

THE 'ORIGINULL' PRIVACY HACK



Critical Issue Affects Privacy of 1-Billion Facebook Messenger Users; Potentially Affects Millions of Other Websites

An estimated 1.8-billion active monthly users trust Facebook to keep their accounts, user details and communications secure. On the one hand, the social network is based on sharing: users post some 350-million photos daily and nearly 300-thousand statuses per minute.

On the other hand, there is Facebook Messenger, one of the network's most popular features, with 1-billion active monthly users. Unlike photo and status features designed specifically for sharing and publishing, the power of Messenger is in the ability to communicate privately.

In this post, we will describe a severe security vulnerability found on Facebook, which also potentially affects millions of websites using origin null restriction checks, threatening user privacy and opening site visitors up to malicious entities. The hack, dubbed "**Originull**," enables an attacker to access and view all of a user's private chats, photos and other attachments sent via Facebook Messenger. The issue was discovered and reported to Facebook by team researcher Ysrael Gurt.

The 'Untechnical' Explanation

The vulnerability discovered is a cross-origin bypass-attack which allows the hacker to use an external website to access and read a user's private Facebook messages. Normally, the browser protects Messenger users from such occurrences by only allowing Facebook pages to access this information. However, Facebook opens a "bridge," in order to enable "subsites" of Facebook.com to access Messenger information. A vulnerability in the manner in which Facebook manages the identity of these subsites makes it possible for a malicious website to access private Messenger chats.

For example, if the user opens a website to which the hacker has directed them (via a malicious ad, a security issue, or the hacker's own website), the hacker can then see all the Facebook Messenger chats, photos and other attachments which the user sends or

THE 'ORIGINULL' PRIVACY HACK

receives. This happens even if the user sends the messages by way of another computer, or from their personal mobile device!

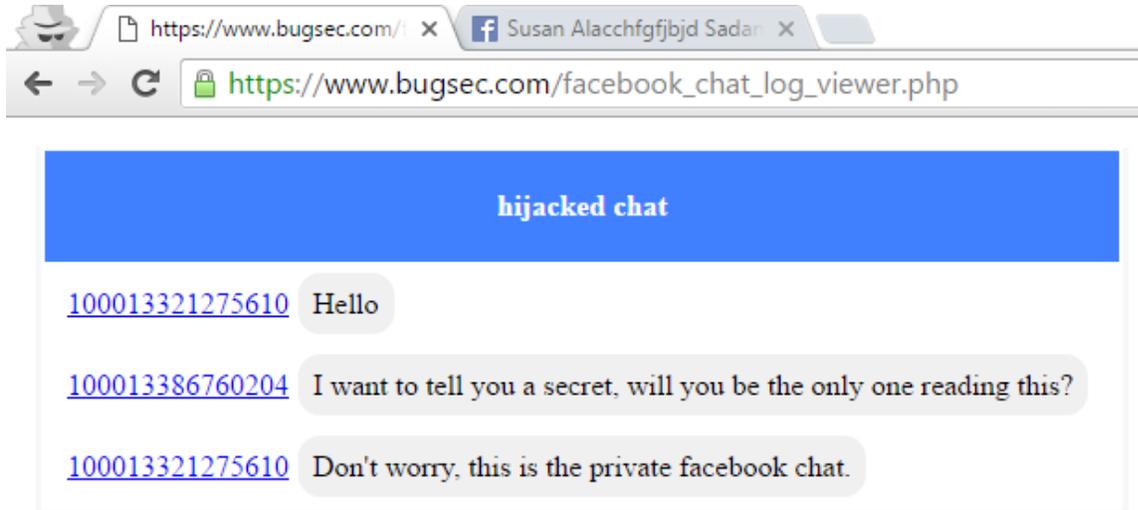


Image 1: The chat appears on the BugSec website. The user ID is shown to the left.

The Technical Explanation

This is a deeper technical explanation of the issue above.

Facebook Messenger chats are managed from a separate server located at the address: {number}-edge-chat.facebook.com. The chat itself runs on the domain www.facebook.com.

Communication between the JavaScript and the server is done by XML HTTP Request (XHR).

