BEAST: A Surprising Crypto Attack Against HTTPS

Thai Duong    Juliano Rizzo

January 12, 2012
B.E.A.S.T: Browser Exploit Against SSL/TLS

- A new way to exploit a decade-old known vulnerability in SSL/TLS.
- The attack combines crypto and browser security weaknesses.
- Demo: BEAST decrypts HTTPS requests and obtains secret cookies.
HTTP over an encrypted SSL/TLS connection provides

- **Confidentiality** (Encryption)
- Integrity (Message Authentication Code)
- Authenticity (Certificates)
NOT a MiTM attack

- Attack against the confidentiality
- Encrypted data is not modified
- No certificates were harmed
B.E.A.S.T in the network

Victim

BEAST Control Connection

Attacker

HTTPS Server
Encryption in SSL/TLS

- Unique symmetric encryption keys negotiated by handshake
- Block ciphers in CBC mode (3DES, AES)
- Stream ciphers (RC4)
Block Ciphers

- Operate on fixed-length groups of bits (64, 128, 256)
- One secret key and two algorithms \((E_k, D_k)\)
- Messages are padded and broken into blocks
Cipher Block Chaining (CBC)
Cipher Block Chaining (CBC)

Encrypt:

\[ C_i = E_k(C_{i-1} \oplus P_i) \]
\[ C_0 = IV \]

Decrypt:

\[ C_0 = IV \]
\[ P_i = D_k(C_i) \oplus C_{i-1} \]
CBC Initialization Vector (IV)

- Same input (key and plaintext) but different IV = different output
- IV need NOT be secret
- IV MUST be unpredictable before attackers can chose plaintext
Dai’s Attack against CBC (1)

- Two assumptions:
  - Adversary can choose $P_i$.
  - Adversary can see $C_i$.

- Idea: use $P_4$ to make a guess for previous plaintext blocks. Suppose he suspects that $P_1$ is $X$, he then sets $P_4 := C_3 \oplus C_0 \oplus X$.

- If $P_1$ is $X$, then:

\[
C_4 = E_k(C_3 \oplus P_4) \\
= E_k(C_3 \oplus C_3 \oplus C_0 \oplus X) \\
= E_k(C_0 \oplus P_1) \\
= C_1.
\]
If $P_1$ takes $W$ possible values, it can be decrypted after at most $W$ guesses.

In practice, $W$ is often too large ($2^{128}$).

How to make $W$ small?
How to make it practical?

How!?
Chosen-Boundary Attack against CBC

- Idea: move block boundaries around to shrink $W$ to 256.
- Assumption: the adversary can prepend some bytes to the plaintext.

Block size $= 8$
$\text{len}(m) = 9$
$\text{len}(r) = 7$
HTTP over SSL. Used to protect cookies in requests, and responses.
SSL receives the HTTP message from the Application Layer as raw data, which is then fragmented into records of length less than or equal to $2^{14}$ bytes.

Vulnerability: each record is encrypted in CBC mode with chained IVs; i.e., the CBC IV for each record except the first is the previous records’ last ciphertext block.

Alice visits http://mallory.com operated by Mallory.

Mallory can sniff to see network traffic from Alice to https://bob.com.
Application: Decrypting HTTPS requests

- known bytes added by browsers
- known bytes that adversary controls
- unknown bytes that adversary wants to decrypt

block size = 8

POST / AAAA AAA HTTP

block boundary

/ 1 . 1 \r \n REQUEST

block boundary

RS [...] \r \n REQUEST BODY [...]
Plug-ins make it easier
Implementations

- Java Applet URLConnection API: confirmed.
- HTML5 WebSocket API: confirmed.
- SilverLight WebClient API: unconfirmed, doesn’t work with the obvious API
- XHTMLRequest: could be possible, we didn’t have luck
- Flash: poor Flash!, too many problems already, please leave him alone!
A Brief History of The Attack

1995: P. Rogaway observed that CBC mode is not secure against chosen-plaintext attack if the IV is predictable.

1996: SSL 3.0 was born, IV is predictable.

1999: TLS 1.0 was born, IV is predictable.

2002: W. Dai then Bellare et al. extended Rogaways attack to SSH. B. Moller then realized that Dais attack can also be used against SSL. A workaround was implemented in OpenSSL.

2004 and 2006: G. Bard tried the attack to SSL in web browsers. Bards work has been largely ignored, since his attacks dont really work.

2006: TLS 1.1 was born, IV is unpredictable.

2010: Predictable IV was alleged as the backdoor in OpenBSDs IPSEC implementation.

2011: BEAST: Chosen-boundary attack was invented.
(Broken) Countermeasures (1)

**TLS 1.1 and TLS 1.2**

- Not enough good TLS servers: just over 3000 servers supporting TLS 1.1 or higher.
- Counter-countermeasure: drop browsers’ TLS ClientHello.

Half of all trusted servers support the insecure SSL v2 protocol

- Modern browsers won’t use it, but wide support for SSL v2 demonstrates how we neglect to give any attention to SSL configuration
- Virtually all servers support SSLv3 and TLS v1.0
- Virtually no support for TLS v1.1 (released in 2006) or TLS v1.2 (released in 2008)
- At least 10,462 servers will accept SSLv2 but only deliver a user-friendly error message over HTTP

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Support</th>
<th>Best protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL v2.0</td>
<td>302,860</td>
<td>302,860</td>
</tr>
<tr>
<td>SSL v3.0</td>
<td>607,249</td>
<td>3,249</td>
</tr>
<tr>
<td>TLS v1.0</td>
<td>604,242</td>
<td>603,404</td>
</tr>
<tr>
<td>TLS v1.1</td>
<td>838</td>
<td>827</td>
</tr>
<tr>
<td>TLS v1.2</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>
Stream Cipher (RC4)

- RC4 is not FIPS-approved encryption
- Counter-countermeasure: Google’s SSL optimization False Start.
Counter-countermeasure: Google’s SSL optimization False Start.
OpenSSL’s fix

- Idea: preventing the attacker from controlling next plaintext block.
- Prepend an empty record to each message (OpenSSL 0.9.6d, May 2002)
- Compatibility issues, turned off by default in most products
Same idea as OpenSSL.

Let’s break each message into two records: the first record contains the first byte of the message, and the second record contains the rest.

Some compatibility issues with applications: ssl_read() = ”G”
What’s next?

- Is it possible to decrypt HTTPS responses?
- More SSL applications: SSL VPN, Instant Messaging, etc.
Conclusion

- Crypto is hard let’s go party!
- Questions?