PADDİNG ORACLE ATTACK

Boreas37

Table of Contents

Introduction-Impact of Vulnerability……………………….……………..3
Basic Technical Overview……………………………………………4
Mitigation Overview…………………………………………………6
Personal Reflections…………………………………………………7
Proof of Work..……………………………………………………8
Final…………………………………………………………………12

1-Introduction - Impact of Vulnerability

Padding Oracle Attack is an attack that uses the padding validation of an encrypted message to decrypt the ciphertext. Varying message lengths get padded to be compatible with the block cipher encryption algorithm, the attack depends on the Oracle server which responds to queries about whether a message is padded correctly or not. The attack is mostly associated with CBC mode decryption used with block ciphers.

Why is it important?

1. **Exploitation of Cryptographic Vulnerabilities**: Padding oracle attacks reveal flaws in the implementation of cryptographic protocols, enabling attackers to decrypt data without access to the encryption key.

2. **Widespread Applicability**: Systems that use block ciphers with padding schemes, such as TLS (Transport Layer Security), are vulnerable if not implemented correctly. This makes the attack relevant to many systems, especially web applications and APIs.

3. **Exposure of Sensitive Information**: By exploiting this attack, adversaries can retrieve encrypted data such as credentials, financial details, or confidential organizational communications.

4. **Highlighting Implementation Flaws**: It emphasizes the need for secure implementation of cryptographic protocols beyond the theoretical safety of encryption algorithms.

Impact on Organizations:

1. **Data Breach**: Sensitive data like customer details or intellectual property can be exposed.
2. **Financial Loss**: Fraud, fines, legal costs, and recovery expenses.
3. **Reputation Damage**: Loss of trust from customers and stakeholders.
4. **Operational Disruption**: Downtime for remediation affects business continuity.
5. **Regulatory Issues**: Non-compliance leads to fines and stricter oversight.
6. **Amplified Risks**: Opens doors for broader cyberattacks.

Impact on Users:

1. **Data Theft**: Personal and financial data can be stolen and misused.
2. **Financial Harm**: Fraudulent transactions or identity theft.
3. **Loss of Trust**: Confidence in affected organizations diminishes.
4. **Privacy Invasion**: Sensitive communications or personal details exposed.
5. **Targeted Attacks**: Data used for phishing or other malicious activities.

2- Basic Technical Overview

A. Cipher Block Chaining (CBC)

Block cipher mode of operation that provides information security such as confidentiality or authenticity. It is an algorithm that uses a block cipher to securely transform amounts of data larger than a block. In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point.

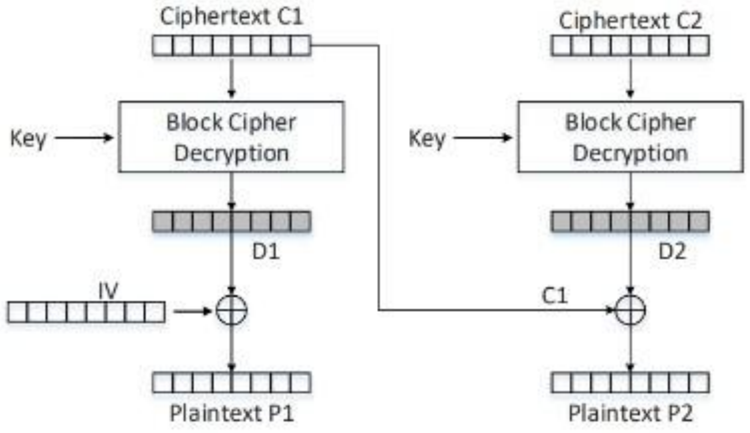
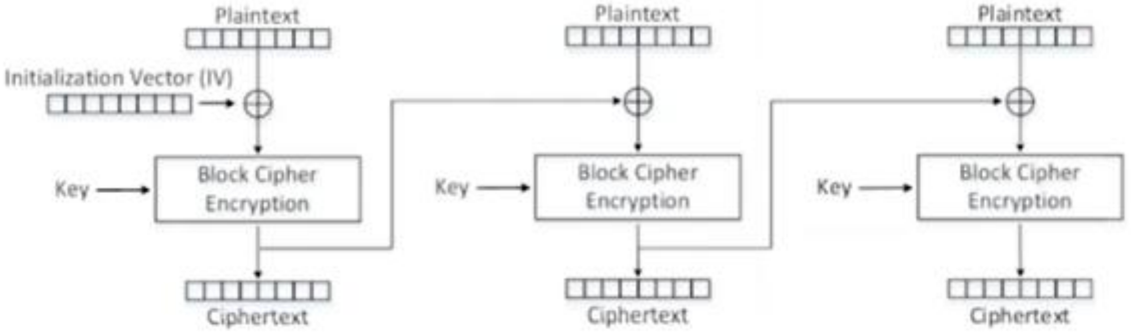


