CONVOCATION

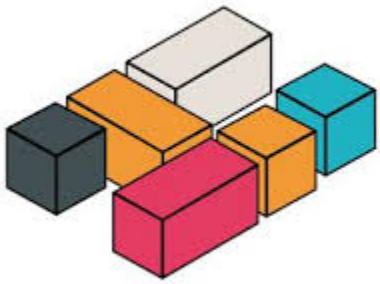
RESEARCH + DESIGN

Horizontal x Convocation Research+Design: A Comprehensive Security Audit for Tella 3.0.0+

July 2026

Prepared for: Horizontal
Prepared by: Convocation Research + Design
Funded by: Open Technology Fund's Security Lab

276 Fifth Avenue, Suite 704-33
New York, NY 10001, US
general@convocation.design
https://convocation.design/



Prepared for: Horizontal
Prepared by: Convocation Research+Design
Funded by: Open Technology Fund's Security Lab
Client Project: Tella 3.0.0+
Client Website: https://tella-app.org/
Client Github Repository: https://github.com/Horizontal-org
Tella Android: https://github.com/Horizontal-org/Tella-Android/
Tella iOS: https://github.com/Horizontal-org/Tella-iOS/
Tella Desktop: https://github.com/Horizontal-org/Tella-Desktop
Contract Reference: IDIQ No. G00524-1092-00 **Task Order:** # T-4
Task Order Effective Date: September 25, 2025 - July 30th, 2026
Remediation Fix Verification Period: May 1st, 2026 - July 6th, 2026
Security Researchers: Sam Smith, Chelsea Manning, Caroline Sinders, Cooper Quintin, and Antonia Valencia

Table of Contents

0.0 Executive Overview

Convocation Research+Design (CoRD) conducted a comprehensive security assessment of Horizontal's Tella application between September 25 and June 30th, 2026 (which includes remediation and fix verification processes), under the Open Technology Fund's Security Lab program. Tella serves as a critical documentation and evidence collection tool for journalists, human rights defenders, and activists operating under surveillance and repression, providing encrypted storage, secure communication channels, and emergency data deletion capabilities across Android, iOS, and desktop platforms.

This audit placed particular emphasis on evaluating Tella's new Nearby Sharing feature, a peer-to-peer protocol enabling offline file transfers between devices without internet connectivity. This capability proves essential for users operating in environments with restricted or monitored internet access, allowing secure documentation sharing through local WiFi networks or personal hotspots using QR code and PIN-based authentication.

The assessment identified multiple critical and high-severity vulnerabilities requiring immediate attention. The most concerning were fundamental flaws in the application's network security implementation. On Android, the application bypassed SSL certificate validation through TrustAllCerts classes in multiple components (FingerprintFetcher.kt, ServerPinger.kt, and TellaPeerToPeerClient.kt), effectively nullifying protection against man-in-the-middle attacks; combined with enabled cleartext traffic permissions in AndroidManifest.xml, this exposed users to interception and surveillance precisely when they needed protection most. The iOS implementation showed similar weaknesses, storing encryption keys in the Documents directory where they were accessible through device backups or forensic tools, while the desktop version used predictable math/rand for PIN generation instead of cryptographically secure random number generation.

Tella's core encryption architecture, built on SQLCipher and platform-specific secure storage (Android Keystore, iOS Secure Enclave), adhered to industry standards, but these network security failures created attack vectors that sophisticated adversaries could exploit. The Nearby Sharing feature showed both promise and peril: although the protocol was intended to use TLS 1.3 with self-signed certificates, the implementation lacked proper certificate validation and pinning across platforms. Additional findings included excessive permission requests on Android (including MANAGE_EXTERNAL_STORAGE and location access), insufficient authentication rate limiting that allowed brute-force attempts, and missing file size validation that could enable denial-of-service attacks. Because Tella's user base operates under extreme threat conditions, where device seizure, network surveillance, and state and non-state actor threats represent daily risks, these vulnerabilities posed immediate dangers to user safety, and their remediation was treated as a priority by both Horizontal and CoRD.

