

**Title:**

*Asking the Oracle for Cyber Secrets*

**Abstract:**

Lab 4 demonstrated the importance of padding oracle attacks, as well as using nc and docker to create multiple containers to extract information. The lab allowed us to use a version of this oracle attack in an informative way, using the manual aspect to truly understand the inner-workings of this cyber security attack method and how much effort it may take.

**Aim:**

The aim of the lab was to get hands on experience with another crypto systems cyber attack. This lab focused on utilizing the *padding oracle attack*, published by Serge Vaudenay which entails verifying whether or not the ciphertext has valid padding or not. We were tasked with deciphering a message using this method.

## **Introduction and Background:**

The padding oracle attack involved utilizing two oracle servers side-by-side, in order to decrypt and figure out the plaintext and encryption key, knowing only the ciphertext and IV. The responses gained from the oracle allow us to figure out if the padding is valid and therefore gain access to the plaintext. Essentially, this method will allow us to ask the oracle for extra information which will allow us to decrypt a message, although it is laborious, it is an effective way to gain an introduction into padding oracle attacks.

## **Method:**

In this lab we utilized a container to run the padding oracle. Naturally, the best method in order to do this is utilizing docker. Building a docker container allowed us to create the container, which then resulted in the creation of files in 5, 10 and 16 bytes.

```
seed@VM: ~/.../Labsetup 2
[10/12/22]seed@VM:~/.../Labsetup 2$ nc 10.9.0.80 5000
[10/12/22]seed@VM:~/.../Labsetup 2$ ls
docker-compose.yml  manual_attack.py
[10/12/22]seed@VM:~/.../Labsetup 2$ docker-compose build
web-server uses an image, skipping
[10/12/22]seed@VM:~/.../Labsetup 2$ docker-compose up
Starting oracle-10.9.0.80 ... done
Attaching to oracle-10.9.0.80
oracle-10.9.0.80 | Server listening on 5000 for padding_oracle_L1
oracle-10.9.0.80 | Server listening on 6000 for padding_oracle_L2
oracle-10.9.0.80 | Connect to 5000, launching padding_oracle_L1
█

seed@VM: ~/.../Labsetup 2
[10/12/22]seed@VM:~/.../Labsetup 2$ nc 10.9.0.80 5000
01020304050607080102030405060708a9b2554b0944118061212098f2f238cd779ea0aae3d9d020
f3677bfc3cda9ce
█
```

## Screenshot section 1

Using nc to connect to the docker container, the oracle can now accept our input.

Providing our own secret key and IV, the oracle will tell us if the padding is valid or not.

```
[10/12/22]seed@VM:~/Desktop$ ls
p10 p10_new p16 p16_new p5 p5_new
[10/12/22]seed@VM:~/Desktop$ xxd p5_new
00000000: 5361 6c74 6564 5f5f 4783 1e9e 388b f5a5  Salted_G...8...
00000010: 938b 69a0 48dd 153b 5461 b2cf 2de6 5da2  ..i.H.;Ta..].
[10/12/22]seed@VM:~/Desktop$ xxd p10_new
00000000: 5361 6c74 6564 5f5f 36dc 4dd3 9f53 ec99  Salted_6.M..S..
00000010: 3193 4699 b63f 35d4 5d42 73fa 573b d025  1.F..?5.]Bs.W;.%
[10/12/22]seed@VM:~/Desktop$ xxd p16_new
00000000: 5361 6c74 6564 5f5f 8c4f c0b0 76a4 a2e2  Salted_..0..v...
00000010: 4fee f964 3384 063f 14f7 da94 c4c9 84ce  0..d3..?.....
[10/12/22]seed@VM:~/Desktop$ █
```

