

[\[TOC\]](#)

概述

本片分析文章通过一道看雪CTF题讲述作者的整个分析流程，学习WebAssemble、Z3库、IDC脚本、多元线性方程等内容

分析流程

安装应用后，出现一个输入框和一个按钮

android

jadx反编译apk后先查看manifest清单文件的注册组件,只有一个入口活动类，进入查看

```
<application android:theme="@style/AppTheme" android:label="@string/app_name" android:icon="@mipmap/ic_launcher" android:allowBackup="false" android:usesCleartextTraffic="true">
    <activity android:name="com.example.assemgogogo.MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

初现迷雾

第一眼：看到Congratulations，我们的目标是打印出这里的字符串，也就是点击按钮后调用本地方法check_key返回值为1即可

再仔细看看：这里有个WebView组件，这个组件表示有访问网络的操作，但是手机界面并没有看页面，我们去布局文件中看看，我们只需要看一个属性android:visibility

```
public String u = gogogoJNI.sayHello();
    static {
        System.loadLibrary("gogogo");
    }

    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView((int) R.layout.activity_main);
        this.eText1 = (EditText) findViewById(R.id.editText);
        this.txView1 = (TextView) findViewById(R.id.textView);
        ((WebView) findViewById(R.id.text1View)).loadUrl(this.u);
        ((WebView) findViewById(R.id.text1View)).getSettings().setJavaScriptEnabled(true);
        this.button1 = (Button) findViewById(R.id.button);
        this.button1.setOnClickListener(new OnClickListener() {
            public void onClick(View view) {
                if (gogogoJNI.check_key(MainActivity.this.eText1.getText().toString()) == 1) {
                    MainActivity.this.txView1.setText("Congratulations!");
                } else {
                    MainActivity.this.txView1.setText("Not Correct!");
                }
            }
        });
    }

    <EditText android:id="@+id/editText" android:visibility="invisible" android:layout_width="wrap_content" android:layout_height="wrap_content">
    <Button android:id="@+id/button" android:visibility="invisible" android:layout_width="wrap_content" android:layout_height="wrap_content">
    <TextView android:id="@+id/textView" android:visibility="invisible" android:layout_width="wrap_content" android:layout_height="wrap_content">
    <WebView android:id="@+id/text1View" android:visibility="visible" android:layout_width="390dp" android:layout_height="733dp">
```

探索URL

打开lib文件夹，出现四个abi架构对应的so文件，基本现在手机的芯片都是支持的，这里ARM64在ida6.8不能使用F5大法，所以我们就分析armeabi-v7这个就行了

我们可以看看第一步我们排除的check_key方法，这里逻辑是输出的32位数都为1即可返回1，实际尝试是错误的，混淆视听


```

7
3  v3 = (*a1)->GetStringUTFChars(a1, (jstring)0, 0);
7  if ( v3 )
8  {
1   v4 = v3;
2   ((void (__cdecl *)(JNIEnv *)))(*a1)->ReleaseStringUTFChars)(a1);
3   srand(0x32u);
4   v5 = "d584a68d4e213d88w511v48e61g8d6e8";
5   v6 = 0;
6   while ( v4[v6] == (rand() % 128 != *v5) )
7   {
8   ++v6;
9   ++v5;
10  if ( v6 > 0x1F )
11  {
12  close(sock_fd_g);
13  return 1;
14  }
15  }
16 }
17 return 0;
18 }

```

在导出表中找到sayhello方法，要使用F5大法先右键将这个区域代码创建为函数。接着讲这个字节数组异或计算即可得到URL地址。下面写了一个简短的idc脚本获取到URL地址：
<http://127.0.0.1:8000>

分析到这里，虽然我们探索URL已经完成，但是却没有看见服务端处理的函数，这个才是我们访问URL的时候，处理我们访问请求的函数



