ByteCTF2021 Crypto - abusedkey writeup

原创

风好衣轻 于 2021-10-21 12:08:37 发布 53 收藏

分类专栏： CTF 文章标签： python wp

abusedkey

首先把用到的数据放在了task_data.py，方便些其他脚本时直接导入：

```
URL = "http://39.105.181.182:30000"
msg11 = URL+"/abusedkey/server/msg11"
msg13 = URL+"/abusedkey/server/msg13"
msg21 = URL+"/abusedkey/server/msg21"
msg23 = URL+"/abusedkey/ttp/msg23"
msg25 = URL+"/abusedkey/server/msg25"

# -------------------------------- Secp256k1 --------------------------------
p = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F
a, b = 0, 7
G = (0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798,
    0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8)
# ------------------ https://en.bitcoin.it/wiki/Secp256k1 -------------------

Pc = (0xb5b1b07d251b299844d968be56284ef32dffd0baa6a0353baf10c90298dfd117,
     0xea62978d102a76c3d6747e283091ac5f2b4c3ba5fc7a906fe023ee3bc61b50fe)
```

协议2的部分，想要拿到hint很简单，只要按照描述实现出来，就拿到了hint，hint.sage:

```
import requests, os, random
from Crypto.Cipher import AES
from Crypto.Util.number import long_to_bytes
from task_data import p, a, b, G, msg21, msg23, msg25
from hashlib import sha256

E = EllipticCurve(IntegerModRing(p), [a, b])
G = E(G)

# sid2 = hex(random.getrandbits(256))[2:]
sid2 = "8d1a95ce724141a0ea7c8ffa7eddc48605b3117c8aa886bcc2aff3b0c2175b56"
msg22 = requests.get(msg21, data=sid2).text
Qs_hex = msg22

rc = 1 # random.randint(1, p)
Rc = rc*G

Pic = long_to_bytes(int('FFFF', 16))
hc = int(sha256(Pic).hexdigest(), 16)
Qc = hc*Rc
Qc_hex = hex(Qc[0])[2:].rjust(64) + hex(Qc[1])[2:].rjust(64)
assert len(Qc_hex) == 128

msg24 = requests.get(msg23, data=Qc_hex+Qs_hex).text
assert len(msg24) == 256
Yc_hex, Ys_hex = msg24[:128], msg24[128:]

msg26 = requests.get(msg25, data=sid2+Yc_hex).text

Ys = E((int(Ys_hex[:64], 16), int(Ys_hex[64:], 16)))
Zcs = rc*Ys
Zcsx = long_to_bytes(int(Zcs[0]))
sk2 = sha256(Zcsx).digest()

msg26 = bytes.fromhex(msg26)
iv, ciphertext, mac = msg26[:12], msg26[12:-16], msg26[-16:]
cipher = AES.new(sk2, mode=AES.MODE_GCM, nonce=iv)
try:
    m = cipher.decrypt_and_verify(ciphertext, mac)
    print(m.decode())
except ValueError:
    print("MAC check failed")
# off-line guessing on protocol_II, and key compromise impersonation on protocol_I
```

Hint: off-line guessing on protocol_II, and key compromise impersonation on protocol_I

hint和题目描述都在说明，两个协议共用一个Server端的key，那么大概思路就是通过协议2拿到key，再将这个key用于解协议1的flag，可以先简单分析一下：

```
已知 rc-(随机), hc-H(c口令)
未知 rs-(随机), hs-H(s口令)
Qc = rc * hc * G --- 已知
Qs = rs * hs * G --- 已知

Yc = rc * rt * G --- 已知
Ys = rs * rt * G --- 已知

Zcs = rc * rs * rt * G --- 已知 公共密钥
```