## Screenshot section 1

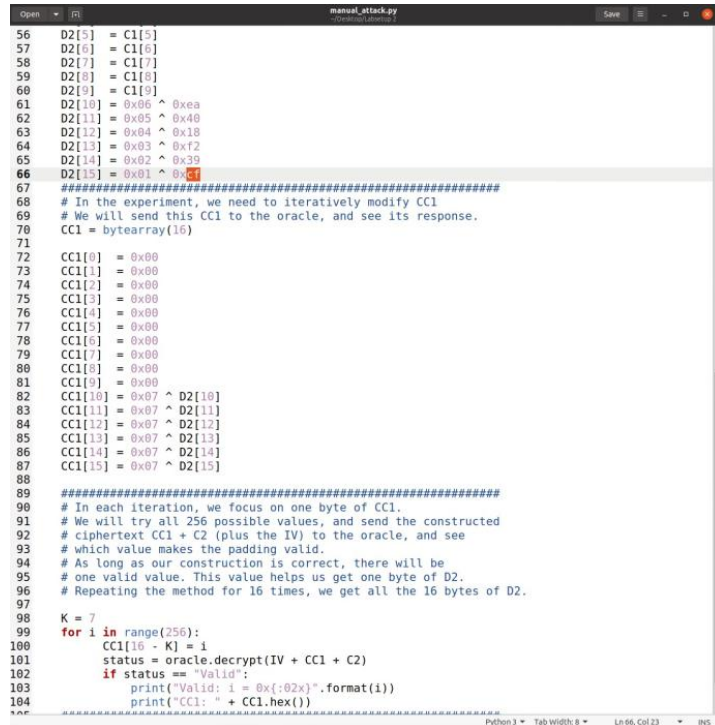
```
C1: a9b2554b0944118061212098f2f238cd
P2: 779ea0aae3d9d020f3677fbcb3cda9ce
C2: 000000000000000000000000000000303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1: a9b2554b0944118061212098f2f238cd
C2: 779ea0aae3d9d020f3677fbcb3cda9ce
Valid: i = 0xf2
CC1: 000000000000000000000000000000f238cd
P2: 000000000000000000000000000000303
[10/12/22]seed@VM:~/.../Labsetup 2$
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1: a9b2554b0944118061212098f2f238cd
C2: 779ea0aae3d9d020f3677fbcb3cda9ce
Valid: i = 0x18
CC1: 00000000000000000000000000000018f53fca
P2: 00000000000000000000000000000030303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1: a9b2554b0944118061212098f2f238cd
C2: 779ea0aae3d9d020f3677fbcb3cda9ce
Valid: i = 0x18
CC1: 00000000000000000000000000000018f53fca
P2: 000000000000000000000000000000ee030303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1: a9b2554b0944118061212098f2f238cd
C2: 779ea0aae3d9d020f3677fbcb3cda9ce
P2: 00000000000000000000000000000030303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1: a9b2554b0944118061212098f2f238cd
C2: 779ea0aae3d9d020f3677fbcb3cda9ce
Valid: i = 0x18
CC1: 00000000000000000000000000000018f53fca
P2: 000000000000000000000000000000ee030303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1: a9b2554b0944118061212098f2f238cd
C2: 779ea0aae3d9d020f3677fbcb3cda9ce
Valid: i = 0x40
CC1: 00000000000000000000000000000019f43ecb
P2: 000000000000000000000000000000ee030303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1: a9b2554b0944118061212098f2f238cd
C2: 779ea0aae3d9d020f3677fbcb3cda9ce
Valid: i = 0xea
CC1: 000000000000000000000000000000ea431af73dc8
P2: 000000000000000000000000000000ddee030303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1: a9b2554b0944118061212098f2f238cd
C2: 779ea0aae3d9d020f3677fbcb3cda9ce
Valid: i = 0x9d
CC1: 0000000000000000000000000000009deb421bf63cc9
P2: 000000000000000000000000000000ccdee030303
[10/12/22]seed@VM:~/.../Labsetup 2$
```

## Screenshot section 2

Lastly, we derived the plain text manually, given the python file included in the lab.

This involved figuring out the 16 bytes of D2, as shown in the screenshot below.

We were able to unlock the last 6 bytes, which were the padding with this method.



```
56 D2[5] = C1[5]
57 D2[6] = C1[6]
58 D2[7] = C1[7]
59 D2[8] = C1[8]
60 D2[9] = C1[9]
61 D2[10] = 0x06 ^ 0xea
62 D2[11] = 0x05 ^ 0x40
63 D2[12] = 0x04 ^ 0x18
64 D2[13] = 0x03 ^ 0xf2
65 D2[14] = 0x02 ^ 0x39
66 D2[15] = 0x01 ^ 0xc4
67 #####
68 # In the experiment, we need to iteratively modify CC1
69 # We will send this CC1 to the oracle, and see its response.
70 CC1 = bytearray(16)
71
72 CC1[0] = 0x00
73 CC1[1] = 0x00
74 CC1[2] = 0x00
75 CC1[3] = 0x00
76 CC1[4] = 0x00
77 CC1[5] = 0x00
78 CC1[6] = 0x00
79 CC1[7] = 0x00
80 CC1[8] = 0x00
81 CC1[9] = 0x00
82 CC1[10] = 0x07 ^ D2[10]
83 CC1[11] = 0x07 ^ D2[11]
84 CC1[12] = 0x07 ^ D2[12]
85 CC1[13] = 0x07 ^ D2[13]
86 CC1[14] = 0x07 ^ D2[14]
87 CC1[15] = 0x07 ^ D2[15]
88
89 #####
90 # In each iteration, we focus on one byte of CC1.
91 # We will try all 256 possible values, and send the constructed
92 # ciphertext CC1 + C2 (plus the IV) to the oracle, and see
93 # which value makes the padding valid.
94 # As long as our construction is correct, there will be
95 # one valid value. This value helps us get one byte of D2.
96 # Repeating the method for 16 times, we get all the 16 bytes of D2.
97
98 K = 7
99 for i in range(256):
100     CC1[16 - K] = i
101     status = oracle.decrypt(IV + CC1 + C2)
102     if status == "Valid":
103         print("Valid: i = 0x{:02x}".format(i))
104         print("CC1: " + CC1.hex())
```

## Screenshot section 2

## Results and Discussion

This lab provided great insight into the inner workings of padding oracle attacks. Using docker to build containers and ask the oracle to provide information felt more advanced in our cybersecurity knowledge journey. The manual derivation of the plaintext provided a very laborious method to decrypt the information with this method and would be very taxing in the real world. Furthermore, only finding the first 6 bytes, which was labor intensive, revealed the padding to us and not the

entire message, which demonstrates the effort required in this method. I hope to be able to complete the optional and additional tasks in future as well.

## **References:**

“Padding oracle attack,” *Wikipedia*, 19-May-2022. [Online]. Available: [https://en.wikipedia.org/wiki/Padding\\_oracle\\_attack](https://en.wikipedia.org/wiki/Padding_oracle_attack). [Accessed: 14-Oct-2022].