In order to access the data that arrives from 5-edge-chat.facebook.com in JavaScript, Facebook must add the "Access-Control-Allow-Origin" header with the caller's origin, and the "Access-Control-Allow-Credentials" header with "true" value, so that the data is accessible even when the cookies are sent.

THE 'ORIGINULL' PRIVACY HACK

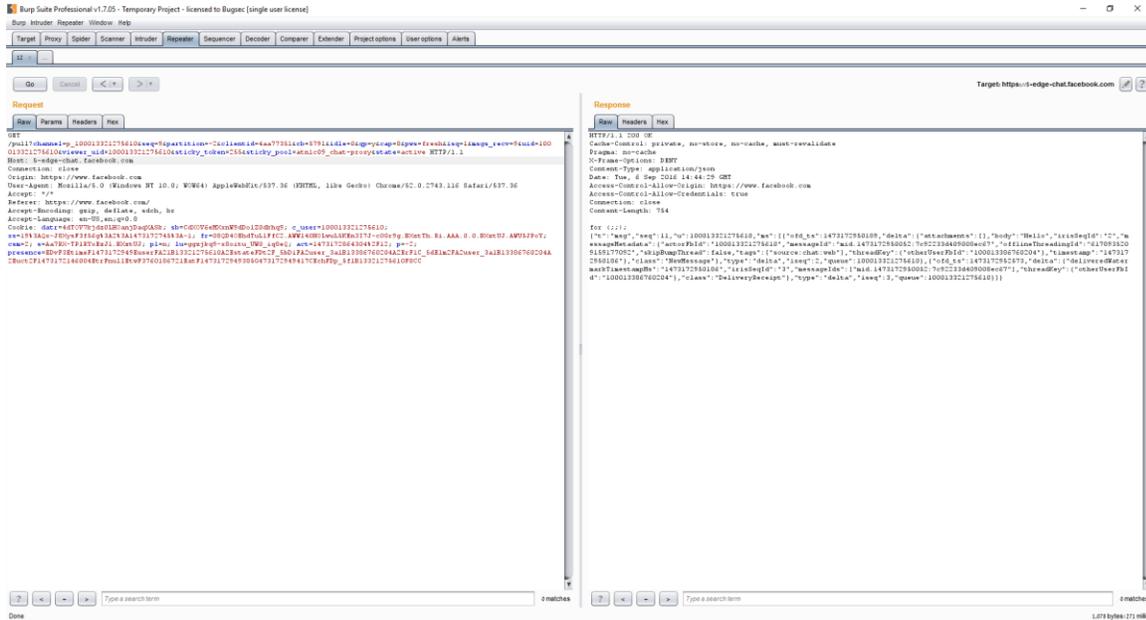


Image 2: The original request.

So far, this appears to be a normal CORS process. In order to prevent other sites from accessing the data, Facebook checks the origin header. If the request came from an unauthorized origin, the server returns 400 with the value “badorigin” in the header “x-fb-chat-failure-reason.”

