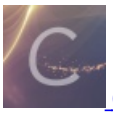
实验吧 CTF 题目之 密码学 Writeup 通关大全 - 1

原创

DarkN0te 于 2020-02-24 19:46:50 发布 862 收藏 3

分类专栏： CTF 文章标签： 安全

本文链接：https://blog.csdn.net/m0_46232048/article/details/104483771



文章目录

概述

Writeup

变异凯撒

传统知识+古典密码

what's wrong with this

try them all

概述

解除CTF也有很多年了，但是真正的将网上的题目通关刷题还是没有过的，同时感觉水平下降的太厉害，这两个月准备把网上目前公开有的CTF环境全部刷一遍，同时收集题目做为素材，为后面的培训及靶场搭建做好准备。本文是实验吧2018年7月8日前所有密码类的题目通关Writeup。

Writeup

变异凯撒

普通凯撒密码参考，https://en.wikipedia.org/wiki/Caesar_cipher。

```
加密密文：afZ_r9VYfScOeO_UL^RWUc
格式：flag{ }
```

解题过程分这几部分，首先`afZ_r9VYfScOeO_UL^RWUc`的ascii码为

```
97 102 90 95 114 57 86 89 102 83 99 79 101 79 95 85 76 94 82 87 85 99
```

`flag{ }`的前几位ascii码为

```
102 108 97 103 123
```

按位做一个比较就可以发现102-97=5，108-102=6，97-90=7，所以此题目凯撒的规律为第一个字符ascii加5，其他每个字符按位ascii自增1，所以解密代码如下。解密代码如下

```python
#!/usr/bin/env python
#-*- coding: utf-8 -*-
"""
@Author : darkN0te
@Create date : 2018-07-07
@description : 凯撒轮转密码解密
@Update date :
"""

INIT_ADD = 5

input = raw_input()
output = ""
for char in input:
    output += chr(ord(char) + INIT_ADD)
    INIT_ADD += 1
print output
```

输出

```
afZ_r9VYfScOeO_UL^RWUc
flag{Caesar_variation}
```

结束。

传统知识+古典密码

题目描述：

```
小明某一天收到一封密信，信中写了几个不同的年份
          辛卯，癸巳，丙戌，辛未，庚辰，癸酉，己卯，癸巳。
          信的背面还写有“+甲子”，请解出这段密文。
key值：CTF{XXX}
```

根据天干地支纪年法

```
1. 甲子 2.乙丑 3.丙寅 4.丁卯 5.戊辰 6.己巳 7.庚午 8.辛未 9.壬申 10.癸酉
11.甲戌 12.乙亥 13.丙子 14.丁丑 15.戊寅 16.己卯 17.庚辰 18.辛巳 19.壬午 20.癸未
21.甲申 22.乙酉 23.丙戌 24.丁亥 25.戊子 26.己丑 27.庚寅 28.辛卯 29.壬辰 30.癸巳
31.甲午 32.乙未 33.丙申 34.丁酉 35.戊戌 36.己亥 37.庚子 38.辛丑 39.壬寅 40.癸卯
41.甲辰 42.乙巳 43.丙午 44.丁未 45.戊申 46.己酉 47.庚戌 48.辛亥 49.壬子 50.癸丑
51.甲寅 52.乙卯 53.丙辰 54.丁巳 55.戊午 56.己未 57.庚申 58.辛酉 59.壬戌 60.癸亥
```