```

IDA View-A | Pseudocode-B | Pseudocode-A | Hex View-1 | Structures
1 int __fastcall Java_com_example_assemgogogo_gogogoJNI_sayHello(JNIEnv *env)
2 {
3     int v1; // r11@0
4     JNIEnv *mEnv; // r4@1
5     int i; // r0@1
6     int result; // r0@4
7     _BYTE byteArr[21]; // [sp+0h] [bp-98h]@1
8     char v6; // [sp+15h] [bp-83h]@4
9     int v7; // [sp+84h] [bp-14h]@1
10    int v8; // [sp+88h] [bp-10h]@1
11
12    v8 = v1;
13    mEnv = env;
14    v7 = _stack_chk_guard;
15    _asahi_memcpy8(byteArr, 128);
16    for ( i = 0; i != 21; ++i )
17        byteArr[i] = byte_2028[i] ^ 0x66;
18    v6 = 0;
19    result = ((int (__fastcall *)(JNIEnv *, _BYTE *))(*mEnv)->NewStringUTF)(mEnv, byteArr);
20    if ( _stack_chk_guard != v7 )
21        _stack_chk_fail(result);
22    return result;
23 }

```

```
#include <idc.idc>
```

```

static main()
{
    auto addr = 0x2d28;
    auto i;
    for(i=0; i !=21; ++i)
    {
        Message("%c", Byte(addr+i)^0x66);
    }
}

```

探索服务端处理函数

从java层分析的逻辑中并没有服务端的线索，而so层也只有初始化的JNIonload、init节还没有探索，这是我们接下来的目标

第一步排除init节，so加载后首先执行的节代码，这里可以看出没有这个节，所以排除，那么就直接分析JNIOnload方法，java中调用loadlibrary的时候调用的方法

Choose segment to jump																	
Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	es	ss	ds	fs	
.plt	000007C0	00000920	R	.	X	.	L	para	0001	public	CODE	32	FFFFFFFF	FFFFFFFF	0009	FFFFFFFF	
.text	00000920	00001078	R	.	X	.	L	para	0002	public	CODE	32	FFFFFFFF	FFFFFFFF	0009	FFFFFFFF	
.rodata	00001080	0000111E	R	.	.	.	L	para	0003	public	CONST	32	FFFFFFFF	FFFFFFFF	0009	FFFFFFFF	
.eh_frame	00001120	00001448	R	.	.	.	L	dword	0004	public	CONST	32	FFFFFFFF	FFFFFFFF	0009	FFFFFFFF	
.eh_frame_hdr	00001448	0000148C	R	.	.	.	L	dword	0005	public	CONST	32	FFFFFFFF	FFFFFFFF	0009	FFFFFFFF	
.fini_array	00002E8C	00002E94	R	W	.	.	L	dword	0006	public	DATA	32	FFFFFFFF	FFFFFFFF	0009	FFFFFFFF	
.got	00002F94	00002FA0	R	W	.	.	L	dword	0007	public	DATA	32	FFFFFFFF	FFFFFFFF	0009	FFFFFFFF	
.got.plt	00002FA0	00003000	R	W	.	.	L	dword	0008	public	DATA	32	FFFFFFFF	FFFFFFFF	0009	FFFFFFFF	
.data	00003000	0000B5F7	R	W	.	.	L	dword	0009	public	DATA	32	FFFFFFFF	FFFFFFFF	0009	FFFFFFFF	
.bss	0000B5F8	0000B5FC	R	W	.	.	L	dword	000A	public	BSS	32	FFFFFFFF	FFFFFFFF	0009	FFFFFFFF	
extern	0000B5FC	0000B64C	?	?	?	.	L	para	000B	public		32	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	
abc	0000B67C	0000B688	?	?	?	.	L	para	000C	public		32	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF	

JNI_Onload分析：往进call两层，最终调用下面这个函数

```
int (__cdecl *inti_proc(void))(int)
{
    return inti_proc();
}
```

这个函数

一开始就对data段中一块大小为34291的数据进行异或0x67解密，接着创建线程用socket链接将刚才解密的内容构造称HTTP的响应数据包，一旦有socket链接连接过来就发

逻辑分析清除，下面我们针对细节进行解决

