Compromising sensitive information through Padding Oracle and Known Plaintext attacks in Encrypt-then-TLS scenarios

Daniel Espinoza Figueroa

Department of Electronic Engineering,
Universidad Técnica Federico Santa María, Chile
daniel.espinozaf@sansano.usm.cl

November 20, 2023

Abstract

Let's consider a scenario where the server encrypts data using AES-CBC without authentication and then sends only the encrypted ciphertext through TLS (without IV). Then, having a padding oracle, we managed to recover the initialization vector and the sensitive data, doing a cybersecurity audit for a Chilean company.

1 Introduction

In the World Wide Web, Transport Layer Security (TLS) is the well-known cryptosystem that handles all the interactions between different services on the Internet, providing Confidentiality, Integrity, and Authentication as a complete cryptographic protocol [1]. Depending on the implementation, some services may require encrypting data even if TLS is active, namely the Encrypt-then-TLS approach, due to end-to-end encryption solutions or some specific requirements, where data is encrypted at the application layer [2]. Having advantages and disadvantages, some implementations don't provide authentication, so even if we are protected with TLS to prevent Man in the Middle (MitM) attacks, users can modify their HTTP requests on the way. That was the case for a web application audited for a well-known Chilean company, where the AES-CBC encryption algorithm was used. With a padding oracle and a known plaintext attack, we recovered the initialization vector considered private and sensitive information, namely a card number.

2 Padding Oracle attack in AES-CBC

Encrypting with AES in Cipher Block Chaining (CBC) mode works by making each block influence the next encrypted block through the following formula:

$$C_i = \mathrm{AES}_k(P_i \oplus C_{i-1})$$

where P_i is the plaintext and C_i the ciphertext. Decryption is computed as

$$P_i = \mathrm{AES}_k^{-1}(C_i) \oplus C_{i-1}$$

If authentication is not presented, an attacker can modify C_{i-1} in order to change P_i directly. Also, padding is used because the information has to be a multiple of 16 bytes in length, typically using PKCS #7 [3]. This padding scheme works as follows:

- Consider the following message: "D-Cryp7 was here!" with 17 bytes long. For the message to be a multiple of 16 bytes long, we have to append 32 - 17 = 15 bytes with the value "15" in hex (0f). Then, the padded message will be

  "D-Cryp7 was here!\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f\x0f"

Even if the message is a multiple of 16 bytes, the message will be padded, adding 16 bytes of "16" in hex (10). As an attacker, we send the encrypted message and the server decrypts it, returning an error if the padding cannot be removed. Modifying a message will return a padding error, as noted in Figure 1, but for the last byte, we can change it until the server removes the padding of the modified message correctly. In that case, the last byte will be \x01, and we will recover the last byte of $\mathrm{AES}_k^{-1}(C_i)$ and P_i. We continue changing the next byte searching for a \x02 and so on. Of course,

