Penetration Test Report

Open Technology Fund

V 1.0
Amsterdam, December 18th, 2024
Public

Document Properties

Client	Open Technology Fund
Title	Penetration Test Report
Target	Letro for Android
Version	1.0
Pentester	Abhinav Mishra
Authors	Abhinav Mishra, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	December 8th, 2024	Abhinav Mishra	Initial draft
0.2	December 10th, 2024	Marcus Bointon	Review
1.0	December 18th, 2024	Marcus Bointon	1.0

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between November 25, 2024 and November 29, 2024, Radically Open Security B.V. carried out a penetration test for Open Technology Fund

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of work

The scope of the penetration test was limited to the following target:

- Letro for Android

The scoped services are broken down as follows:

- Pentest Of Android App: 2.5 days
- Reporting: 1 days
- **Total effort: 3.5 days**

1.3 Project objectives

ROS will perform a penetration test of the Letro Android app with OTF in order to assess the security of Android app. To do so ROS will access the code at `https://github.com/relaycorp/letro-android` and guide OTF in attempting to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

1.4 Timeline

The security audit took place between November 25, 2024 and November 29, 2024.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 2 Moderate and 1 Low-severity issues.

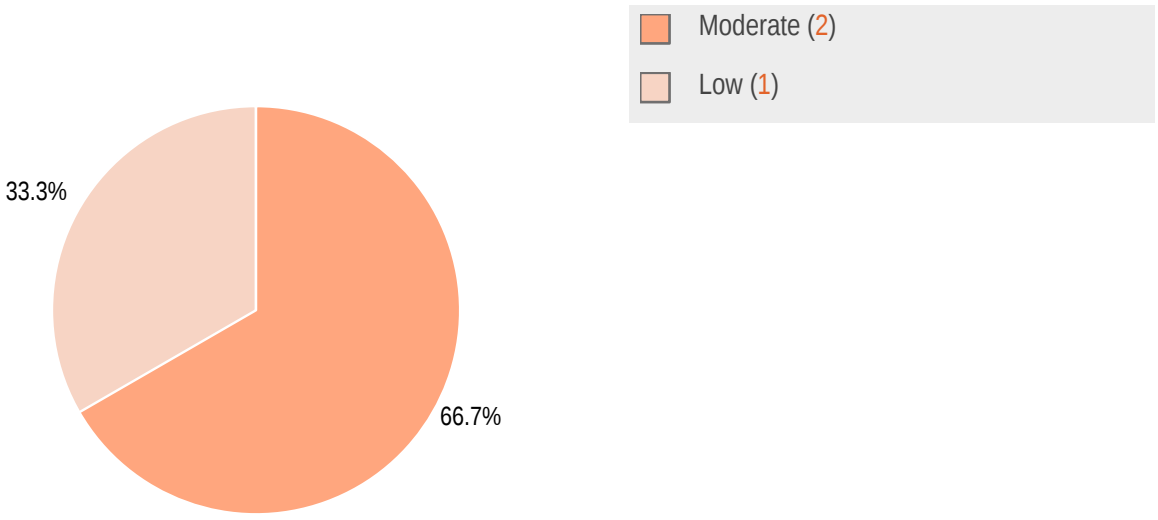
The security review of the Letro Android application was conducted over approximately 20 hours, focusing specifically on its implementation. The application offers a limited attack surface, as it provides minimal functionality, with the core operations being handled by the Awala component, which is the subject of a separate review. During the assessment, we identified three security concerns: sensitive information being logged in logcat OTF-001 (page 10), the absence

of PIN or biometric authentication OTF-002 (page 12), and the lack of screenshot protection OTF-003 (page 12). While these findings are important, the overall risk is considered moderate due to the limited scope and functionality of the Android app. Additionally, we have provided a non-exhaustive list of non-findings in the report, which includes other important test cases that we performed during the assessment. Addressing the identified vulnerabilities will improve the apps security without requiring significant changes to its structure.

