BREACH attack

Jeff Costlow, 2013-09-08

There were some interesting developments with regard to crypto and TLS this year at the BlackHat security conference. This first article is about the BREACH attack, CVE-2013-3587. For background, take a look at my earlier article that talks about TLS encryption algorithms.

BREACH is a refinement of the CRIME and TIME attacks. This attack takes advantage of the side channel information leak caused by compression. In the earlier CRIME attack, the attacker had to use TLS compression, but this attack uses HTTP compression which is very common and used on most pages. The attack recovers secrets from within the HTTP response body. The authors used it onstage to obtain a CSRF token and then to forge a request. I have no doubt this could be used in other ways with other secrets obtained from the response. Note that authentication cookies should not be in most responses, so they are safe for now.

The attack needs a man in the browser (MitB) who can send multiple requests to the same page. The MitB appends some data to the request and watches the size of the response. He makes sequential guesses and looks at the compressed size of the returned page to see if he guessed right.

For example, if your CSRF token on a particular page load is "CSRFToken=badguy", the attacker will cause the response to send an extra "CSRFToken=a" in the response. Let's assume "CSRFToken=" is compressed into 1 byte. Our response page would be 2 bytes larger; one byte for the compressed "CSRFToken=" and one byte for "a". The attacker then causes another load with "CSRFToken=b". This time, the response page is only 1 byte longer, which means that the compression algorithm successfully compressed the whole token. Now the attacker knows the first byte of the token. He starts iteratively guessing again at "CSRFtoken=ba" until he finds the whole string.

Note that this works against HTTP compression or TLS compression. It also works against any cipher.

The authors offer some mitigation options on their page. The easiest is to disable HTTP compression on any user input pages. This may be expensive. Two other suggestions are to separate user content from secrets and to randomize secrets per request. This last may be doable with an iRule, but I haven't tried yet. I also wonder if adding a few bogus secrets on each page would be able to slow the attacker.

So what does this mean to you? My suggestion would be to disable HTTP compression on any user input pages. Static content can continue to be compressed.

We knew this was coming; the CRIME & TIME attacks pointed out that it was possible and they even left room for better attacks. Now we have a proof of concept.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com