TLS Compression

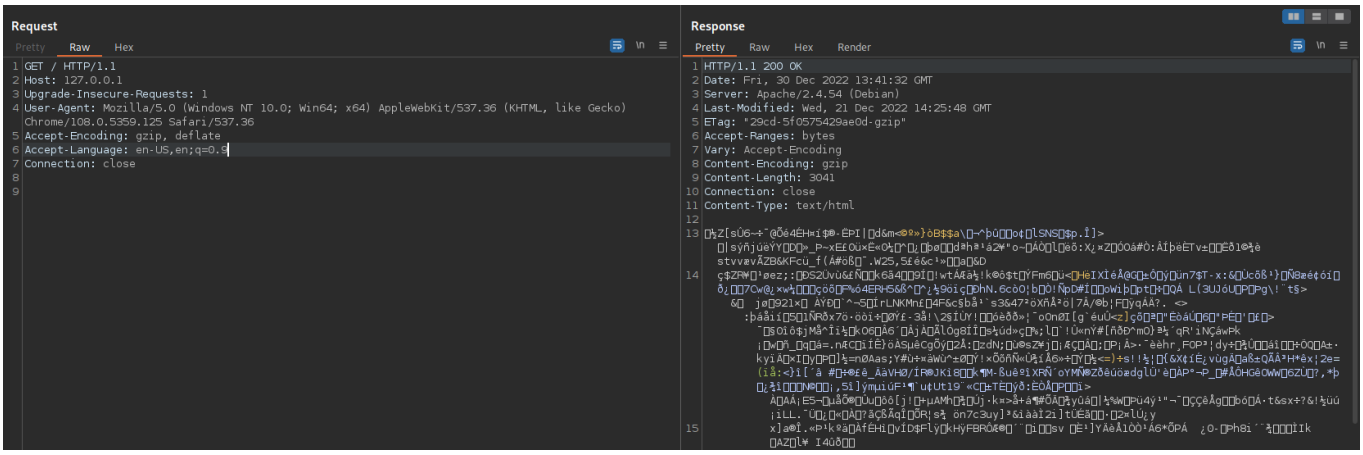
Intro to Compression

Compression can be used to reduce the size of data. This can be particularly important when transmitting data, as a reduced size enables communication over connections of limited strength and speeds up the transmission.

HTTP Compression

Compression can be applied at the application layer level. In a web context, this means applying compression at the HTTP level. More specifically, HTTP requests can be compressed by the webserver. This is indicated by the `Content-Encoding` HTTP header. This header can be set to the values `gzip`, `compress`, or `deflate` to inform the web browser what kind of compression method was used to compress the data. The web browser is then able to unpack the compressed data and display the web page correctly.

If compression is applied at the HTTP level, the compressed response looks similar to this:



Since the compression is applied only to the HTTP body, all headers are transmitted uncompressed and in their original state.

Note: Most proxies like Burp automatically detect compressed responses and unpack the response by default. So to view the compressed response, this option needs to be disabled.

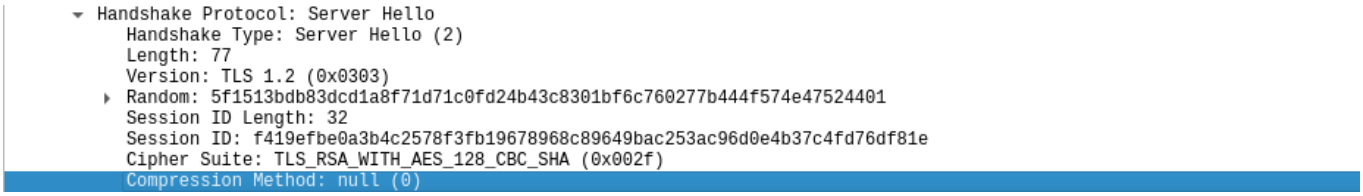
TLS Compression

Instead of applying compression at the application layer level, it can also be applied at the TLS level. This means that not only the application layer payload but all application layer data is compressed. In a web context, this means that the whole response is compressed, including all HTTP headers.

Since the compression is applied at the TLS level, it is completely transparent to any web server or web proxy such that we cannot detect it in Burp. However, whether TLS compression is used or not is negotiated in the TLS handshake.

