Lab-report 5

Moritz Rupp

June 15, 2022

Contents

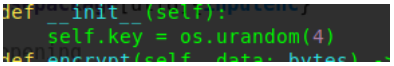
Abstract

Lab 5 Dokumentation

1 Lab 5

1.1 Exercise 5.1 - Basic repeated XOR encryption

We are provided with a small python programm and a .txt file that includes the encrypted flag. The programm is a self made crypto implementation that should be easy to crack.
After analysing the code we realize that the key lenght is only 4 characters long.



Thanks to the provided hint we also see the we are given the first 4 characters of the encrypted stream, since the HTB flags always follow the same format. This matches the key lenght.

HTB{SOME_FLAG}
⇒ HTB{

We also know that xor operations are beeing performed. Hence we take the ASCII values of the known plaintext characters and xor them with the first 4 characters of the encrypted stream.

4854427b xor 134af6e1
⇒ 5b1eb49a

This should be the key to decrypt the whole sequence. With the help of an online tool we get the correct key.

HTB{rep34t3d_x0r_n0t_s0_s3cur3}

1.2 Exercise 5.2 - AES-CBC Decryption Oracle

(In collaboration with Maximilian Heim!)
We are provided with an executable and a ciphertext, that includes an initialisation vector and 2 AES cipher blocks.
First we make the programm executable. When we run it without parameters it gives us a useage hint.

```
sleven  parrot  ../Offsec-lab/Lab5  chmod 700 aes128-cbc-crackme

sleven  parrot  ../Offsec-lab/Lab5  ./aes128-cbc-crackme
Usage: aes128-cbc-crackme iv ciphertext
- iv is the initialization vector as hex string (32 characters out of 0-9,a-f)
- ciphertext is the cipher text as hex string (a multiple of 32 characters out of 0-9,a-f)
```

When passing the iv with different parameters the programm prompts valid or invalid which can be used to decrypt the sequence.

The attack format is run by passing:

$$\text{IV} + \text{padding} + \text{cipher block}$$

Following the lecture slides we came up with the following. We generate the padding, pass it, listen to the output if valid or invalid. If valid we use the index byte xor it with r and the byte of the first parameter. We do this with the follow up bytes and apply it on both blocks. When converted to an ASCII string and formated properly we get the success output!

```python
import random
import subprocess
def pad_byte(b):
    if len(hex(b)) == 3:
        return "0" + hex(b)[2:]
    else:
        return hex(b)[2:]
plaintext_result = []
blocks = ["00112233445566778800112233445566", "9949F0ED9C7D4FB79B600B8854B18FDC",\
          "51A894A2BA12939DCC36F074F63CF60B"]
def oracle(c0, c1):
    plaintext = []
    pad_bytes = []
    padding = ""
    for b in range(16, 0, -1):
        for i in range(256):
            r = subprocess.run(["./aes128-cbc-crackme", c0, "00" * (b-1) + pad_byte(i) + padding + c1], \
                               capture_output=True)
            if r.stdout == b"padding valid\n":
                plaintext.append((17-b) ^ i ^ int(c0[(b-1)*2:(b-1)*2 + 2], 16))
                pad_bytes.append(i)
                padding = ""
                for j in range(len(pad_bytes)):
                    pad_bytes[j] = pad_bytes[j] ^ ((18-b) ^ (17-b))
                    padding = pad_byte(pad_bytes[j]) + padding
    return plaintext
for i in range(2):
    res = oracle(blocks[i], blocks[i+1])
    res.reverse()
    plaintext_result += res
print("".join([chr(c) for c in plaintext_result]))
```

```
Success! You have decrypted me!
```