Figure 1: CBC Decryption Diagram



B. PKCS5 Padding

If the size of each block is denoted as B, the process involves adding N padding bytes to the input to reach the next multiple of B, taking into account the following conditions. When the input length is not an exact multiple of B, N padding bytes, each having the value of N, are appended. However, if the input length is already a multiple of B, B bytes of value B are added. This ensures that padding, ranging from one to B bytes in length, is consistently applied without any ambiguity. Once the decryption is complete, a verification step is performed by checking that the last N bytes of the decrypted data all have the value N, where N is greater than 1 but not exceeding B. If this condition is met, N bytes are removed; otherwise, a decryption error is thrown.

-Which Systems or Areas Padding Oracle Attack Affects

Padding oracle attacks primarily affect systems or areas that use block ciphers in encryption protocols, especially when these rely on CBC (Cipher Block Chaining) mode with padding schemes. Some specific systems and areas include:

1. Web Applications**:**

- Websites using encrypted cookies or tokens to manage authentication or session data.
- APIs that rely on encrypted payloads for secure communication.

2. Email Systems**:**

- Email encryption protocols (e.g., S/MIME) that use block ciphers susceptible to padding oracle attacks.

3. Cryptographic Libraries**:**

- Poorly implemented encryption libraries that expose padding validation errors can be targeted.

4. Secure Communication Protocols**:**

- **TLS/SSL (Transport Layer Security)**: Earlier versions like SSLv3 and TLS 1.0 are especially vulnerable.
- **VPN Protocols**: Some implementations relying on vulnerable encryption mechanisms.

5. File Encryption Systems**:**

- Systems that encrypt files or databases using CBC mode with padding, exposing vulnerabilities during decryption.

6. Payment Systems**:**

- Payment gateways and systems that use encrypted data for transactions.

7. IoT Devices**:**

- Devices with limited cryptographic implementations, often susceptible to padding-related flaws.

8. Legacy Systems**:**

- Older systems that haven't been updated to use secure encryption practices, such as authenticated encryption modes (e.g., AES-GCM).

By targeting these areas, attackers can decrypt sensitive data, compromise authentication mechanisms, or perform further exploitation.

Mitigation Overview

To protect against padding oracle attacks, organizations and developers should implement the following measures:

1. **Use Authenticated Encryption**:

Replace vulnerable encryption modes (e.g., CBC) with authenticated encryption modes like **AES-GCM** or **AES-CCM**, which provide both encryption and integrity protection.

2. **Validate Cryptographic Libraries**:

Ensure cryptographic libraries used in the system are up-to-date and have no known vulnerabilities. Use reputable libraries that follow modern standards, such as OpenSSL (latest versions).

3. **Avoid Detailed Error Messages**:

Do not return error messages that differentiate between padding and other decryption failures. Use generic error messages instead.

4. **Implement Robust Padding Validation**:

Ensure padding is validated securely and uniformly to avoid revealing any clues to attackers.

5. **Patch and Update Systems Regularly**:

Apply patches and updates provided by vendors to address known vulnerabilities in cryptographic protocols (e.g., upgrading from older versions of TLS to TLS 1.2 or 1.3).

