Penetration Test Report

Taler

V 1.0
Amsterdam, April 17th, 2025
Confidential

Document Properties

Client	Taler
Title	Penetration Test Report
Target	Taler Wallet iOS app
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	April 10th, 2025	Abhinav Mishra	Initial draft
0.2	April 10th, 2025	Marcus Bointon	Review
0.3	April 11th, 2025	Abhinav Mishra	Retest Report
1.0	April 17th, 2025	Abhinav Mishra	Final Report

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 February 24, 2025 and March 14, 2025, Radically Open Security B.V. carried out a penetration test for Taler.

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:

- Taler Wallet iOS app

The scoped services are broken down as follows:

- Penetration test of Taler Wallet iOS app.: 4 days
- Reporting: 2 days
- **Total effort: 6 days**

1.3 Project objectives

ROS will perform a penetration test of the Taler Wallet iOS app with Taler in order to assess the security of the application. To do so ROS will access the source code and the app from the iOS app store, and guide Taler 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 February 24, 2025 and March 14, 2025.

1.5 Results In A Nutshell

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

The penetration test of the iOS application revealed a few security findings, none of which were classified as critical or high severity, indicating that the application demonstrates an acceptable baseline of security.

The key issues identified include the absence of local authentication mechanisms TW-001 (page 12), sensitive data being stored in an unencrypted SQLite database TW-003 (page 10), transaction data inadvertently exposed on error

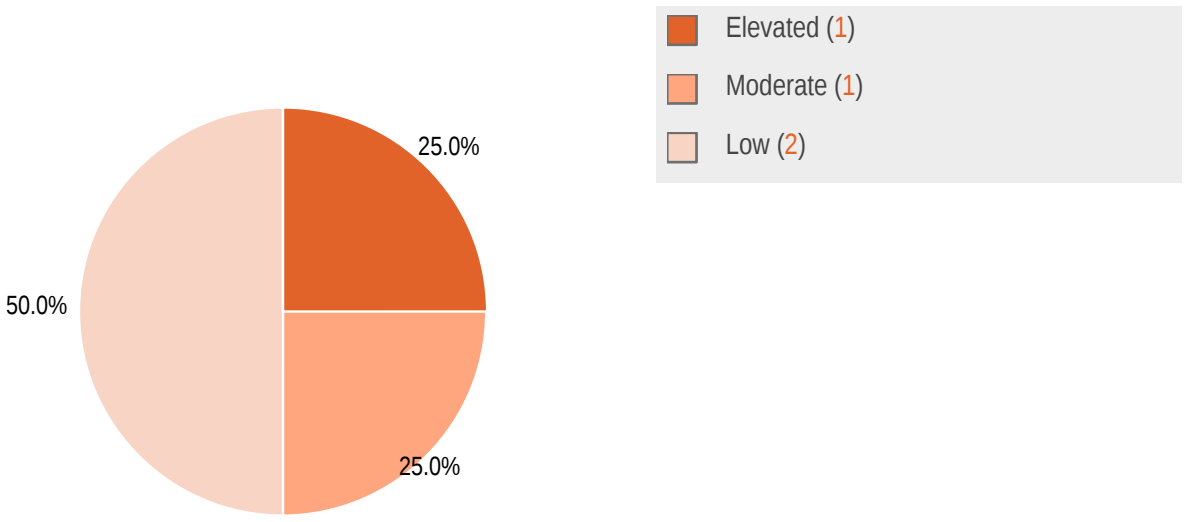
screens for unknown commands TW-004 (page 13), and logging of URI information in both debug and production environments TW-005 (page 15). These findings highlight opportunities for improvement, particularly in securing local storage and enhancing error-handling practices.

It is important to note that the scope of this assessment was limited to the iOS application and did not include an evaluation of the Wallet-core component.