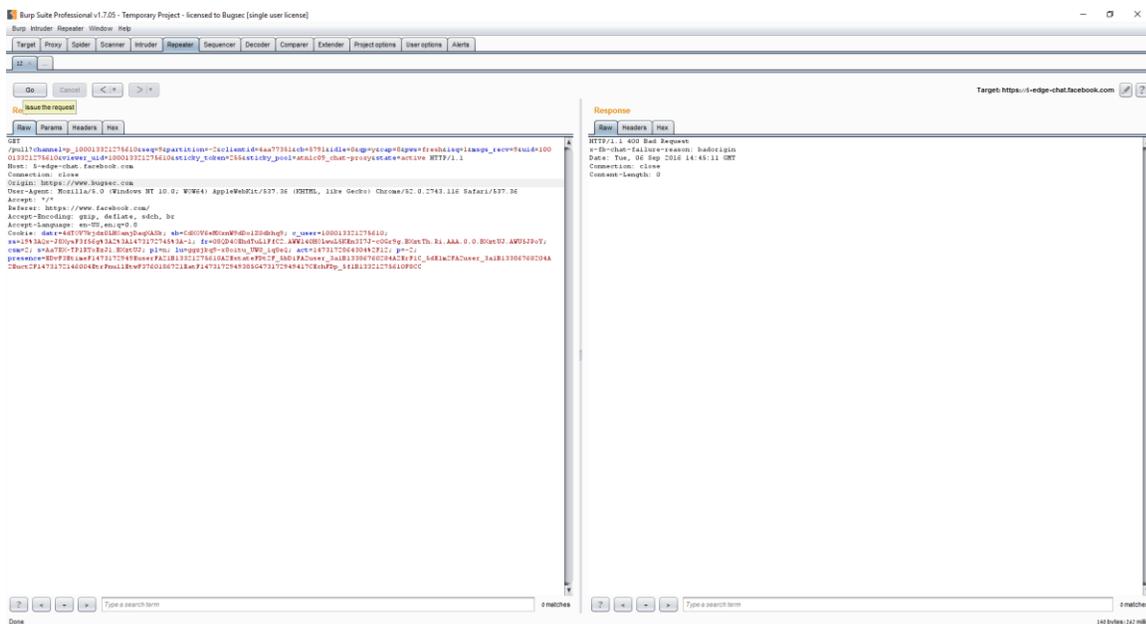


Image 3: Request from a different origin.

THE 'ORIGINULL' PRIVACY HACK

Now for the good stuff: Facebook also allows normal GET requests to the chat domain. But normal GET requests do not come with an origin header. The origin header is a special header sent by the browser only with XHR requests.

Thus, when the server receives a GET request, it does not include the “origin” header. In many development languages, nonexistent headers are represented by the “null” value. If Facebook expected to receive “null” in the “origin” header, it would not block requests from this “origin.”

Most likely, the filtering mechanism is separated from the responder mechanism, and the responder assumes that the value in the “origin” header is allowed, because if not, the filter would already have dropped the request. This development design allows Facebook to add authorized origins by changing code in one position only.

In conclusion, the “null” origin passes the filter check, allowing it to pass as a normal “GET” request. The responder took the value of the “origin” header from the request, and placed it as the value for “Access-Control-Allow-Origin” header in the response.

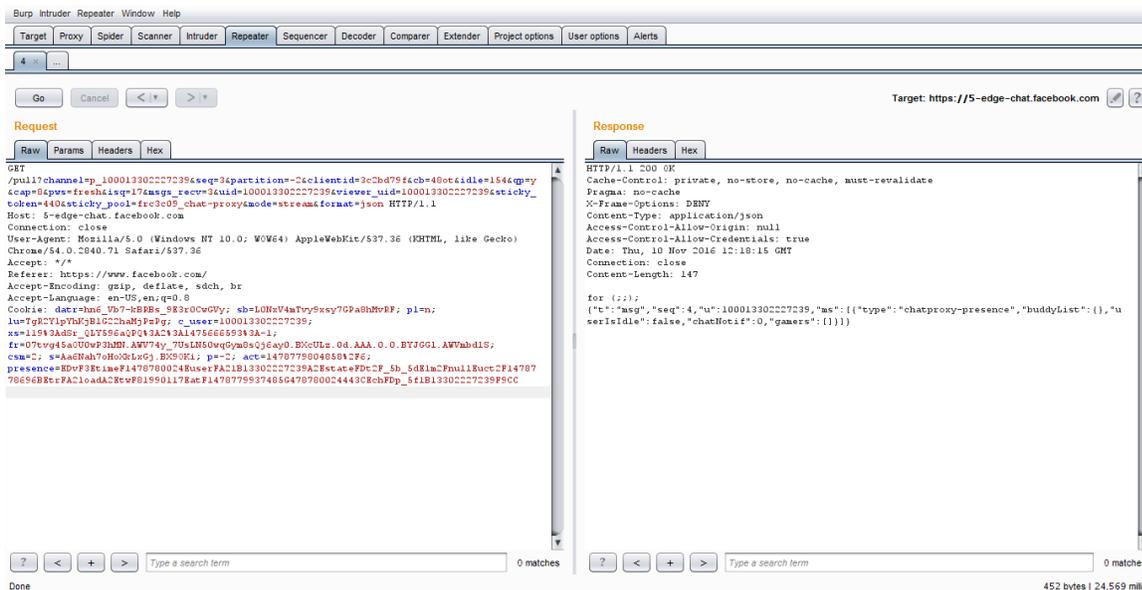


Image 4: Request without origin.