6. **Use End-to-End Testing**:

Test applications for vulnerabilities like padding oracle attacks using security tools or services, such as penetration testing or vulnerability scanners.

7. **Encrypt Data Properly**:

Follow best practices for encryption, including key management and using recommended encryption algorithms.

8. **Enable Strong Security Configurations**:

Disable outdated or insecure protocols (e.g., SSLv3, TLS 1.0) in servers and applications.

9. **Monitor and Audit Systems**:

Continuously monitor systems for unusual behavior that could indicate padding oracle exploitation attempts.

10. **Educate Developers**:

Train developers to recognize cryptographic vulnerabilities and follow secure coding practices.

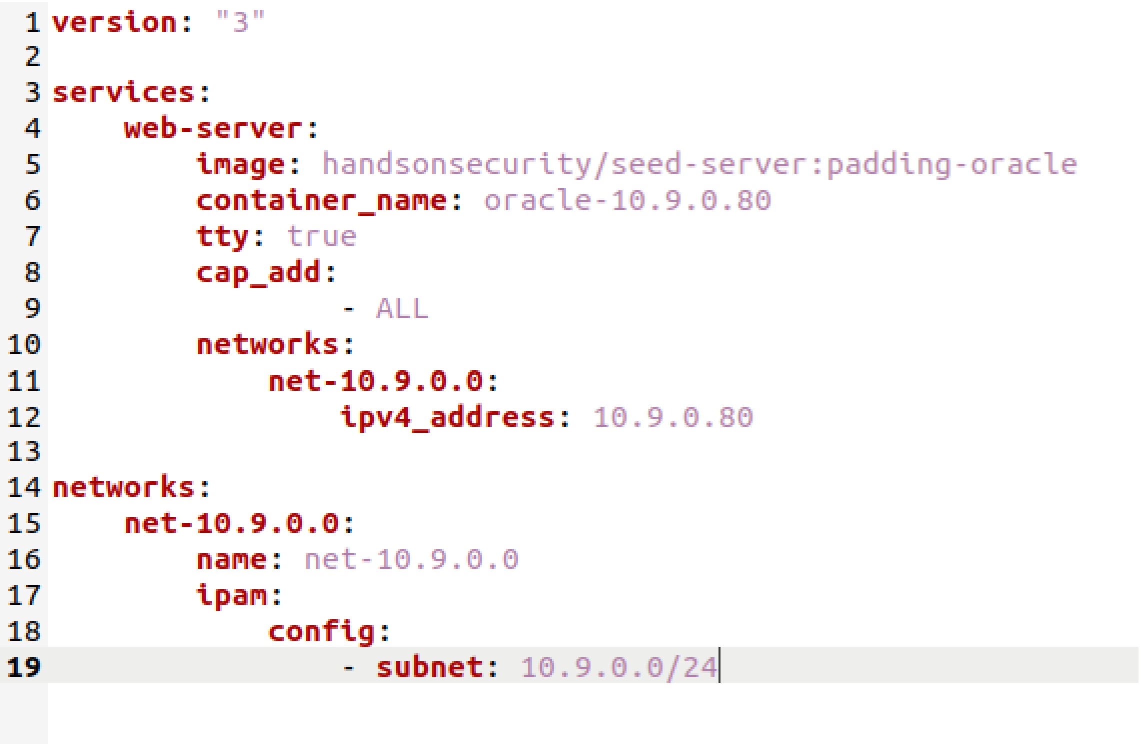
By implementing these steps and regularly reviewing cryptographic implementations, organizations can significantly reduce the risk of padding oracle attacks.

Personal Reflections

This project helped me understand how small weaknesses in encryption can lead to serious security problems. I learned the importance of using proper encryption methods and regularly updating systems to protect against attacks like the padding oracle. Protecting against these vulnerabilities is essential because they can expose sensitive data, harm users, and damage an organization’s reputation. This experience showed me how important it is to stay alert and take steps to prevent security risks.