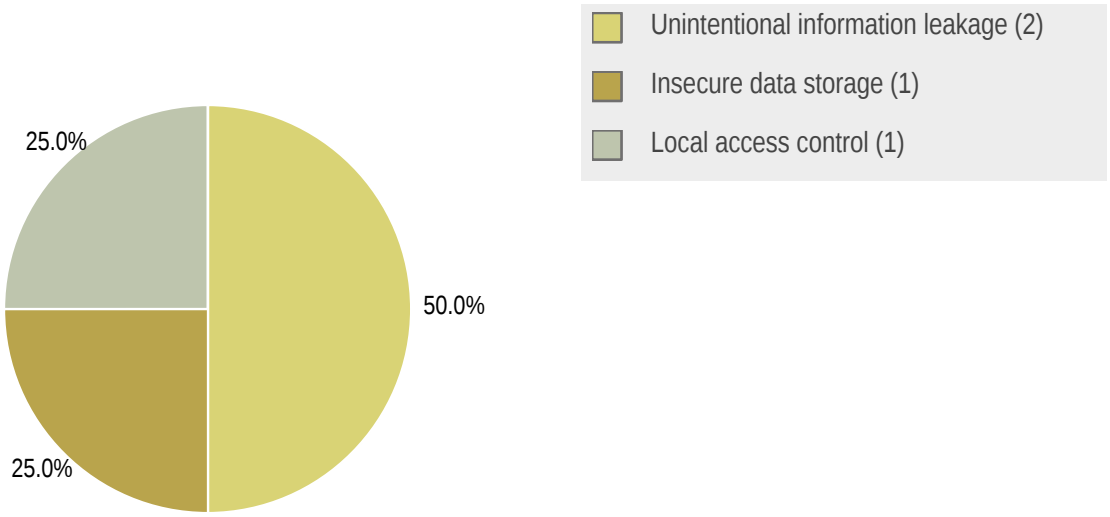
1.6 Summary of Findings

ID	Type	Description	Threat level
TW-003	Insecure Data Storage	The iOS app stores sensitive details like transaction info and all requests, in the Cache.db file created by the use of NSURLSession in default mode.	Elevated
TW-001	Local Access Control	The app lacks biometric or PIN-based access protection, exposing sensitive financial functionality to unauthorized access.	Moderate
TW-004	Unintentional Information Leakage	The application displays sensitive transaction data in plaintext when an unknown command is encountered in the URL scheme.	Low
TW-005	Unintentional Information Leakage	The app logs the entire URL object in both debug and production modes	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
TW-003	Insecure Data Storage	• For requests involving sensitive data, it is recommended to use NSURLSession with the `.ephemeral` configuration to prevent persistent storage of cache files, such as Cache.db, on disk and enhance security. In general, iOS apps should encrypt sensitive data stored in SQLite databases and, where applicable, encrypt specific fields at the application level.
TW-001	Local Access Control	• Implement strong local access control mechanisms, such as biometric authentication.
TW-004	Unintentional Information Leakage	• Do not display error messages containing sensitive information such as DATA or other transaction details; replace sensitive values with generic placeholders or omit them entirely.
TW-005	Unintentional Information Leakage	• Sanitize URLs before logging by removing or masking sensitive components.

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.

- nmap – https://nmap.org
- frida – https://github.com/frida/frida
- objection – https://github.com/sensepost/objection
- SonarQube – https://github.com/SonarSource/sonarqube
- mobXplore – https://github.com/enciphers-team/mobXplore

4 Findings

We have identified the following issues:

4.1 TW-003 — Sensitive Data Stored in Unencrypted SQLite Database

Vulnerability ID: TW-003	**Status:** Resolved
Vulnerability type: Insecure Data Storage	
Threat level: Elevated	