Horizontal reviewed the findings and carried out a coordinated remediation effort across the Android, iOS, and Desktop codebases. CoRD verified the resulting fixes in a first fix-verification round in May 2026, against release candidate builds Tella Android 3.0.0+

(236), Tella iOS 3.0.0+ (143), and Tella Desktop 1.0.0+. That round confirmed the majority of the fixes but left two findings unresolved: the iOS portion of CVE-2025-TELLA3-008, where encryption keys were still written to the iOS Documents directory, and CVE-2025-TELLA3-018, self-signed certificates without validation. Horizontal remediated both, and CoRD conducted a final fix-verification round under the same OTF Security Lab contract in June 2026, against Tella iOS 3.1.0 (149), Tella Android 3.2.0 (244), and internal Desktop builds.

As of that final verification, 21 of the 24 findings documented in Section 4.6 are Fixed and Verified and no findings remain open. The iOS portion of CVE-2025-TELLA3-008 was resolved by migrating encryption-key storage to the iOS Keychain, and CVE-2025-TELLA3-018 was resolved by a new mutual-TLS (mTLS) peer-to-peer protocol implemented across iOS, Android, and Desktop. The three remaining findings are Risk Accepted by Horizontal on the affected platforms: CVE-2025-TELLA3-005 (Fixed on Desktop; Risk Accepted on Android and iOS), CVE-2025-TELLA3-007 (Android), and CVE-2025-TELLA3-013 (Android and iOS). Section 6 documents the remediation work and the fix-verification methodology, and Section 4.5 and the Status field of each finding in Section 4.6 reflect these final results.

1.0 Introduction

1.1 About Tella

Horizontal's Tella is a documentation and data collection tool specifically designed for users operating under surveillance and repression. The application provides encrypted storage, camouflage features, and secure data transmission capabilities to protect sensitive information and user identities. In Horizontal's own words, "In challenging environments, with limited or no internet connectivity or in the face of repression, Tella is an app that makes it easier and safer to document human rights violations and collect data." The latest version of Tella we tested will be published as 3.0.0+ on both Android and iOS; and Tella Desktop will be launched for the first time 1.0.0+ after fixes from this audit.

1.2 Previous Security Assessments

CoRD's security audit of Tella builds upon previous audits from 2023 and 2024:

- Radically Open Security Audit (August 2024): Link
- Subgraph Technologies, Inc. Audits (May 2023/2024): Link

2.0 Audit Objectives

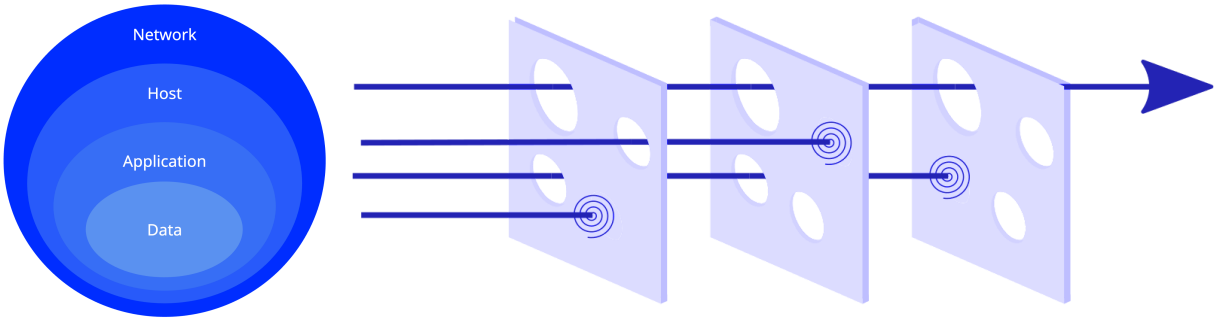
Our objectives with this security audit are to make sure Tella 3.0.0 is the most secure version it can be in order to better serve our users. The userbase of Tella is uniquely at risk from both state and non-state bad actors and with this in mind we are attempting to

