COL 759

Assignment-3 Report

Pushpam Anand (2016CS10347)

POODLE VULNERABILITY IN SSL v3

I have simulated the POODLE attack in ssl v3 protocol. For the proof of concept, I have shown how an attacker who has access to the encrypted message can use this vulnerability to decode the message.

1. What is SSL

SSL (Secure Sockets Layer), is an encryption-based Internet security protocol, developed for the purpose of ensuring privacy, authentication, and data integrity in Internet communications.

Features

a. **Data Encryption :** In order to provide a high degree of privacy, SSL encrypts data that is transmitted across the web. Anyone intercepting the data will not get the ciphertext, impossible to decrypt.
b. **SSL Handshake :** SSL initiates an authentication process called a handshake between two communicating devices to ensure that both devices are really who they claim to be.
c. **Data Integrity :** SSL also digitally signs data in order to provide data integrity, verifying that the data is not tampered with before reaching its intended recipient.

2. SSL v3.0 Protocol

Any cryptographic protocol like SSL consists of a encryption and a decryption function. SSLv3.0 uses something called **MAC** (**M**essage **A**uthentication **C**ode) verification process to ensure data integrity in the communication.

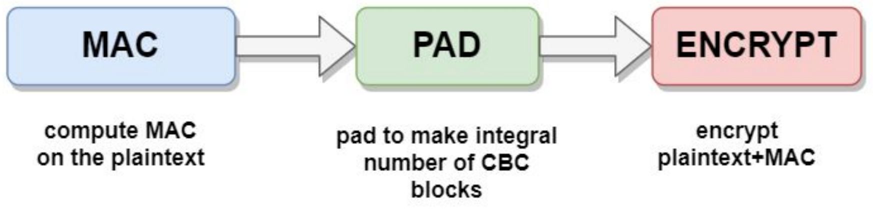
MAC : When one sends encrypted data, one does encryption as well as calculate keyed hash (HMAC) using MAC secret. This is done to avoid tampering with encrypted data.

Encryption :

- SSLv3 encryption uses the **MAC-then-encrypt** method, which means that the MAC of the plaintext is calculated and appended to the plaintext before encrypting it.

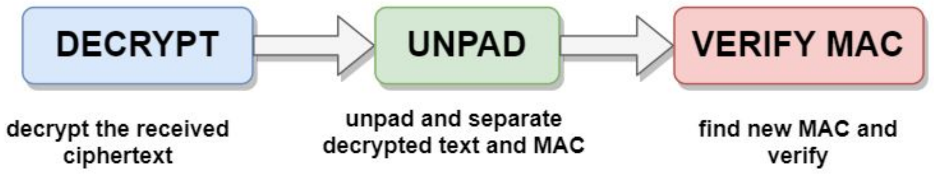
- **CBC mode of encryption :** Cipher Block Chaining (CBC) mode is a block mode of DES that XORs the previous encrypted block of ciphertext to the next block of plaintext to be encrypted

- When SSL 3.0 is used with a block cipher CBC mode of encryption, the SSL implementations must add padding bytes to make the total data (that includes the plaintext and the MAC of the plaintext) integral number of cipher blocks in length


MAC
PAD
ENCRYPT
compute MAC on the plaintext
pad to make integral number of CBC blocks
encrypt plaintext+MAC

Decryption :

- Decryption occurs at the receiver's side, when the ciphertext is received.
- First, decryption of the ciphertext occurs, after padding blocks have been removed, using information from last padding byte.
- MAC and the decrypted text are splitted apart.
- Then the new MAC on the decrypted text is calculated.
- The new and the old MAC are matched to verify the integrity of the message, as if the message had been tampered with then the new MAC would not match with the received MAC.


DECRYPT
UNPAD
VERIFY MAC
decrypt the received ciphertext
unpad and separate decrypted text and MAC
find new MAC and verify

3. POODLE vulnerability explanation

POODLE stands for (**P**adding **O**racle **O**n **D**owngraded **L**egacy **E**ncryption). In this vulnerability, an attacker has to be a Man-in-the-Middle(MiTM).
When cipher block chaining is combined with SSLv3.0, padding is required to make the cipher blocks of constant size. This attack exploits the padding method to get access to 1 character from the message at a time.

3.1 Attacker must have following capabilities:

1. the attacker must be able to control portions of the client side of the SSL connection (varying the length of the input) and
2. the attacker must have visibility of the resulting ciphertext.

These are achieved by acting **Man in the Middle**

3.2 Design Flaw in SSL v3:

The attack is possible due to two SSL design flaws:

1. **SSL 3.0 uses the MAC-THEN-ENCRYPT** mode of authenticated encryption. This means a receiver has to decrypt the message before applying the MAC integrity check. This allows an attacker to change the ciphertext to launch active attacks to learn more about the plaintext.
2. **SSL 3.0 padding definition** : When SSL 3.0 is used with a block cipher CBC mode of encryption, it adds padding bytes to make the total data (that includes the plaintext and the MAC of the plaintext) integral number of cipher blocks in length. SSL 3.0 defines only the last byte of the padding. It does not define what the rest of the bytes of the padding need to be. And therein lies the second design problem.