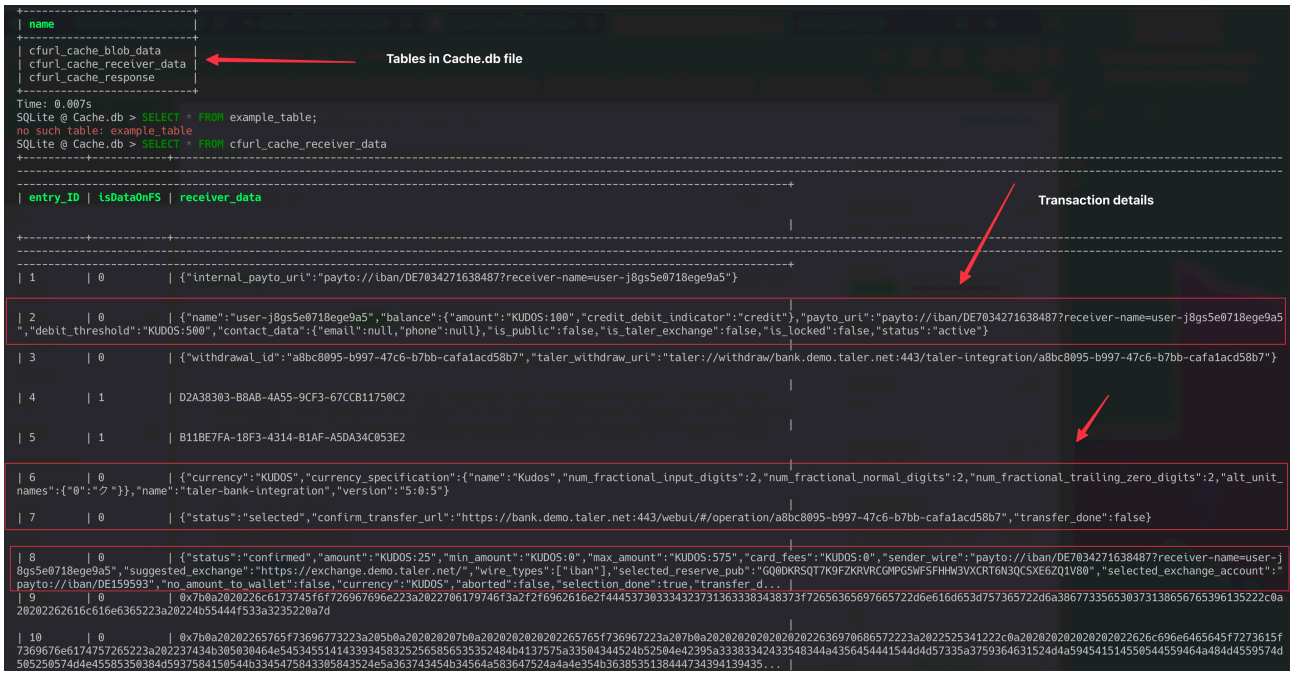
Description:

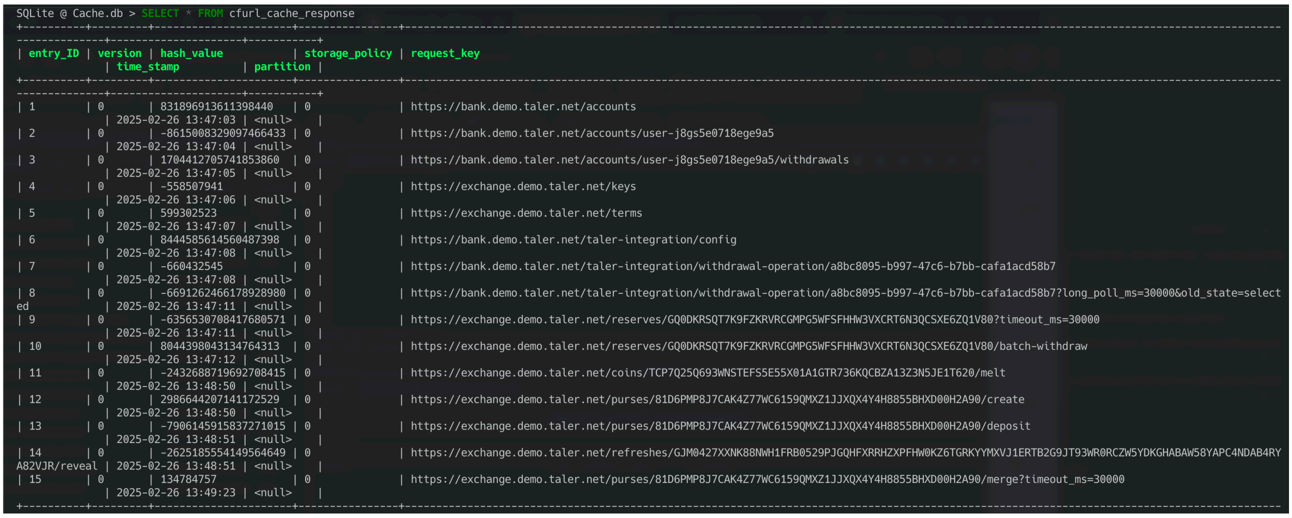
The iOS app stores sensitive details like transaction info and all requests, in the `Cache.db` file created by the use of `NSURLSession` in default mode.