In this manner, we were able to ascertain that were we to send a request from the page with a “null” origin, we would most likely get the “Access-Control-Allow-Origin” header in response to the “null” origin.

THE 'ORIGINULL' PRIVACY HACK

First, we tested this assumption with a “burp” – a tool enabling us to modify every request for information. When we sent the request with the origin “null,” Facebook responded with a “null” value on “Access-Control-Allow-Origin.”

This meant that if we could cause the browser to send “null” in the “origin” header, we would get a “null” value in the “Access-Control-Allow-Origin.”

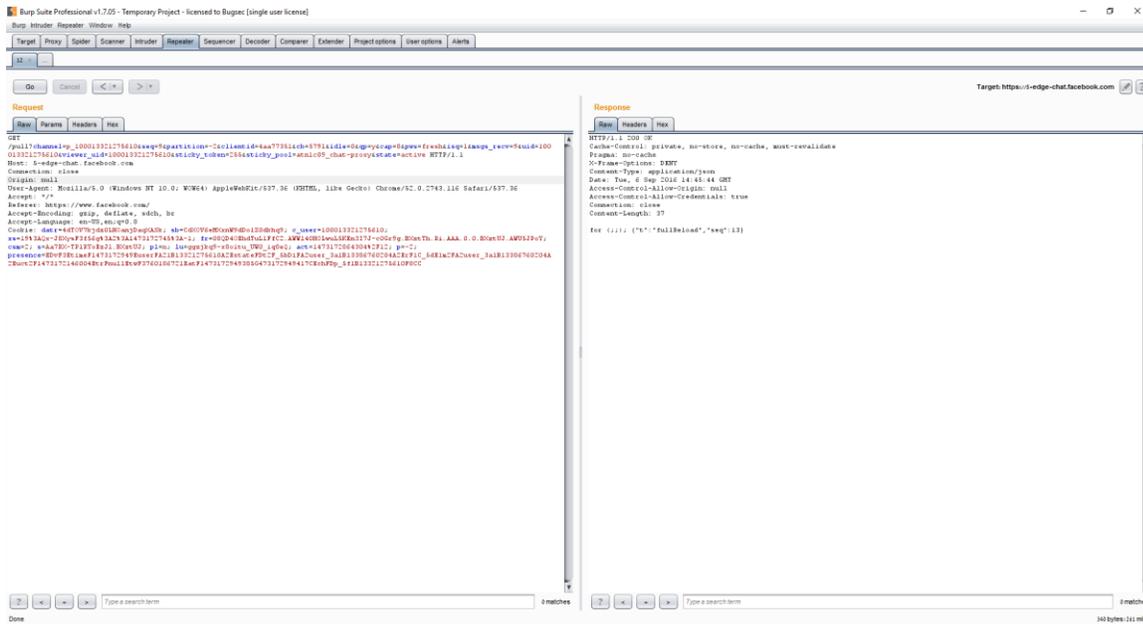


Image 5: Request with “null” origin.

In our testing, we also found that it was possible to use the data scheme in order to send requests with the origin “null.” When a data scheme is used, the browser sets the origin to “null” for security purposes.

THE 'ORIGINULL' PRIVACY HACK

Encode to Base64 format

Simply use the form below

```
<script>alert("document.domain = " + document.domain + "")</script>
```

> ENCODE < UTF-8 (You may also select output charset.)

```
PHNjcmlwdD5hbGVydCgnZG9jdW1lbnQuZG9tYWludD0glicgKyBkb2N1bWVudC5kb21haW4gKyAnIicpPC9zY3JpcHQ+
```

Image 6: The example code.

```
<html>
<head>
<html><head><meta charset="UTF-8"><script>try{window.opener = null;}catch(e){}</script><meta http-equiv="Refresh" content="0";
URL=data:text/html;base64,PHNjcmlwdD5hbGVydCgnZG9jdW1lbnQuZG9tYWludD0glicgKyBkb2N1bWVudC5kb21haW4gKyAnIicpPC9zY3JpcHQ+ />
</head>
<body>
</body>
</html>
```

Image 7: The final HTML.