provide a comprehensive security audit in order to help keep Tella users safer online and offline. This security audit will assess Tella's core protection mechanisms across three priority tiers, beginning with critical (P0) objectives covering the new Nearby Sharing peer-to-peer file transfer feature: including its TLS implementation, session management, authentication mechanisms, and protection against man-in-the-middle attacks, alongside comprehensive evaluation of data protection architecture (encryption at rest, key management, secure deletion), authentication systems (PIN/password security, brute force protection, data deletion triggers), and application masking features (calculator disguise effectiveness, Quick Delete functionality, forensic resistance, and screen security). High-priority (P1) objectives encompass platform-specific security reviews across Android, iOS, and Desktop codebases, server integration security for Uwazi, ODK, Tella Web, and cloud storage connections, and development of comprehensive threat modeling using STRIDE analysis for activist/journalist use cases alongside user-facing OPSEC documentation including WiFi security best practices and emergency protocols. Secondary (P2) objectives include static code analysis, dependency vulnerability assessment, and dynamic runtime behavior analysis to ensure code quality and security-by-design principles throughout the application.

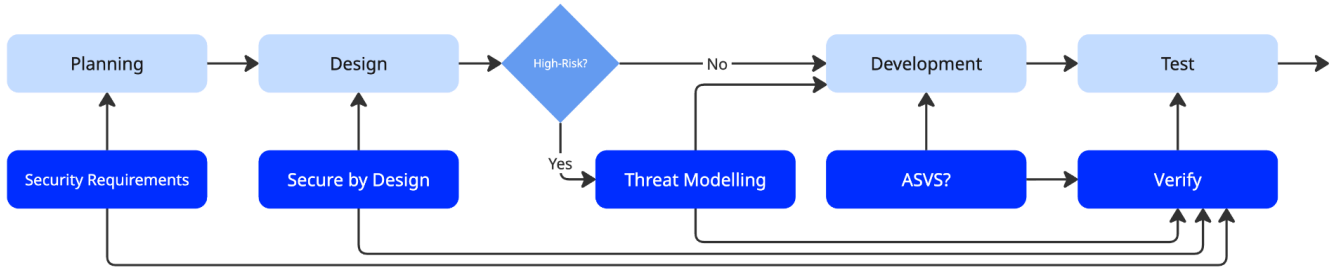
- Comprehensive security audit of the Tella application ecosystem
- Focused evaluation of the new Nearby Sharing (P2P feature)
- Verification of previous audit remediation efforts
- Threat modeling aligned with real-world adversary capabilities
- Development of user-facing security guidance

2.1 Defense-in-Depth and Security-by-Design Approach

This security audit will implement a Defense-in-Depth (DiD) and Security-by-Design (SbD) approach across the following technical domains, prioritized according to criticality. Defense in Depth (DiD) is a concept used in information security in which multiple layers of security controls (defense) are placed throughout an information technology system. Its intent is to provide redundancy in the event a security control fails or a vulnerability is exploited. Think of this like an onion with layers, or like pieces of Swiss cheese stacked together, allowing data to flow through only if it knows the correct route.



Security by Design (SbD) is a concept used in information security and foundational approach in the software development lifecycle that ensures security is engineered into applications and services from the outset, making them resilient to threats and aligned with both regulatory and organizational policies.



2.2 Priority 0 (P0) - Critical Security Objectives

2.2.1 Nearby Sharing Feature Security Assessment

New offline peer-to-peer file sharing capability based on LocalSend-inspired protocol

A.1 Core Protocol Security:

- Review the implementation of the architecture
- HTTP/HTTPS server integration and self-signed certificate handling
- Session management and encryption key derivation
- QR code and PIN-based authentication mechanisms
- Protection against man-in-the-middle attacks on local networks
- TLS v1.3 implementation with ephemeral Diffie-Hellman for perfect forward secrecy

A.2 Integration with Security Architecture:

- File transfer to encrypted vault workflow
- Impact on existing security features during active sharing
- Memory management during file transfers
- Temporary file cleanup and secure deletion
- Compatibility with Quick Delete and camouflage features
- Metadata preservation during P2P transfers

A.3 Network and Connection Security:

- Personal hotspot vs public WiFi security implications
- Error handling that prevents information disclosure
- Protection against malicious peers and rogue access points
- Port configuration and HTTP route security

A.4 Data Transfer Security:

- File metadata transmission security (prepare-upload phase)
- Binary data upload encryption verification
- Session cancellation and resumption mechanisms
- Integration with the encrypted vault post-transfer

2.2.2 Data Protection and Encryption Architecture

A.1 File Encryption Implementation:

- Cryptographic algorithm selection and implementation review
- Key generation and management assessment

- Secure deletion and data wiping verification
- Memory management and data residue analysis
- Verification of encryption at rest implementation

A.2 Authentication Systems:

- PIN/password/pattern security evaluation
- Brute force protection mechanisms
- Lock timeout implementation review
- Failed attempt restrictions and data deletion triggers
- Re-authentication requirements for security settings changes

A.3 Data Storage Security:

- Database encryption implementation
- File system security analysis
- Media file metadata stripping verification
- Temporary file handling and cleanup
- File export/import security and original file deletion verification
- Third-party app sharing data leakage assessment

2.2.3 Nearby Sharing Tests Conducted:

A.1 Integration Security Tests

1. **File Transfer to Vault**
    - Encryption during transfer: Verified
    - Temporary file cleanup: Incomplete
    - Metadata preservation: Secure
    - Memory management: Leaks Found
2. **Quick Delete During Transfer**
    - Interruption handling: Problematic
    - Data remnants: Found
    - Recovery possibility: Yes
    - 5-second countdown respected: Yes
3. **Camouflage Mode Sharing**
    - Calculator interface maintained: No
    - Feature visibility: Exposed
    - Forensic traces: Found
    - App size changes: Detectable

A.2 Network Environment Tests

1. **Public WiFi Security**
    - Attack vectors tested: MITM attempts, rogue AP tests
    - MITM success rate: Prevented by TLS
    - Error handling: No Info Disclosure
    - User warnings: Yes, but insufficient
2. **Personal Hotspot Configuration**
    - Set up security: Manual
    - Password requirements: Weak (OS-level dependent)
    - Isolation from other devices: Effective
    - Network visibility: Exposed
3. **Rogue Access Point Detection**

- Evil twin resistance: Strong
- Certificate warnings: Acceptable
- Connection verification: Implemented
- User education: Adequate

A.3 High-Risk Scenario Tests

1. **Device Seizure Simulation**
   - Quick Delete effectiveness: A few seconds
   - Data recovery attempts: Partially successful (Android)
   - Forensic resistance: Strong (iPhone), Intermediate (Android)
   - Vault encryption strength:
2. **Surveillance Resistance**
   - Network traffic patterns: Visible (local), TLS but not no obfuscation
   - Nearby Sharing discovery visibility: Hidden
   - Transfer fingerprinting: Possible but difficult to implement
   - Metadata leakage: Partial (size, time)
3. **Emergency Response Testing**
   - Quick Delete activation methods: Tested
   - Reliability under stress: 100%
   - False activation prevention: Effective
   - App self-deletion: Adequate

A.4 Data Protection Tests

1. **Encryption Implementation**
   - At rest: AES-256-GCM (32-byte key = 256 bits
   - In transit: TLS 1.2
   - Key generation: Secure crypto/rand.Read()
   - PBKDF2 iteration count: Unused for password hashing
2. **Authentication Security**
   - PIN/Password strength: 6 characters (backend), 8 characters (frontend)
   - Brute force protection: 3 per nonce
   - Lock timeout: None
   - Re-authentication for settings: No re-auth required, access persists until manual lock
3. **Media File Handling**
   - EXIF stripping: None
   - Location data removal: None
   - Third-party app sharing: Files encrypted in TVault
   - Original file deletion: Secure (single pass overwrite) in Desktop; partial/incomplete in iOS and Android