写出题中所给组合的数字编码

```
28    30    23    8    17    10    16    30
```

加上一个甲子(60)

```
88    90    93    68    77    70    76    90
```

转换成ASCII字母：
XZSDMFLZ
栅栏密码（两栏）：
XMZFSLDZ
凯撒：
SHUANGYU
最后按格式提交`CTF{SHUANGYU}`即可。

what's wrong with this

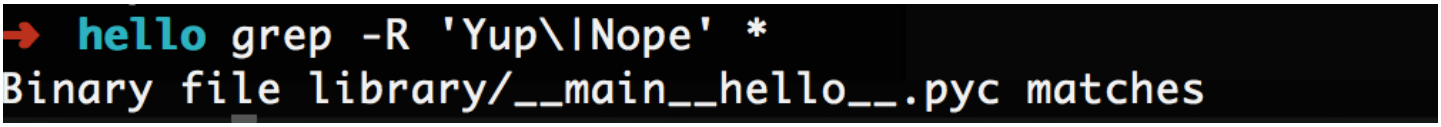
题目描述：

```
We managed to get this package of the robots servers. We managed to determine that it is some kind of compiled b
ytecode. But something is wrong with it. Our usual analysis failed - so we have to hand this over to you pros. W
e only know this: The program takes one parameter and it responds with "Yup" if you have found the secret code,
with "Nope" else. We expect it should be obvious how to execute it.

解题链接： http://ctf5.shiyanbar.com/crypto/What-s-wrong-with-this/hello.tar.gz

中文题干：我们设法获得了这个机器人服务器包。 我们设法确定它是某种编译的字节码。 但它有些问题。 我们通常的分析失败了 - 所以我们不得不把它交给你们。 我们只知道这个：程序接受一个参数，如果你找到了密码，它会以“Yup”响应，而“Nope”则是。 我们期望它应该是如何执行它的。
```

网站给出了一个答案pdf，请查看
http://hebin.me/wp-content/uploads/2017/09/2017090715264378.pdf
解压题目给出的文件`hello.tar.gz`，我们知道了一些程序特征会打印`Yup`和`Nope`，然后使用命令`grep -R 'Yup\|Nope'`进行搜索，找到匹配文件。



使用`uncompyle`进行反编译不可以，使用`Decompyle++`进行反编译。安装过程是这样。

```
git clone https://github.com/zrax/pycdc.git
cd pycdc
cmake .
make
make install
pycdc __main__hello__.pyc
```

我们用过pycdc反编译出`__main__hello__.pyc`的源码

```
# Source Generated with Decompyle++
# File: __main__hello__.pyc (Python 2.7)

import sys
import dis
import multiprocessing
import UserList

def encrypt_string(s):
Unsupported opcode: <255>
    new_str = []
# WARNING: Decompyle incomplete


def rot_chr(c, amount):
    None = chr(((ord(c) + 33) % amount) / 94 % 33)

SECRET = 'w*0;CNU[\\gwPWk}3:PWk"#&:ABu/:Hi,M'
if encrypt_string(sys.argv - 1) == SECRET:
    print
    print >>'Yup'
else:
    print
    print >>'Nope'
None = None
```