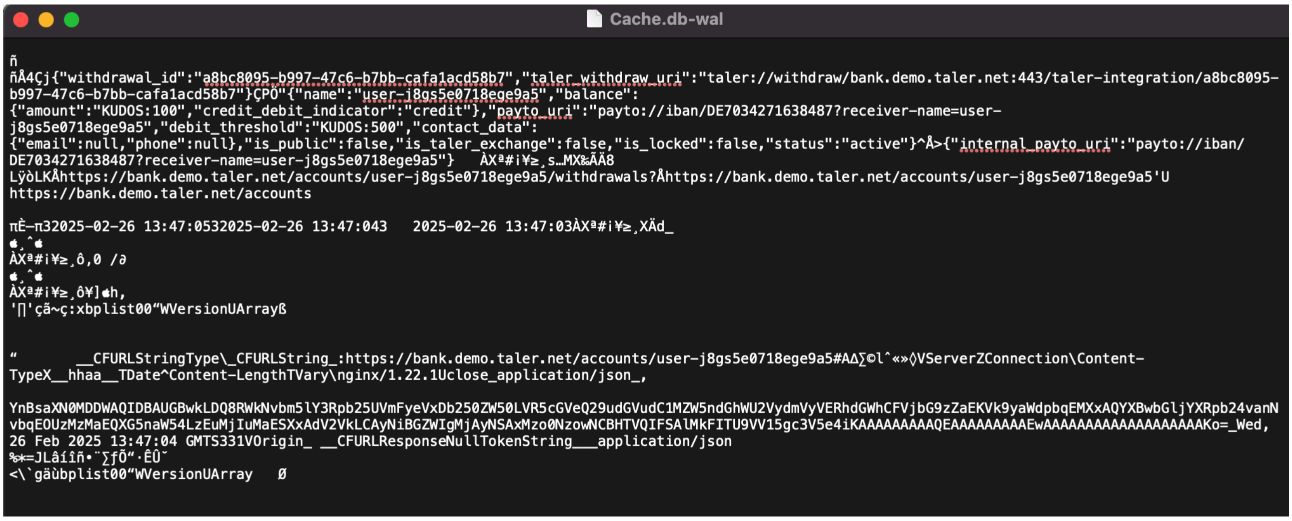
Technical description:

Transaction details, including user information, are stored on the device, in an unencrypted SQLite database file located at `CachesDirectory`: in `/var/mobile/Containers/Data/Application/[identifier]/Library/Caches`

Reading the content of Cache.db file



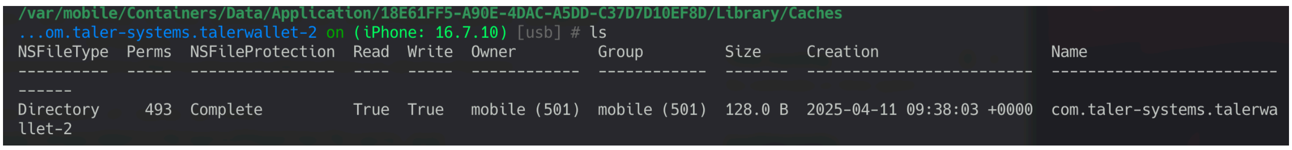




The NSURLSession is used for app-backend communication, which by default, caches network traffic in an unencrypted SQLite database called Cache.db.

Update :

Retest Status: As per the retest done on 11th April, 2025, the iOS app version 0.14.7 does not store the Cache.db file.



```
/var/mobile/Containers/Data/Application/18E61FF5-A90E-4DAC-A5DD-C37D7D10EF8D/Library/Caches/com.taler-systems.talerwallet-2
...om.taler-systems.talerwallet-2 on (iPhone: 16.7.10) [usb] # ls
NSFileType  Perms  NSFileProtection  Read  Write  Owner         Group         Size     Creation                  Name
----------  -----  ----------------  ----  -----  ------------  ------------  -------  ------------------------  -----------------
Directory     493  Complete          True  True   mobile (501)  mobile (501)  192.0 B  2025-04-11 09:38:03 +0000  com.apple.metal
Directory     493  Complete          True  True   mobile (501)  mobile (501)  64.0 B   2025-04-11 09:38:03 +0000  com.apple.metalfe
```

Impact:

Storing sensitive data in an unencrypted format increases the risk of unauthorized access, especially in scenarios involving jailbroken devices, or physical theft of the device. Attackers could extract and misuse sensitive transaction information, violating user privacy and breaking regulatory compliance such as GDPR or PCI DSS.