2.2.4 Application Masking and Security Features

A.1 Camouflage Features:

- Calculator disguise security effectiveness
- App name/icon customization review
- Android Settings visibility assessment (known limitation)
- Forensic resistance evaluation
- App size detection risks with large media files

A.2 Emergency Response Features:

- Quick Delete button implementation review
- 5-second countdown and cancellation mechanism
- Data deletion completeness verification
- App self-deletion functionality

A.3 Privacy Features:

- Screen security (screenshot/recording blocking)
- Camera silent mode implementation
- Recent apps preview hiding
- Divvi Up analytics implementation review (privacy-preserving telemetry)

2.3 Priority 1 (P1) - High Security Objectives

2.3.1 Platform-Specific Security Review

A.1 Android Security (Tella Android):

- Java/Kotlin code security audit
- Android permission model review
- Root detection and anti-tampering measures
- Tracker implementation review (CrashLytics/Firebase for Android)
- Verification of previous audit fixes
- PBKDF2 iteration count verification
- Clear-text traffic prevention validation

A.2 iOS Security (Tella iOS - Swift):

- Swift code security assessment
- iOS sandbox escape prevention
- Keychain implementation review
- WebView implementation security

A.3 Desktop Security (Tella Desktop - Go/Java):

- Cross-platform security considerations
- Desktop-specific attack vectors
- File system permission handling
- Memory protection mechanisms

2.3.2 Server Integration and TLS Security

A.1 Platform Integrations:

- Uwazi connection security
- Open Data Kit (ODK) integration assessment
- Tella Web communication protocols
- Cloud storage integrations (Google Drive, Dropbox, Nextcloud)

A.2 Transport Security:

- TLS implementation verification (mandatory encryption in transit)
- Certificate validation and pinning
- API authentication mechanisms
- Offline queue security
- Retry logic and error handling
- Prevention of non-TLS data transmission

2.3.3 Threat Modeling and Operational Security

A.1 Comprehensive Threat Modeling:

- STRIDE analysis for activist/journalist use cases
- Physical security considerations
- Device seizure scenarios
- Nearby Sharing specific threat vectors

2.3.4 Threat Actor Considerations

- **Nation-State Actors**
- **Criminal Organizations**
- **Insider Threats**
- **Opportunistic Attackers**

A.1 Documentation Requirements:

- Plain language for non-technical users
- Step-by-step instructions with visual aids
- Regional considerations for specific threat landscapes
- Quick reference cards for emergencies
- Decision trees for security choices
- Common mistakes and how to avoid them

2.4 Priority 2 (P2) - Important Security Objectives

2.4.1 Code Review for Quality and Security by Design

A.1 Static Code Analysis:

- Automated vulnerability scanning across all codebases
- Manual code review of critical paths
- Dependency vulnerability assessment
- Third-party library and protocol security review

A.2 Dynamic Analysis:

- Runtime behavior analysis
- Memory leak detection
- Performance under adversarial conditions
- Crash resistance testing

3.0 Methodology

Convocation Research+Design (CoRD) conducted a focused security assessment of the Tella ecosystem in an isolated test environment from October 20th, 2025 to June 1st, 2026, with full access to Android, iOS, Desktop, Nearby Sharing (P2P), and associated server components. Testing was performed against production-intent builds sourced from the public GitHub repositories at the specified commits for Android 3.0.0+, iOS 3.0.0+, and Desktop 1.0.0+, using representative hardware and operating systems, including a Framework 13 laptop running NixOS and an iPhone 16 Pro on iOS 26.0.1. All testing was conducted in an isolated laboratory environment to safely reproduce attack scenarios, ensuring that no real user data or production systems were exposed. The audit employed both static and dynamic techniques, guided by established security standards and

frameworks, including the OWASP Testing Guide, the NIST Framework, ISO 27001/27002, and CIS Controls. Static Application Security Testing (SAST) included automated scanning and manual review of critical code paths for Android (Kotlin/Java), iOS (Swift), Desktop (Go/Typescript/React), and backend components. Particular emphasis was on cryptography, key management, authentication flows, and file handling. Dynamic Application Security Testing (DAST) exercised running applications and services under realistic usage patterns, validating encryption in transit, session management, error handling, and secure integration with Uwazi, ODK, and cloud storage providers.