可以看到，有一部分字节码没有被反编译出来，这是由于一部分字节码没有被识别造成的，使用`pycdas`查看一下信息。

```
➜  what's wrong with this ~/Safe/03_tools/pycdc/pycdas __main__hello__.pyc
__main__hello__.pyc (Python 2.7)
[Code]
    File Name: chall.py
    Object Name:
    Arg Count: 0
    Locals: 0
    Stack Size: 3
    Flags: 0x00000040 (CO_NOFREE)
    [Names]
        'sys'
        'hashlib'
        'sha256'
        'dis'
        'multiprocessing'
        'UserList'
        'encrypt_string'
        'rot_chr'
        'SECRET'
        'argv'
    [Var Names]
    [Free Vars]
    [Cell Vars]
    [Constants]
        -1
        None
        (
            'sha256'
        )
        [Code]
            File Name: chall.py
            Object Name: encrypt_string
```

```
Object Name: encrypt_string
Arg Count: 1
Locals: 4
Stack Size: 8
Flags: 0x00000043 (CO_OPTIMIZED | CO_NEWLOCALS | CO_NOFREE)
[Names]
    'enumerate'
    'append'
    'rot_chr'
    'ord'
    'join'
[Var Names]
    's'
    'new_str'
    'index'
    'c'
[Free Vars]
[Cell Vars]
[Constants]
    None
    0
    10
    1
    ''
[Disassembly]
    0       BUILD_LIST              0
    3       STORE_FAST              1: new_str
    6       SETUP_LOOP              99 (to 108)
    9       LOAD_GLOBAL             0: enumerate
    12      LOAD_FAST               0: s
    15      CALL_FUNCTION           1
    18
    19      FOR_ITER                85 (to 107)
    22      UNPACK_SEQUENCE         2
    25      STORE_FAST              2: index
    28      STORE_FAST              3: c
    31      LOAD_FAST               2: index
    34      LOAD_CONST              1: 0
    37      COMPARE_OP              2 (==)
    40      POP_JUMP_IF_FALSE       68
    43      LOAD_FAST               1: new_str
    46      LOAD_ATTR               1: append
    49      LOAD_GLOBAL             2: rot_chr
    52      LOAD_FAST               3: c
    55      LOAD_CONST              2: 10
    58      CALL_FUNCTION           2
    61      CALL_FUNCTION           1
    64      ROT_TWO
    65      JUMP_ABSOLUTE           19
    68      LOAD_FAST               1: new_str
    71      LOAD_ATTR               1: append
    74      LOAD_GLOBAL             2: rot_chr
    77      LOAD_FAST               3: c
    80      LOAD_GLOBAL             3: ord
    83      LOAD_FAST               1: new_str
    86      LOAD_FAST               2: index
    89      LOAD_CONST              3: 1
    92      BINARY_ADD
    93      BINARY_SUBTRACT
    94      CALL_FUNCTION           1
```

```
            97      CALL_FUNCTION           2
            100     CALL_FUNCTION           1
            103     ROT_TWO
            104     JUMP_ABSOLUTE           19
            107     END_FINALLY
            108     LOAD_CONST              4: ''
            111     LOAD_ATTR               4: join
            114     LOAD_FAST               1: new_str
            117     CALL_FUNCTION           1
            120     IMPORT_STAR
    [Code]
        File Name: chall.py
        Object Name: rot_chr
        Arg Count: 2
        Locals: 2
        Stack Size: 3
        Flags: 0x00000043 (CO_OPTIMIZED | CO_NEWLOCALS | CO_NOFREE)
        [Names]
            'chr'
            'ord'
        [Var Names]
            'c'
            'amount'
        [Free Vars]
        [Cell Vars]
        [Constants]
            None
            33
            94
        [Disassembly]
            0       LOAD_GLOBAL             0: chr
            3       LOAD_GLOBAL             1: ord
            6       LOAD_FAST               0: c
            9       CALL_FUNCTION           1
            12      LOAD_CONST              1: 33
            15      BINARY_ADD
            16      LOAD_FAST               1: amount
            19      BINARY_MODULO
            20      LOAD_CONST              2: 94
            23      BINARY_DIVIDE
            24      LOAD_CONST              1: 33
            27      BINARY_MODULO
            28      CALL_FUNCTION           1
            31      IMPORT_STAR
    'w*0;CNU[\\gwPWk}3:PWk"#&amp;:ABu/:Hi,M'
    1
    'Yup'
    'Nope'
[Disassembly]
    0       LOAD_CONST              0: -1
    3       LOAD_CONST              1: None
    6       IMPORT_NAME             0: sys
    9       STORE_NAME              0: sys
    12      LOAD_CONST              0: -1
    15      LOAD_CONST              2: ('sha256',)
    18      IMPORT_NAME             1: hashlib
    21      IMPORT_FROM             2: sha256
    24      STORE_NAME              2: sha256
    27      ROT_TWO
    28      LOAD_CONST              0: -1
```

```
28      LOAD_CONST              0: -1
31      LOAD_CONST              1: None
34      IMPORT_NAME             3: dis
37      STORE_NAME              3: dis
40      LOAD_CONST              0: -1
43      LOAD_CONST              1: None
46      IMPORT_NAME             4: multiprocessing
49      STORE_NAME              4: multiprocessing
52      LOAD_CONST              0: -1
55      LOAD_CONST              1: None
58      IMPORT_NAME             5: UserList
61      STORE_NAME              5: UserList
64      LOAD_CONST              3: <CODE> encrypt_string
67      MAKE_FUNCTION           0
70      STORE_NAME              6: encrypt_string
73      LOAD_CONST              4: <CODE> rot_chr
76      MAKE_FUNCTION           0
79      STORE_NAME              7: rot_chr
82      LOAD_CONST              5: 'w*0;CNU[\\gwPWk}3:PWk"#&amp;:ABu/:Hi,M'
85      STORE_NAME              8: SECRET
88      LOAD_NAME               6: encrypt_string
91      LOAD_NAME               0: sys
94      LOAD_ATTR               9: argv
97      LOAD_CONST              6: 1
100     BINARY_SUBTRACT
101     CALL_FUNCTION           1
104     LOAD_NAME               8: SECRET
107     COMPARE_OP              2 (==)
110     POP_JUMP_IF_FALSE       121
113     LOAD_CONST              7: 'Yup'
116     PRINT_NEWLINE
117     PRINT_ITEM_TO
118     JUMP_FORWARD            5 (to 126)
121     LOAD_CONST              8: 'Nope'
124     PRINT_NEWLINE
125     PRINT_ITEM_TO
126     LOAD_CONST              1: None
129     IMPORT_STAR
```

