CZ4010: Exploring TLS/SSL Attacks

Group 48:
Garg Astha
Ramasubramanian Nisha

Summary

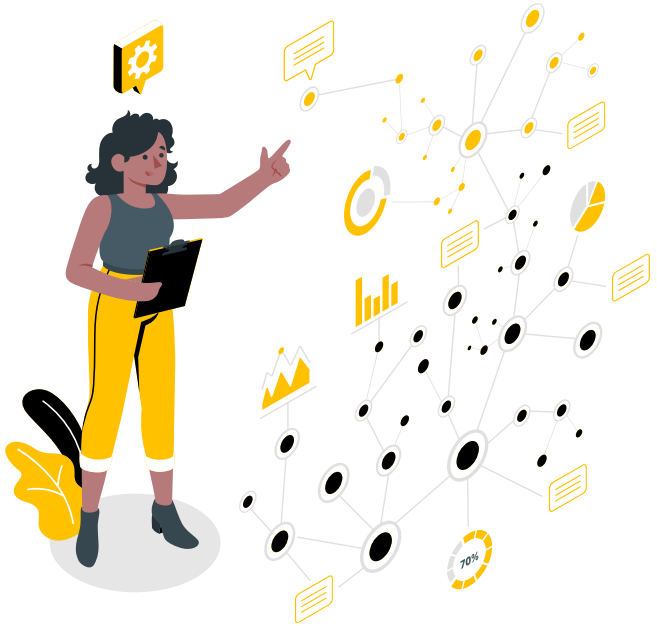
01
Introduction

02
BEAST Attack

03
CRIME Attack

1. Feature of SSL/TLS that is exploited.
2. How is it exploited (Theory)?
3. How is it exploited (Demonstration)?
4. What are some strategies to mitigate this attack?

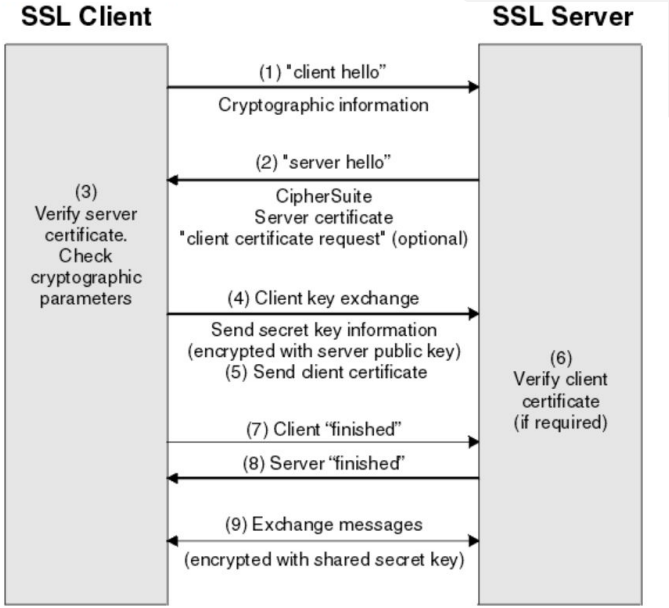
Introduction



01

What is TLS?

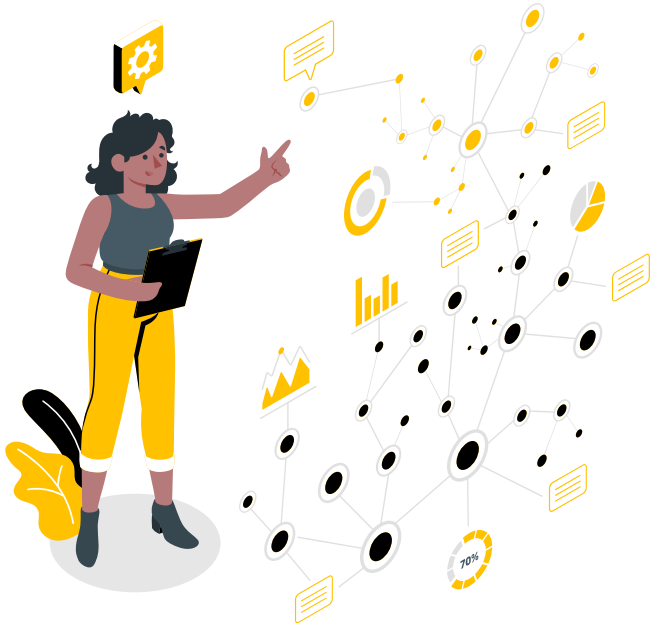
- TLS is a cryptographic protocol that provides end-to-end security of data sent between applications over the Internet.