3.0.1 In-Scope Components

Component	Version	Platform	Priority
Tella Android	3.0.0+	Android 5.0+ (v16 Baklava)	P0/1
Tella iOS	3.0.0+	iOS 14.0+ (iOS v26.0.1)	P0/1
Tella Desktop	1.0.0+	Desktop	P0/1
Nearby Sharing (P2P)	Beta	All	P0
Server Connections	All	All	P1

3.0.2 Testing Environment

- **Environment Type:** Isolated Test
- **Access Level:** Full
- **Testing Period:** October 20th, 2025 - June 1st, 2026

3.0.3 Versions Examined

- Testing environment: Android 12-14, iOS 15-17, Windows 10/11, macOS 13-14
- Android Application :feat/stable_peer_to_peer
    - Android Application: Tella Android v3.2.0
    - Source repository: https://github.com/Horizontal-org/Tella-Android/pull/393
    - Source release: commit ce376b9641f273d93e1a8012f5adf661fefc427f
    - Testing environment
        - Framework 13 Laptop (2025)
        - AMD Ryzen AI 9 HX 370
        - AMD Radeon 880M (Integrated)
        - NixOS 25.05 (Warbler) Linux 6.16.4
        - androidenv - Android SDK 2025.1.4.8
        - wireshark - 4.4.7
- iOS Application :feat/p2p
    - iOS Application: Tella iOS v3.1.0
    - Source repository: https://github.com/Horizontal-org/Tella-iOS/pull/264

    - Source release: commit 8313238d5a6b57642923c73ad6ff1f67d6bc39b9
    - Testing environment: iPhone
        - iPhone 16 Pro
        - iOS 26.0.1
        - wireshark - 4.4.7
- Desktop Application :main
    - Desktop Application: Tella Desktop v1.0.0
    - Source repository: https://github.com/Horizontal-org/Tella-Desktop
    - Source release: commit de1cbd560d3c26053cf91f265c289e57392c9559
    - Testing environment:
        - Framework 13 Laptop (2025)
        - AMD Ryzen AI 9 HX 370
        - AMD Radeon 880M (Integrated)
        - NixOS 25.05 (Warbler) Linux 6.16.2
        - wireshark - 4.4.7

3.0.4 Versions Examined: Post-Remediation

Platform	Version	Distribution Method
iOS	Tella iOS 3.1.0	TestFlight
Android	Tella Android 3.2.0	Firebase App Distribution
Desktop	Builds from tella-mtls-2026-audit-target/PR #22	GitHub Actions internal artifacts

3.1.1 Testing Methodology

3.1.2 Standards and Frameworks

- OWASP Testing Guide: https://owasp.org/www-project-web-security-testing-guide/
- NIST Cybersecurity Framework: https://www.nist.gov/cyberframework
- ISO 27001/27002: https://www.iso.org/standard/27001
- CIS Controls: https://www.cisecurity.org/controls

3.1.3 Testing Types Performed

- Static Application Security Testing (SAST)
- Dynamic Application Security Testing (DAST)
- Nearby Sharing Protocol Assessment
- Encryption Implementation Review
- Manual Code Review
- Configuration Review
- Architecture Review
- Platform-Specific Security Testing (Android/iOS/Desktop)
- Integration Security Testing (Uwazi/ODK/Cloud Storage)

- Threat Modeling (STRIDE for activist/journalist case studies)
- 2025 Digital Security Best Practices Confirmation

