CTF-misc-压缩包解题思路

原创

OceanSec 于 2020-11-01 13:52:57 发布 12944 收藏 16

分类专栏： # CTF 文章标签： linux 运维 服务器

本文链接：https://blog.csdn.net/q20010619/article/details/109425270

CTF 专栏收录该内容

66 篇文章 29 订阅

订阅专栏

misc-压缩包

文章目录

文件头

格式	文件头（16进制）	文件头（ascii）
zip	504B0304	PK
rar	52617221	Rar!
7z	377ABCAF271C	7z¼¯'

注意：wordx文件其实是一种zip

关于文件两个简单命令

- file命令，根据文件头来识别文件类型
- strings，输出文件中的可打印字符串

可以发现一些提示信息或特殊编码信息

strings filename

配合-o 参数获取所有ascii 字符偏移，即字符串在文件中的位置

```
root@kali:~/桌面/crc# strings 1.txt |grep "flag"
```

看属性

命令分离文件

binwalk、foremost

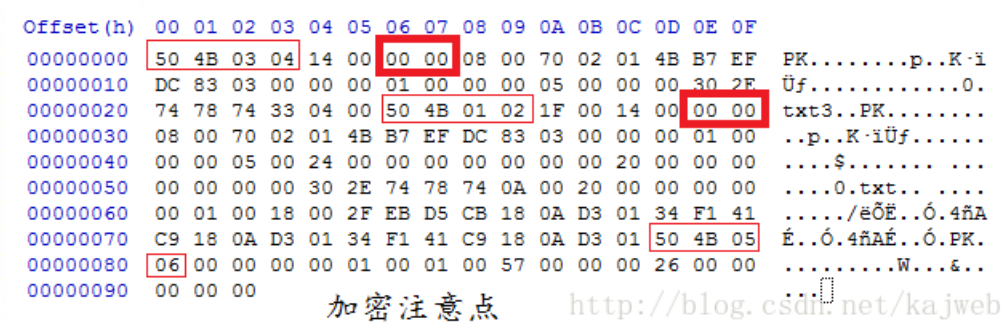
暴力破解

最简单、最直接的攻击方式，适合密码较为简单或是已知密码的格式或者范围时使用

工具：apchpr(windows)、fcrackzip(linux)

zip伪加密

原理：一个zip文件由三部分组成：压缩源文件数据区+压缩源文件目录区+压缩源文件目录结束标志。



全局方式位标记的四个数字中只有第二个数字对其有影响，其它的不管为何值，都不影响它的加密属性！

第二个数字为奇数时 –>加密

第二个数字为偶数时 –>未加密

无加密

压缩源文件数据区的全局加密应当为`00 00`（504B0304两个bytes之后）

且压缩源文件目录区的全局方式位标记应当为`00 00`（504B0304四个bytes之后）

假加密

压缩源文件数据区的全局加密应当为`00 00`

且压缩源文件目录区的全局方式位标记应当为`09 00`

真加密

压缩源文件数据区的全局加密应当为`09 00`

且压缩源文件目录区的全局方式位标记应当为`09 00`

修复方法：

1. 修改通用标志位
2. winrar修复
3. binwalk -e 命令可以无视伪加密，从压缩包中提取文件，macos可以直接打开伪加密zip压缩包
4. ZipCenOp.jar(win)

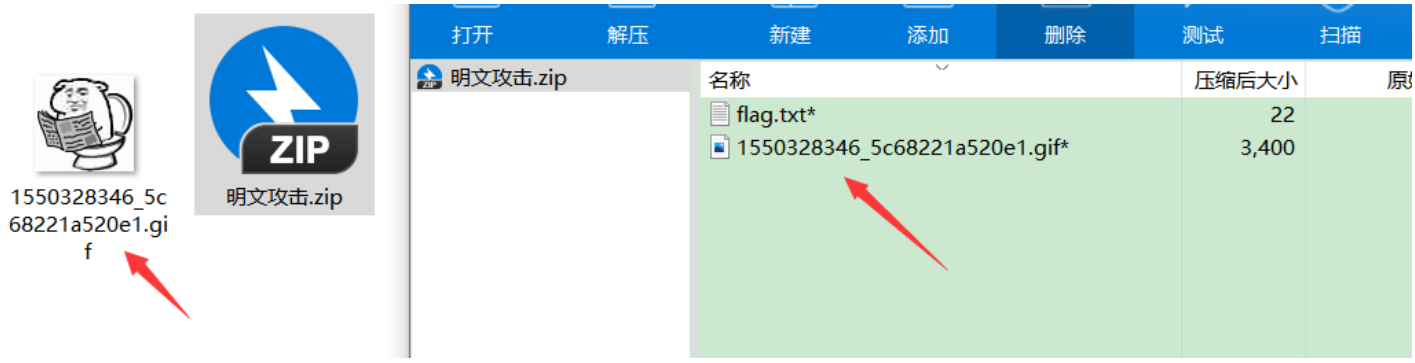
找到所在文件夹，在地址栏输入cmd

```
java -jar ZipCenOp.jar r 文件名
```

已知明文攻击

我们为ZIP压缩文件所设定的密码,先被转换成了3个4字节的key,再用这3个key加密所有文件。如果我们能通过某种方式拿到压缩包中的一个文件,然后以同样的方式压缩,选择不去爆破密码。这种攻击方式便是已知明文攻击。