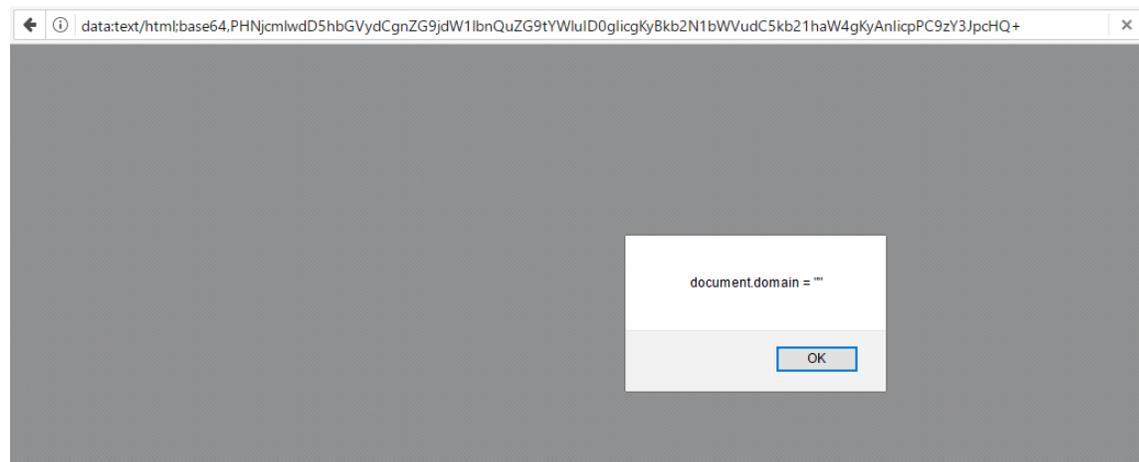


Image 8: The HTML in Firefox.

THE 'ORIGINULL' PRIVACY HACK



Image 9: The HTML in Chrome.

Thus, based on data scheme testing, and our knowledge of Facebook handling methodology, we could produce a valid attack on Messenger chat users.

Some Background on the Facebook Chat API

Facebook uses a continually repeated XHR request to the server to receive newly arrived messages. The server responds only when a message arrives, or at timeout. Using this method means that there is always an XHR request waiting for a server response. When the server responds to a request, JavaScript code opens a new request to the server.