We can see the compression methods supported by the client in the `ClientHello` message in the `Compression Methods` Field:

The compression method is then chosen by the server in the `ServerHello` message:



Example: LZ77

As an example of how compression works, let's look at the LZ77 algorithm. LZ77 works by keeping a dictionary of recently encountered character strings and replacing repeatedly encountered sequences with a reference to the first occurrence. As a simplified example, consider the following sentence: `I like HackTheBox's HackTheBox Academy`. This would be compressed as `I like HackTheBox's <13,10> Academy`. We can see that the second occurrence of `HackTheBox` has been replaced with a back reference of two numbers, the back pointer and the length. To unpack the original sentence, we follow the back pointer by moving backward 13 characters and replacing the reference with the following 10 characters, resulting in the word `HackTheBox`.

It is important to understand that LZ77 uses a sliding window, so it only considers a recent history of words for compression and does not operate on the text as a whole. This is important for the upcoming compression attacks.

CRIME & BREACH

CRIME

Compression Ratio Info-leak Made Easy (CRIME) is a compression-based attack that targets TLS compression. As such, it can target sensitive information present in the HTTP body and HTTP headers such as session cookies. To successfully exploit CRIME, an attacker needs to be able to intercept traffic from the victim, as well as force the victim to adjust the request parameters slightly, for instance via malicious JavaScript code. The attacker also needs to know the name of the session cookie and the length of its value.

Let's look at an example to illustrate how the attack works. For our example we make the following assumptions:

- Let's assume the session cookie is called `sess` and has a length of 6 characters. The victim's session cookie's value is `abcdef`
- Our target website is called `crime.local` and we are attacking the path `/crime.html`
- A sliding-window compression algorithm is used that works similarly to LZ77 as discussed in the previous section

The attacker then forces the victim to request the target website but appends an extra HTTP parameter to the URL with the same name as the session cookie and an arbitrary value with the correct length. An exemplary request could look like this:

Code: http

```
GET /crime.html?sess=XXXXXX HTTP/1.1
Host: crime.local
Cookie: sess=abcdef
```

This request is now compressed using a sliding window compression algorithm, meaning that the `sess=` string present in the `Cookie` Header is replaced with a back-reference to the `sess=` string appended by the attacker to the query string. The compressed data is then encrypted. Since the attacker can intercept the ciphertext, he denotes the ciphertext size.

In the second step, the attacker changes the query parameter slightly to brute-force the value of the session cookie character by character from left to right. So, the next request might look like this:

Code: http

```
GET /crime.html?sess=aXXXXX HTTP/1.1
Host: crime.local
Cookie: sess=abcdef
```

In this case, the compression algorithm can now replace the string `sess=a` with a back-reference, since an additional character is the same in the cookie's value and query string. This means the resulting compressed data is smaller, potentially resulting in a smaller ciphertext. The attacker notices the smaller ciphertext and knows that the current character is correct. He can therefore move on to the next character:

Code: http

```
GET /crime.html?sess=aaXXXX HTTP/1.1
Host: crime.local
Cookie: sess=abcdef
```

The attacker can apply this technique recursively to brute-force all characters of the cookie, thereby leaking the session cookie. Depending on the length of the session cookie, a lot of requests are required to perform this attack.

BREACH

Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext (BREACH) is a variant of CRIME that targets HTTP-level compression. Since HTTP-level compression can only compress the HTTP body, BREACH is unable to target session cookies that are transmitted in HTTP headers. Therefore, potential targets of BREACH are sensitive information contained in the HTTP body such as CSRF-tokens.

Conceptually, BREACH works identically to CRIME with the slight difference that the webserver's response needs to contain a reflected value in the body for the attack to work since the attacker cannot simply adjust the query string as it is not part of the HTTP body.

Tools & Prevention

The simplest countermeasure to prevent CRIME attacks is to disable TLS-level compression. Alternatively, compression algorithms that do not fulfill the requirements needed for the successful exploitation of CRIME can be used to mitigate this attack. As of today, up-to-date webservers and libraries are not vulnerable to CRIME as patches have been applied.

Similarly, the simplest countermeasure to prevent BREACH attacks is to disable HTTP-level compression.