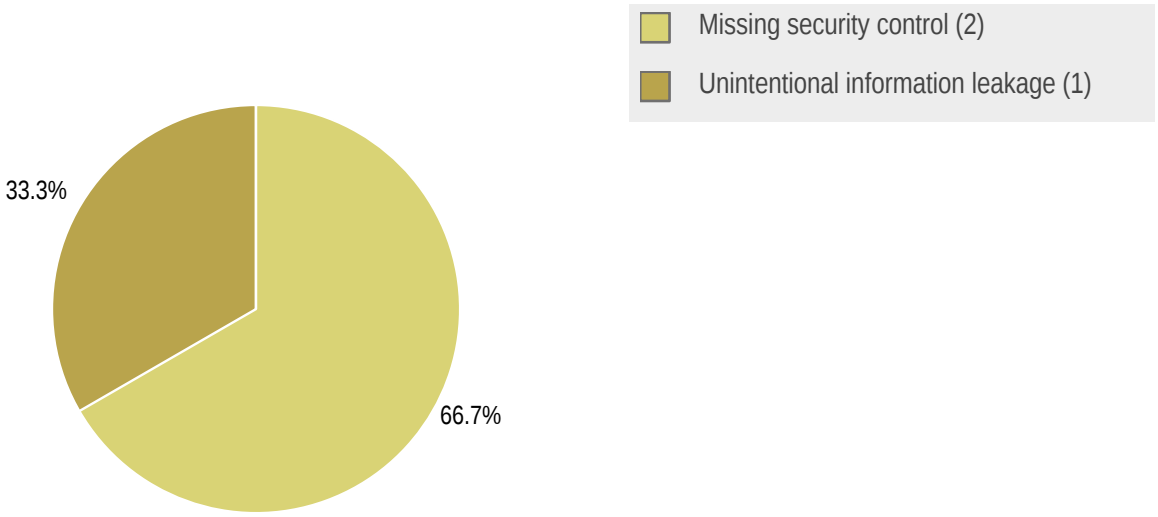
1.6 Summary of Findings

ID	Type	Description	Threat level
OTF-001	Unintentional information leakage	The Android app leaks sensitive information to logcat, potentially exposing confidential data to other applications or processes.	Moderate
OTF-002	Missing security control	The application does not implement a PIN or biometric lock mechanism, which leaves the app data accessible to unauthorized users if the device is unlocked.	Moderate
OTF-003	Missing security control	The application does not implement screenshot protection, which might allow other applications with the necessary permissions to capture the app's screen content.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
OTF-001	Unintentional information leakage	• Sanitize all log output in non-debug builds. • Handle logging from inside the app instead of using system logging.
OTF-002	Missing security control	• Implement a PIN or biometric lock feature that is required after a timeout or app launch.
OTF-003	Missing security control	• Implement the FLAG_SECURE flag to prevent screenshots or screen recording.

2 Methodology

2.1 Planning

Our general approach during penetration tests is as follows:

1. **Reconnaissance**
   We attempt to gather as much information as possible about the target. Reconnaissance can take two forms: active and passive. A passive attack is always the best starting point as this would normally defeat intrusion detection systems and other forms of protection afforded to the app or network. This usually involves trying to discover publicly available information by visiting websites, newsgroups, etc. An active form would be more intrusive, could possibly show up in audit logs and might take the form of a social engineering type of attack.
2. **Enumeration**
   We use various fingerprinting tools to determine what hosts are visible on the target network and, more importantly, try to ascertain what services and operating systems they are running. Visible services are researched further to tailor subsequent tests to match.
3. **Scanning**
   Vulnerability scanners are used to scan all discovered hosts for known vulnerabilities or weaknesses. The results are analyzed to determine if there are any vulnerabilities that could be exploited to gain access or enhance privileges to target hosts.
4. **Obtaining Access**
   We use the results of the scans to assist in attempting to obtain access to target systems and services, or to escalate privileges where access has been obtained (either legitimately though provided credentials, or via vulnerabilities). This may be done surreptitiously (for example to try to evade intrusion detection systems or rate limits) or by more aggressive brute-force methods. This step also consist of manually testing the application against the latest (2021) list of OWASP Top 10 risks. The discovered vulnerabilities from scanning and manual testing are moreover used to further elevate access on the application.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.

- **High**

  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**

  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**

  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**

  Low risk of security controls being compromised with measurable negative impacts as a result.

3 Reconnaissance and Fingerprinting

We were able to gain information about the software and infrastructure through the following automated scans. Any relevant scan output will be referred to in the findings.