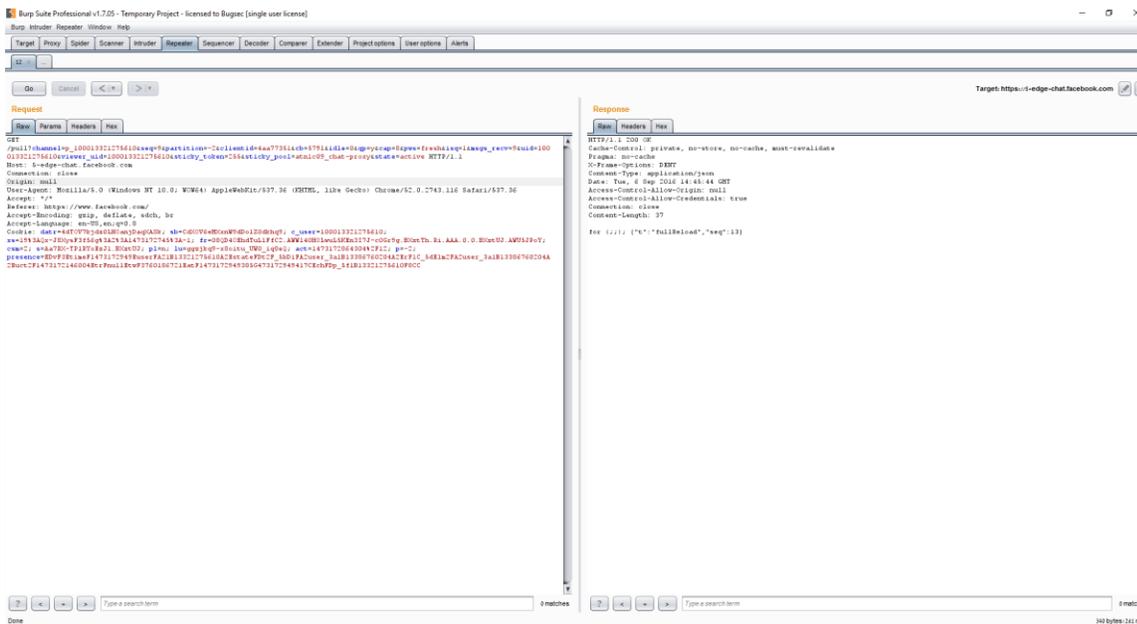


Image 10: Server response at timeout.

THE 'ORIGINULL' PRIVACY HACK

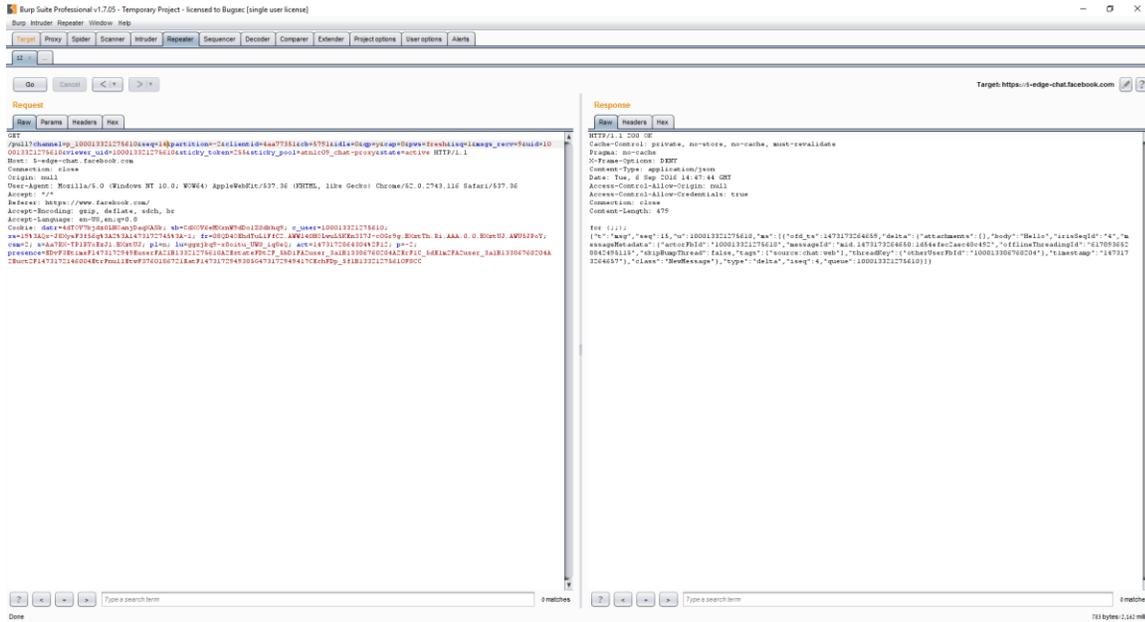


Image 11: Server response for a new message.

To ensure that the messages arrive in the correct order, every request has a sequence number (represented by “seq” in the requests).

The right “pool” value are built using the following steps:

1. Every request is part of a “pool.”

The right “pool” value is sent as a response to an empty request.