Recommendation:

- For requests involving sensitive data, it is recommended to use NSURLSession with the `.ephemeral` configuration to prevent persistent storage of cache files, such as Cache.db, on disk and enhance security. In general, iOS apps should encrypt sensitive data stored in SQLite databases and, where applicable, encrypt specific fields at the application level.

4.2 TW-001 — App Lacks Local Authentication Controls

Vulnerability ID: TW-001 **Status:** Not Retested

Vulnerability type: Local Access Control

Threat level: Moderate

Description:

The app lacks biometric or PIN-based access protection, exposing sensitive financial functionality to unauthorized access.

Technical description:

The Taler Wallet iOS app (ver 0.14.5) does not implement any local access controls (PIN, Face ID, Touch ID) upon opening the app or resuming from background.

Impact:

Without any form of local access control, anyone with physical access to the device can open and use the wallet without any restriction. This exposes users to financial theft, especially in the case of device loss, theft, or shared devices.

Recommendation:

Implement strong local access control mechanisms, such as:

- Biometric authentication (Face ID / Touch ID) using `LocalAuthentication.framework`
- A custom PIN if biometrics are unavailable or disabled.

4.3 TW-004 — Transaction Data Exposed On Error Screen for Unknown Commands

Vulnerability ID: TW-004 **Status:** Resolved

Vulnerability type: Unintentional Information Leakage

Threat level: Low

Description:

The application displays sensitive transaction data in plaintext when an unknown command is encountered in the URL scheme.

Technical description:

During the handling of talerURI, the app logs and displays an error message for unknown commands. However, the message also includes the `DATA` value from the URL, which contains sensitive information related to the transaction. For instance, the URL: `taler://pay-push-fail/exchange.demo.taler.net/[DATA]` results in an error message on the app's screen:

This behavior inadvertently exposes sensitive transaction information to unauthorized users who can view the error message. An attacker with access to the screen could extract this information and misuse it.

Update :

`Retest Status:` As of the retest done on 11th April, 2025, the iOS app version 0.14.7 does not reveal the `DATA` value on the error screen.

Impact:

Sensitive transaction data can be intercepted or viewed by unauthorized individuals. Exposed information may be used for unauthorized access, fraud, or replay attacks. Violates data protection principles and may lead to compliance issues with privacy regulations such as GDPR or CCPA.

Recommendation:

- Do not display error messages containing sensitive information such as DATA or other transaction details; replace sensitive values with generic placeholders or omit them entirely.

4.4 TW-005 — Logging URI in Debug and Production

Vulnerability ID: TW-005 **Status:** Resolved

Vulnerability type: Unintentional Information Leakage

Threat level: Low

Description:

The app logs the entire URL object in both debug and production modes

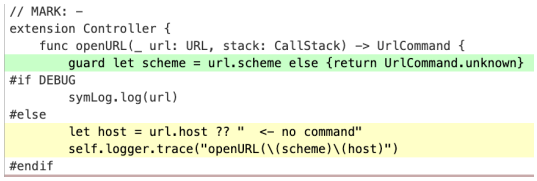
Technical description:

The function logs the entire URL object in both debug and production modes. This logging occurs in the following lines of code: in `Controller.swift`:

```
#if DEBUG
    symLog.log(url)
#else
    self.logger.trace("openURL(\(url))")
#endif
```

The URL may contain sensitive information, including the `DATA` value. For example: `taler://pay-push/exchange.demo.taler.net/[DATA]`. When logged, this data is stored in plaintext, which poses a security risk if the logs are accessed by unauthorized individuals or shared inappropriately.

`Retest Status:` As per the retest done on 11th April, 2025, the iOS app version 0.14.7 only logs the `scheme ("taler://")` and the `host ("command")` of the URL in production mode.



```
// MARK: -
extension Controller {
    func openURL(_ url: URL, stack: CallStack) -> UrlCommand {
        guard let scheme = url.scheme else {return UrlCommand.unknown}
#if DEBUG
        symLog.log(url)
#else
        let host = url.host ?? "  <- no command"
        self.logger.trace("openURL(\(scheme)\(host)")
#endif
```

Impact:

Logs in production environments may inadvertently store sensitive data. This data could be retrieved by attackers if logs are accessible. Regulations like GDPR, CCPA, or HIPAA mandate secure handling of sensitive data. Logging such data in plaintext may lead to non-compliance.