- SonarQube – https://github.com/SonarSource/sonarqube
- Frida – https://github.com/frida
- Burp Suite Professional – https://portswigger.net/burp/pro

4 Findings

We have identified the following issues:

4.1 OTF-001 — (Letro) Sensitive Information Being Logged

Vulnerability ID: OTF-001

Vulnerability type: Unintentional information leakage

Threat level: Moderate

Description:

The Android application `tech.relaycorp.letro (version 0.4.0)` captures and stores information, some of which may be sensitive, in the system logs. This practice could expose confidential data, making it accessible to other applications or processes that have permission to read these logs, potentially leading to privacy or security concerns.

Technical description:

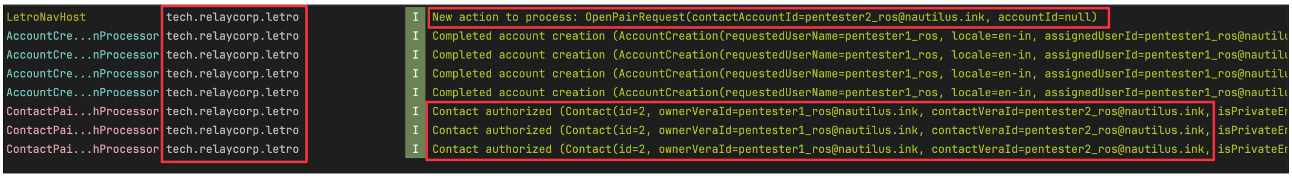
The app enables users to communicate securely with end-to-end encryption, even in areas where internet access may be censored. This makes it crucial to ensure that data remains protected while at rest. Android's `Log.*` statements write to a shared memory buffer called logcat. Since Android 4.1 (API level 16), only privileged system applications can access logcat by declaring the `READ_LOGS` permission. However, Android supports a vast array of devices, and some pre-installed applications may still include the `READ_LOGS` privilege.

During the audit of the Letro Android application, we observed that the app writes a significant amount of information to logcat. Some notable instances include:

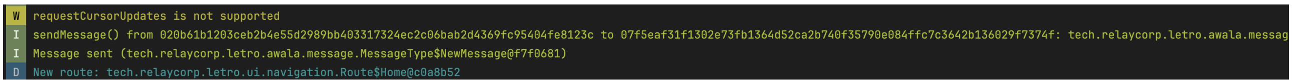
Account creation

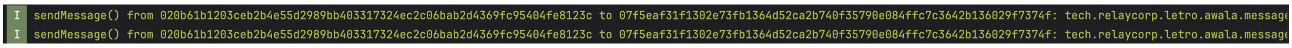
```
Receive message: application/vnd.relaycorp.letro.account-creation
uid=10222(tech.relaycorp.letro) AwalaManagerMes identical 1 line
Receive message: application/vnd.relaycorp.letro.account-creation
Completed account creation (AccountCreation(requestedUserName=abhinavmishra, locale=en-in, assignedUserId=abhinavmishra@nautilus.ink))
Message application/vnd.relaycorp.letro.account-creation processed
Completed account creation (AccountCreation(requestedUserName=abhinavmishra, locale=en-in, assignedUserId=abhinavmishra@nautilus.ink))
Message application/vnd.relaycorp.letro.account-creation processed
```

Pairing process details



Message sender and receiver id

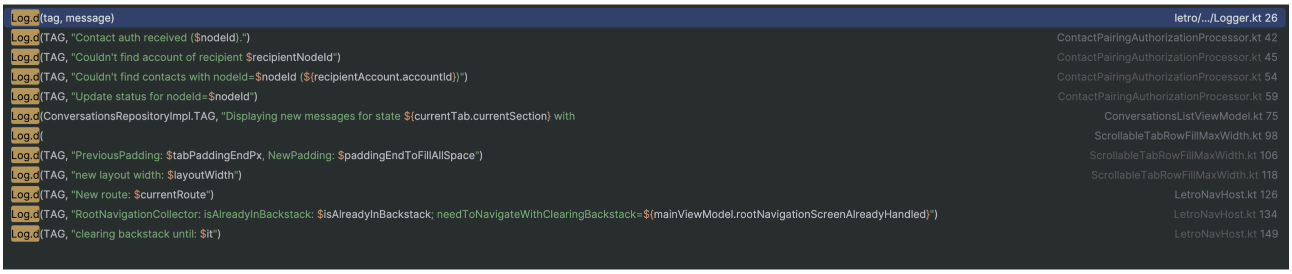




There are other points in the code which create log entries. For example:

```
Log.w(TAG, IllegalStateException("You cannot edit this contact. Contact belongs to
    ${contacts.firstOrNull()?.ownerVeraId}, but yours is $currentAccountId")) // TODO: log?
```