```
16|
17|  ByteArray = &mm0;
18|  v18 = _stack_chk_guard;
19|  v1 = 34291;
20|  v14 = 1;
21|  while ( v1 )           解密
22|  {
23|      --v1;
24|      *ByteArray++ ^= 0x67u;
25|  }
26|  __asm { VM0V.I32      q0, #0 }
27|  __R0 = (char *)&req.ai_protocol;
28|  i = 1;
29|  __asm { UST1.32      {D16-D17}, [R0] }
30|  req.ai_family = 0;
31|  req.ai_flags = 1;
32|  req.ai_socktype = 1;
33|  req.ai_next = 0;      IP地址、端口号
34|  if ( !getaddrinfo(0, "8000", &req, &pai) )
35|  {
```

```

v1 = a1;
addr_len = 128;
while ( 1 )
{
    do
    {
        v2 = accept(*v1, &addr, &addr_len);
        while ( v2 == -1 );
        v3 = &addr.sa_data[6];
        if ( addr.sa_family == 2 )
            v3 = &addr.sa_data[2];
        inet_ntop(addr.sa_family, v3, &buf, 0x2Eu);
        v4 = strlen((const char *)&mm0);
        snprintf(
            &v6,
            0xC350u,
            "HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=utf-8\r\nContent-Length: %ld\r\n\r\n%s",
            v4,
            &mm0);
        v5 = strlen(&v6);
        if ( send(v2, &v6, v5, 0) == -1 )
            perror("send");
        close(v2);
    }
}

```



针对需要解密的字节流，通过idc脚本进行处理，解密后的数据是html页面，使用到了WebAssembly技术，web汇编的灵魂就是将其其他语言如C汇编成前端可以解释的语言

```
#include <idc.idc>
```

```

static main()
{
    auto addr = 0x4004;
    auto i = 34291;
    while(i)
    {
        --i;
        Message("%c", Byte(addr++)^0x67);
    }
}

```

分析这里的逻辑得知，我们需要让输入内容为32位并且check_key()函数返回结果为1，即可完成这道题

```

<!DOCTYPE html>
<html><head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<meta charset="utf-8">
<style>
    body {
        background-color: rgb(255, 255, 255);
    }
</style>
</head>
<script>

```

```
var instance;
```

```

WebAssembly.compile(new Uint8Array(`
00 61 73 6D 01 00 00 00 01 1B 05 60 00 00 60 04
7F 7F 7F 7F 01 7F 60 02 7F 7F 01 7F 60 01 7F 01
.....
66 6C 61 67 0A 12 73 65 74 5F 69 6E 70 75 74 5F
66 6C 61 67 5F 6C 65 6E 0B 09 63 68 65 63 6B 5F
6B 65 79 0C 03 78 78 78
`.trim().split(/[\\s\\r\\n]+/g).map(str => parseInt(str, 16))
)).then(module => {
    new WebAssembly.instantiate(module).then(results => {
        instance = results;
    }).catch(console.error);})

```

```
function check_flag(){
    var value = document.getElementById("key_value").value;
    if(value.length != 32)
    {
        document.getElementById("tips").innerHTML = "Not Correct!";
        return;
    }
    instance.exports.set_input_flag_len(value.length);
    for(var ii=0;ii<value.length;ii++){
        instance.exports.set_input_flag(value[ii].charCodeAt(),ii);
    }
    var ret = instance.exports.check_key();

    if (ret == 1){
        document.getElementById("tips").innerHTML = "Congratulations!"
    }
    else{
        document.getElementById("tips").innerHTML = "Not Correct!"
    }
}
</script>
<body>
    <div>Key: <input id="key_value" type="text" name="key" style="width:60%" ;="" value=""> <input type="submit" value="check" o
    <div> <label id="tips"></label></div>

</body></html>
```

下面我们进入web汇编来探索内部实现逻辑

webassemble

我们在这部分探索的目标就是先用16进制内容构成对应的wasm二进制文件，然后将wasm二进制文件转成c，接着生成ELF文件，用IDA进行分析。

先生成data.bin二进制文件

