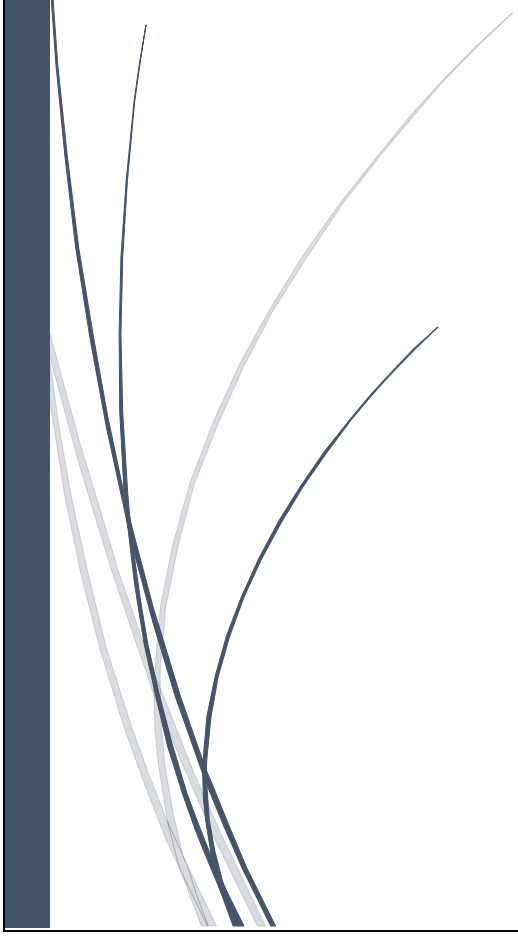
Assignment 3 - Report

COL759: Cryptography & Computer Security

ENTRY NO. 2019SIY7580

Aman Bhardwaj
SIT, IIT DELHI

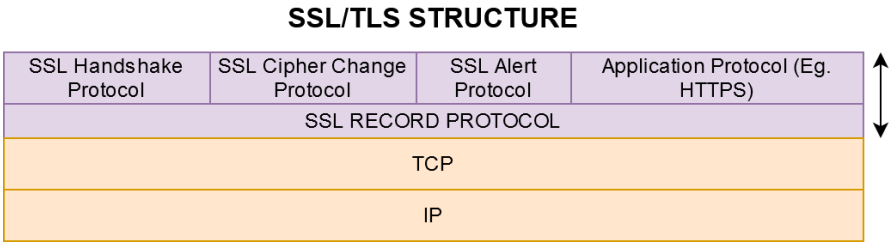


A. SSL/TLS PROTOCOL EXPLAINED

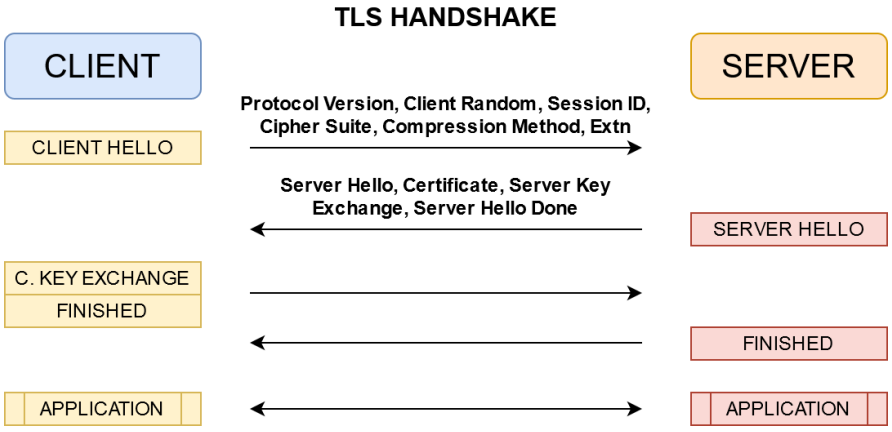
INTRODUCTION: A lot of data being transmitted over the Internet consists of sensitive information. **SSL/TLS (Secure Sockets Layer/Transport Layer Security),** is one of the major secure cryptographic communications protocols on the Internet, used by a variety of applications such as web browsers, electronic mail, voice over-IP and more. Typically, when a service uses a secure connection the letter **S** is appended to the protocol name, for example, HTTP**S**, SMTP**S**, FTP**S**, SIP**S**. In most cases, SSL/TLS implementations are based on the **OpenSSL library.**

SSL/TSL HISTORY: SSL was first introduced **in 1994 by Netscape**, later SSLv2.0 in 1995 and SSLv3.0 in 1996. Which was replaced by TLS 1.0 in 1999. **TLS 1.2** is the latest version.

SSL/TLS STRUCTURE: It is an agile protocol that allows peers to negotiate their highest supported protocol version, as well as the **combination of ciphers** used in a session. It is located above the transport layer in the TCP/IP layout.