https://www.ibm.com/docs/en/ibm-mq/7.5?topic=ssl-overview-tls-handshake

BEAST Attack

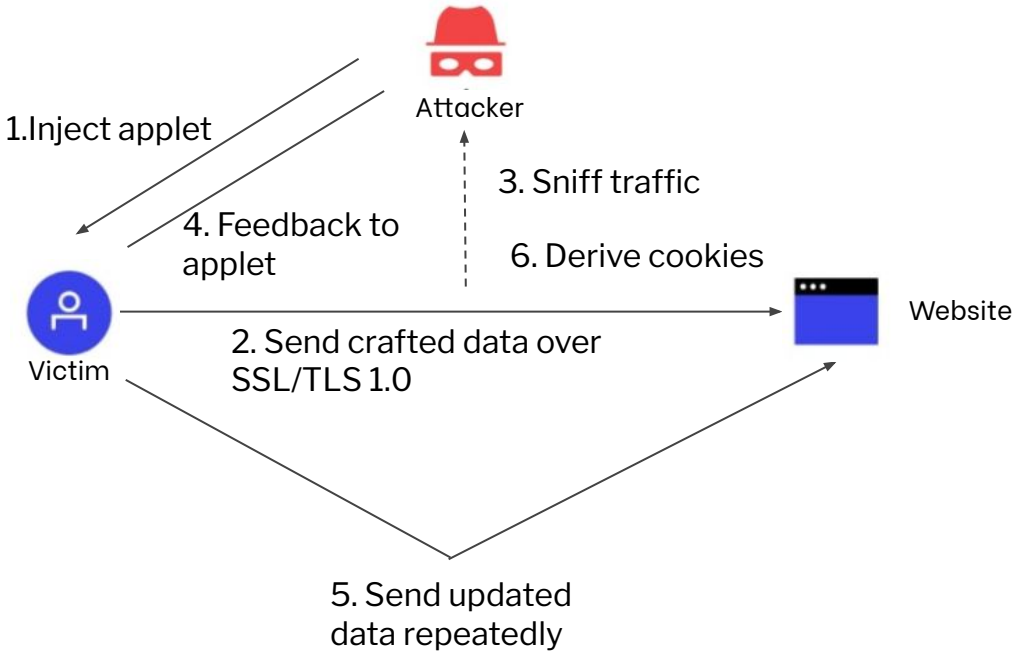
02



BEAST – Browser Exploit Against SSL/TLS

- It exploits a vulnerability in the TLS 1.0 and older SSL protocols, using the cipher block chaining (CBC) mode encryption.
- It allows attackers to capture and decrypt HTTPS client-server sessions and obtain authentication tokens. It combines a man-in-the-middle attack (MitM), record splitting, and chosen boundary attack.
- The probability of this attack is very low, and it can, at best, be used to read short strings of plaintext.

How does the attack work?



https://www.wallarm.com/what/what-is-a-beast-attack

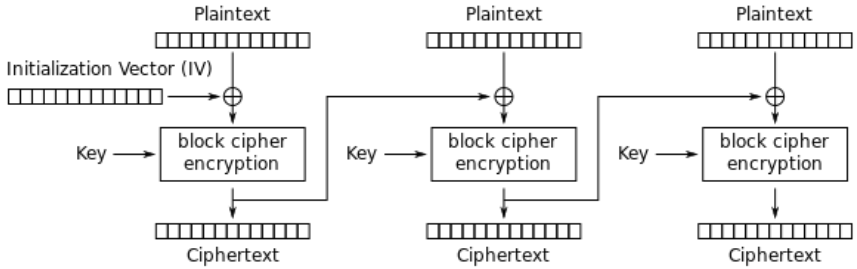
How does the attack work?

- For BEAST to be possible, several conditions need to be met:
  - An attacker must be able to monitor or “sniff” the network connection and the sessions between the client and server.
  - A vulnerable version such as TLS 1.0 or an older SSL protocol must be in use (or a downgrade must be possible) with a block cipher.
  - An applet or JavaScript injection must be likely, overriding the same-origin policy of a website.

CBC – Cipher Block Chaining

- Block Cipher - The data is split into fixed length blocks and then encrypted.
- Padding - The last block may contain data that does not meet the length requirement. In this case, padding (usually with random data) is done.
- Initialization Vector (IV) - It is a “random” fixed-size input used in encryption method.
- CBC - Each block is XORed with the previous cipher text before encryption.