Proof of Work

The first thing I did was navigating through lab setup file provided to us by SeedLabs to dockercompose.yml file which allows me to run a server on my Linux terminal.

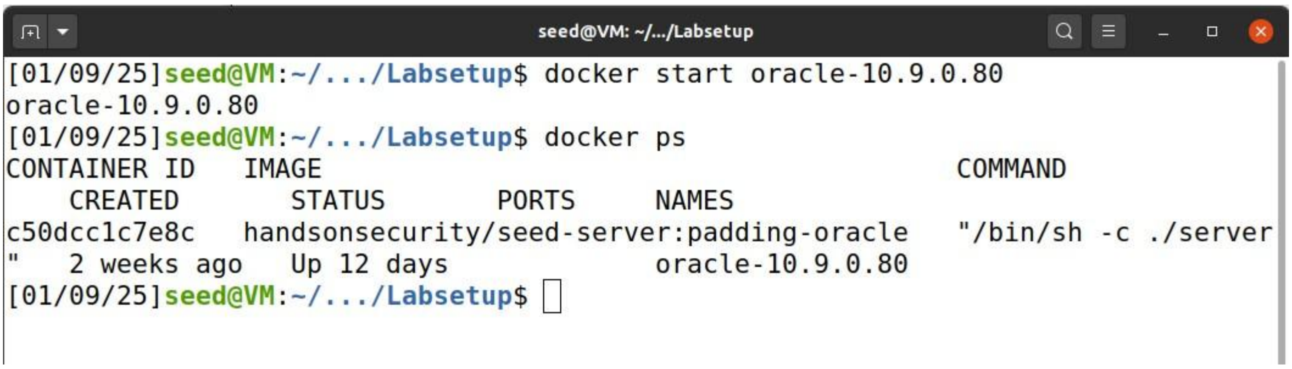


```
version: "3"

services:
    web-server:
        image: handsonsecurity/seed-server:padding-oracle
        container_name: oracle-10.9.0.80
        tty: true
        cap_add:
                - ALL
        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.80

networks:
    net-10.9.0.0:
        name: net-10.9.0.0
        ipam:
            config:
                - subnet: 10.9.0.0/24
```

After finding this file i started docker through my terminal.