```
import array, struct

hexstring = "\x00\x61\x73\x6D\x01\x00\x00\x00\x01\x1B\x05\x60\x00\x00\x60\x04\x7F\x7F\x7F\x7F\x01\x7F\x60\x02\x7F\x7F\x01\x7F\x
.....
\x6C\x61\x67\x0A\x12\x73\x65\x74\x5F\x69\x6E\x70\x75\x74\x5F\x66\x6C\x61\x67\x5F\x6C\x65\x6E\x0B\x09\x63\x68\x65\x63\x6B\x5F\x
\x6C\x61\x67\x0A\x12\x73\x65\x74\x5F\x69\x6E\x70\x75\x74\x5F\x66\x6C\x61\x67\x5F\x6C\x65\x6E\x0B\x09\x63\x68\x65\x63\x6B\x5F\x

f = open('c:\\Users\\xxx\\Desktop\\data.bin', 'wb')
f.write(hexstring)
f.close()
```

接着用wasm2c.exe生成c文件

```
wasm2c.exe data.bin -o test.c
```

直接gcc wasm.c会报错，因为很多wasm的函数没有具体的实现。所以只编译就好了

```
gcc -c test.c -o test.o
```

用IDA打开.o文件

首先JS中调用将输入的字符长度保存到内存中，接着将输入的字符也保存到内存0x400处

```
1 __int64 __fastcall set_input_flag_len(unsigned int a1, __int64 a2, __int64 a3)
2 {
3     if ( ++wasm_rt_call_stack_depth > 500u )
4         wasm_rt_trap(7LL, a2, a3);
5     i32_store((__int64)&memory, 1224LL, a1);    // 将输入的字符串长度写到内存1224处
6     --wasm_rt_call_stack_depth;
7     return 0LL;
8 }
```

```

1 __int64 __fastcall set_input_flag(unsigned int onechar, __int64 char_index, __int64 a3)
2 {
3     if ( ++wasm_rt_call_stack_depth > 500u )
4         wasm_rt_trap(7LL, char_index, a3);
5     i32_store8((__int64)&memory, (unsigned int)(char_index + 1024), onechar); // 将输入的字符串保存到内存1024(0x400)出
6     --wasm_rt_call_stack_depth;
7     return 0LL;
8 }

```

接着就是主要的check_key函数，最终目标是xxx函数返回结果为1，即可完成逆向工作

这里前8个o函数对我们输入的32内容依次进行了处理，我们具体分析一下

```

7     unsigned int v2, // 3100_4020
8
9     if ( ++wasm_rt_call_stack_depth > 0x1F4u )
10         wasm_rt_trap(7LL, a2, a3);
11     o(1024u, 1025LL, 1026LL, 1027u); // 匹配算法，如果条件失败，就对内存中保存的四个输出参数重新赋值
12     oo(0x404u, 1029LL, 1030LL, 0x407u);
13     ooo(0x408u, 1033LL, 1034LL, 0x40Bu);
14     oooo(0x40Cu, 1037LL, 1038LL, 0x40Fu);
15     ooooo(0x410u, 1041LL, 1042LL, 0x413u);
16     oooooo(0x414u, 1045LL, 1046LL, 0x417u);
17     ooooooo(0x418u, 1049LL, 1050LL, 0x41Bu);
18     v3 = oooooooo(1052u, 1053LL, 1054LL, 1055u);
19     v5 = xxx(1052LL, 1053LL, v4); // 32元线性方程
20     --wasm_rt_call_stack_depth;
21     return v5;
22 }

```

一重加密

经过简单分析，这里其实是对输入内容进行了异或计算，然后将结果替换内存中原来的数据。下面图中的条件是肯定满足的，因为我们输入的内容在33到127之间，最小的3