这里面的rc是我们可以控制的，所以可以令rc=1让问题看起来简单一点。

```
rc = 1 时：

Qc = hc * G --- 已知
Qs = rs * hs * G --- 已知

Yc  =     rt * G --- 已知
Ys  = rs * rt * G --- 已知
Zcs = rs * rt * G --- 已知 公共密钥
```

hs是两个字节的sha256结果，显然是让我们爆破的，也就是说我们需要得到一组形式为`hs*Point`和`Point`的数据，这样去爆两个字节就可以了，为了得到这样的数据，我们需要构造一下发送的数据。

```
发送假的 Qc = hc * rs * hs * G = hc * Qs
得到   Yc = hs * rs * rt * G

发送 Qs = rs * hs * G
得到 Ys = rs * rt * G
```

这样以来，Ys和Yc刚好是我们需要的一组数据，

$Yc = hs \cdot Ys$，然后爆破一下两个字节的哈希，如果某两个字节的sha256乘Ys等于Yc，那么这个sha256的值就

```
import requests, os, random, tqdm
from Crypto.Cipher import AES
from Crypto.Util.number import long_to_bytes
from task_data import p, a, b, G, msg21, msg23, msg25
from hashlib import sha256

E = EllipticCurve(IntegerModRing(p), [a, b])
G = E(G)

sid2 = "8d1a95ce724141a0ea7c8ffa7eddc48605b3117c8aa886bcc2aff3b0c2175b56"
msg22 = requests.get(msg21, data=sid2).text
Qs = E((int(msg22[:64], 16), int(msg22[64:], 16)))


rc = 1 # random.randint(1, p)
Rc = rc*G

Pic = long_to_bytes(int('FFFF', 16))
hc = int(sha256(Pic).hexdigest(), 16)
fake_Qc = hc * Qs # hc * rs * hs * G
fake_Qc_hex = hex(fake_Qc[0])[2:].rjust(64) + hex(fake_Qc[1])[2:].rjust(64)

msg24 = requests.get(msg23, data=fake_Qc_hex+msg22).text
assert len(msg24) == 256
Yc_hex, Ys_hex = msg24[:128], msg24[128:]

# hs * rs * rt * G
Yc = E((int(Yc_hex[:64], 16), int(Yc_hex[64:], 16)))
#      rs * rt * G
Ys = E((int(Ys_hex[:64], 16), int(Ys_hex[64:], 16)))

for pis in tqdm.tqdm(range(0xff, 0xffff+1)):
    hs = int(sha256(long_to_bytes(pis)).hexdigest(), 16)
    if ((hs*Ys) == Yc):
        print(f'pis = {pis}\nhs = {hs}')
        break

'''
pis = 36727
hs = 67294392667457530634966084521984708026794776225602296684920633502274376489620
'''
```

协议2搞到了hs，也就是协议1中的服务端私钥ds，所以服务端的公钥也很容易得到，这样就有了

```
import requests, random
from Crypto.Util.number import *
from Crypto.Cipher import AES
from task_data import p, a, b, G, msg11, msg13, Pc
from hashlib import sha256



E = EllipticCurve(IntegerModRing(p), [a, b])
G = E(G)
sid1 = "8d1a95ce724141a0ea7c8ffa7eddc48605b3117c8aa886bcc2aff3b0c2175b56"

msg12 = requests.get(msg11, data=sid1).text
ds = 67294392667457530634966084521984708026794776225602296684920633502274376489620
Ps = ds*G
Pc = E(Pc)
invPc = -1*Pc
print(invPc)
invPc_hex = hex(invPc[0])[2:].rjust(64) + hex(invPc[1])[2:].rjust(64)
msg14 = requests.get(msg13, data=sid1+invPc_hex).text

Kcs = ds*invPc
sk1 = sha256(long_to_bytes(int(Kcs[0]))).digest()

msg26 = bytes.fromhex(msg14)
iv, ciphertext, mac = msg26[:12], msg26[12:-16], msg26[-16:]
cipher = AES.new(sk1, mode=AES.MODE_GCM, nonce=iv)
try:
    m = cipher.decrypt_and_verify(ciphertext, mac)
    print(m.decode())
except ValueError:
    print("MAC check failed")
```