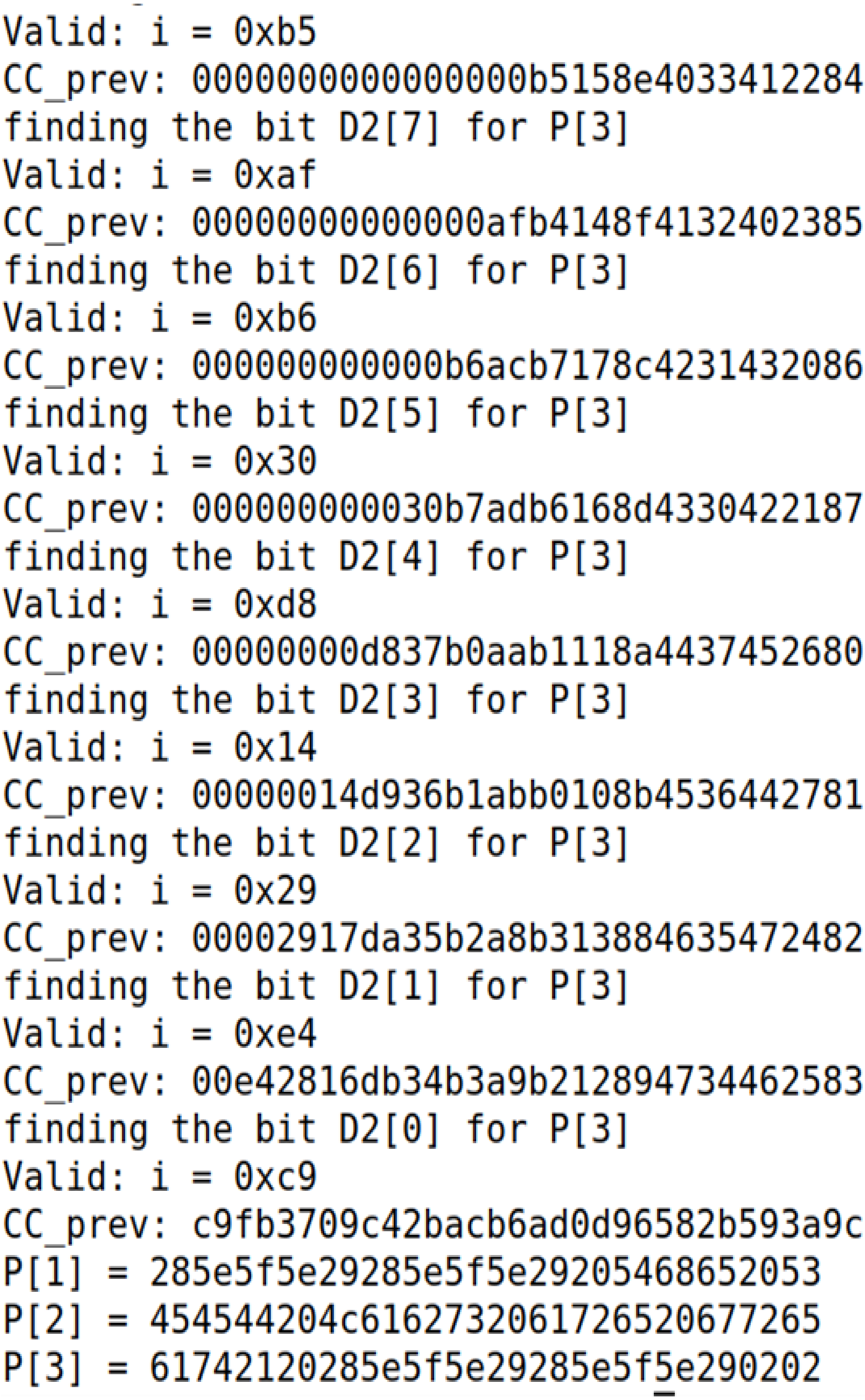
seed@VM: ~/.../Labsetup

```
[01/09/25]seed@VM:~/.../Labsetup$ docker start oracle-10.9.0.80
oracle-10.9.0.80
[01/09/25]seed@VM:~/.../Labsetup$ docker ps
CONTAINER ID   IMAGE                                         COMMAND
    CREATED        STATUS        PORTS      NAMES
c50dcc1c7e8c   handsonsecurity/seed-server:padding-oracle   "/bin/sh -c ./server
"    2 weeks ago    Up 12 days              oracle-10.9.0.80
[01/09/25]seed@VM:~/.../Labsetup$ 
```

The aftermath was relatively easy compared to this part, despite seeming so easy at above with zero knowledge on topic it was hard to even start docker.

Screenshot doesnt fit all of the output of the python scricpt so im gonna paste the code I run here:

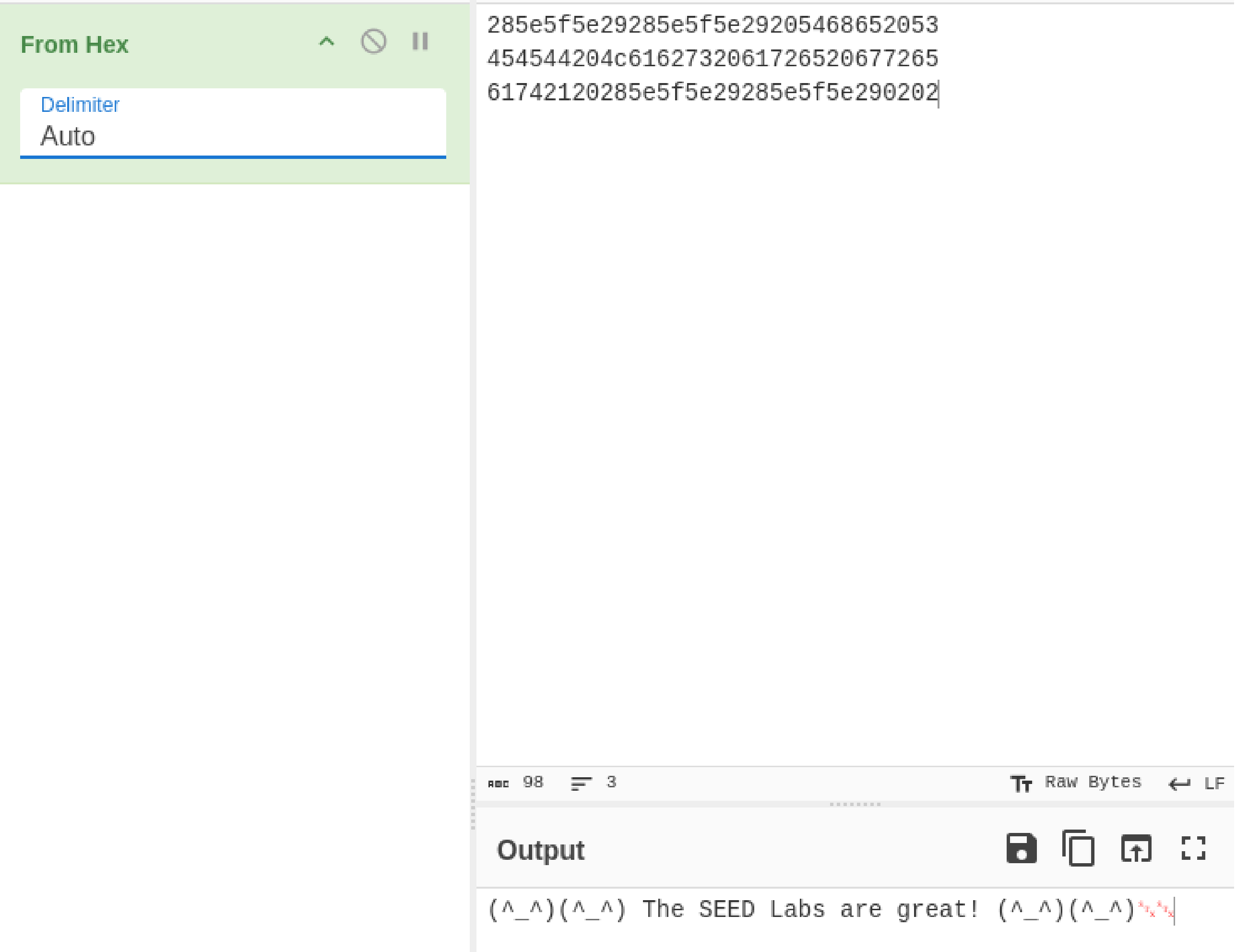
[01/09/25]seed@VM:~/.../Proje$ python3 automatedAttack.py 5000



```
Valid: i = 0xb5
CC_prev: 0000000000000000b5158e4033412284
finding the bit D2[7] for P[3]
Valid: i = 0xaf
CC_prev: 00000000000000afb4148f4132402385
finding the bit D2[6] for P[3]
Valid: i = 0xb6
CC_prev: 000000000000b6acb7178c4231432086
finding the bit D2[5] for P[3]
Valid: i = 0x30
CC_prev: 000000000030b7adb6168d4330422187
finding the bit D2[4] for P[3]
Valid: i = 0xd8
CC_prev: 00000000d837b0aab1118a4437452680
finding the bit D2[3] for P[3]
Valid: i = 0x14
CC_prev: 00000014d936b1abb0108b4536442781
finding the bit D2[2] for P[3]
Valid: i = 0x29
CC_prev: 00002917da35b2a8b313884635472482
finding the bit D2[1] for P[3]
Valid: i = 0xe4
CC_prev: 00e42816db34b3a9b212894734462583
finding the bit D2[0] for P[3]
Valid: i = 0xc9
CC_prev: c9fb3709c42bacb6ad0d96582b593a9c
P[1] = 285e5f5e29285e5f5e2920546865205 3
P[2] = 454544204c6162732061726520677265
P[3] = 61742120285e5f5e29285e5f5e290202
```

As we can see above the code tries every character combination on crypted message until it gets a positive answer from the server, doing this to all encrypted characters gives us the unencrypted message.