```

69     while ( v5 );
70 }
71 if ( char_param4 + char_param2 + (char)pparam1 + (char)ppparam3 == -1 )
72 {
73     v19 = v21
74         + v20
75         + ((char)pparam1 != -char_param4 || char_param4 != -45 || ppparm3 != 0 || (pparam1 & char_param4) != 0)
76         + v14
77         + 1;
78 }
79 else
80 {
81     i32_store8((__int64)&memory, param1, pparam1 ^ 24);
82     v6 = i32_load8_u((__int64)&memory, (unsigned int)param2);
83     i32_store8((__int64)&memory, (unsigned int)param2, v6 ^ 9);
84     v7 = i32_load8_u((__int64)&memory, pparam3);
85     i32_store8((__int64)&memory, pparam3, v7 ^ 3);
86     v8 = i32_load8_u((__int64)&memory, pparam4);
87     i32_store8((__int64)&memory, pparam4, v8 ^ 107);
88     v19 = (unsigned __int64)i32_load8_s((__int64)&memory, param1) ^ 112;
89 }
90 --wasm_rt_call_stack_depth;
91 return (unsigned int)v19;
92 }

```

32元线性方程组

接着我们分析xxx函数，我们的目标也是满足xxx函数返回值为1

从内存中读一重加密后的输入内容读取到变量中，可以看到顺序做过修改

```

7     wasm_rt_trap(7LL, params0, 40);
8     v37 = 0;
9     v36 = i32_load8_s((__int64)&memory, 1054LL);
10    v35 = i32_load8_s((__int64)&memory, 1055LL);
11    v34 = i32_load8_s((__int64)&memory, 1025LL);
12    v33 = i32_load8_s((__int64)&memory, 1024LL);
13    v32 = i32_load8_s((__int64)&memory, 1026LL);
14    v31 = i32_load8_s((__int64)&memory, 1027LL);
15    v30 = i32_load8_s((__int64)&memory, 1028LL);
16    v29 = i32_load8_s((__int64)&memory, 1029LL);
17    v28 = i32_load8_s((__int64)&memory, 1030LL);
18    v27 = i32_load8_s((__int64)&memory, 1031LL);
19    v26 = i32_load8_s((__int64)&memory, 1032LL);
20    v25 = i32_load8_s((__int64)&memory, 1033LL);
21    v24 = i32_load8_s((__int64)&memory, 1034LL);
22    v23 = i32_load8_s((__int64)&memory, 1035LL);
23    v22 = i32_load8_s((__int64)&memory, 1036LL);
24    v21 = i32_load8_s((__int64)&memory, 1037LL);
25    v20 = i32_load8_s((__int64)&memory, 1038LL);
26    v19 = i32_load8_s((__int64)&memory, 1039LL);
27    v18 = i32_load8_s((__int64)&memory, 1040LL);
28    v17 = i32_load8_s((__int64)&memory, 1041LL);
29    v16 = i32_load8_s((__int64)&memory, 1042LL);
30    v15 = i32_load8_s((__int64)&memory, 1043LL);
31    v14 = i32_load8_s((__int64)&memory, 1044LL);
32    v13 = i32_load8_s((__int64)&memory, 1045LL);
33    v12 = i32_load8_s((__int64)&memory, 1046LL);
34    v11 = i32_load8_s((__int64)&memory, 1047LL);
35    v10 = i32_load8_s((__int64)&memory, 1048LL);
36    v9 = i32_load8_s((__int64)&memory, 1049LL);
37    v8 = i32_load8_s((__int64)&memory, 1050LL);
38    v7 = i32_load8_s((__int64)&memory, 1051LL);
39    v6 = i32_load8_s((__int64)&memory, 1052LL);
40    v5 = i32_load8_s((__int64)&memory, 1053LL);
41    )

```

接下来就是下图中看到的32元方程组，如果有兴趣和数学基础的同学可以用矩阵解法写个类似的小脚本，这里我用的是z3库解决

```

if ( 45 * v5
+ 248 * v6
+ 20 * v7
+ 67 * v8
+ 90 * v9
+ 135 * v10
+ 106 * v11
+ 112 * v12
+ 40 * v13
+ 231 * v14
+ 153 * v15
+ 233 * v16
+ 19 * v17
+ 188 * v18
+ 232 * v19
+ 127 * v20
+ 15 * v21
+ 67 * v22
+ 50 * v23
+ 161 * v24
+ 103 * v25
+ 144 * v26
+ 81 * v27
+ 126 * v28
+ 240 * v29
+ 124 * v30
+ 194 * v31
+ 92 * v32
+ 108 * v33
+ 111 * v34
+ 174 * v35
+ 48 * v36 == 359512
&& 131 * v5
+ 120 * v6
+ 149 * v7
+ 244 * v8
+ 56 * v9
+ 154 * v10
+ 156 * v11
+ 94 * v12
+ 169 * v13 |
+ 32 * v14
+ 209 * v15
+ 225 * v16
+ 26 * v17
+ 178 * v18
+ 90 * v19

```

解密

一重解密

pip安装z3-solver

接着用python脚本写一个求解语句，先初始化32个变量，接着将ida的内容拷贝过来，将符号修改一下即可