3.3 Implementation of the attack:

1. The attacker tweaks the plaintext message (**P**) to change the size. Note that, in SSLv3, if the plaintext message and the MAC are exactly integral number of blocks in length, then a full padding block is added. This is exploited by the attacker as he knows the last byte of the padding block now (since full block is added, last byte is 15)

2. Once encrypted with a key *K* and an initialization vector *IV*, the plaintext message will be converted into a series of ciphertext blocks. Lets call the encrypted message **E,** the cipher blocks as $\mathbf{C_1, C_2, \ldots C_n}$

3. we know that the last ciphertext block, when decrypted, will yield the padding block (whose last byte has a value of 15).
   This fact is exploited as follows:
   a. Suppose, attacker decides to decode second block ($\mathbf{C_2}$). We send **E,** with the last block (the padding block, $\mathbf{C_n}$) replaced by $\mathbf{C_2}$**.**
   b. Since, the new decryption of the last block does not guarantee that correct MAC is extracted. (as MAC location in **E** is decided by the padding length , which is decided by the last byte of **E**).
   c. But, if the process is repeated, there is 1 in 256 chance that the last byte of modified E is decrypted to 15 (the correct padding length). In that case the MAC verification will be validated.
   d. And the attacker gains the information that last byte of **E** (last byte of $\mathbf{C_2}$), is decrypted to 15. Using this information we can find the corresponding plaintext byte, using the formula :

$$P_2[15] = C_{n-1}[15] \ \mathbf{xor} \ C_1[15] \ \mathbf{xor} \ 15$$

4. We can rotate the plaintext **P** to place previous byte at the last position of $\mathbf{P_2}$, and repeat the above steps, and finally decode the whole block $\mathbf{C_2}$**,** and the whole plaintext.

3.4 Safeguards:

Disabling SSL 3.0 support in system/application configurations is the most viable solution currently available.

1. Disable SSL 3.0 support from browsers.
2. Disable SSL 3.0 support in the web server.
3. Both clients and servers need to support TLS_FALLBACK_SCSV to prevent downgrade attacks.

4. My Simulation of POODLE attack

1. Since we needed a proof of concept of attack, the simulation is done without involving explicit server, client, and attacker with SSL handshakes between ports.

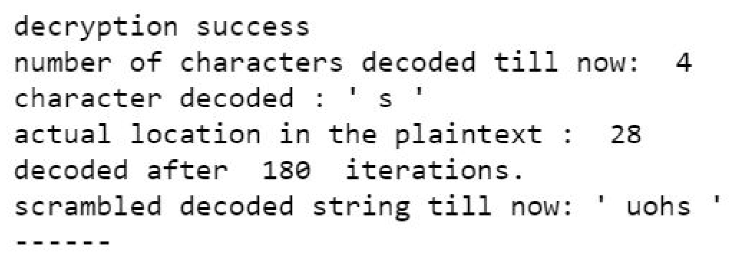
2. The library used for encryption function and MAC generation is "pycryptodome"

3. The overall design of the simulation, based on the idea provided by an instructor on piazza, is as follows:
   a. Generate a random message **m**, and random key
   b. Tweak the message length so that it becomes an integral number of blocks (for CBC encryption). This is required as it enables the POODLE attack by giving info about the last byte of padding block (which is 16, as block size is 16).
   c. Define an encryption function
   d. Define a decryption function, such that it returns success if MAC checksum is verified, else not.
   e. Encrypt **m** using the SSLv3 protocol. Let the encrypted message be **e.**
   f. modify **e** according to the algorithm discussed above in section 3.3. In the code, I have fixed the last byte of second block of **e** (**C2**) to be computed.
   g. Keep modifying the message "e" and send the modified message to the Decrypt function until a success code is received and a character is decrypted.
   h. Keep rotating the plaintext one character at a time, such that new character to be decoded comes as the last byte of C2, and is decoded one by one. And finally the full plaintext is decoded.

4. More detailed description about the functions and the exact implementation, is given as comments in the code itself.

About the output on command line:

1. The output of decoding of one character looks like:



```
decryption success
number of characters decoded till now:  4
character decoded : ' s '
actual location in the plaintext :  28
decoded after  180  iterations.
scrambled decoded string till now: ' uohs '
------
```

Here **iterations** is the number of times the plaintext is sent for encryption/decryption before that decryption is finally successful and the character is decoded.

2. Similar output is given for each character decoded.
3. Then the whole string is printed, which is not arranged properly as characters were not decoded in correct order.
4. Finally rearranged string is printed at the last.

How to Run :

1. The code can be simply run by the typing the following command on cmd:

   python assignment_3.py

   It shows the simulation of attack on the default message
   msg = "hello this is secret and it should not be disclosed in any situation."

2. To run on any other plaintext:
   change the line:102 in the file "assignment_3.py" as:
   msg = any string you want with length > 32

   The plaintext length should be at least greater than 32, as in the attack, character at the index 32 is decoded one-by-one.