在研究一下如何修复的

修复后反编译得到的结果为

```
# Source Generated with Decompyle ++
# File: __main__hello__.pyc (Python 2.7)
import sys
from hashlib import sha256
import dis
import multiprocessing
import UserList

def encrypt_string(s):
    new_str = []
    for (index , c) in enumerate(s):
        if index == 0:
            new_str.append(rot_chr(c, 10))
            continue
        new_str.append(rot_chr(c, ord(new_str[index - 1])))
    return ''.join(new_str)

def rot_chr(c, amount):
    return chr(((ord(c) - 33) + amount) % 94 + 33)

SECRET = 'w*0;CNU[\\gwPWk}3:PWk"#&amp;:ABu/:Hi,M'
if encrypt_string(sys.argv[1]) == SECRET:
    print 'Yup'
else:
    print 'Nope'
```

写出解密代码：

```
SECRET = 'w*0;CNU[\\gwPWk}3:PWk"#&amp;:ABu/:Hi,M'
def decrypt_string(s):
    new_str = []
    for (index , c) in enumerate(s):
        if index == 0:
            new_str.append(rot_chr(c, 10))
            continue
        new_str.append(rot_chr(c, ord(s[index - 1])))
    return ''.join(new_str)

def rot_chr(c, amount):
    return chr(((ord(c) - 33) - amount) % 94 + 33)

print decrypt_string(SECRET)
# output : modified_in7erpreters_are_3vil!!!
```

try them all

题目描述：

```
You have found a passwd file containing salted passwords. An unprotected configuration file has revealed a salt of 5948. The hashed password for the 'admin' user appears to be 81bdf501ef206ae7d3b92070196f7e98, try to brute force this password.

您找到了一个有盐的密码表。已知密码的明文结尾为5948，密码表中哈希值为81bdf501ef206ae7d3b92070196f7e98，尝试暴力破解此密码。
```

原题干为英文，但是感觉和题目本身的意思不一样，写了一段和原题目意思一致的中文。题目的意思就是一个简单的爆破md5。难点是不知道到底这个明文到底有多少位，都包含什么字符。

这里直接使用在cmd5上查到的结果`sniper5948`。

脚本

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
"""
@Author : darkN0te
@Create date : 2018-07-07
@description : md5爆破 单进程
@Update date :
"""



import string
import hashlib

endOutput = "5948"

file=open("output.txt","a")
md5input=raw_input("请输入md5：\n")
md5input=md5input.lower()

# apt=string.printable[:-6]
apt=string.letters

def dfs(s,num):
    m=hashlib.md5()
    print s + endOutput
    m.update(s + endOutput)
    md5temp=m.hexdigest()
    if md5temp==md5input:
        file.write("md5是："+md5input+"   明文是："+s+"\n")
        file.close()
        exit(-1)
    if len(s)==num:
        return
    for i in apt:
        dfs(s+i,num)

myinput=7               #生成字符的位数
for j in range(1,myinput):
    dfs("",j)
```