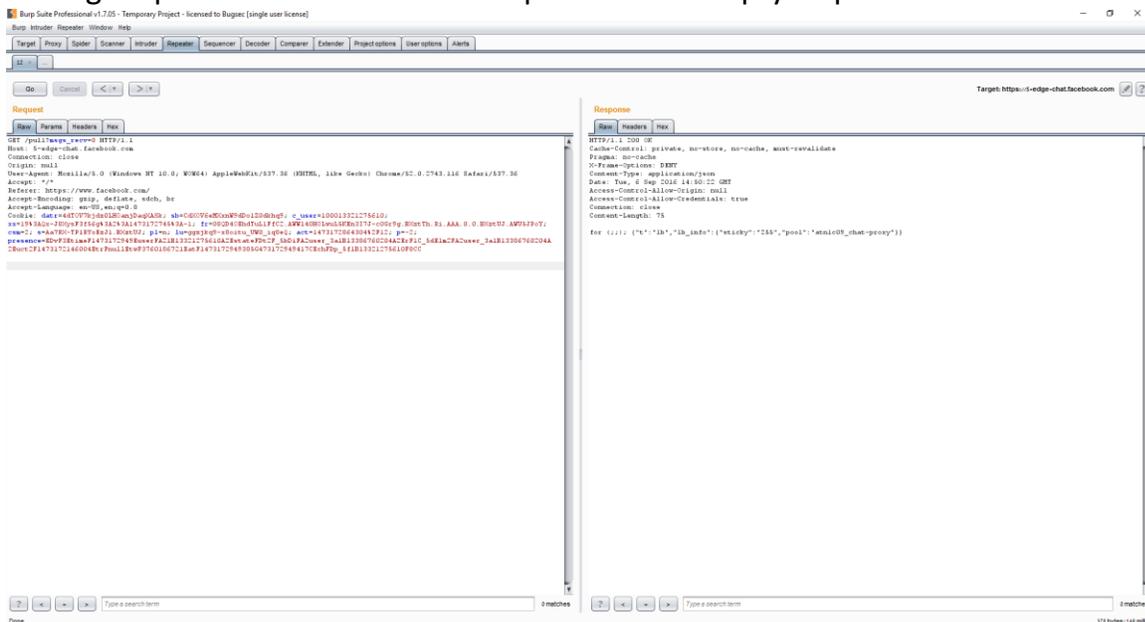


Image 12: Receiving the “pool” value.

THE 'ORIGINULL' PRIVACY HACK

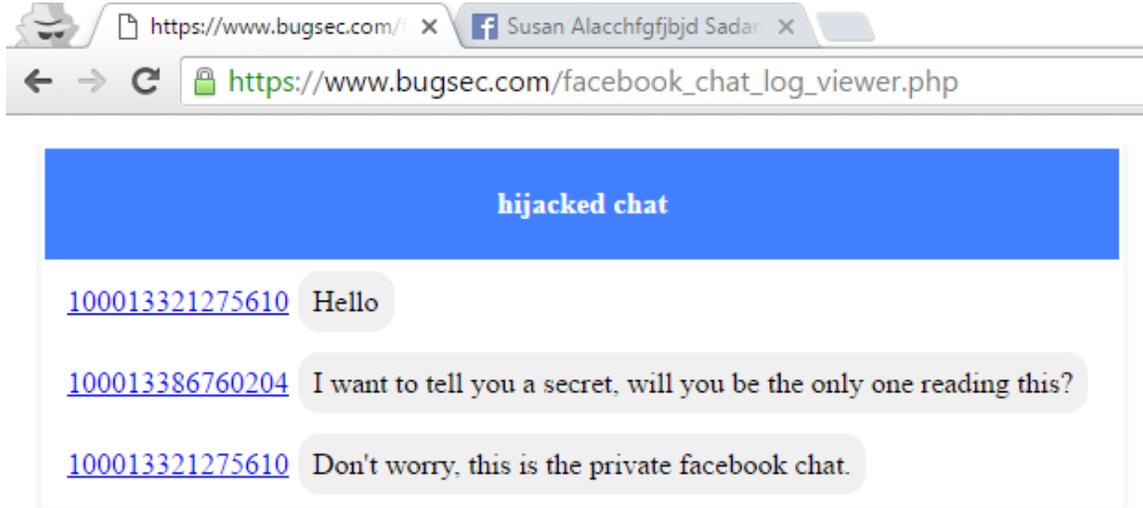


Image 17: The chat appears on BugSec website. The user ID is shown on the left.

Conclusion

The vulnerability was reported to Facebook through its Bug Bounty program. They responded quickly, and had fixed the vulnerability within several days.