4.0 Key Security Audit Findings

After extensive code review and testing, CoRD was able to determine that Tella's overall security posture is not yet acceptable for its high-risk user base, despite a generally sound architectural intent and the use of standard cryptographic components. The most serious findings relate to implementation flaws rather than the core design of the new Nearby Sharing feature: on all platforms, the P2P workflow is essentially a straightforward local HTTPS client-server arrangement built on existing TLS protocols, not a novel or exotic mechanism, but it is undermined by weaknesses such as disabled or incomplete certificate validation, predictable PIN generation, lack of robust brute-force protections, and weak session lifecycle management. Across the ecosystem, additional critical issues were identified in key management and storage (including insecure key storage on iOS and logging of database keys on desktop), insufficient secure deletion and memory wiping, world-readable file exports, missing size validation for uploads, and excessive or loosely justified permissions on Android. Taken together, these vulnerabilities create credible opportunities for interception, unauthorized access, and data exposure, meaning that in its current state Tella cannot be considered secure enough for journalists, activists, and human rights defenders facing capable adversaries until these implementation defects in cryptography, network security, and sensitive data handling are fully remediated.

4.1 Android Application

On Android, while encryption and key management are robust, severe issues were found in SSL/TLS validation, cleartext traffic allowances, and debug logging. These weaknesses expose the app to interception and credential theft, undermining an otherwise industry-standard security model. Permissions are found to be excessive, and input validation is insufficient, which raises the risk of misuse and an avoidable attack surface.

Core Architecture	
Main Application	MyApplication.java
Vault System	SQLCipher
Key Management	Android Keystore
Authentication	PIN, pattern, password, biometrics
Network Layer	NextCloud, Uwazi, Google Drive, Dropbox
Peer-to-Peer	Direct device-to-device communication

4.2 Desktop Application

The Tella Desktop application demonstrates a solid foundation for security with proper encryption implementation and secure file handling. However, critical vulnerabilities in debug logging, file upload validation, and PIN generation pose immediate risks that must be addressed before deployment in production. The application would benefit from a comprehensive security review focusing on input validation, resource management, and secure coding practices. With the recommended fixes, this application could provide a secure peer-to-peer file sharing solution.

Core Architecture	
Main Application	React with Typescript
Vault System	Custom Encryption with TVault
Key Management	Encrypted Database Keys, SQLite
Authentication	Argon2 password hash
Network Layer	HTTPS with self-signed certificates
Peer-to-Peer	Direct device-to-device communication

4.3 iOS Application

Tella iOS implements a security architecture with strong cryptographic foundations. The primary concerns center around key storage security and certificate validation. With the recommended fixes, particularly moving to secure keychain storage and fixing certificate pinning, the application would achieve a high security posture suitable for its sensitive use case of protecting human rights documentation.

Core Architecture	
Main Application	SwiftUI
Vault System	SQLCipher
Key Management	Secure Enclave
Authentication	PIN, pattern, password, biometrics
Network Layer	NextCloud, Uwazi, Google Drive, Dropbox

Peer-to-Peer	Direct device-to-device communication

Tella's new Nearby Sharing feature, which is a device-to-device transfer system, enables users to securely share files between nearby devices without relying on the internet or external servers. This capability is built on a unified peer-to-peer (P2P) architecture shared across its desktop, iOS, and Android platforms, with slight variations in implementation per operating system. The process ensures privacy, encryption, and direct communication over local networks.

All devices participating in a transfer must be on the same local Wi-Fi network or hotspot, though internet access is not required. The device that receives files acts as a temporary HTTPS server, while the sending device connects as a client. When a user initiates a "receive" session, Tella creates a local HTTPS server that listens on a specific port (commonly 53317 or 8080) and generates a unique self-signed TLS certificate. Alongside this, a random six-digit PIN is produced for authentication.

Connection details, including the local IP address, port, certificate hash, and PIN, are bundled into a QR code. The sender can connect either by scanning this QR code or by manually entering the details. The use of QR codes simplifies discovery and minimizes human error in manual setup.

