BREACH Attack

Xing Liu (xl76)

I. INTRODUCTION

At ekoparty security conference 2012, Juliano Rizzo and Thai Duong announced CRIME (Compression Ratio Info-leak Made Easy), which can obtain the information leaked by compression to recover the wanted plaintext. This method provided a way to recover the headers of an HTTP request, since we can always find cookies in headers and cookies plays crucial roles of authentication, this presents a significant attack.

But, this attack relied on a rarely used feature of TLS (Transport Layer Security): compression. as long as the browsers disable TLS/SSL (Secure Sockets Layer) compression then the attack will be completely mitigated.

However, hackers found actually TLS-protected traffic are still vulnerable, they change their attack target from HTTP requests to HTTP responses. BREACH (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext) is a security exploit against HTTPS when using HTTP compression. HTTP level are very common to use gzip, and it's likely the user input and secrets (such as CSRF [Cross-Site Request Forgery] tokens) are in the same HTTP responses, therefore in the same compression context. So this gave us a chance to get the secrets.

II. ATTACK PRINCIPLE

2.1 Overview

Many of the web services deliver secrets (like CSRF tokens) in the bodies of HTTP response. And it is also common for web services reflect user input in HTTP bodies. Since the gzip uses DEFLATE - a lossless data compression algorithm and associated file format that uses a combination of the Huffman coding and LZ77 algorithm - to compress the response, so the attacker can use URL parameter to guess the secrets.

We take Microsoft Outlook Web Access as an example: if the user makes following request:

GET /owa/?ae=Item&t=IPM.Note&a=New&id=**canary=<guess>**

Then the HTTP response might look like this:

https://malbot.net:443/owa/forms/ basic/
BasicEditMessage.aspx?ae=Item&t=IPM.Note&
amp;a=New&id=**canary=<guess>**
...
<td nowrap id="tdErrLgf"><a href="logoff.owa?
canary=d634cda866f14c73ac135ae858c0d894">Log Off</td>

Because DEFLATE can take advantages when repetition happens, so when the first character of guess matches the real secret ('d' in this case), then the algorithm will

compress body further. We only need check the bytes then know if our guess are correct or not. The attacker can guess the secrets byte by byte in this way.

