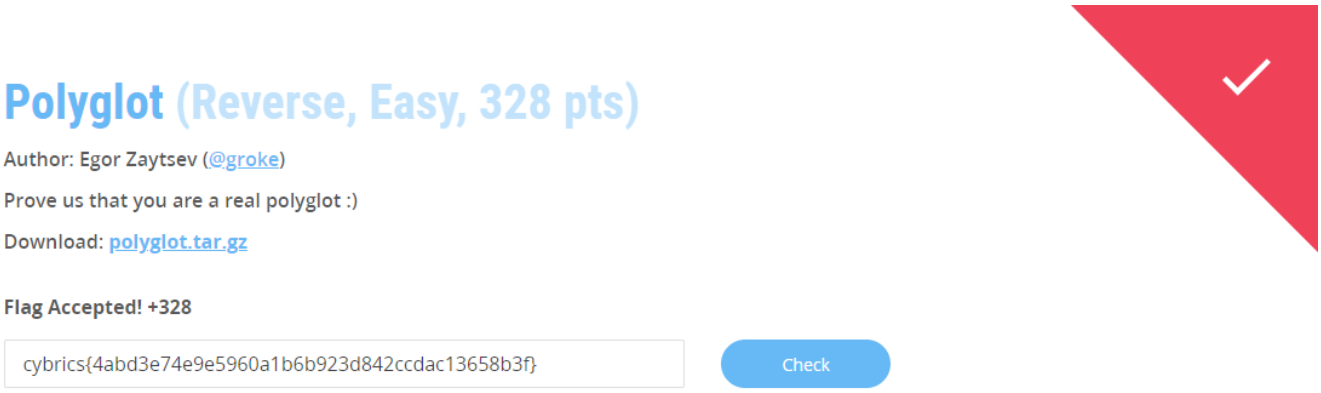
【CYBRICS】Polyglot WriteUp

原创

古月浪子 于 2020-07-29 16:20:46 发布 109 收藏

文章标签： CTF

本文链接：https://blog.csdn.net/tqydyqt/article/details/107584907



这道题有毒。。。

附件是一个.c源代码

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char flagged[] = {78,6,10,80,30,69,0,68,77,83,13,92,1,68,22,68,12,2,90,62,120,68,1,76,29,3,5

int main(){

    char *key = getenv("XKEY");
    if((!key) ||strncmp("mod3r0d!",key,8 )){
        puts(";[");
        return 1;
    }
    unsigned long long val = *(unsigned long long *)key;
    unsigned long long *ptr = (unsigned long long *)flagged;
    while (*ptr != 0) {
        *ptr = *ptr ^ val;
        ptr += 1;
    }
    puts(flagged);
}
```

我们自己写一下代码跑一下，把key直接写死

```
int main() {
 char flagged[] = { ..... };
 char key[] = "mod3r0d!";
 unsigned long long val = *(unsigned long long*)key;
 unsigned long long *ptr = (unsigned long long*)flagged;
 while (*ptr != 0) {
  *ptr = *ptr ^ val;
  ptr += 1;
 }
 puts(flagged);
}
```

跑出来是另一段源码

```
#include <iostream>
template <unsigned int a, unsigned int b>
struct t1 {
        enum { value = b + t1<a-1, b>::value };
};
template <unsigned int b>
struct t1<0, b> {
        enum { value = 0 };
};
template <unsigned int a, unsigned int b>
struct t2 {
        enum { value = 1 + t2<a-1, b>::value };
};
template <unsigned int b>
struct t2<0, b> {
        enum { value = 1 + t2<0, b-1>::value };
};
template<>
struct t2<0, 0>{
    enum { value = 0};
};
void decode(unsigned char *data, unsigned int val){
    unsigned int *ptr = reinterpret_cast<unsigned int *>(data);
    while (*ptr != 0) {
        *ptr = *ptr ^ val;
        val = (val ^ (val << 1)) ^ 0xc2154216;
        ptr += 1;
    }
}
unsigned char flagged[] = { ..... };
int main(){
    decode(flagged, t2<0xcaca0000, t2<444, t1<t2<100, t1<4,3>::value>::value, t2<44, t1<11,3>::value>::value>::v
alue>::value>::value);
    std::cout << flagged <<std::endl;
}
```

我们对源码稍做改动