TLS HANDSHAKE: The Handshake protocol is used to **negotiate security parameters** of a TLS session. It consists of various messages which are shown in following figure:



DURING HANDSHAKE: following things are achieved.

i. Validation of Identity of both the parties. And **Session ID** is generated.
ii. Decision is made on following things: **Protocol version, Cipher Suite (to encrypt the messages), Compression Method** which will be used for the communication.
iii. Then **Certificates and Secret key** is exchanged.
iv. And the Handshake finished message exchange signifies a **secure connection has been established** and applications can communicate securely by using above parameters which were discussed during the handshake.

POST HANDSHAKE: After the process of negotiation of parameters and generating of necessary, from the application level Record protocol takes the data, breaks it into smaller chunks, optionally compresses it, applies the agreed hash function, encrypts it with the agreed algorithm. Then sends it through the transport layer to the other side in communication.

B. VULNERABILITY EXPLAINED: PADDING ORACLE ATTACK

VULNERABILITY EXPLOITED:

CBC Cipher Block Chaining Mode along with **PKCS7** Public Key Cryptography Standards 7 padding to complete the last block of message, used for **AES128 Cipher** used for SSL/TLS Protocol.

In network protocols best practice is to send a informative error message to clients. **Example**: **SMTP** gives "Unable to deliver because no such mailbox exists" lets the sender know that they have misaddressed an email, **HTTP** server gives **404** error "Page not found" or **401** error "Access denied".

In cryptography, this is **not a good idea**. It leaks information about if a block of cipher text is correctly padded or not. It can enable us to decipher the whole text.

PKCS7 PADDING EXPLAINED: Public Key Cryptography Standards 7

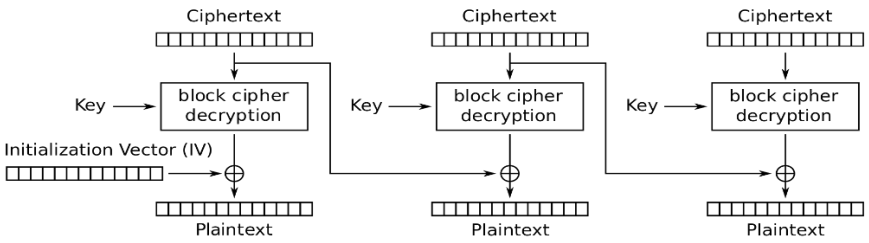
As per PKCS7, the text block needs to be in a complete byte by-octet i.e. each block should have 16 bytes. If the last message falls short to complete an octet, then the bytes need to be added to complete the same. The value of each added byte is equal to the total number of bytes being added to complete the octet.

Example: Cipher Text: CIPHER_TEXT ----------> It has **11 Bytes**.

Block 1: C | I | P | H | E | R | _ | T | E | X | T ----------> **11 Bytes. Therefore, short of 05 bytes to complete 16 Bytes block**

Padded Block 2: C | I | P | H | E | R | _ | T | E | X | T **| 05 | 05 | 05 | 05 | 05 ------->** 5 bytes added with value = 05

CBC DECRYPTION: *Block Diagram Reference: Wikipedia.*



Cipher Block Chaining (CBC) mode decryption

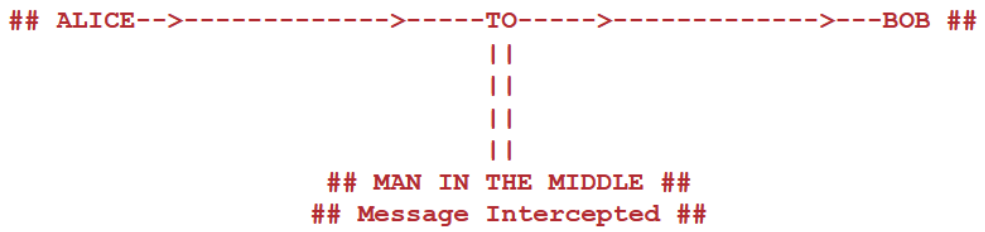
Standard CBC Decryption performs following tasks

- **Decrypt** all blocks of cipher text
- **Validate** PKCS7 Padding and return error message in case of padding invalid.
- **Remove** Padding
- **Return** Plain Text Message

VULNERABILITY: After validation of PKCS7 Padding. Some servers implement **return** message as "**Invalid Padding or 404 not found etc".** This lets the attacker know that the padding is not correct.

EXPLOIT: Attackers exploit such servers for decrypting the cipher text and sometimes encrypt the mocked message and send it to the server to mislead it. This vulnerability can be very disastrous, as the attacker might be able to decipher the encryption and 100% of plain text can be retrieved from the same.