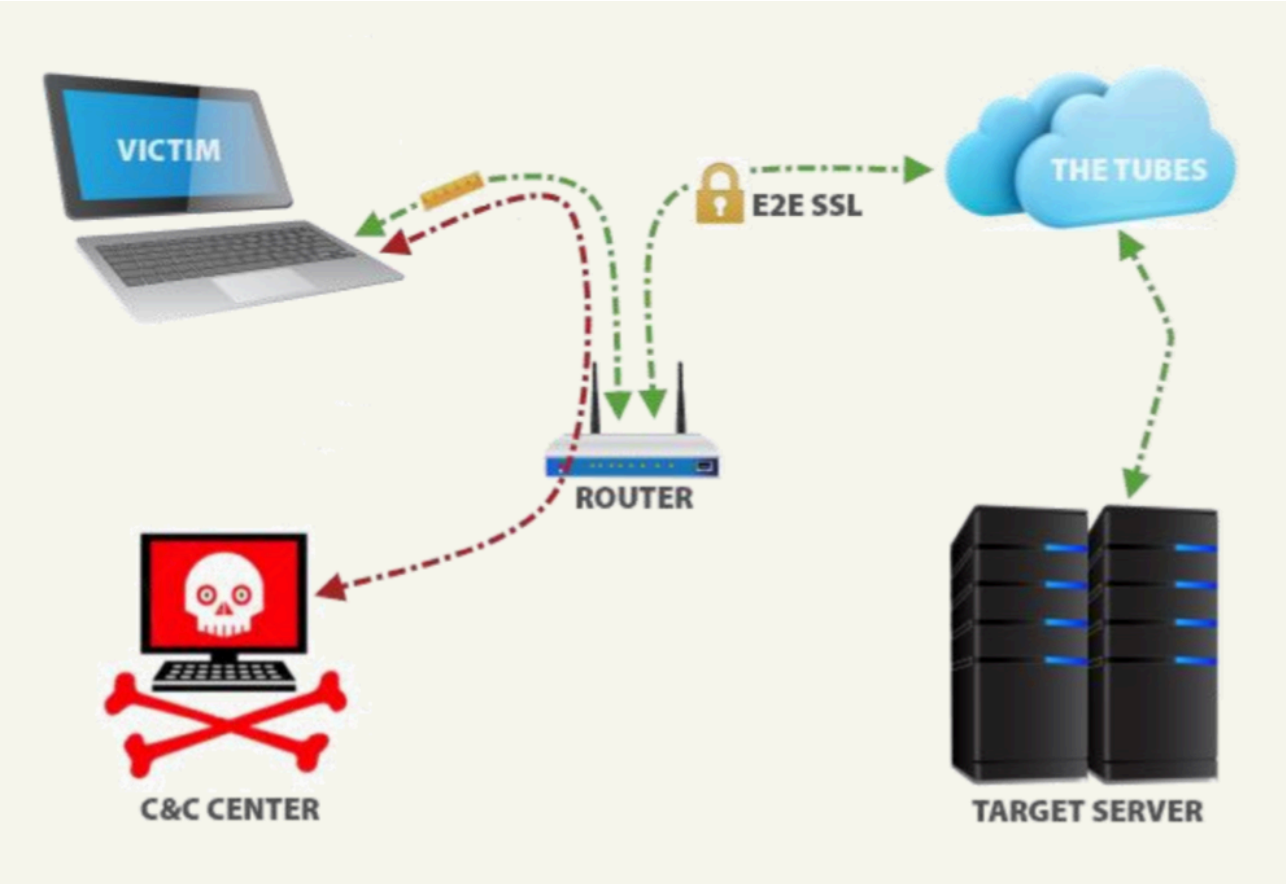
2.2 Requirements

In order to conduct this attack, we need the web service satisfied following requirements:

- Use a server that uses HTTP-level compression
- Reflect a secret in HTTP response bodies
- Reflect user input parameter in HTTP response bodies

2.3 Setup

We should assume the attacker has the ability to watch the victim's traffic, and make the victim to send HTTP requests to the vulnerable web server.



This attack is active online attack, it proceeds byte by byte. The attacker make the victim continuously send small packets to server to guess the secret.

III. Technical Challenges

3.1 Huffman Coding and Two-Tries

DEFLATE algorithm has a very important component: Huffman coding. It has a feature: encode those characters appearing more frequently into shorter codes. Note that our attack use the compressed length to determine if our guess are correct or not, so if the attacker guess a character which is incorrect but it happens

to be a heavy symbol, then it is possible that the compressed data is smaller than when the attack guess it correct.

To solve this problem, Rizzo and Duong used a “Two-Tries” method, instead of send a single request for each guess, we send two. We can see how it works:

GET /owa/?ae=Item&t=IPM.Note&a=New&id=
canary=<guess><padding>

GET /owa/?ae=Item&t=IPM.Note&a=New&id=
canary=<padding><guess>

The padding is a collection of characters from the complement of the alphabet for the secret. For instance, if the secret is known to be hex, then a candidate for padding could be “!$”.

If the guess is incorrect, the both guesses should result in compressed data of the same size. If the guess is correct, then the first request should has smaller compressed data size.

3.2 Charset Pools

We can also add all candidate characters to each request, for instance, suppose the secret is a hex string. Then we should format each request like this:

canary=4bf**d**!$-a-b-c-e-f-0-1-2-3-4-5-6-7-8-9

canary=4bf**e**!$-a-b-c-d-f-0-1-2-3-4-5-6-7-8-9

By doing this, we can also avoid the confusion Huffman Coding brings.

3.3 Guess Swap

This is also another variation of Two-Tries method. If the attacker has already known the first two characters fo the secret is 3a. Then we can submit two requests per guess with payload like:

canary=3a**c**

and

canary=3**c**a

we can calculate the score for each guess: the difference of size between the later response and the former. If ‘c’ is the correct guess, then ‘c’ should has the highest score.

3.4 Others

There are lots of other methods to deal with the problem: Compression Ratio for Guesses, Block Ciphers, Guess Windows, Encoding Issues, False Negatives and Dynamic Padding, False Positives and Looking Ahead.

IV. MITIGATION

4.1 Length Hiding

A crucial information the attacker should know is the length of the secret. But in fact this can only let the attacker use a little bit more time to measure the true length of the secret.

4.2 Separating User Input from Secrets

This method can solve this attack completely, let the secret and user-controlled text in separate compressed text. Like an application can use a secrets servlet.

4.3 Masking Secrets

The attack are based on the assumption that the secret remains the same for each requests. But we could use a new onetime pad P, to embed the secret S into $P||(P \oplus S)$.

V. OPINION and SUMMARY

The author admits that the side-channel is poorly understood and there may be lots of new variants to be discovered.

Like the attacker used a padding in Two-Tries method, now they just choose the padding randomly but maybe there is a formula which can calculate which padding will has the best performance. I think the white hats should also learn more about DEFLATE mechanism to prevent more attack methods.

Another issue is TLS and HTTP are not particularly special in the side-channel attack. Whenever the attacker can inject their text inside the text that will be compressed, the CRIME-like attack can be conducted. So all other protocols should also be aware of this attack as long as they using composition of encryption with compression.

Actually during the study of this topic I found the security issues are quite fun and you have to do game with yourself to learn the strategies of attacking and defending, I decided to take Computer Security course next semester.

VI. REFERENCE

1. https://en.wikipedia.org/wiki/BREACH
2. http://breachattack.com/
3. http://breachattack.com/resources/BREACH%20-%20SSL,%20gone%20in%2030%20seconds.pdf
4. https://github.com/nealharris/BREACH