

crypt 450

拿到流量包，只有三个http请求：

- 主页 (特别加载了网站图标)
- 网站图标 (favicon.ico)
- 下载了一个加密的zip文件 (Somethingneeded.zip)

压缩包有密码，爆破不出来也没有提示；主页上就一个下载压缩包的链接，和一个网站图标的请求，然后看favicon.ico。一个简单的隐写，改了文件高度= 没有想到这里会卡主一部分人，估计解压出来压缩包的都卡在密码上，没解压缩包的，都卡在这个隐写上上面了。

用010editor打开图片，并载入png模板。找到文件的宽高字段，修改高度为一个略大的值，比如400，重新计算CRC并用新得到的CRC值替换原来字段的CRC值保存文件，打开就会得到zip加密的key。

修改前：

Startupico.png

0123456789ABCDEF0123456789ABCDEF

0000h: 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52 %PNG.....IHDR

0010h: 00 00 00 F0 00 00 00 F0 08 04 00 00 00 94 5C 21 ...8...8...:"!

0020h: 19 00 00 00 09 70 48 59 73 00 00 00 27 00 00 00pHYs...'...

0030h: 27 01 2A 09 91 4F 00 00 00 20 63 48 52 4D 00 00 '.*.'O...cHRM..

0040h: 7A 25 00 00 80 83 00 00 F9 FF 00 00 80 E9 00 00 z%.ef..ÿ..éé..

0050h: 75 30 00 00 EA 60 00 00 3A 98 00 00 17 6F 92 5F u0..è'...'...'o'

0060h: C5 46 00 00 09 92 49 44 41 54 78 DA EC DC 5B 8C ÅF...'IDATxÜiÜ(

0070h: 5C 65 01 C0 F1 FF 37 67 F6 BE DB BD 76 7B 59 9A \e.Åñÿ7gö%Ü*sv{Yš

0080h: 96 6B 29 05 65 E5 1E 08 18 09 62 10 24 3E 91 80 -k).eä....b.\$>'€

Template Results - PNG.bt

Name	Value	Start	Size	Color
> struct PNG_SIGNATURE sig		0h	8h	Fg: Bg:
▼ struct PNG_CHUNK chunk[0]	IHDR (Critical, Public, Unsafe to Copy)	8h	19h	Fg: Bg:
uint32 length	13	8h	4h	Fg: Bg:
> union CTYPE type	IHDR	Ch	4h	Fg: Bg:
▼ struct PNG_CHUNK_IHDR ihdr	240 x 240 (x8)	10h	Dh	Fg: Bg:
uint32 width	240	10h	4h	Fg: Bg:
uint32 height	240	14h	4h	Fg: Bg:
ubyte bits	8	18h	1h	Fg: Bg:
enum PNG_COLOR_SPAC...	AlphaGrayScale (4)	19h	1h	Fg: Bg:
enum PNG_COMPR_MET...	Deflate (0)	1Ah	1h	Fg: Bg:
enum PNG_FILTER METH...	AdaptiveFiltering (0)	1Bh	1h	Fg: Bg:
enum PNG_INTERLACE_...	NoInterlace (0)	1Ch	1h	Fg: Bg:
uint32 crc	945C2119h	1Dh	4h	Fg: Bg:
> struct PNG_CHUNK chunk[1]	pHYs (Ancillary, Public, Safe to Copy)	21h	15h	Fg: Bg:
> struct PNG_CHUNK chunk[2]	cHRM (Ancillary, Public, Unsafe to Copy)	36h	2Ch	Fg: Bg:
> struct PNG_CHUNK chunk[3]	IDAT (Critical, Public, Unsafe to Copy)	62h	99Eh	Fg: Bg:
> struct PNG_CHUNK chunk[4]	IEND (Critical, Public, Unsafe to Copy)	A00h	Ch	Fg: Bg:

修改后：

Startupico.png*

0123456789ABCDEF0123456789ABCDEF

0000h: 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52 %PNG.....IHDR

0010h: 00 00 00 F0 00 00 01 90 08 04 00 00 00 3A DF 25 ...8...8...:B%

0020h: F7 00 00 00 09 70 48 59 73 00 00 00 27 00 00 00pHYs...'...

0030h: 27 01 2A 09 91 4F 00 00 00 20 63 48 52 4D 00 00 '.*.'O...cHRM..

0040h: 7A 25 00 00 80 83 00 00 F9 FF 00 00 80 E9 00 00 z%.ef..ÿ..éé..