C. ATTACKER: ON PAPER IMPLEMENTATION OF STEPS TO EXPLOIT



STEP 1 - INTERCEPTION: A message being communicated from Alice to Bob needs to be intercepted by the attacker.

STEP 2 - ASSUMPTION: The intercepted message is apparently a Cipher Text, encrypted via AES128 which implements CBC mode with PKCS7 Padding being communicated via SSL/TLS Protocol.

STEP 3 – PADDING ORACLE ATTACK LAUNCHED:

Prepare text blocks: divide the cipher text into blocks of 16 bytes as C_0 **|** $C_{...}$ **|** C_{i-1} **|** C_i

C_0 **=** First block**,** C_{i-1} **=** Second Last Block, C_i **=** Last Block

STEP 4 – IDENTIFY THE CORRECT PADDING (First Byte from the end): Note: demonstration done on block size of 8 bytes. Implementation has been done on block size of 16 bytes.

This attack will be carried out from last byte to first byte of each block.

- Then prepare dummy block like:

  $C'_{i-1} = C_{i-1} \oplus 00000001 \oplus 0000000X \mid C_i$

  X = Character (Range 0 - 255)

- Send $C'_{i-1} \mid C_i$ to the Server (**Oracle**). And Server will try to decrypt this by following:
  $D_k(C_i) \oplus C'_{i-1}$
  $= D_k(C_i) \oplus C_{i-1} \oplus 00000001 \oplus 0000000X$
  $= P_i \oplus 00000001 \oplus 0000000X$
  Middle block is 00000001 which signifies 1 byte as padding.

- Brute force server by *0-255* values of X one by one.
  If Server returns decrypted some message: Our mocked byte is correct. Now we have the padding value, or the first byte has been deciphered.
  If Server returns error message: Then our mocked byte is incorrect. Continue with the brute force.

- Now let us consider that we have the first byte known. Say **X.**
- If server sends error message for each try or does not blocks the brute force, then we will get the correct byte for sure.

STEP 5 - DECIPHER THE SECOND BYTE:

- This can be done in the same way as we have done for the first byte.
- Here the dummy cipher block will be prepared as follows
- Fix the Byte **X** and the for the second byte we will apply brute force on Byte **Y**
  $C'_{i-1} = C_{i-1} \oplus 00000022 \oplus 000000YX \mid C_i$

- Send C'_{i-1} | C_i to the Server (**Oracle**). And Server will try to decrypt this by following:

$$D_k(C_i) \oplus C'_{i-1}$$
$$= D_k(C_i) \oplus C_{i-1} \oplus 00000022 \oplus 000000YX$$
$$= P_i \oplus 00000001 \oplus 00000YX$$

  Here second block is 00000022. This implies we are using 2 bytes as padding.
- Brute force server by *0-255* values of Y one by one.
  If Server returns decrypted some message: Our mocked byte is correct. Now we have the padding value, or the first byte has been deciphered.
  If Server returns error message: Then our mocked byte is incorrect. Continue with the brute force.
- We will get byte **Y,** by this attack.

STEP 6 – REPEAT STEP 4-5 FOR ALL 16 BYTES OF TARGET BLOCK:

By now we should be having the target block deciphered.

STEP 7 – REPEAT STEP 4-6 FOR ALL THE BLOCKS:

Repeat the steps 4-6 for all the message blocks of the intercepted cipher text.

By now we have all the blocks deciphered.

STEP 8 – RETRIEVE FULL PLAIN TEXT:

Join all the blocks of plain text retrieved by above attack and append them in order to retrieve the whole plain text message being sent from Alice to Bob.

Below is the snippet from the console output.

```
$$$$$$********************---PADDING ORACLE ATTACK INITIATED---**********************$$$$$$


   ------------------------------# LET THE HACK BEGIN #---------------------------------

Total Blocks in Cipher Text = 33

[+]--- Cracking Block Number:  0
    Deciphered Block:  Harry Potter is

[+]--- Cracking Block Number:  1
    Deciphered Block:  a British Americ

[+]--- Cracking Block Number:  2
```

.

.

.

```
$$$$$$********************---PADDING ORACLE ATTACK SUCCESSFUL---********************$$$$$$


CRACKED MESSAGE BY MAN IN THE MIDDLE:

  b'Harry Potter is a British American film series based on the eponymous novels by author J. K. Rowling. The series
is distributed by Warner Bros. and consists of eight fantasy films, beginning with Harry Potter and the Philosophers
Stone (2001) and culminating with Harry Potter and the Deathly Hallows Part 2 (2011). A spin-off prequel series will
consist of five films started with Fantastic Beasts and Where to Find Them (2016), marking the beginning of the Wizar
ding World shared media franchise.'
```

WORST CASE COMPLEXITY OF ATTACK

Worst case complexity for the Padding Oracle Attack on AES128 system implemented on CBC Mode PKCS7 Padding = **Number of Bytes in Cipher Text * 256**

(2^8 = 256 guesses per byte)

It can be seen that this is cost efficient. So, it does not require any high-performance machine to implement this attack.