With this code I attacked the docker server i created at the port 5000 and the output was p[1-3]. With hexadecimal to plaintext tools I found on web I decrypted the message:



From Hex

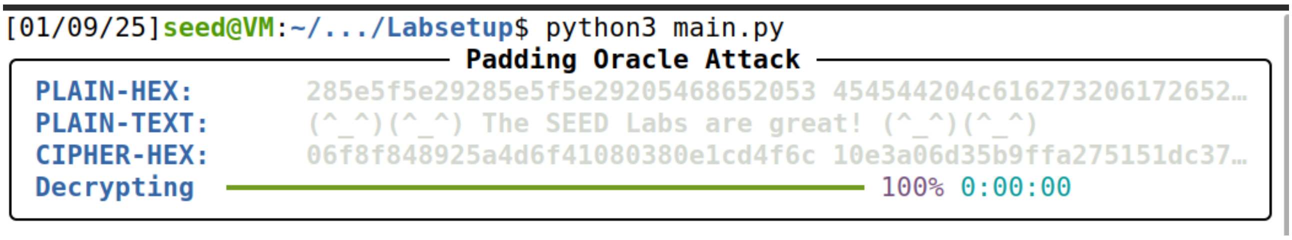
Delimiter
Auto

```
285e5f5e29285e5f5e29205468652053
454544204c6162732061726520677265
61742120285e5f5e29285e5f5e290202
```

