Project 1 - CBC-padding oracle

Felix Mölder - au737970

February 2024

1 obtain secret plaintext:

As a first step, we tried to understand the server structure and workflow. This was achieved by examine the given code of the server (main.py) and start the server locally and examine the behaviour in the browser. We found out that the server has two webpages:

index The index page responds with a cookie (authtoken) which describes the severs secret encrypted with AES-128 bit in CBC mode.

quote The quote page responds with a quote **if** the authtoken sent in the request can be decrypted and the plaintext equals to the servers secret **or** responds with "No quote for you!" otherwise.

The first task is to recover the servers secret (plaintext) from a given ciphertext by using a padding oracle. The idea works as follows:

1. Obtain a ciphertext by requesting the index webpage.
2. Seperate the ciphertext into 128 bit blocks. The first block is the IV, the last block is filled up with padding.
3. Take a IV full of 0's. XOR the ith byte with the ith byte of the ciphertext decryption for all 256 possible values until the server responds with valid padding.
4. XOR the valid padding intermediate IV with the proper padding. For example for the last byte the proper padding ist 0x01 at the last byte and the rest 0. Save the result in the zeroing IV and decrease all IV bytes by 1.
5. Repeat step 3 and 4 until all bytes are found and thus the zeroing IV (the decrypted ciphertext block) is found.
6. Repeat step 1 to 5 until all ciphertext blocks are decrypted.

The source code can be found attached. After testing the outputs, we found out that only for one byte in step 3, the server responds with an utf-8 error. Thus our oracle tests if the responded text starts with "'utf" (utf-8 error) or "No" if the responds is "No quote for you". Together with the structure of the attack, we obtained the plaintext:
"You never figure out that "I should have used authenticated encryption because ...". :)"

2 create new ciphertext:

The goal is to create a new ciphertext without knowing the key that can be decrypted properly. This can be especially interesting in the context of JSON Web tokens (JWTs) to authenticate yourself to an API. Due to the first part of this exercise, we are able to decrypt a ciphertext that was sent to us via the authtoken cookie. Now we can prepare a ciphertext that would lead to a specific plaintext without knowing the key. The idea is the following:

1. Apply padding to the wanted plaintext and seperate it into n 128-bit (16 bytes) blocks.
2. Choose random 16 bytes as final ciphertext block.
3. Ask the padding oracle to find the decrypted ciphertext.
4. XOR the decrypted ciphertext with the wanted plaintext block to obtain the necessary penultimate ciphertext block.
5. Repeat Step 2 and 3 until we obtain the IV as the first block.

This algorithm gives us IV and $C_0, \ldots, C_n$ such that the decryption would lead to $P_0, \ldots, P_n$. If we send a request to the quote webpage with IV followed by the ciphertext blocks, the server decrypts the ciphertext leading to the prepared plaintext. If this plaintext is indeed the secret, the attacker authenticated against the server without knowing the secret key with a selfmade plaintext.