题目特征：有一个加密压缩包一个未加密压缩包（或者是一个文件）这个文件是加密压缩包的一部分



注意使用的压缩软件和压缩格式，压缩完对比crc32校验码

crc32碰撞

CRC校验是在数据存储和数据通讯领域，为了保证数据的正确，就不得不采用检错的手段。在诸多检错手段中，CRC是最著名的一种。CRC的全称是循环冗余校验。

总之每个文件都有唯一的CRC32值，即便数据中一个bit发生变化，也会导致CRC32值不同。若是知道一段数据的长度和CRC32值，便可穷举数据，与其CRC32对照，以此达到暴力猜解的目的。但通常只适用于较小文本文件。

zip文件中crc32为未加密文件的校验码

比如这里有一个加密的压缩包，直接双击就可以看见其中信息，而且我知道其中全是数字，便可使用脚本爆破。

题目特征：文件本身内容很小，密码很复杂

crc.py(需要linux环境)

```
python2 crc.py 2.zip
```

crc.py

```
#!/usr/bin/env python3
import sys
import os
import string
import collections

import argparse
parser = argparse.ArgumentParser()
parser.add_argument('file', nargs='*')
parser.add_argument('--hex', action='append')
parser.add_argument('--dec', action='append')
parser.add_argument('--limit', type=int)
parser.add_argument('--compiler', default='gcc')
```

```
parser.add_argument('--compiler', default='g++')
parser.add_argument('--alphabet', type=os.fsencode, default=string.printable.encode())
args = parser.parse_args()

targets = collections.OrderedDict()
limit = 0
crcs = []

if args.limit:
    limit = max(limit, args.limit)
if args.hex or args.dec:
    if not args.limit:
        parser.error('Limit of length not specified')

if args.hex:
    for s in args.hex:
        crc = int(s, 16)
        targets[s] = crc
        for l in range(args.limit + 1):
            crcs += [( crc, l )]
if args.dec:
    for s in args.dec:
        crc = int(s)
        targets[s] = crc
        for l in range(args.limit + 1):
            crcs += [( crc, l )]

if args.file:
    print('reading zip files...', file=sys.stderr)
    import zipfile
    for zipname in args.file:
        fh = zipfile.ZipFile(zipname)
        for info in fh.infolist():
            targets['%s / %s' % ( zipname, info.filename )] = ( info.CRC, info.file_size )
            crcs += [( info.CRC, info.file_size )]
            limit = max(limit, info.file_size)
            print('file found: %s / %s: crc = 0x%08x, size = %d' % (zipname, info.filename, info.CRC, info.file_
size), file=sys.stderr)

if not crcs:
    parser.error('No CRCs given')

# compiling c++ in python script is the easy way to have the both a good interface and better speed
code = ''
code += r'''
#include <cstdio>
#include <vector>
#include <array>
#include <string>
#include <set>
#include <cstdint>
#include <cctype>
#define repeat(i,n) for (int i = 0; (i) < (n); ++(i))
using namespace std;

uint32_t crc_table[256];
void make_crc_table() {
    repeat (i, 256) {
        uint32_t c = i;
        repeat (j, 8) {
```

```
            c = (c & 1) ? (0xedb88320 ^ (c >> 1)) : (c >> 1);
        }
        crc_table[i] = c;
    }
}
const uint32_t initial_crc32 = 0xffffffff;
uint32_t next_crc32(uint32_t c, char b) {
    return crc_table[(c ^ b) & 0xff] ^ (c >> 8);
}
const uint32_t mask_crc32 = 0xffffffff;

const char alphabet[] = { ''' + ', '.join(map(str, args.alphabet)) + r''' };
const int limit = ''' + str(limit) + r''';

array<set<uint32_t>, limit+1> crcs;
string stk;
void dfs(uint32_t crc) {
    if (crcs[stk.length()].count(crc ^ mask_crc32)) {
        fprintf(stderr, "crc found: 0x%08x: \"", crc ^ mask_crc32);
        for (char c : stk) fprintf(stderr, isprint(c) && (c != '\\') ? "%c" : "\\x%02x", c);
        fprintf(stderr, "\"\n");
        printf("%08x ", crc ^ mask_crc32);
        for (char c : stk) printf(" %02x", c);
        printf("\n");
    }
    if (stk.length() < limit) {
        for (char c : alphabet) {
            stk.push_back(c);
            dfs(next_crc32(crc, c));
            stk.pop_back();
        }
    }
}

int main() {
'''
for crc, size in crcs:
    code += '    crcs[' + str(size) + '].insert(' + hex(crc) + ');\n'
code += r'''
    make_crc_table();
    dfs(initial_crc32);
    return 0;
}
'''

import tempfile
import subprocess
with tempfile.TemporaryDirectory() as tmpdir:
    cppname = os.path.join(tmpdir, 'a.cpp')
    with open(cppname, 'w') as fh:
        fh.write(code)
    binname = os.path.join(tmpdir, 'a.out')
    print('compiling...', file=sys.stderr)
    p = subprocess.check_call([args.compiler, '-std=c++11', '-O3', '-o', binname, cppname])
    print('searching...', file=sys.stderr)
    p = subprocess.Popen([binname], stdout=subprocess.PIPE)
    output, _ = p.communicate()

print('done', file=sys.stderr)
```

```
print(file=sys.stderr)
result = collections.defaultdict(list)
for line in output.decode().strip().split('\n'):
    crc, *val = map(lambda x: int(x, 16), line.split())
    result[( crc, len(val) )] += [ bytes(val) ]
for key, crc in targets.items():
    for s in result[crc]:
        print('%s : %s' % (key, repr(s)[1:]))
```

多个压缩文件合并

cat 文件名(按需) > 保存文件名

docx文件

docx文件就是包含xml文件的zip压缩包

可能隐藏文件、信息在压缩包里面，word直接打开是看不见的