0050h: 75 30 00 00 EA 60 00 00 3A 98 00 00 17 6F 92 5F u0..è'...'...'o'

0060h: C5 46 00 00 09 92 49 44 41 54 78 DA EC DC 5B 8C ÅF...'IDATxÜiÜ(

0070h: 5C 65 01 C0 F1 FF 37 67 F6 BE DB BD 76 7B 59 9A \e.Åñÿ7gö%Ü*sv{Yš

0080h: 96 6B 29 05 65 E5 1E 08 18 09 62 10 24 3E 91 80 -k).eä....b.\$>'€

Template Results - PNG.bt

Name	Value	Start	Size	Color
> struct PNG_SIGNATURE sig		0h	8h	Fg: Bg:
▼ struct PNG_CHUNK chunk[0]	IHDR (Critical, Public, Unsafe to Copy)	8h	19h	Fg: Bg:
uint32 length	13	8h	4h	Fg: Bg:
> union CTYPE type	IHDR	Ch	4h	Fg: Bg:
▼ struct PNG_CHUNK_IHDR ihdr	240 x 400 (x8)	10h	Dh	Fg: Bg:
uint32 width	240	10h	4h	Fg: Bg:
uint32 height	400	14h	4h	Fg: Bg:
ubyte bits	8	18h	1h	Fg: Bg:
enum PNG_COLOR_SPAC...	AlphaGrayScale (4)	19h	1h	Fg: Bg:
enum PNG_COMPR_MET...	Deflate (0)	1Ah	1h	Fg: Bg:
enum PNG_FILTER METH...	AdaptiveFiltering (0)	1Bh	1h	Fg: Bg:
enum PNG_INTERLACE_...	NoInterlace (0)	1Ch	1h	Fg: Bg:
uint32 crc	3ADF25F7h	1Dh	4h	Fg: Bg:
> struct PNG_CHUNK chunk[1]	pHYs (Ancillary, Public, Safe to Copy)	21h	15h	Fg: Bg:
> struct PNG_CHUNK chunk[2]	cHRM (Ancillary, Public, Unsafe to Copy)	36h	2Ch	Fg: Bg:
> struct PNG_CHUNK chunk[3]	IDAT (Critical, Public, Unsafe to Copy)	62h	99Eh	Fg: Bg:
> struct PNG_CHUNK chunk[4]	IEND (Critical, Public, Unsafe to Copy)	A00h	Ch	Fg: Bg:

CRC Calculator

HexASCII

49484452000000F00000001900804000000

Copy

Paste

CRC-32x32+x26+x23+x22+x1

CRC: 3ADF25F7HexBinCopy

CRC-32x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x1



key:3%7@pj!!aYqL!XJ8

用得到的key解压缩包会得到三个文件。

1. 发现RSA公钥中存在低解密指数攻击（加密指数过长）：通过Wiener Attack 攻击公钥n, e , 得到RSA私钥。恢复出RSA明文：

```
AES_key:>>Jfff0aqYHvuSikfU
```

```
First sign: ElGamal; First HASH: SHA[in python:>>int(SHA.new(msg).digest().encode('hex'),16)]
```

2. 使用1. 中得到的AES的密钥解密文本，得到：

```
ESIGN
```

```
HASH: SHA
```

```
Alice in Sun Sep 25 02:40:07 CST 2016
```

```
in these Signatures, important things r different....but....they r one and the same. indeed.
```

通过1. 中所给消息，知道本次信息传递中使用的是elgamal签名算法。发现几条对消息签名时使用了相同的随机指数（A值相同）。根据A, B, P, 以及1. 中给出的hash计算方法，解出elgamal的私钥。

1. 通过2. 中的AES明文，知道本次信息传递中使用的是ESIGN签名算法。用AES解密后，发现明文是两篇演讲的片段。

发现没有找到flag，回头检查3，通过计算签名s的长度，发现其长度为4094位。长度有些诡异，怀疑用来计算N的私钥有问题。通过以前的积累，或查询资料，发现本次esign中存在阙下信道。通过2. 中所给信息，猜测本次阙下信道中的额外私钥r与2. 中elgamal签名的私钥相同。对3. 中签名信息计算后，得到信息：

```
1. flag:>>pleas3_b00m\_**__b0om_shak4l4ka_lam\_****
```

```
2. hash:>>b96648cc3f097c6faf581386b97d7e043521abe2369d1141d9a98d4897770ea7
```

```
3. map(lambda x:x in string.letters[:26]+string.digits, flag)==[True for x in len(flag)]
```

通过得到的信息得知：flag的格式、flag的hash值、flag包含的字符。其中，根据散列值长度得知，hash算法应为SHA256。此处有两种解法：

1. 猜测flag的值
2. 爆破

爆破耗时: in python 7966.50890207s