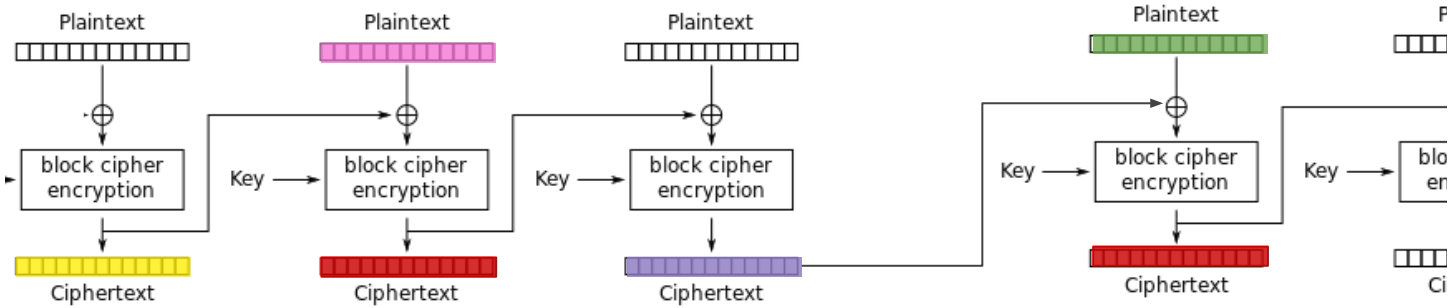
Encryption	Decryption
$C_i = E_k(P_i \oplus C_{i-1})$, and $C_0 = IV$	$P_i = D_k(C_i) \oplus C_{i-1}$, and $C_0 = IV$



Cipher Block Chaining (CBC) mode encryption

Explaining the vulnerability

- In TLS 1.0, next message IV is chosen as the last cipher block of previous message.
- What is known:
  - All Ciphertext is known (known - ,).
  - Next message IV is previous block (known -).
  - Attacker can inject some plain text (we control -).



Explaining the vulnerability

- Suppose the attacker wants to guess that message m_i may be secret x.
- She picks the next plaintext block to be $m_j = m_i \oplus c_{(i-1)} \oplus c_{(j-1)}$.
- c_j then becomes:

  $c_j = E(k; c_{(j-1)} \oplus m_j) = E(k; c_{(j-1)} \oplus c_{(j-1)} \oplus m_i \oplus c_{(i-1)}) = E(k; m_i \oplus c_{(i-1)})$.
- If $c_j = c_i$ then the guess of secret x is correct.

Explaining the vulnerability

- AES has 16 bytes block size i.e. 128 bits and it will be tough to guess those many bytes, instead we craft the request such that we only have to guess 1 byte.
- HTTP request example:

```
GET /index.html HTTP/1.1
Host: mysite.com
Cookie: Session=12345678
Accept-Encoding: text/html
Accept-Charset: utf-8
```

GET /index.html	HTTP/1.1\r\nHost	: mysite.com\r\n	Cookie:Session=	12345678
GET /index.jsp H	TTP/1.1\r\nHost:	mysite.com\r\nC	ookie: Session=1	2345678
GET /index.js HT	TP/1.1\r\nHost:	mysite.com\r\nCo	okie: Session=12	345678

Implementation details

- For demonstration have four .py files:
  - **website.py** - Simulate the response from the server victim is connected to.
  - **user_victim.py** - Sends requests to website which include the guess for value of cookie. In each iteration (total 8), the request is shifted to include only one byte of unknown cookie value and the rest is padded. Different guesses (alphanumeric) are sent one a time before moving to the next iteration.
  - **attacker.py** - It “sniffs” the packets being sent to the website and since it knows the location of the cookie, it compares the actual with the guessed value and keeps updating the cookie value found till that point.
  - **cryptography.py** - It provides helping functions relating to padding, XORing, AES encryption and decryption.

Demonstration

Feasibility

- A BEAST attack is not easy to perform.
- The attacker must use a different exploit to become a man-in-the-middle and to inject content into the stream.
- The web application is by default protected using same-origin policy and this makes injection difficult (unless the web application has server-side CORS headers that override the default policy).

Mitigation

- Permitting TLS 1.1 or 1.2 was the most secure method for keeping up with security since they fixed the fundamental TLS 1.0 issue.
- Browser vendors had attempted to implement a workaround while still remaining compatible with the SSL 3.0/TLS 1.0 protocol. These included inserting empty fragments into the message in order to randomize the IV.
- Supported browsers
  - Google: Update to Chrome 16 or later.
  - Microsoft: Ensure that MS12-006 has been applied.
  - Mozilla: Update to Firefox 10 or later.

Software supporting the mitigation

- Browsers
  - Google: Update to Chrome 16 or later.
  - Microsoft: Ensure that MS12-006 has been applied.
  - Mozilla: Update to Firefox 10 or later.

References

- https://www.nccgroup.com › ssl_attacks_survey
- https://github.com/lennysec/awesome-tls-hacks
- http://commandlinefanatic.com/cgi-bin/showarticle.cgi?article=art027
- https://crashtest-security.com/ssl-beast-attack-tls/
- https://vnhacker.blogspot.com/2011/09/beast.html
- https://medium.com/@c0D3M/beast-attack-explained-f272acd7996e
- https://github.com/mpgn/BEAST-PoC
- https://github.com/mpgn/CRIME-poc
- https://github.com/DennyHsieh/crime-attack

Thank you!

Any questions?