Debug logs



With only access to these logs, it's possible to see details like the accounts being used (or created) in the app, pairing details (veraID), and details (veraId) of message sender and receiver.

Impact:

Internal data, including unique identifiers of both message senders and receivers of messages may be exposed, possibly leading to identification of individuals, or metadata leakage revealing connections between users.

Recommendation:

Make sure all logging to logcat is sanitized in non-debug builds of your application, removing any potentially sensitive data. To further enhance security, strip out all log levels except warnings and errors. For more detailed logging

needs, avoid using the system log and instead manage your logs directly in internal storage. Reference: `https://developer.android.com/privacy-and-security/risks/log-info-disclosure`

4.2 OTF-002 — (Letro) Lack of PIN/Biometric Lock on the Application

Vulnerability ID: OTF-002

Vulnerability type: Missing security control

Threat level: Moderate

Description:

The application does not implement a PIN or biometric lock mechanism, which leaves the app data accessible to unauthorized users if the device is unlocked.

Technical description:

The application can be accessed directly without any client side authentication, as it lacks a PIN or biometric lock feature. Given that the app is intended for users in regions where internet access is monitored or restricted, implementing such a security control is crucial to safeguard user data and privacy.

Impact:

Without a PIN or biometric lock, sensitive email data stored or accessed within the app can be exposed, compromising user privacy and the confidentiality of communications.

Recommendation:

Implement a PIN or biometric lock feature to provide an additional layer of protection for accessing the application. This lock should trigger after a configurable timeout or upon the app's re-launch.

4.3 OTF-003 — (Letro) Lack of Screenshot Protection in the Application

Vulnerability ID: OTF-003

Vulnerability type: Missing security control

Threat level: Low

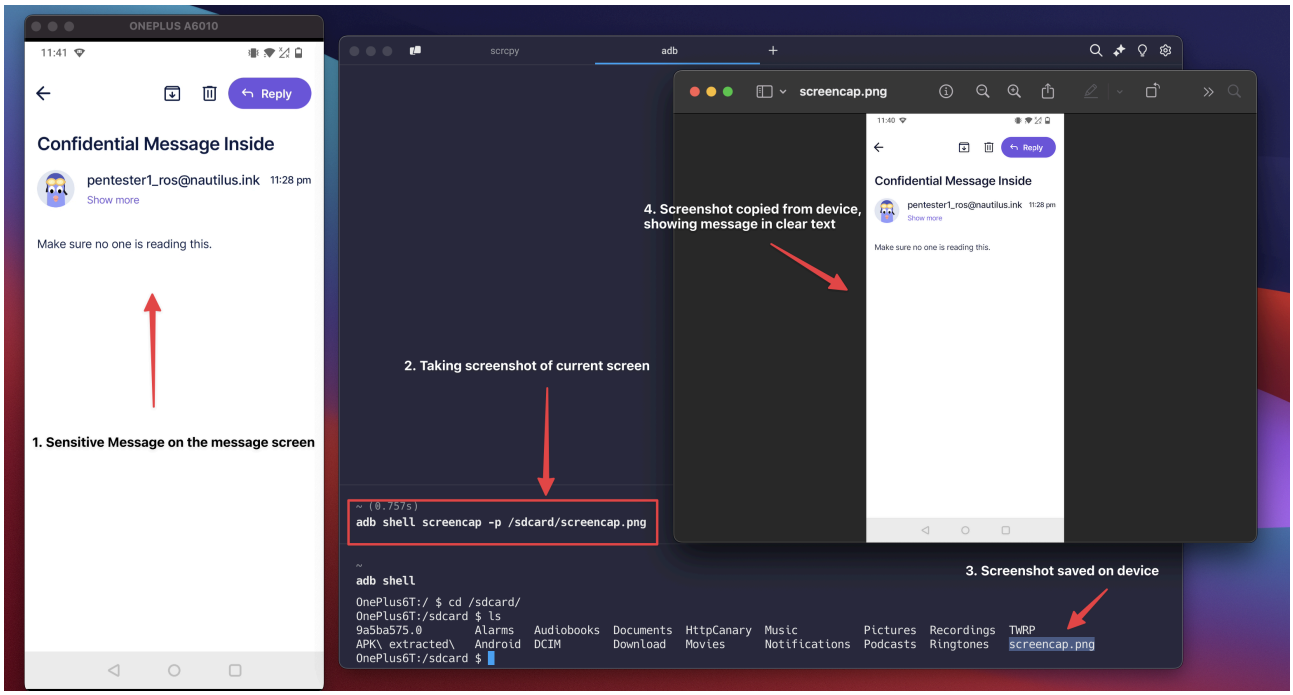
Description:

