BEAST & CRIME

How TLS was attacked

SAAG Meeting – IETF 85

Agenda

- What is BEAST?
- What is CRIME?
- “Best Practice” Mitigations
- Going forward

What is BEAST

- BEAST is an attack demonstrated by Rizzo and Duong at Ekoparty on September 2011.
- Using Java, a network sniffer and a popular browser they managed to mount a chosen plaintext attack and recover session cookies for an HTTPS session.
- The session cookies could then be used to impersonate the user.

What is BEAST

- To do this, they used two vulnerabilities:
  - SSLv3 and TLS 1.0, when used with a CBC cipher, use the last ciphertext block of the previous record as the Initialization Vector of the next record. So after one record has been emitted, an eavesdropper can know the IV of the next record.
  - A bug in SOP. A script, applet or any other active content from one origin MUST NOT be able to run queries to another origin using the browser context (such as cookies). It is this violation of the SOP in Java that allowed them to inject plaintext.

What is BEAST

- In HTTP(S), authorization is granted based on session, and sessions are recognized using session cookies, that the browser sends in requests.
- The form of the header is something like this:

```
Cookie: sessid=5ec20c5e8c2ebe595ab0
```

- If you can inject a request, everything up to the cookie is predictable, so you could guess the block with the cookie, but there’s too many possibilities.

What is BEAST

- In CBC the plaintext is split into blocks.
- Rizzo and Duong sent requests in different lengths, so as to play with the alignment.
- Suppose you align things so that the blocking of the cookie header is as follows:

  |Cookie: |sessid=5|ec20c5e8|c2ebe595|ab0
- Note that the entire block is predictable, except for the last byte. Let’s guess that it’s 5.

What is BEAST

- In hex, the block would look like this: `7365737369643d35`
- Let’s assume that the previous block of ciphertext looked like this: `de3bedcee3ade70a`
- XORing them together, we get this: `ad5e9ebd8ac9da3f`
- Let’s assume that this block of ciphertext was this: `0165b037d2e44954`
- If we’re right, then the browser encrypted `ad5e9ebd8ac9da3f` and got `0165b037d2e44954`.

What is BEAST

- We can prove ourselves right by having the browser use the same key to encrypt that same block again.
- The obvious candidate was WebSockets. Once you have upgraded to WS, you can pretty much send anything, and SOP is weak.
  - But then a new spec of WS ruined it.
- Turns out Java had a flaw.

What is BEAST

- The URLConnection class in Java has an API where you could create a request, and send information in small chunks.
- When the SSL stack decides it’s had enough data to fill a record, it sends that record.
- Now the attacker can sniff the last ciphertext block of the just-emitted record. Assume it’s `f4e52a1a9f11f116`.
- This block is going to get XOR‘ed with the next block that the attacker sends.

What is BEAST

- The attacker chooses `59bbb4a715d82b29` as the data it adds next.
  - Why? Because XOR-ing that with the last plaintext block (=the IV of the next record) yields `ad5e9ebd8ac9da3f`.
- If the first block of ciphertext on the next record is equal to `0165b037d2e44954`, it means we were right, and the first character in the cookie is ‘5’.
- Repeat to get the rest of the cookie.

What is BEAST

- So what’s wrong?
  - The big issue here is the violation of the SOP.
  - Even without discovering the cookie, sending arbitrary requests allows you to buy yourself stuff on many e-commerce sites.
  - This leads to a mixture of attacker-controlled and browser-controlled data.
  - Also, CBC in TLS 1.0 is vulnerable. TLS 1.1 (and using stream ciphers / CTR / GCM) solve this.

What is CRIME

- CRIME is another attack presented at this year’s Ekoparty by the same Juliano Rizzo and Thai Duong.
- This one works with any TLS version and relies on the compression function revealing patterns in the plaintext.
- Unlike BEAST, this does not rely on breaking the SOP.

What is CRIME

- Assume you have a connection to bank.com.
- The TLS session has compression enabled.
- As before, every request includes a cookie, which is used to associate the request with a user: `Cookie: sessid=5ec20c5e8c2ebe595ab0`
- Compression functions work by finding repeated patterns, and including them only once.

What is CRIME

- What this means, is that a request that includes "`sessid=5`" will compress better than a request that includes "`sessid=7`", because it has a longer shared part
  with the real cookie.
- It's not necessary to control the data, all you need to do is to make sure these requests are sent together with the cookie

What is CRIME

```
<html><head><title>Evil.com</title></head><body>
    <img src="https://bank.com/foo.jpg?sess=1>
    <img src="https://bank.com/foo.jpg?sess=2>
    <img src="https://bank.com/foo.jpg?sess=3>
    <img src="https://bank.com/foo.jpg?sess=4>
    <img src="https://bank.com/foo.jpg?sess=5>
    <img src="https://bank.com/foo.jpg?sess=6>
    <img src="https://bank.com/foo.jpg?sess=7>
</body></html>
```

What is CRIME

- The HTML in the previous page causes 7 requests to be made in succession.
- One of them will result in a shorter query than the others.
- That’s the one with the correct guess.
- Repeat for the next byte.

What is CRIME

- So what is the root cause here?
- Ivan Ristić says it’s that attacker-controlled and secret data are compressed together.

Mitigations

- After BEAST, there were two suggestions going around: Don't use CBC or upgrade to TLS 1.1.
- Since some major browsers still don't support TLS 1.1 and 1.2, this effectively means RC4
  - Which, if you've been listening to NIST, is not recommended.
- For CRIME, we can disable compression
  - Enabled in 42% of websites, but…
  - By now disabled in 93% of browsers.
    - Thanks to patches

Going Forward

- It's been suggested to accelerate the adoption of TLS 1.1/1.2
  - Microsoft has enabled it in their browsers.
  - OpenSSL has finally shipped it.
  - So did some other vendors.
- Do we really need compression?
  - HTTP offers body compression anyway.
  - HTTP/2.0 is discussing different encoding for the header, including compressed and binary-efficient encodings.