VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

HTTPS POODLE attack
Research Report

Advisor: Nguyễn An Khương
Student: Nguyễn Phú Nghĩa 1952355

HO CHI MINH CITY, JUNE 2021

Contents

1 Abstract

This document covers my research on the POODLE (Padding Oracle On Downgraded Legacy Encryption) attack on HTTPS, which was discovered by Bodo Möller, Thai Duong and Krzysztof Kotowicz back in 2014. The communications using SSL version 3.0 with CBC mode ciphers are shown to be vulnerable. An active attacker can decrypt the client's secret cookie byte-by-byte given that he can act as a Man in the Middle (MitM) between the targeted client and the server. Although that privilege must be achieved via a different attack, environments such as public WiFi would ease that burden on the attacker. After a successful attack, the attacker can impersonate the targeted client to communicate with the server by using the victim's cookie, which is dangerous.

This was meant to be a personal project documentation and not in any way a formal description of the attack.

2 Overview

This attack exploit the encryption scheme of SSLv3, which will be explained later in this document.

Before digging in to the details, a good question to ask is "Why is this attack discovered in 2014 still relevant? What did it teach us?". Well, the lesson lies in the phrase "backward compatibility".

We have long get used to building applications, services, and virtually everything in the field of Information Technology with "backward compatibility" in mind, i.e. we build new technology but also support the ability to reuse old codes if the client did not yet support the new technology, for example.

It sounds like a good idea at first, because we need not to re-implement the functionalities supported by the old technology as the old saying goes: "don't reinvent the wheel", e.g. an arbitrary `C` program will be compiled well using a `C++` compiler. But this attack has proven it wrong since the enhanced version of SSLv3, which is TLS 1.0 has been defined in RFC 2246 back in 1999 and not until 2014 was the POODLE attack introduced, which can still exploits SSLv3 in real life, 15 years after the enhancement of SSLv3 was introduced! This is possible because people still support SSLv3 in the name of backward compatibility despite their TLS support, just in case one of the parties communicating has not supported TLS yet. If a MitM can trick both the client and server into downgrading their protocol, then they are ready to be biten by the POODLE.

3 The POODLE attack

3.1 SSLv3

As mentioned earlier, SSLv3 is a protocol vulnerable to the POODLE attack. Basically, SSLv3 is a Secure layer in HTTPS.

If Alice wants to send a message m to Bob, they will agree on a shared key K ahead of time (using a key sharing scheme such as Diffie-Hellman). Using K, Alice would compute the HMAC of her message $\text{HMAC}(K, m)$ as a mean to prove the message integrity and authenticity and append the HMAC to m.

In the end, she wants to send $m \mid \text{HMAC}(K, m)$ to Bob using a block cipher, say AES, with CBC mode. Before being encrypted, the data must be padded to have a multiple of the cipher block size, which is 16 in AES case. The padding scheme SSLv3 suggested is padding the data by $0 - 15$ random bytes and 1 additional bytes having the numerical value of the number of random bytes appended to data earlier such that the final data's length is divisible by the block size. So the last byte of the plain text data sent using SSLv3 always ranges from 0 to 15, corresponding the padding length of 1 to 16. After encrypting the data using K as the key for AES-CBC, Alice can send the encrypted data to Bob, which is $\mathrm{Enc}_K\,(m \mid \mathrm{HMAC}(K, m) \mid \mathrm{padding})$.

When Bob receives the encrypted data, he first decrypts it using K to get $m \mid \mathrm{HMAC}(K, m) \mid \mathrm{padding}$. Then base on the last byte, he can unpad the data and split the data into m and $\mathrm{HMAC}(K, m)$ since the length of a HMAC (with a hash function specified) is a constant. He can then compute the HMAC of m using K and compare it to the received $\mathrm{HMAC}(K, m)$ to verify the message integrity and authenticity.

3.2 Cryptographic flaw

There is an exploitable flaw when SSLv3 pads the data with random bytes, given that the attacker can freely prepend and append anything to Alice's m before the message was sent to Bob, i.e. changing it from m to $H \mid m \mid T$, the attack can be carried out as follows:

1. First, the attacker makes Alice send $H \mid m$ to Bob and adjusts H's length so that $H \mid m \mid \mathrm{HMAC}$ is padded with 16 bytes. The appropriate length can be found by increasing H's length until the length of the resulting ciphertext increases. We can do this because SSLv3 uses a block cipher for encrypting data. Now the last byte of the plaintext is guaranteed to be 15 and we have **the last block full of padding**.

2. Assume that there are n plaintext blocks P_1 to P_n, and n corresponding ciphertext blocks C_1 to C_n. A valuable observation is that P_n is all-random except for the last byte (15). The attacker now change the last ciphertext block C_n to another C_i and forwards the tampered ciphertext to Bob.

   (a) If the last byte of the plaintext of the tampered ciphertext is not 15, when Bob unpads the data, he would assume a wrong message with a wrong "expected HMAC" because the padding length is changed while it is in fact the same, thus resulting in Bob cannot trust "Alice" and close the connection immediately.

   After receiving an error message from Bob, which acts as a padding oracle, the attacker will try to re-establish the connection with Bob and makes Alice send the same request, but this time with a different shared session key between Alice and Bob because the old session has been closed. The attacker can try this move 256 times on average before the last byte of the plaintext of the tampered ciphertext happens to be 15 once, then the attack would proceed to its next cycle.

   (b) If the last byte of the plaintext of the tampered ciphertext happens to be 15, the process of unpadding the message and verifying its validity by Bob is carried out as if the ciphertext has not been tampered at all since the padding scheme of SSLv3 only use the last byte of the plaintext to unpad the message, so it cannot appropriately verify the padding's integrity.

   In this case, the attacker immediately knows that $C_{n-1}[15] \oplus 15 = \mathrm{Dec}_K(C_i)[15]$ while it is true that $C_{i-1}[15] \oplus P_i[15] = \mathrm{Dec}_K(C_i)[15]$.

So he can compute $P_i[15] = C_{i-1}[15] \oplus C_{n-1}[15] \oplus 15$. In other words, 1 byte of m has been exposed! The attacker can then appropriately choose H and T to shift m the right amount so that every bytes in m would be the last byte of a block at least once, then he can decrypt the whole m.

With that approach, instead of trying to decrypt the data with 256^{16} possible keys, the attacker only need 256 requests on average to decrypt one byte of m now. An exponential-time problem has been converted to a linear-time one with this attack.

In SSLv3, we have to blindly decrypt the data to be able to get the HMAC and check for the message's integrity, or in Moxie Marlinspike's words: if you have to perform any cryptographic operation before verifying the MAC on a message you've received, it will somehow inevitably lead to doom. And the above analysis has pointed out that "doomness" of SSLv3.

4 Summary

The POODLE attack has taught us that it is not always a good idea to care about backward compatibility. This is the reason why SSLv3 was deprecated in 2015 as in RFC 7568.