The application does not implement screenshot protection, which might allow other applications with the necessary permissions to capture the app's screen content.

Technical description:

Android provides a flag (`FLAG_SECURE`) that can be set on application windows to prevent screenshots or screen recording. This flag is not implemented in the Letro application, leaving it vulnerable to screen capture by apps with privileges like `MEDIA_PROJECTION` or accessibility permissions.

Screenshot capture



When a window is flagged with `FLAG_SECURE`, Android prevents screenshots from being taken and prevents the window from being displayed on a non-secure display. This helps to protect the information that is being displayed in the window from being accessed by unauthorized people.

Reference: https://developer.android.com/security/fraud-prevention/activities.

Impact:

A malicious app or entity could potentially capture background screenshots of the application.

Recommendation:

To mitigate this risk, implement the `FLAG_SECURE` flag. This ensures that when the app transitions to the background or a screenshot attempt is made, the resulting image is blank, safeguarding sensitive content.

5 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

5.1 NF-001 — Test Cases

During the pentest, we executed a series of test cases on the Android app. While some tests led to the discovery of vulnerabilities, the majority did not. The following is a non-exhaustive list of test cases we performed.

- Test for insecure data storage: Check whether sensitive data is being stored locally in the internal storage without encryption.
- Test for hardcoded sensitive information: Inspect the application code for hardcoded sensitive information like credentials.
- Test for insecure authentication mechanisms: Ensure that authentication methods are implemented securely. We noticed that the application does not implement any app-based authentication like a PIN or biometric login; This has been reported as a finding.
- Test for unprotected app components: Check if any exported activities, services or broadcast receivers can be accessed or manipulated by other apps. We noticed some of the broadcast receivers which do not need any permission, however, they could not be exploited.
- Test for improper input validation: Test input fields within the app for injection attacks.
- Test for leakage in logs: Analyze the apps logs to check if sensitive information is being logged. We discovered that the application does log some sensitive information, and this has been reported as a finding.
- Test for improper intent handling: Test if the app properly validates intents to prevent intent spoofing or unauthorized access to app components via malicious intents.
- Test whether the app uses insecure or deprecated cryptographic algorithms (e.g., MD5, SHA1) for any critical application features.
- Test whether the app requests unnecessary permissions (e.g., camera, location) that are not relevant to its functionality.
- Verify if it is possible to crash the app by performing an attack on an activity

6 Future Work

- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is a process that must be continuously evaluated and improved; this penetration test is just a single snapshot. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security.

7 Conclusion

The penetration test focused solely on the implementation of the Letro Android application and did not include an analysis of the Awala component. Our assessment identified three key findings: sensitive information being logged in logcat, which could be accessed by malicious applications or attackers with device log access; the absence of a PIN or biometric lock, leaving sensitive data vulnerable if the device is compromised; and the lack of screenshot protection, allowing sensitive screens to be captured and potentially exposing private information. Addressing the issues mentioned in this report would enhance the application's security posture and better protect user data. We recommend remedying these findings in accordance with Android security best practices.

Appendix 1 Testing team

Abhinav Mishra	Abhinav has an extensive expereince of 13+ years working on web, mobile and infrastructure pentests. He is also known for delivering security trainings on web, mobile and infrastructure hacking.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.