Brutal Mogging - UMassCtf 2024

by Luca Boscarato and Emmanuel Scopelliti

Overview

The challenge provides a server that lets us encrypt or decrypt any data. The encryption (as well as the decryption) is done in blocks of four.

```
def encrypt(data, key):
    data = pad(data)
    return encrypt_data(encrypt_data(data, key[0:2]), key[2:4])

def decrypt(data, key):
    plain = decrypt_data(decrypt_data(data, key[2:4]), key[0:2])
    return unpad(plain)
```

As you can see, the encryption is performed by encrypting an already encrypted version of the plaintext, using for the inner encryption the first two bytes of the key and for the outer the remaining two. The opposite is used for decryption.

Structure

Our goal is to decrypt the encrypted flag that is displayed whenever someone connects to the server.

```
from binascii import unhexlify

r = remote('brutalmogging.ctf.umasscybersec.org', 1337)
r.recvuntil(b'flag is: ')
flag = unhexlify(r.recvline().strip().decode())
```

One can notice that the key is only 4 bytes, therefore, if we wanted to bruteforce it we could by trying a total of 256^4 keys. Implementing such a bruteforce, especially in an optimized language like Rust or C could be feasible, but a better approach exists.

Exploitation

Namely, we could try a "Meet me in the middle" approach. In this case, we want to encrypt some text and to decrypt a second, and check whether they match. In such a way, we can significantly reduce the amount of operations to be performed. In particular, we will perform twice a bruteforce of 256^2.

Let us dive into the code and see how we can implement such an idea:

```
from Crypto.Util.number import long_to_bytes

# Assume we have a couple plain - ciphertext provided by the server
# In this case we will use the following

plain  = b"UMAS"
ciphertext = b"\xb4\xcd\x49\x49"

alphabet = [long_to_bytes(i) for i in range(256)]
p01 = {}
for k0 in alphabet:
    for k1 in alphabet:
        p01[encrypt_data(plain,k0+k1)] = k0+k1
        p23 = {}


for k2 in alphabet:
    for k3 in alphabet:
        p23[decrypt_data(ciphertext,k2+k3)] = k2+k3

# Now we combine the results and find an intersection between
# the two dictionaries:

for i in p01.keys():
    if i in p23.keys():
        key=p01[i]+p23[i]

print(decrypt(encrypted_flag,key))
```

> **Note**: Sometimes, the script could fail due to hash collisions, namely, two different keys could produce the same ciphertext. This happens because sha256 is used in the *encrypt* function.

Flag

> UMASS{1_h4ve_b33n_m3w1ng_f0r_my_l1f3_733061741}