BEAST TITLE TO-FILL

Yang Shichu
Huazhong University of Science and Technology
TO-FILL

Shu Yi
Huazhong University of Science and Technology
TO-FILL

Su Haochen
Sichuan University
TO-FILL

Li Yucong
Shandong University
TO-FILL

Liao Haicheng
University of Electronic Science and Technology of China
TO-FILL

ABSTRACT

Transport Layer Security (TLS) is an protocol that provides communication security over networks. However, there is a flaw in TLS 1.0 where the initial vectors for block ciphers are predictable. The BEAST attack, with some prerequisites and efforts, allows attackers in the middle to decrypt those encrypted messages without knowing the key. This paper will demonstrate the procedures of the BEAST attack, and propose methods in simulation and vulnerability detection.

Keywords

BEAST attack, TLS flaws, CBC exploits, vulnerability detection

1. INTRODUCTION

Transport Layer Security (TLS) has several versions. The specification for TLS 1.0 is RFC 2246[1]. In this paper we will show a flaw in one of the common modes of operation used in block ciphers and how it allows for a specific kind of attack[2] on HTTPS.

2. BACKGROUND

2.1 A glance at TLS

TLS is a protocol for safe data transferring that works between the transport layer and the application layer. The cipher suites used in TLS often involve an asymmetric cipher (e.g. RSA) for key exchanging and a symmetric block cipher (e.g. AES) for message encryption. The protocol is widely used together with data transfer applications such as HTTP, FTP and SMTP.

2.2 CBC in block ciphers

Cipher Block Chaining (CBC) is one of the modes of operation used in block ciphers. In order to reduce the time spent on generating random initialization vectors (IVs), CBC always takes the previous encrypted ciphertext block and use it as the IV for the current plaintext block before the block cipher encrypts, except for the first block as shown in Figure 1.

Suppose that $P_1, P_2, \cdots P_n$ are the plaintext blocks, with a initialization vector IV, we have:

$$C_1 = E_k(P_1 \oplus IV)$$
$$C_i = E_k(P_i \oplus C_{i-1})(i \geq 2)$$

to obtain ciphertext blocks $C_1, C_2, \cdots, C_n$.

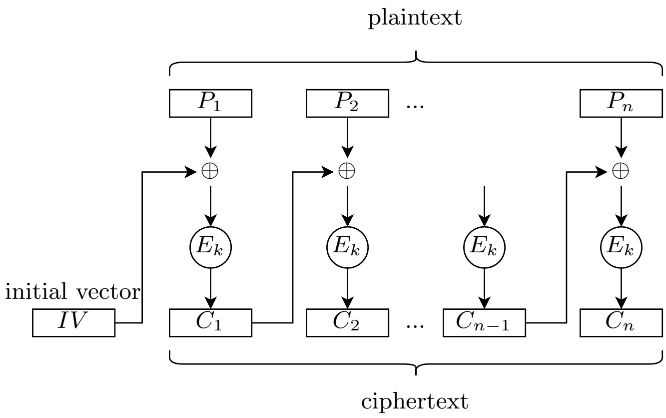

Figure 1: CBC encryptor

3. THE BEAST ATTACK

3.1 Predictable IV and Consequences

As we mentioned in Section 2.2, a block cipher using CBC always takes its previous cipher block as its next IV. This means that an attacker who has been eavesdropping the whole encrypted conversation can infer all the IVs in the conversation except for the first one. If the attacker has control over such an encryption machine (i.e. he has chosen plaintext privilege), in an attempt to guess the plaintext of block C_k with a guessed plaintext block P'_k, assuming the encryption machine is about to encrypt the $i+1$th block, he can pass

$$P_{i+1} = P'_k \oplus C_i \oplus C_{k-1}$$

to the machine. So that the ciphertext block would be

$$C_{i+1} = E_k(P'_k \oplus C_i \oplus C_{k-1} \oplus C_i) = E_k(P'_k \oplus C_{k-1})$$

Since the block cipher algorithm used in TLS is deterministic (i.e. same plaintext encrypts to same ciphertext),

and $C_k = E_k(P_k \oplus C_{k-1})$,

If $C_{i+1} = C_k$, then

$$P_k \oplus C_{k-1} = P'_k \oplus C_{k-1}$$
$$P_k = P'_k$$

The procedure shown above is actually a validation oracle which tells whether the attacker's guess on P_k is correct. In conclusion, an attacker with chosen plaintext privilege in this case can use brute force to obtain the plaintext of any cipher block $C_k (k \geq 2)$.

3.2 Chosen Boundary Attacks

4. THREAT MODEL

4.1 Prerequisites for attackers

- TO-DO

5. DEMONSTRATION

6. FEASIBILITY AND DEFENSE

6.1 Feasibility

While BEAST attacks are theoretically feasible, with the enhancement of security features of browsers and other clients, BEAST attacks are less and less practical for an attacker to exploit.