Output

```
(^_^)(^_^) The SEED Labs are great! (^_^)(^_^)
```

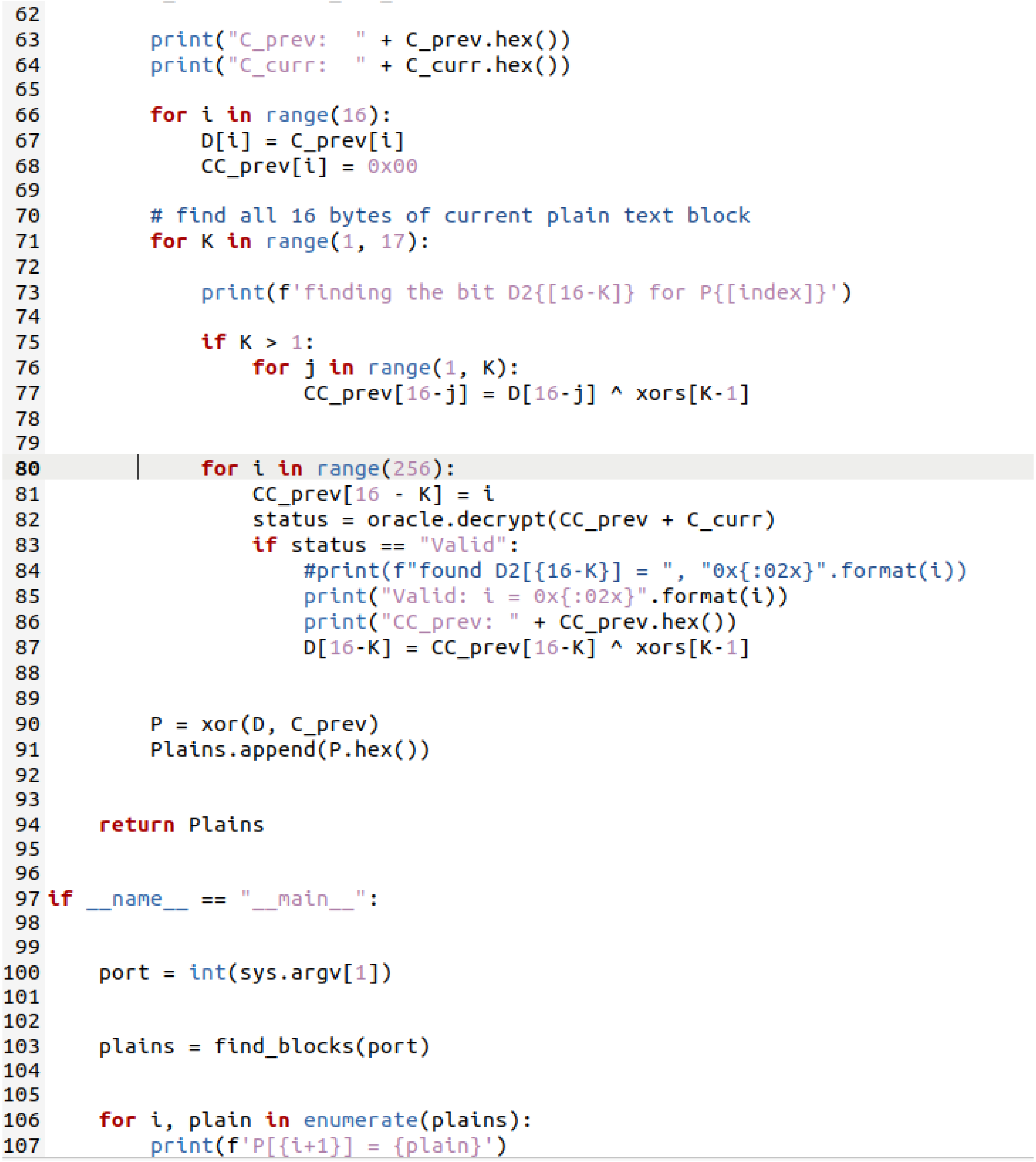
Also while surfing on internet I found another interesting code related to this lab:



```
[01/09/25]seed@VM:~/.../Labsetup$ python3 main.py
 ──────────────────── Padding Oracle Attack ────────────────────
  PLAIN-HEX:       285e5f5e29285e5f5e29205468652053 454544204c616273206172652…
  PLAIN-TEXT:      (^_^)(^_^) The SEED Labs are great! (^_^)(^_^)
  CIPHER-HEX:      06f8f848925a4d6f41080380e1cd4f6c 10e3a06d35b9ffa275151dc37…
  Decrypting  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:00
```

This code gave me the answer straight away without needing to solve hex.

As for the code i used, because that main.py is so long due to visualization lib, im going to give automatedAttack.py files codes:



```
        print("C_prev:  " + C_prev.hex())
        print("C_curr:  " + C_curr.hex())

        for i in range(16):
            D[i] = C_prev[i]
            CC_prev[i] = 0x00

        # find all 16 bytes of current plain text block
        for K in range(1, 17):

            print(f'finding the bit D2{[16-K]} for P{[index]}')

            if K > 1:
                for j in range(1, K):
                    CC_prev[16-j] = D[16-j] ^ xors[K-1]


            for i in range(256):
                CC_prev[16 - K] = i
                status = oracle.decrypt(CC_prev + C_curr)
                if status == "Valid":
                    #print(f"found D2[{16-K}] = ", "0x{:02x}".format(i))
                    print("Valid: i = 0x{:02x}".format(i))
                    print("CC_prev: " + CC_prev.hex())
                    D[16-K] = CC_prev[16-K] ^ xors[K-1]


        P = xor(D, C_prev)
        Plains.append(P.hex())


    return Plains


if __name__ == "__main__":


    port = int(sys.argv[1])


    plains = find_blocks(port)


    for i, plain in enumerate(plains):
        print(f'P[{i+1}] = {plain}')
```

Final

That was all of my documentation, thank you for reding this far.