Paul Garewal
V00803658

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.

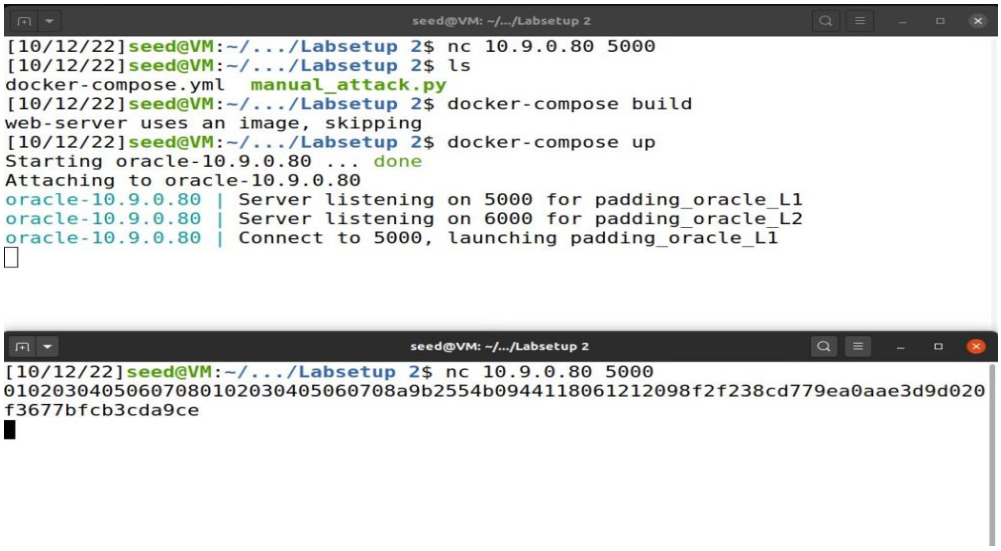
Paul Garewal
V00803658

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:

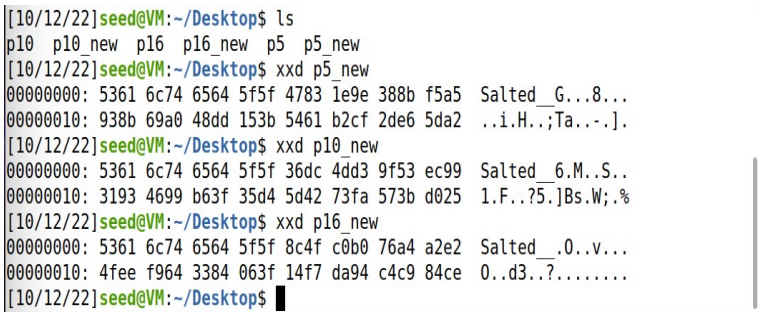
In this lab we utilized a container to run the padding oracle. Naturally, the best method in order to do this is utilizing docker. Building a docker container allowed us to create the container, which then resulted in the creation of files in 5, 10 and 16 bytes.



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.

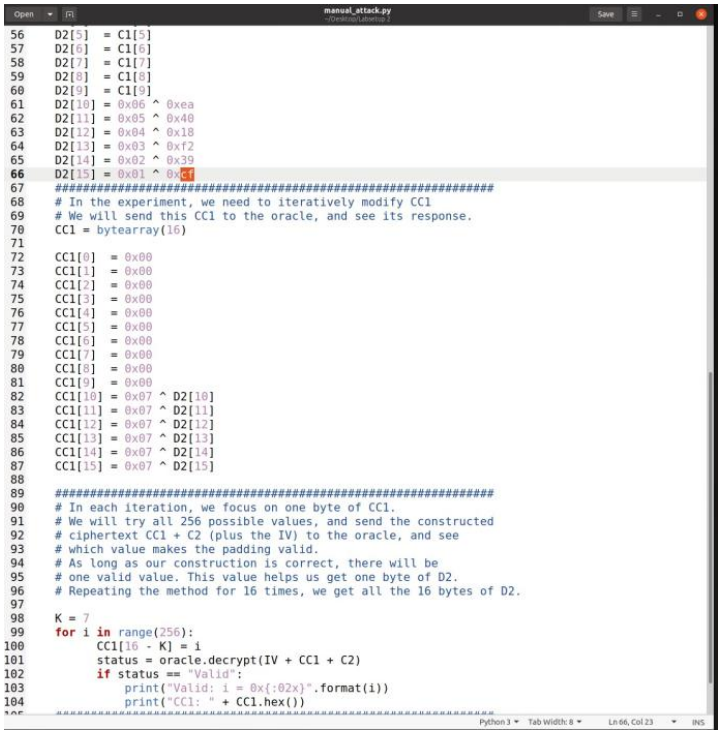


Screenshot section 1

```
C1:  a9b2554b09441180612120 98f2f238cd
C2:  779ea0aae3d9d020f3677bfcb3cda9ce
P2:  00000000000000000000000000000303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1:  a9b2554b09441180612120 98f2f238cd
C2:  779ea0aae3d9d020f3677bfcb3cda9ce
Valid: i = 0xf2
CC1: 0000000000000000000000000f238cd
P2:  00000000000000000000000000000303
[10/12/22]seed@VM:~/.../Labsetup 2$
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1:  a9b2554b09441180612120 98f2f238cd
C2:  779ea0aae3d9d020f3677bfcb3cda9ce
Valid: i = 0x18
CC1: 00000000000000000000000018f53fca
P2:  00000000000000000000000000030303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1:  a9b2554b09441180612120 98f2f238cd
C2:  779ea0aae3d9d020f3677bfcb3cda9ce
Valid: i = 0x18
CC1: 00000000000000000000000018f53fca
P2:  000000000000000000000000ee030303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1:  a9b2554b09441180612120 98f2f238cd
C2:  779ea0aae3d9d020f3677bfcb3cda9ce
P2:  00000000000000000000000000030303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1:  a9b2554b09441180612120 98f2f238cd
C2:  779ea0aae3d9d020f3677bfcb3cda9ce
Valid: i = 0x18
CC1: 00000000000000000000000018f53fca
P2:  00000000000000000000000000030303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1:  a9b2554b09441180612120 98f2f238cd
C2:  779ea0aae3d9d020f3677bfcb3cda9ce
Valid: i = 0x40
CC1: 00000000000000000000004019f43ecb
P2:  000000000000000000000000ee030303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1:  a9b2554b09441180612120 98f2f238cd
C2:  779ea0aae3d9d020f3677bfcb3cda9ce
Valid: i = 0xea
CC1: 0000000000000000000ea431af73dc8
P2:  00000000000000000000ddee030303
[10/12/22]seed@VM:~/.../Labsetup 2$ python3 manual_attack.py
C1:  a9b2554b09441180612120 98f2f238cd
C2:  779ea0aae3d9d020f3677bfcb3cda9ce
Valid: i = 0x9d
CC1: 000000000000000009deb421bf63cc9
P2:  0000000000000000000ccddee030303
[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.



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. [Accessed: 14-Oct-2022].