6.1.1 CORS

CORS is a group of policies to regulate the contents of cross-origin requests. An attacker cannot make a request to other sites using JavaScript. If an attacker wants to send some requests to Facebook, they will be rejected by browser's policies.

That is to say, when attackers want the clients to forge a request to websites with credentials. These requests will not be sent. Thus, BEAST attacks will not work any longer.

6.1.2 WebSocket mask

WebSocket is not subject to CORS policies, and it can be also wrapped by TLS. It seems that WebSocket will be a good choice to implement BEAST attack.

However, in modern clients, WebSocket payloads are masked by a value shown in 2.

Besides, WebSocket prepends extra bits before real payload. It is still hard to control the block boundary.

The mask makes it hard for attackers to do BEAST attacks, since the attack requires the first sent block mostly controllable by attackers.

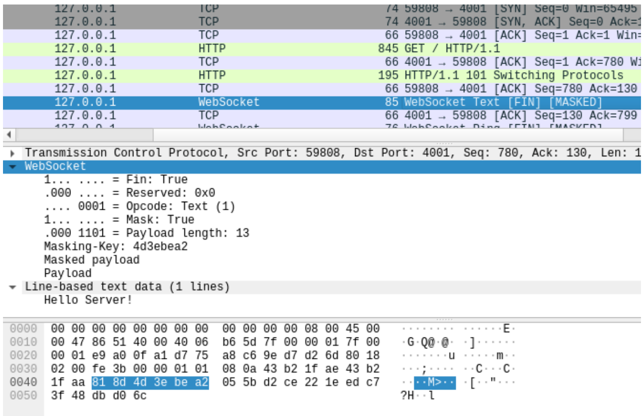

Figure 2: WebSocket mask

6.2 Defense

BEAST attacks make use of a flaw in the specification of TLS 1.0, and the attack only works for block ciphers. That is to say, stream ciphers with TLS 1.0 are not vulnerable to BEAST attacks.

However, TLS 1.0 is still vulnerable to other attacks when using stream ciphers (e.g. RC4). Therefore, a much more direct way is just to abandon TLS 1.0, and update to later TLS versions.

Many modern browsers and clients have also limited users to browse those sites with TLS 1.0 enabled alone. This kind of action will boost organizations to update their websites TLS versions. Today there are only few sites supporting TLS 1.0.

7. DETECTION

We will propose a method to detect BEAST vulnerability of a server, together with a Python script which is easy to use.

At the stage of TLS handshake, a cipher suite will be selected through steps:

1. (Client Hello) Client sent a list of accepted cipher suites.
2. (Server Hello) Server chose a best accepted cipher suite, or a handshake failure occured.

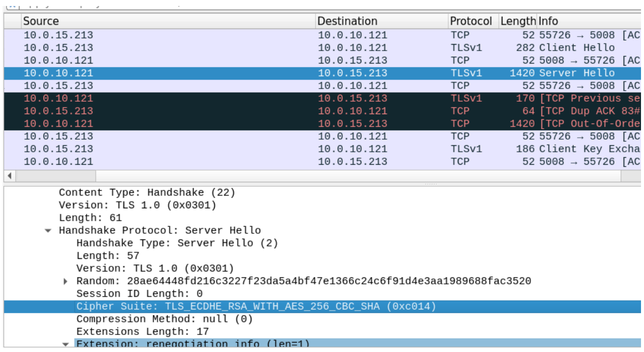

Figure 3: Negotiation on the cipher suite

Based on this, a scanner could change the list of cipher suites to enumerate all cipher suites that the server will accept.

The server is vulnerable to BEAST attacks if it accepts TLS 1.0 handshake and support cipher suites with CBC modes.

```
python scan.py host port
```

The `openssl` utility is able to start a TLS server with many options.

```
openssl s_server \
    -CAfile ca_cert.pem \
    -cert server_cert.pem \
    -key server_key.pem \
    -HTTP -port 5008 -tls1
```

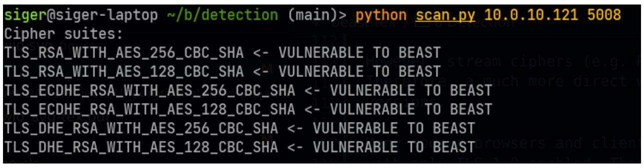

Figure 4: Detection output on a TLS 1.0 server

8. REFERENCES

[1] C. Allen and T. Dierks. The TLS Protocol Version 1.0. RFC 2246, Jan. 1999.

[2] T. Duong and J. Rizzo. Here come the $\oplus$ ninjas!, May 2011.

APPENDIX