```
# -*- coding:utf-8 -*-
```

```
from z3 import *
```

```

# ■■■32■■■
v5 = Int('m53')
v6 = Int('m52')
v7 = Int('m51')
v8 = Int('m50')

```



```

v9 = Int('m49')
v10 = Int('m48')
v11 = Int('m47')
v12 = Int('m46')
v13 = Int('m45')
v14 = Int('m44')
v15 = Int('m43')
v16 = Int('m42')
v17 = Int('m41')
v18 = Int('m40')
v19 = Int('m39')
v20 = Int('m38')
v21 = Int('m37')
v22 = Int('m36')
v23 = Int('m35')
v24 = Int('m34')
v25 = Int('m33')
v26 = Int('m32')
v27 = Int('m31')
v28 = Int('m30')
v29 = Int('m29')
v30 = Int('m28')
v31 = Int('m27')
v32 = Int('m26')
v33 = Int('m24')
v34 = Int('m25')
v35 = Int('m55')
v36 = Int('m54')

```

```

# ■■■■■■■■■■
s = Solver()

```

```

s.add(And(45 * v5
    + 248 * v6
    + 20 * v7
    + 67 * v8
    + 90 * v9
    + 135 * v10
    + 106 * v11
    + 112 * v12
    + 40 * v13
    + 231 * v14
    + 153 * v15
    + 233 * v16
    + 19 * v17
    + 188 * v18
    + 232 * v19
    + 127 * v20
    + 15 * v21
    + 67 * v22
    + 50 * v23
    + 161 * v24
    + 103 * v25
    + 144 * v26
    + 81 * v27
    + 126 * v28
    + 240 * v29
    + 124 * v30
    + 194 * v31
    + 92 * v32
    + 108 * v33
    + 111 * v34
    + 174 * v35
    + 48 * v36 == 359512
    ....
    , 244 * v5
    + 196 * v6

```

```

+ 30 * v7
+ 100 * v8
+ 168 * v9
+ 7 * v10
+ 249 * v11
+ 84 * v12
+ 252 * v13
+ 171 * v14
+ 210 * v15
+ 206 * v16
+ 108 * v17
+ 153 * v18
+ 67 * v19
+ 189 * v20
+ 141 * v21
+ 239 * v22
+ 177 * v23
+ 10 * v24
+ 15 * v25
+ 164 * v26
+ 142 * v27
+ 97 * v28
+ 27 * v29
+ 173 * v30
+ 146 * v31
+ 133 * v33
+ 105 * v34
+ 75 * (v32 + v35)
+ 197 * v36 == 393331 ) )

```

```

s.add(185 * v5
      + 196 * v6
      + 135 * v7
      + 218 * (v24 + v9)
      + 241 * v8
      + 210 * v10
      + 127 * v11
      + 221 * v12
      + 47 * v13
      + 179 * v14
      + 61 * v15
      + 59 * v16
      + 197 * v17
      + 204 * v18
      + 198 * v19
      + 75 * v20
      + 146 * v21
      + 156 * v22
      + 235 * v23
      + 63 * v25
      + 220 * v26
      + 3 * v27
      + 167 * v28
      + 230 * v29
      + 69 * v30
      + 186 * v31
      + 57 * v32
      + 147 * v33
      + 221 * v34
      + 79 * v35
      + 53 * v36 == 430295)

```

```

# sat■■■■■■■■
if s.check() == sat:
    t = []
    print "compute result: "
    m = s.model()
    t.append(str(m[v33]))
    t.append(str(m[v34]))

```

