**POODLE Attack Overview**

- POODLE: Padding Oracle On Downgraded Legacy Encryption

- a variant of BEAST attack

- Attack takes advantage of vulnerability of SSL version 3.0 with CBC-mode

- Discovered Sep 2014 by Google Security team

- SSL 3.0 created 20 years ago, but still commonly supported by browsers

- If issues connecting with current TLS protocols, browsers would fallback to older protocol versions such as SSL3.

- one instigation method: MitM attacker causes connection faiulres, triggering client and server to fallback to SSL3

connection, then execute POODLEattack

- Solution: disable SSL 3.0 support altogether in all browsers

-- or disabling CBC-mode to support only RC4 stream cipher mode (Google also disabled RC4)

-- original Google Chrome fix was to only disallow fallback due to retry after connection failures in order not to

cause compatibility issues with valid legacy sites still using SSL3.

1. Attacker established as MitM (the hard part)

- Attacker must be able to modify network transmissions between the victim client and the server

2. Attacker uses a “downgrade dance” between target victim and the victim’s destination web server

- First handshake offers highest protocol supported by client

- If first handshake fails, retry with earlier protocol versions

- Network glitches and active attackers can also trigger a downgrade

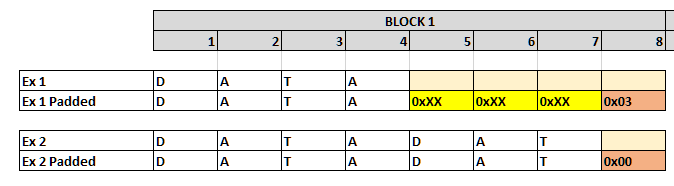
3. SSL3 CBC vulnerability

- The block cipher padding is not deterministic and not covered by the Message Authentication Code

- This means the validity of the padding cannot be fully verified during decryption

- Padding 1 to B bytes (B is block size in bytes) is applied to create integral number of blocks before performing

CBC encryption on the blocks

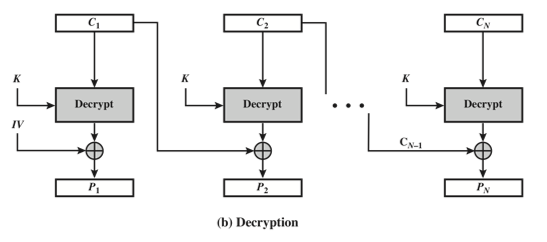


- Easiest to exploit if there’s an entire block of padding L­1 arbitrary bytes followed by a single byte of value L­1.



- IV||C0||C1||...||Cn are transmitted from client to server

- Server decrypts each block Pi = DK(Ci) ⊕ Ci-1 (symmetric session key, K, is the same per-connection)



- Server then checks for padding in Cn

- Server checks and removes MAC (recall MAC doesn’t include padding)

\* If there’s a full block of padding, and attacker replaces Cn (padding block) by any earlier ciphertext block Ci from

the same encrypted stream, the ciphertext will still be accepted if DK(Ci) ⊕ Cn­1 has 0x07 as its final byte

and rejected otherwise (SSL3 server becomes the Oracle)

- Use in HTTPS web setting to decrypt secure HTTP cookies using techniques from BEAST attack

- Attacker runs JavaScript agent on **http**://compromised-website.com that victim browses to

- Get the victim’s browser to send cookie-bearing HTTPS requests to **https**://website-to-get-access-to.com

- Attacker intercepts and modifies SSL records from victim browser in a way that the https site may accept

the modified record

- If the https site accepts the modified record, the attacker can decrypt one byte of the cookie

- Assume size of each block Ci is 8 bytes

- Assume that the size of the cookies is known

- MAC size in SSL3 CBC is 20 bytes

- This gives the format of the encrypted POST request:

ECBC[K, POST **/path** **Cookie**: name=value...\r\n\r\n **body** ‖ **20­byte MAC** ‖ **padding**]

- Goal: decrypt Cookie bytes causing client to varying the cookie byte positions within subsequent requests

- Observe that MAC does not factor the padding, so the encrypted padding block can be replaced

- Attacker controls request /path and the request body, so attacker can cause the client to modify the request

before encryption and transmission such that:

1. Padding fills an entire block (encrypted into Cn)

2. Cookie’s first unknown byte appears as the final byte in an earlier block (encrypted into Ci)

- Attacker replaces Cn with Ci and forwards the modified SSL record to the server oracle

- When the server oracle rejects the record, the attacker tries again with a new request