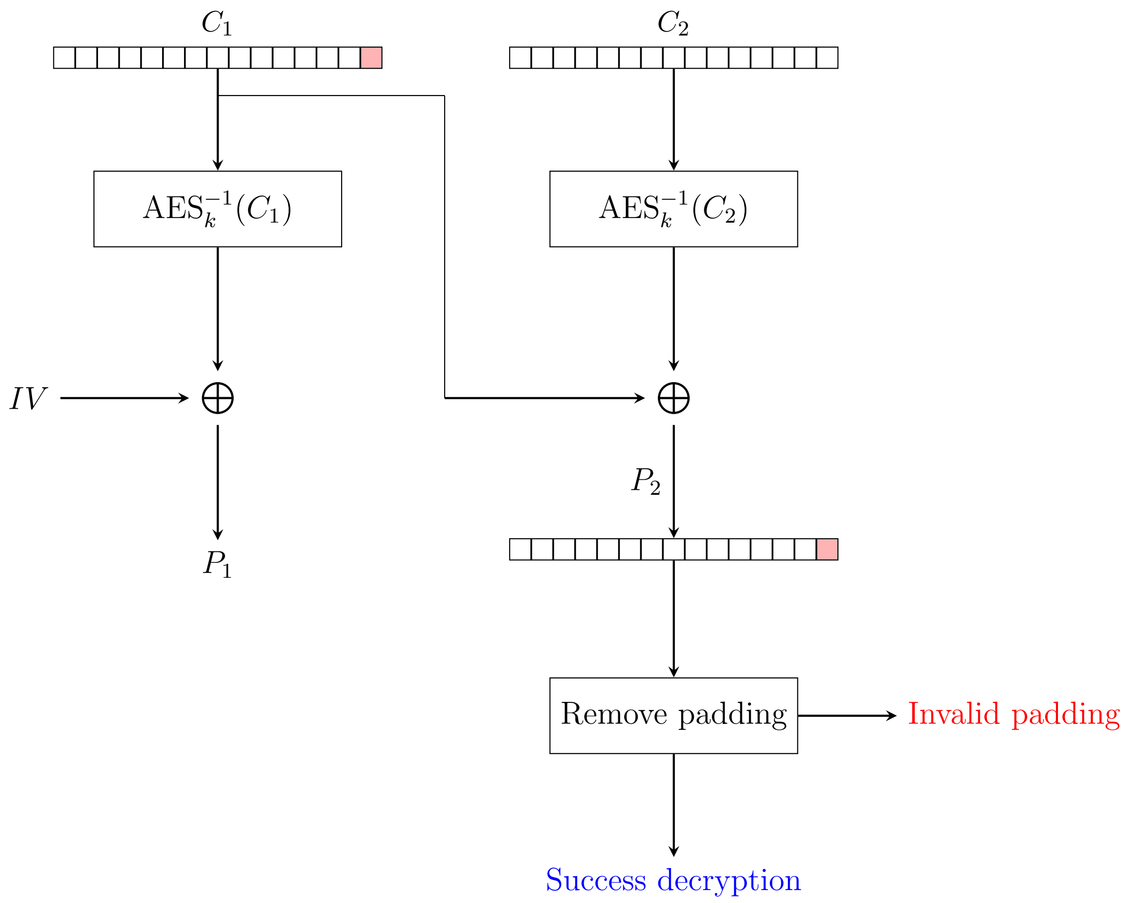

Figure 1: AES in Cipher Block Chaining (CBC) mode decryption diagram. The red square in C_1 means a possible byte modification, which makes a change in P_1.

we can recover whatever plaintext block we want, if and only if we can set the initialization vector (IV) that the server is going to use. If not, we can't recover the first block. This well-known vulnerability is called Padding Oracle attack [4].
For the next section, we consider that the initialization vector (IV) is static, so the server doesn't return it to us. However, we can combine the padding oracle with a known plaintext attack in order to recover the IV, and then the sensitive information.

3 Combining Padding Oracle and Known Plaintext attack in AES-CBC

In a cybersecurity audit for a Chilean company, we found that the server returns two ciphertexts:

- An encrypted card number. The card number itself has 16 bytes, so the resulting ciphertext has 32 bytes assuming PKCS #7 padding.
- An encrypted JSON which contains the user's JWT and some additional information. We executed padding oracle in few blocks in order to came up with this conclusion.

As a note, the server doesn't return the initialization vector (IV), so we can't recover the first block of the encrypted card number directly. Then, we define the following equations, considering two encrypted blocks:

$$\begin{aligned} S_1 &= \mathrm{AES}_k^{-1}(S_1^c) \oplus IV \\ S_2 &= \mathrm{AES}_k^{-1}(S_2^c) \oplus S_1^c \end{aligned}$$

$$\begin{aligned} J_1 &= \mathrm{AES}_k^{-1}(J_1^c) \oplus IV \\ J_2 &= \mathrm{AES}_k^{-1}(J_2^c) \oplus J_1^c \end{aligned}$$

where S_i, S_i^c, J_i and J_i^c are the plaintext and encrypted versions of the card number and the JSON user data, respectively. Knowing J_i and J_i^c, we can recover the IV as follows:

- Send (J_2^c, J_1^c) to the server i.e. the encrypted JSON data, changing the order of the ciphertext blocks. The server computes

$$\begin{aligned} J_2 &= \mathrm{AES}_k^{-1}(J_2^c) \oplus IV \\ J_1 &= \mathrm{AES}_k^{-1}(J_1^c) \oplus J_2^c \end{aligned}$$

- Use the padding oracle attack for recovering the intermediate bytes: $\mathrm{AES}_k^{-1}(J_1^c)$.
- Compute the initialization vector (IV) as

$$IV = J_1 \oplus \mathrm{AES}_k^{-1}(J_1^c)$$

Next, we recover the card number (S_1) as follows:

- Send (S_2^c, S_1^c) to the server i.e. the encrypted card number, changing the order of the ciphertext blocks. The server computes

$$S_2 = \mathrm{AES}_k^{-1}(S_2^c) \oplus IV$$
$$S_1 = \mathrm{AES}_k^{-1}(S_1^c) \oplus S_2^c$$

- Use the padding oracle attack for recovering the intermediate bytes: $\mathrm{AES}_k^{-1}(S_1^c)$.
- Compute the card number (S_1) as

$$S_1 = \mathrm{AES}_k^{-1}(S_1^c) \oplus IV$$

4 Conclusions

Since Transport Layer Security (TLS) provides Confidentiality, Integrity, and Authentication in web applications, some companies have to implement an additional layer of security. Still, they ignore the fact that users can modify the original information before sending the HTTP payload through the secure channel. We named this an Encrypt-then-TLS scenario, where vulnerabilities can occur in legacy applications or when there's a misconception about the total safety that TLS provides. The proposed attack was exploited on a real company during a cybersecurity audit in Chile, so the user's sensitive information is not compromised. However, it is crucial to keep the systems updated with the world's security needs; AES-CBC is an encryption algorithm we don't recommend using without authentication. Finally, we demonstrate that, given a context where we don't have the static initialization vector (IV) and known plaintexts, we can recovering and decrypt sensitive information, being able to recover the card numbers of several users.

References

[1] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[2] Mohamed Nabeel. The many faces of end-to-end encryption and their security analysis. In *2017 IEEE International Conference on Edge Computing (EDGE)*, pages 252–259, 2017.

[3] Burt Kaliski. PKCS #7: Cryptographic Message Syntax Version 1.5. RFC 2315, March 1998.

[4] Juliano Rizzo and Thai Duong. Practical padding oracle attacks. In *4th USENIX Workshop on Offensive Technologies (WOOT 10)*, Washington, DC, August 2010. USENIX Association.