```cpp
unsigned int t1(unsigned int a, unsigned int b) {
 if (a == 0)
  return 0;
 return b + t1(a - 1, b);
}

unsigned int t2(unsigned int a, unsigned int b) {
 if (a == 0) {
  if (b == 0)
   return 0;
  else
   return 1 + t2(0, b - 1);
 }
 return 1 + t2(a - 1, b);
}

void decode(unsigned char *data, unsigned int val) {
 unsigned int *ptr = reinterpret_cast<unsigned int *>(data);
 while (*ptr != 0) {
  *ptr = *ptr ^ val;
  val = (val ^ (val << 1)) ^ 0xc2154216;
  ptr += 1;
 }
}

unsigned char flagged[] = { ..... };

int main() {
 unsigned int tmp = t2(444, t1(t2(100, t1(4, 3)), t2(44, t1(11, 3))));
 // tmp = 9068
 // t2(0xcaca0000, tmp)
 decode(flagged, 0xcaca0000 + 9068);
 cout << flagged << endl;
}
```

我把模板结构体改成了函数
注意到最外层的t2递归太多次了，会爆栈，不过它的逻辑很简单，就是a+b，因此计算出tmp的值硬编码进去即可
这次运行会打印出python源码

```
import types

def define_func(argcount, nlocals, code, consts, names):
    #PYTHON3.8!!!
    def inner():
        return 0

    fn_code = inner.__code__
    cd_new = types.CodeType(argcount,
                            0,
                            fn_code.co_kwonlyargcount,
                            nlocals,
                            1024,
                            fn_code.co_flags,
                            code,
                            consts,
                            names,
                            tuple(["v%d" for i in range(nlocals)]),
                            fn_code.co_filename,
                            fn_code.co_name,
                            fn_code.co_firstlineno,
                            fn_code.co_lnotab,
                            fn_code.co_freevars,
                            fn_code.co_cellvars)
    inner.__code__ = cd_new
    return inner

f1 = define_func(2,2,b'|\x00|\x01k\x02S\x00', (None,), ())
f2 = define_func(1,1,b't\x00|\x00\x83\x01S\x00', (None,), ('ord',))
f3 = define_func(0,0,b't\x00d\x01\x83\x01S\x00', (None,  'Give me flag: '), ('input',))
f4 = define_func(1, 3, b'd\x01d\x02d\x03d\x04d\x05d\x01d\x06d\x07d\x08d\td\x03d\nd\x0bd\x0cd\rd\x08d\x0cd\x0ed\x
0cd\x0fd\x0ed\x10d\x11d\td\x12d\x03d\x10d\x03d\x0ed\x13d\x0bd\nd\x14d\x08d\x13d\x01d\x01d\nd\td\x01d\x12d\x0bd\x
10d\x0fd\x14d\x03d\x0bd\x15d\x16g1}\x01t\x00|\x00\x83\x01t\x00|\x01\x83\x01k\x03r\x82t\x01d\x17\x83\x01\x01\x00d
\x18S\x00t\x02|\x00|\x01\x83\x02D\x00]$}\x02t\x03|\x02d\x19\x19\x00t\x04|\x02d\x1a\x19\x00\x83\x01\x83\x02d\x18k
\x02r\x8c\x01\x00d\x18S\x00q\x8cd\x1bS\x00',
                (None, 99, 121, 98, 114, 105, 115, 123, 52, 97, 100, 51, 101, 55, 57, 53, 54, 48, 49, 50, 56, 1
02, 125, 'Length mismatch!', False, 1, 0, True),
                ('len', 'print', 'zip', 'f1', 'f2'))
f5 = define_func(0, 1,b't\x00\x83\x00}\x00t\x01|\x00\x83\x01d\x01k\x08r\x1ct\x02d\x02\x83\x01\x01\x00n\x08t\x02d
\x03\x83\x01\x01\x00d\x00S\x00',(None, False, 'Nope!', 'Yep!'), ('f3', 'f4', 'print'))
f5()
```

emmm...看起来好像很复杂的样子

不过值得庆幸的是，在上次的ctf比赛中我认识了一个叫dis的python库，没想到这么快就用上了
先对f5进行dis

```
import dis

print(dis.dis(f5))
```

得到的结果是

```
  7           0 LOAD_GLOBAL              0 (f3)
              2 CALL_FUNCTION            0
              4 STORE_FAST               0 (v%d)
              6 LOAD_GLOBAL              1 (f4)
              8 LOAD_FAST                0 (v%d)
             10 CALL_FUNCTION            1
             12 LOAD_CONST               1 (False)
             14 COMPARE_OP               8 (is)
             16 POP_JUMP_IF_FALSE       28
             18 LOAD_GLOBAL              2 (print)
             20 LOAD_CONST               2 ('Nope!')
             22 CALL_FUNCTION            1
             24 POP_TOP
             26 JUMP_FORWARD             8 (to 36)
        >>   28 LOAD_GLOBAL              2 (print)
             30 LOAD_CONST               3 ('Yep!')
             32 CALL_FUNCTION            1
             34 POP_TOP
        >>   36 LOAD_CONST               0 (None)
             38 RETURN_VALUE
None
```

看不懂伪代码的可以上网搜一下，个人感觉比较容易看清楚逻辑了

可以看出，主要逻辑在f4中，因此对f4进行dis