- On average, server will accept 1 in 256 requests as valid when it recognizes valid padding byte

- Attacker concludes DK(Ci)[7] ⊕ Cn­1[7] = 7

Not explained in paper: (diagrams here)

DK(Ci)[7] ⊕ Cn­1[7] = 7 Attacker knows from Oracle

We can find Pi[7]:

D(Ci) = Pi ⊕ Ci-1 Known by CBC Encryption algorithm

D(Ci)[7] = Pi[7] ⊕ Ci-1[7]

Since DK(Ci)[7] ⊕ Cn­1[7] = 7,

( Pi[7] ⊕ Ci-1[7] ) ⊕ Cn­1[7] = 7 Substitute for DK(Ci)[7]

**Pi[7]** ⊕ Ci-1[7] ⊕ Cn­1[7] = 7 Solve for Pi[7]

- So **Pi[7]** = 7 ⊕ Cn­1[7] ⊕ Ci­1[7]

- This reveals the cookies’ first previously unknown byte.

- The attacker proceeds to the next byte by changing the sizes of request path and body simultaneously such that

the request size stays the same but the position of the headers is shifted, continuing until it has decrypted as

much of the cookies as desired

(some browsers send request header and request body in separate SSL records; for this case, only the path size

needs to be changed when proceeding to the next byte)

The expected overall effort is 256 SSL 3.0 requests per byte.

As the padding hides the exact size of the payload, the cookies’ size is not immediately

apparent, but inducing requests GET /, GET /A, GET /AA, ... allows the attacker to

observe at which point the block boundary gets crossed: after at most 16 such requests,

this will reveal the padding size, and thus the size of the cookies.

**POODLE Setup**

- HTTPS Server SSL 3.0

-- unsuspecting accomplice

- client

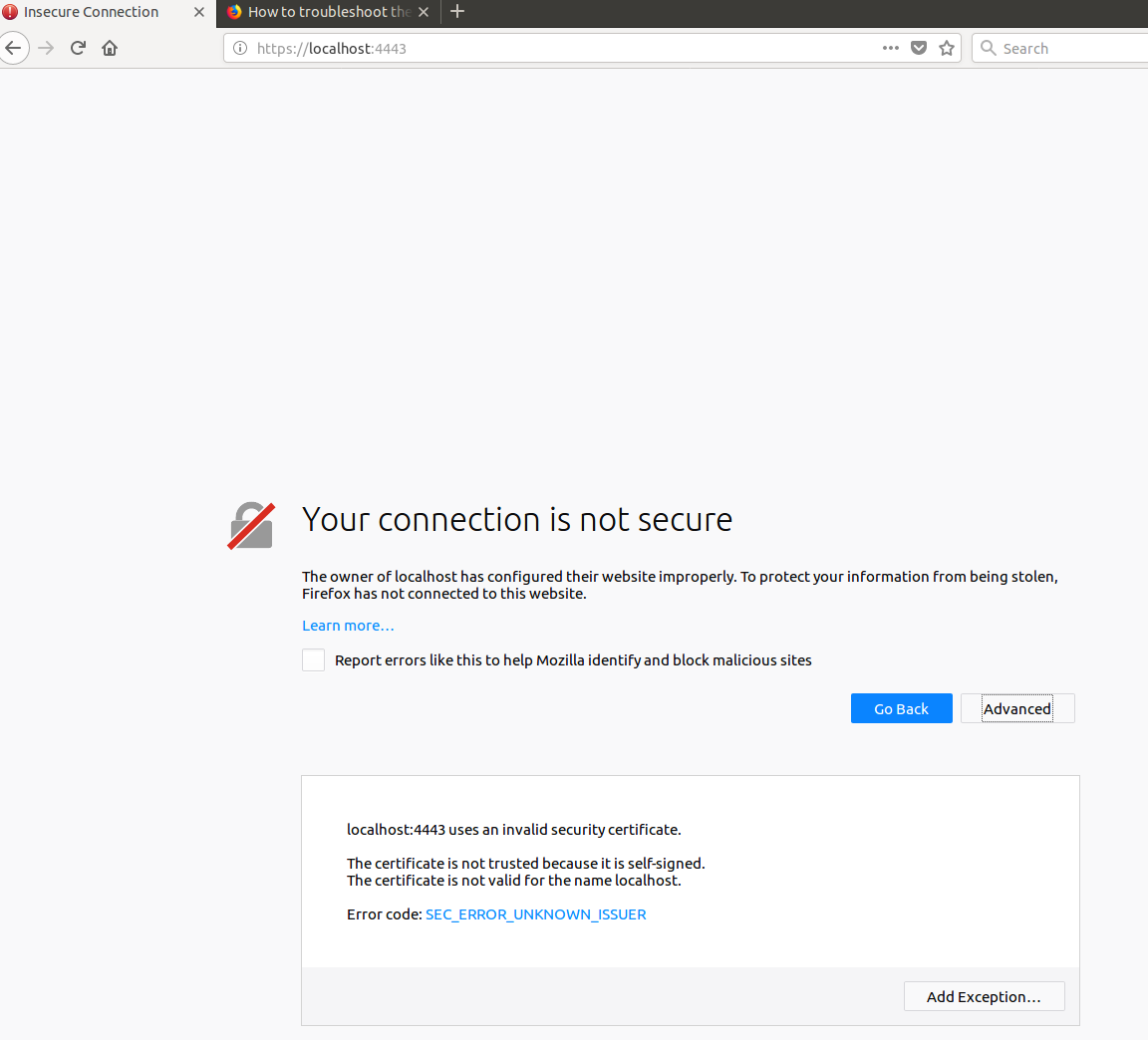
- attacker already established as man-in-the-middle