CYBER-SECUTIRY ANALYST: HOW TO PREVENT PADDING ORACLE

1. While implementation of the server, make sure it does not return error responses such as "Padding error", "MAC error", "decryption failed" in case of invalid padding in the cipher text.
2. Server protection against brute force. Block too many invalid calls from a particular client.
3. Encrypt and MAC your data

D. ABOUT CODE IMPLEMENTATION:

File Name: padding_oracle_attack.ipynb

How to Run:

- Open the jupyter notebook.
- Just run the notebook and simulation can be seen in the console below.
- I have hard coded the plain text message in the main(). So, no need to give any input.
- I have added **sleep time of 0.5 sec** after each step, so that results could be seen as simulation in the console.

IMPLEMENTATION OF CODE EXPLAINED:

I have implemented my code in **3 classes** in order to implement abstraction between CLIENT, SERVER and ATTACKER.

CLASS IMPLEMENTED:

1. **CLIENT_AES:**
   CLIENT - Encrypts message by AES Cipher and get it ready to be transmitted via secured channel. This class encodes and encrypts the plain text message from the client and sends to the server via SSL/TLS.
   Can be accessed only by Alice (Client)
2. **SERVER_AES:**
   SERVER - Receives the message via SSL/TLS and decrypts it by AES128 CIPHER.
   Can be called by both client and attacker.
3. **PADDING_ORACLE_ATTACK:**
   ATTACKER - This class implements function to initialize and facilitate Padding Oracle Attack by Man In The Middle.
   Can be accessed only by the attacker.

FUNCTIONS IMPLEMENTED IN EACH CLASS:

1. **CLASS: CLIENT_AES:**
   - **_pad (self, message):**
     Function to complete the block size (16) for the message blocks. If in case the last block is shorter than 16 Bytes then it adds padding as per PKCS7 Encoding guidelines.
     params:
     self: Class reference
     message: plain text message (Bytes)
   - **encrypt (self, message):**
     Once the message has been encoded with PKCS7 guidelines. It is then encrypted here by AES128 CIPHER. **MODE OF OPERATION**: CBC - Cipher Block Chaining **PRIVATE KEY**: communicated b/w client and server during handshake.
     params:
     self: Class reference
     message: plain text message (Bytes) to encrypt
     return: Encrypted text

2. **CLASS: SERVER_AES:**
   - **_unpad (self, message):**
     Function to remove the extra padding from the message once it has been decoded.
     params:
     self: Class reference
     message: cipher text message (Bytes)
   - **decrypt (self, message):**
     Once the crypted message has been received via SSL/TLS. It is then decrypted here by AES128 CIPHER.
   - **check_padding (self, message):**
     This function checks if the message received from the client has been properly encoded with Block size 16 as per PKCS7 Encoding.
     params:
     self: Class reference
     message: crypted text message (Bytes)
     return: True if encoding is correct / False if encoding is wrong
3. **CLASS: PADDING_ORACLE_ATTACK:**
   - **man_inthe_middle_init (self, en_text):**
     Initialize Padding Oracle Attack:
     params:
     self: Class reference
     en_text: Intercepted crypted text message (Bytes)
   - **padding_oracle_attack(self, en_text):**
     Implements Padding Oracle Attack: Exploits server's check_padding function and PKCS7 encoding if server doesn't have a firewall which blocks frequent calls from an IP Address. Here MITM Acts as a pseudo client and exploits the above vulnerability SSL/TLS security which uses AES Cipher operating in CBC Mode.
     params:
     self: Class reference **en_text:** Intercepted crypted text message (Bytes)
     return: Cracked Cipher Text

4. **NON-CLASS FUNCTIONS:**
   - **main():** Initialize the secure communication between Alice and Bob.
   - **secure_communication(message):**
     Facilitate Secure communication between Client and Server, via SSL/TLS using AES128 in CBC Mode. This function:
     Divides message into blocks, add pad, encrypts the plain text.
     Then send it via Secured SSL/TLS channel from client to server.
     At receivers end it receives the cipher text. Divides into blocks. Validate padding and then decrypts the message.
     All these things have been implemented by the below partial functions
     - **_encrypt_message(aes, message):**
       Encrypts Plain text, on client side using CLIENT_AES class functions.
     - **_send_message():**
       Sends encrypted plain text via secure channel
     - **_decrypt_message(aes, c_text):**
       Receives and decrypts cipher text and returns plain text using SERVER_AES CLASS functions.

CONTACT INFO:

Phone: +91-9882305248

Email: siy197580@cse.iitd.ac.in

****