```
import dis

print(dis.dis(f4))
```

得到的结果是

```
  7           0 LOAD_CONST               1 (99)
              2 LOAD_CONST               2 (121)
              4 LOAD_CONST               3 (98)
              6 LOAD_CONST               4 (114)
              8 LOAD_CONST               5 (105)
             10 LOAD_CONST               1 (99)
             12 LOAD_CONST               6 (115)
             14 LOAD_CONST               7 (123)
             16 LOAD_CONST               8 (52)
             18 LOAD_CONST               9 (97)
             20 LOAD_CONST               3 (98)
             22 LOAD_CONST              10 (100)
             24 LOAD_CONST              11 (51)
             26 LOAD_CONST              12 (101)
             28 LOAD_CONST              13 (55)
             30 LOAD_CONST               8 (52)
             32 LOAD_CONST              12 (101)
             34 LOAD_CONST              14 (57)
             36 LOAD_CONST              12 (101)
             38 LOAD_CONST              15 (53)
             40 LOAD_CONST              14 (57)
             42 LOAD_CONST              16 (54)
             44 LOAD_CONST              17 (48)
             46 LOAD_CONST               9 (97)
             48 LOAD_CONST              18 (49)
             50 LOAD_CONST               3 (98)
             52 LOAD_CONST              16 (54)
             54 LOAD_CONST               3 (98)
```

```
 56 LOAD_CONST              14 (57)
 58 LOAD_CONST              19 (50)
 60 LOAD_CONST              11 (51)
 62 LOAD_CONST              10 (100)
 64 LOAD_CONST              20 (56)
 66 LOAD_CONST               8 (52)
 68 LOAD_CONST              19 (50)
 70 LOAD_CONST               1 (99)
 72 LOAD_CONST               1 (99)
 74 LOAD_CONST              10 (100)
 76 LOAD_CONST               9 (97)
 78 LOAD_CONST               1 (99)
 80 LOAD_CONST              18 (49)
 82 LOAD_CONST              11 (51)
 84 LOAD_CONST              16 (54)
 86 LOAD_CONST              15 (53)
 88 LOAD_CONST              20 (56)
 90 LOAD_CONST               3 (98)
 92 LOAD_CONST              11 (51)
 94 LOAD_CONST              21 (102)
 96 LOAD_CONST              22 (125)
 98 BUILD_LIST              49
100 STORE_FAST               1 (v%d)
102 LOAD_GLOBAL              0 (len)
104 LOAD_FAST                0 (v%d)
106 CALL_FUNCTION            1
108 LOAD_GLOBAL              0 (len)
110 LOAD_FAST                1 (v%d)
112 CALL_FUNCTION            1
114 COMPARE_OP               3 (!=)
116 POP_JUMP_IF_FALSE      130
118 LOAD_GLOBAL              1 (print)
120 LOAD_CONST              23 ('Length mismatch!')
122 CALL_FUNCTION            1
124 POP_TOP
126 LOAD_CONST              24 (False)
128 RETURN_VALUE
>>  130 LOAD_GLOBAL              2 (zip)
132 LOAD_FAST                0 (v%d)
134 LOAD_FAST                1 (v%d)
136 CALL_FUNCTION            2
138 GET_ITER
>>  140 FOR_ITER                36 (to 178)
142 STORE_FAST               2 (v%d)
144 LOAD_GLOBAL              3 (f1)
146 LOAD_FAST                2 (v%d)
148 LOAD_CONST              25 (1)
150 BINARY_SUBSCR
152 LOAD_GLOBAL              4 (f2)
154 LOAD_FAST                2 (v%d)
156 LOAD_CONST              26 (0)
158 BINARY_SUBSCR
160 CALL_FUNCTION            1
162 CALL_FUNCTION            2
164 LOAD_CONST              24 (False)
166 COMPARE_OP               2 (==)
168 POP_JUMP_IF_FALSE      140
170 POP_TOP
172 LOAD_CONST              24 (False)
```

```
            174 RETURN_VALUE
            176 JUMP_ABSOLUTE          140
        >>  178 LOAD_CONST              27 (True)
            180 RETURN_VALUE
None
```

就是很简单的比较字符串
把前面的一大串const打印出来即可

```
int main() {
 char flag[] = { 99,121,98,114,105,99,115,123,52,97,98,100,51,101,55,52,101,57,101,53,57,54,48,97,49,98,54,98,57
,50,51,100,56,52,50,99,99,100,97,99,49,51,54,53,56,98,51,102,125,0 };
 cout << flag;
}
```

吐个槽：题目的名字的中文意思是“通晓多种语言的人”，我还以为会套很多娃，结果就俩啊。。。