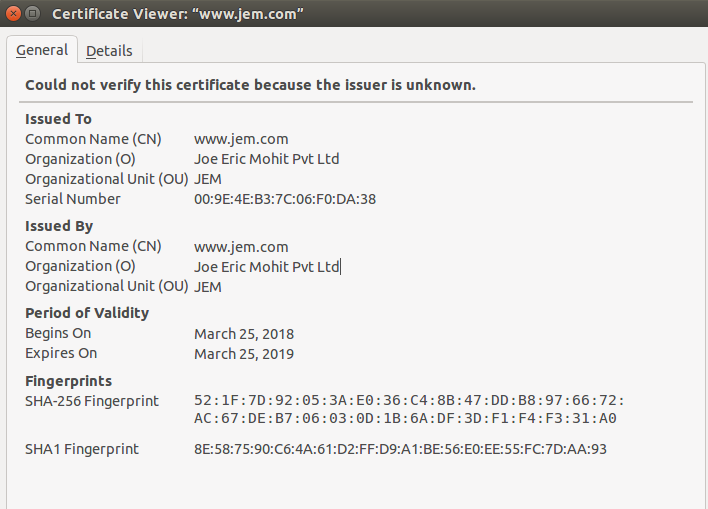
**POODLE Attack Steps**

VM-C Ubuntu becomes SSL v3 webserver

execute web server with

python3 simple-https.py





Modulus (2048 bits):

da 0e d5 c4 19 9a 9b 75 a0 e7 84 56 f8 d6 75 f3

6f 07 c6 67 a7 a1 01 af f3 df 33 0c 8e 7a e5 da

36 bc 05 8b cd 86 b4 98 93 b6 d8 cd ed e6 0c c7

de 87 c7 05 4a 3b 23 87 48 7c 48 a0 fc 61 40 9a

b3 94 d3 37 2f 91 fa bd 15 24 26 a9 79 a4 c8 68

9c 2e 2a 39 5b e4 1a ca 18 48 52 c0 5f 02 99 e3

96 1e 47 ca f7 7f ea b3 2b a9 90 4f 8c 5f 11 94

58 f0 7d 24 5d 4b f6 46 ae b3 9b 2b d4 41 a3 94

6c 8d 25 70 47 24 c7 77 f0 d5 45 67 94 99 54 3f

b8 73 b3 90 5d 1c 29 d8 f9 56 03 a8 79 c7 da c2

f0 66 60 5b 57 9f c6 87 81 27 d5 3c e8 fc 07 67

1d f5 7b d3 cb 13 d7 1d 10 b4 c1 8d 67 46 46 15

8d cf 04 cf eb 72 67 ab d2 fb 9a 2c 9d 94 29 44

41 4a 81 db e6 0b 7c d6 38 2c a1 c5 ef 9d 3e b1

55 c1 f4 c1 5f ab 8d 0c 88 90 14 46 6c 11 26 cd

e9 a4 f9 56 ad 11 ba 2b ac e1 04 c6 8b bf df 85

Exponent (24 bits):

65537

References

This POODLE Bites: Exploiting The SSL 3.0 Fallback, https://www.openssl.org/~bodo/ssl-poodle.pdf

[BEAST] T. Duong, J. Rizzo: “Here Come The ⊕ Ninjas”, 2011.