```

t.append(str(m[v32]))
t.append(str(m[v31]))
t.append(str(m[v30]))
t.append(str(m[v29]))
t.append(str(m[v28]))
t.append(str(m[v27]))
t.append(str(m[v26]))
t.append(str(m[v25]))
t.append(str(m[v24]))
t.append(str(m[v23]))
t.append(str(m[v22]))
t.append(str(m[v21]))
t.append(str(m[v20]))
t.append(str(m[v19]))
t.append(str(m[v18]))
t.append(str(m[v17]))
t.append(str(m[v16]))
t.append(str(m[v15]))
t.append(str(m[v14]))
t.append(str(m[v13]))
t.append(str(m[v12]))
t.append(str(m[v11]))
t.append(str(m[v10]))
t.append(str(m[v9]))
t.append(str(m[v8]))
t.append(str(m[v7]))
t.append(str(m[v6]))
t.append(str(m[v5]))
t.append(str(m[v36]))
t.append(str(m[v35]))
t = map(int, t)
t = map(chr, t)
print "".join(t)

else:
    print "failed"

```

二重解密

这里直接用的大佬的脚本，将上面解密的数据进行异或计算，即可返回最终我们需要输入的内容

```

int main(int argc, char** argv) {
    unsigned char c[33] = "S0m3time_l1ttle_c0de_1s_us3ful33";
    unsigned char in[33] = { 0 };
    unsigned int t1 =0,t2= 0,t3=0,t4=0;

    printf((const char *)c);
    printf("\n");
    in[0] = c[0] ^ 0x18;
    in[1] = c[1] ^ 0x9;
    in[2] = c[2] ^ 0x3;
    in[3] = c[3] ^ 0x6b;

    in[4] = c[4] ^ 0x1;
    in[5] = c[5] ^ 0x5A;
    in[6] = c[6] ^ 0x32;
    in[7] = c[7] ^ 0x57;

    in[8] = c[8] ^ 0x30;
    in[9] = c[9] ^ 0x5d;
    in[10] = c[10] ^ 0x40;
    in[11] = c[11] ^ 0x46;

    in[12] = c[12] ^ 0x2b;
    in[13] = c[13] ^ 0x46;
    in[14] = c[14] ^ 0x56;
    in[15] = c[15] ^ 0x3d;

    in[16] = c[16] ^ 0x02;

```

```
in[17] = c[17] ^ 0x43;
in[18] = c[18] ^ 0x17;
in[19] = c[19];
in[20] = c[20] ^ 0x32;
in[21] = c[21] ^ 0x53;
in[22] = c[22] ^ 0x1F;
in[23] = c[23] ^ 0x26;

in[24] = c[24] ^ 0x2a;
in[25] = c[25] ^ 0x01;
in[26] = c[26];
in[27] = c[27] ^ 0x10;
in[28] = c[28] ^ 0x10;
in[29] = c[29] ^ 0x1E;
in[30] = c[30] ^ 0x40;
in[31] = c[31];
printf((const char *)in);
return 0;
}
```

小结

【1】多元线性方程式可以通过python的z3-solver库快速计算

反思

最开始做这道题我是卡在了最后一步，我用sage并未求出结果。

主要原因是：我甚至未能清除的理解这个算法的本质，当时并未意识到这是个多元方程求解的计算，只想着怎么求出这个结果，结果在网上找到一个相似题的解决方法，用s

结论：解决问题时，不要求对所有细节了如执掌，但是题的主干脉络、根本思路是我们需要探索的

参考

- 【1】[原创]第五题：丛林的秘密<https://bbs.pediy.com/thread-252191.htm>
- 【2】Z3 API in Python <https://nen9ma0.github.io/2018/03/14/z3py/>
- 【3】IDC脚本 - IDC脚本语言官方教程 <https://bbs.pediy.com/thread-219016.htm>
- 【4】线性方程组矩阵解法 <https://www.shuxuele.com/algebra/systems-linear-equations-matrices.html>

点击收藏 | 1 关注 | 1

[上一篇：某拍App算法so层逆向分析](#) [下一篇：Google2019CTF web...](#)

- 0 条回复
 - 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)