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 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 $C_1, C_2, \cdots, C_n$.

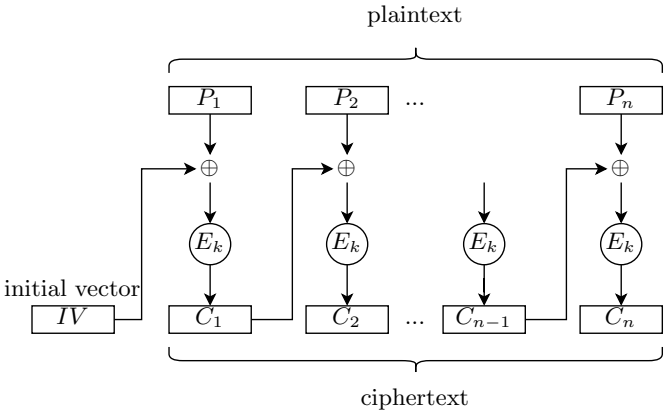

Figure 1: CBC encryptor

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

3.2.1 Failure of chosen plaintext

The chosen plaintext attack shown above requires that the attacker guesses the content of a whole block every time. Although this effectively gets IVs from CBC mode out of the way, it is still not a practical attack as the time required to decrypt a single block would be sufficiently long. For Advanced Encryption Standard (AES), a commonly used block cipher, the block size is 16 bytes[3] which means an attacker would have to guess up to 2^{128} blocks for one ciphertext block. This is considered secure as AES is designed with a 128-bit security level[4] and this security level is unharmed. So for an attack to work, it needs to further reduce the number of guess trials.

3.2.2 The BEAST

Recall that in Section 3.1 we mentioned an attacker with chosen plaintext privilege. Here we will show that if such an attacker further gains chosen boundary privilege, he can decrypt a secret string with far less trials than a simple brute force attack. The chosen boundary privilege means that the attacker can force the victim to send cookie-bearing requests with arbitrary request paths to the HTTPS server. If the attacker gains this privilege, he can carefully craft the request path so that exactly 1 byte of the target string will be placed at the end of a plaintext block while the rest is at the beginning of the next plaintext block. An example of such a request is shown in 2.

G	E	T		/	P	O	C		H	T	T	P	/	1	.
1	CR	LF	C	o	o	k	i	e	:		P	I	N	=	?
?	?	?	?	?	?	?									

Figure 2: chosen boundary request example

In this case, 15 out of 16 bytes in the second plaintext block is already known to the attacker. Now the attacker performs the same chosen plaintext attack described in Section 3.1 except that he only has to guess 256 times to obtain the first byte of the secret string.

After obtaining the first byte of the secret, the attacker reduces the length of the request path by 1 byte, so 2 bytes of the secret are now at the end of the plaintext block. As the first byte is already compromised, the attacker can use the same technique above to obtain the second byte with only 256 guesses. Repeating the steps shown above, the attacker can eventually reveal the whole secret string with chosen boundary requests as in 3 (showing only the block to guess).

1	CR	LF	C	o	o	k	i	e	:		P	I	N	=	?
CR	LF	C	o	o	k	i	e	:		P	I	N	=	1	?
LF	C	o	o	k	i	e	:		P	I	N	=	1	2	?

Figure 3: revealing the secret bytes 1 by 1

In this attack scenario, only $256 \times 16 = 2^{12}$ guesses are needed to decrypt an entire block of secret, effectively reducing the 128-bit security level in AES to only 12 bits. As this attack aims at breaking certain ciphers used in Secure Sockets Layer (SSL) and TLS and requires chosen boundary privilege which is often obtained by injecting malicious JavaScript into a webpage, this kind of attack is therefore named Browser Exploit Against SSL/TLS (BEAST).

4. THREAT MODEL

4.1 Prerequisites for attackers

In order to mount such an attack, the attacker must have these capabilities:

- **Network Eavesdropping** The attacker must be able to capture the whole encrypted conversation to know every IV in the process. This information is obtainable most of the time, yet certain protocols used at link layer (e.g. Wi-Fi with Pre-Shared Key, Virtual Private Networks) may prevent this.
- **Chosen Plaintext** The attacker can construct any known plaintext block and force the client to encrypt and send it via HTTPS. This allows for guessing the content of a ciphertext block.
- **Chosen Boundary** The attacker should be able to force the client to send arbitrary requests with cookie to the HTTPS server. By controlling the resource path, the attacker is able to place the secret cookie anywhere in a plaintext block.

4.2 Server vulnerability

The BEAST attack only works against TLS Version 1.0 and requires a block cipher working in CBC mode. For the attacker to gain chosen plaintext privilege, the server must also host a WebSocket service that allows sending arbitrary plaintext without any formatting. This is further addressed in Section 6.1, where we discuss why BEAST attack fails in front of the latest security standards.

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 4.

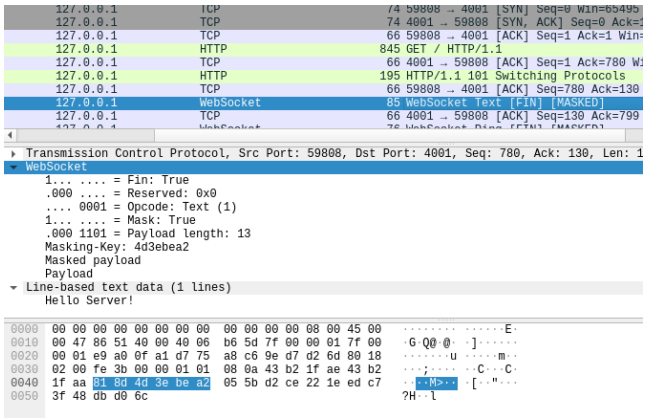

Figure 4: WebSocket mask

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.

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.

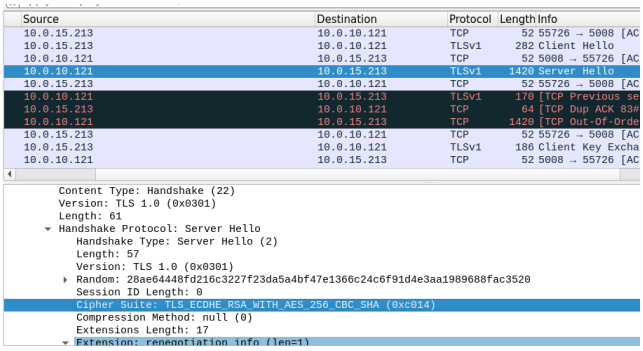

Figure 5: Negotiation on the cipher suite

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
```

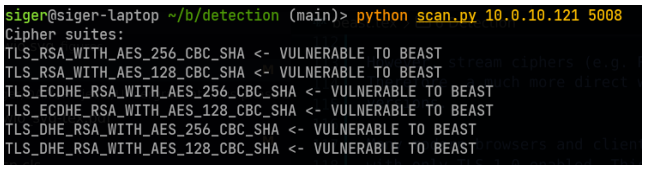

Figure 6: Detection output on a TLS 1.0 server

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.

[3] NIST. Announcing the advanced encryption standard (aes), November 2001.

[4] Wikipedia. Security level.

APPENDIX