Recommendation:

Sanitize URLs before logging by removing or masking sensitive components. In production environments, limit logs to high-level information (e.g., noting a URL was received) without including sensitive details.

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-002 — Non Finding Test Cases

During the audit, we performed a number of test cases to evaluate the security posture of the iOS app (`com.taler-systems.talerwallet-2`). Although we did not identify any critical or high-severity vulnerabilities, some test cases revealed low or medium-severity issues, while the majority did not uncover any vulnerabilities. Below, we highlight a selection of these non-vulnerable test cases to provide insight into the thoroughness of our assessment.

Local storage analysis

To analyze the application's local data storage, we examined the app's sandbox directory (`/var/mobile/Containers/Data/Application/[App-ID]`) to identify stored files such as SQLite databases, `.plist` files, and snapshots. We accessed the SQLite database files using a database viewer to inspect the stored content and verify whether sensitive information like transaction details, user data, and payment information was stored in plaintext. This resulted in discovery of the finding TW-003 (page 10):

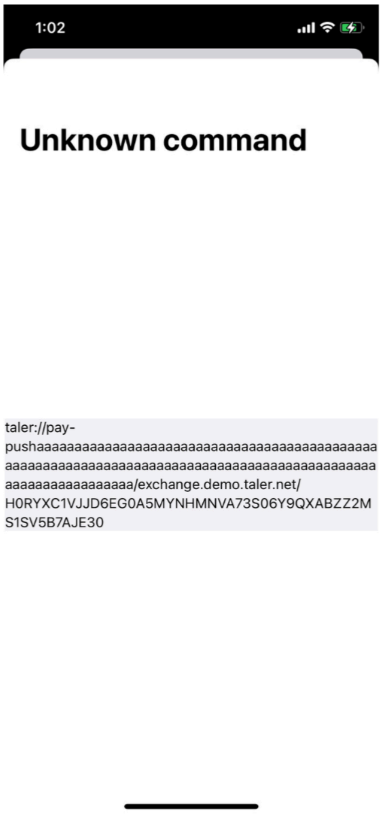
```
inspiring-harvest:/var/mobile/Containers/Data/Application/7AD7019A-FBF4-4925-9909-433E639B17E3/Library root# ls -laR
.:
total 0
drwxr-xr-x 9 mobile mobile 288 Feb 26 06:17 ./
drwxr-xr-x 7 mobile mobile 224 Feb 26 05:46 ../
drwxr-xr-x 3 mobile mobile  96 Apr  9 23:28 Application\ Support/
drwxr-xr-x 3 mobile mobile  96 Feb 26 05:47 Caches/
drwxr-xr-x 3 mobile mobile  96 Feb 26 05:47 HTTPStorages/
drwxr-xr-x 3 mobile mobile  96 Feb 26 06:17 LanguageModeling/
drwxr-xr-x 3 mobile mobile  96 Feb 26 05:48 Preferences/
drwxr-xr-x 3 mobile mobile  96 Feb 26 05:46 Saved\ Application\ State/
drwxr-xr-x 3 mobile mobile  96 Feb 26 05:46 SplashBoard/

./Application Support:
total 1024
drwxr-xr-x 3 mobile mobile      96 Apr  9 23:28 ./
drwxr-xr-x 9 mobile mobile     288 Feb 26 06:17 ../
-rw-r--r-- 1 mobile mobile 1036288 Apr  9 23:28 talerwalletdb-v30.sqlite3

./Caches:
total 0
drwxr-xr-x 3 mobile mobile  96 Feb 26 05:47 ./
drwxr-xr-x 9 mobile mobile 288 Feb 26 06:17 ../
drwxr-xr-x 6 mobile mobile 192 Feb 26 05:47 com.taler-systems.talerwallet-2/

./Caches/com.taler-systems.talerwallet-2:
total 1044
drwxr-xr-x 6 mobile mobile    192 Feb 26 05:47 ./
drwxr-xr-x 3 mobile mobile     96 Feb 26 05:47 ../
-rw-r--r-- 1 mobile mobile  49152 Feb 26 05:47 Cache.db
-rw-r--r-- 1 mobile mobile  32768 Apr  9 22:28 Cache.db-shm
-rw-r--r-- 1 mobile mobile 935272 Apr  9 23:28 Cache.db-wal
drwxr-xr-x 4 mobile mobile    128 Apr  9 10:52 fsCachedData/

./Caches/com.taler-systems.talerwallet-2/fsCachedData:
total 320
drwxr-xr-x 4 mobile mobile    128 Apr  9 10:52 ./
drwxr-xr-x 6 mobile mobile    192 Feb 26 05:47 ../
-rw-r--r-- 1 mobile mobile  33149 Feb 26 05:47 B11BE7FA-18F3-4314-B1AF-A5DA34C053E2
-rw-r--r-- 1 mobile mobile 289606 Apr  9 10:52 D2A38303-B8AB-4A55-9CF3-67CCB11750C2
```

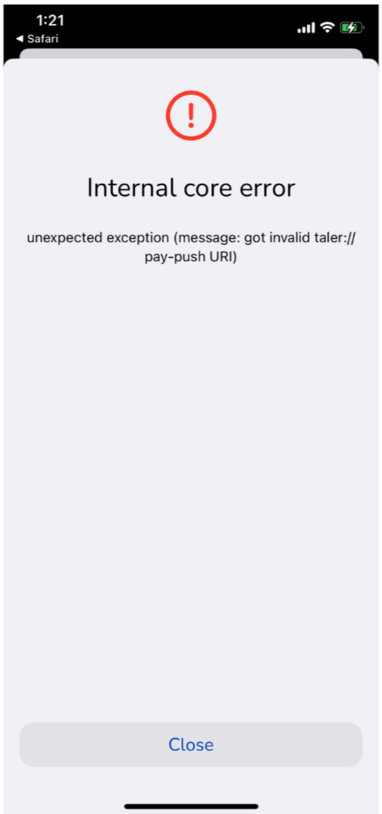
Testing URL Schemes (talerURI)

To test the `talerScheme` function for security issues, we crafted various custom URLs to assess how the app handles valid and invalid commands, ensuring proper mapping and error logging. We tested for URL scheme hijacking by attempting to invoke the custom URL from unauthorized third-party apps. Additionally, we checked for injection

vulnerabilities by passing malicious or unexpected inputs as query parameters or commands in the URL. We also examined how the unencrypted flag is handled, verifying whether unencrypted URLs are logged securely, and that no sensitive data is exposed. Lastly, we checked whether unknown or malformed URLs are handled gracefully without crashing the app or exposing unintended behaviors. Through static analysis we discovered that the applications uses the following URL schemes:

- taler
- taler+http
- ext+taler
- web+taler





This phase of testing revealed that the application handles the `talerURI` properly but the error screen for unknown commands discloses the value of `DATA` from the URI. We reported this in TW-004 (page 13).

Log analysis

While testing the application, we checked whether any sensitive information is written to logs. We did not find any such details in the logs (Using `Console.app` on a production build), however we noticed that the code suggests that it is logged. A finding for this has been reported in TW-005 (page 15).

Hardcoded sensitive info

We reviewed the wallet app's source code to identify any instances of hardcoded sensitive information (e.g. API keys), but did not find any.

Test for unnecessary permissions

We verified whether the app requests permissions that are not relevant to its functionality; it does not.

Other test cases

- We verified that user authentication is implemented securely, ensuring that authentication is enforced locally within the app and is resistant to bypass attempts; this led to one issue in TW-001 (page 12) as no local authentication has been implemented.
- We verified encryption of sensitive data at rest and in transit.
- To ensure the robustness of the iOS app, we performed fuzz testing on the `talerURI` parameters and other input fields. Fuzz testing involved providing the app with unexpected, malformed, or excessively large data inputs to identify vulnerabilities or crash scenarios; this did not lead to any findings.
- We checked whether the app handles unexpected user inputs or actions gracefully, and did not find any problems.

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

We discovered 1 Elevated, 1 Moderate and 2 Low-severity issues during this penetration test.

Our assessment of the iOS application reveals a relatively small attack surface, largely owing to the app's limited functionality. While we did not identify any critical or high-severity issues, the findings highlight areas where security practices can be enhanced, particularly around the handling of sensitive data and local storage mechanisms. Addressing these improvements would further bolster the app's security posture. Overall, the application demonstrates a foundational level of security, and the focused nature of its design contributes to reducing potential risks. It is worth reiterating that this evaluation was confined to the iOS app and did not encompass the Wallet-core component.

Finally, we want to emphasize that security is a process – this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing team

Abhinav Mishra	Abhinav has more than 13 years of extensive experience working on web, mobile, and infrastructure pentests. He is also known for delivering security training 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.