Tella enforces multiple layers of security before any data exchange begins. All communications occur over HTTPS/TLS with certificate verification. The sender validates the receiver's certificate fingerprint against the hash encoded in the QR code, ensuring that the connection cannot be intercepted or redirected. Additionally, PIN-based authentication adds an extra layer of protection against unauthorized access. The receiver enforces rate limiting on PIN entry attempts, blocking brute-force attacks after a few failures. Each device generates short-lived certificates for every session, further enhancing security.

Once authenticated, the sender and receiver establish a session with a unique session ID. The sender then compiles file metadata, such as filenames, sizes, MIME types, and SHA-256 hashes, and sends it to the receiver via a prepare-upload request. The receiver presents this information to the user for review, allowing them to accept or reject the transfer. Upon acceptance, the receiver allocates a session workspace and prepares to receive files, assigning identifiers to track progress and ensure file integrity.

Files are uploaded sequentially through secure HTTPS PUT requests. The sender streams each file directly to the receiver's device, and progress is monitored in real time on both ends. Each chunk of data is verified during transmission, and the receiver performs integrity checks against the SHA-256 hash upon completion. The receiver stores the files temporarily until all transfers are complete, after which they are imported into Tella's encrypted vault. All files remain encrypted during transmission and at rest, guaranteeing that sensitive information is never exposed.

Tella's transfer system includes robust handling for connection failures, timeouts, and interrupted sessions. If a transfer fails, users can retry specific files without restarting the entire session. The application automatically cleans up temporary files after completion or failure to prevent data residue. Error messages are explicit, informing users of problems such as incorrect PINs, network interruptions, or certificate mismatches.

While the underlying principles are consistent across desktop, iOS, and Android, each platform adapts its implementation to its environment. The desktop version relies on standard HTTPS servers and file system access, while the iOS version uses Apple's Network framework and P12 certificates for TLS. The Android version employs its own HTTPS service with X.509 certificates and uses platform-specific classes for encryption and vault integration. Despite these differences, all platforms interoperate over the same secure, local P2P protocol.

4.4 Design Limitations That Cannot Be "Fixed"

- Port 53317/8080 detection (required for compatibility)
- VPN incompatibility with Nearby Sharing (local-only by design)
- Server uploads not end-to-end encrypted (architectural decision)
- Camouflage visible in Settings > Apps (Android limitation)
- Camouflage unavailable in iOS Tella, per Apple TOS

4.5 Vulnerability Overview Catalog

Vuln No.	Title	Severity	Remediation
CVE-2025-TELLA3-001	TrustAllCerts Bypasses SSL Validation	**Critical**	Fixed and Verified
CVE-2025-TELLA3-002	Cleartext Traffic Enabled	**High**	Fixed and Verified
CVE-2025-TELLA3-003	Debug Information Leak	**Medium**	Fixed and Verified
CVE-2025-TELLA3-004	Weak Key Generation Error Handling	**Medium**	Fixed and Verified
CVE-2025-TELLA3-005	Insufficient Input Validation	**Medium**	Risk Accepted
CVE-2025-TELLA3-006	Excessive Permission Requests	**Medium**	Fixed and Verified
CVE-2025-TELLA3-007	Insufficient Memory Wiping	**Low**	Risk Accepted

Vuln No.	Title	Severity	Remediation
CVE-2025-TELLA3-008	Insecure Key Storage in Documents Directory	**Critical**	Fixed and Verified
CVE-2025-TELLA3-009	Weak Keychain Implementation	**High**	Fixed and Verified
CVE-2025-TELLA3-010	Insufficient Certificate Pinning	**High**	Fixed and Verified
CVE-2025-TELLA3-011	Decrypted Files Stored in Temporary Directory	**Medium**	Fixed and Verified
CVE-2025-TELLA3-012	Database Key Passed as Plain String	**Medium**	Fixed and Verified
CVE-2025-TELLA3-013	Insufficient Authentication Rate Limiting	**Medium**	Risk Accepted
CVE-2025-TELLA3-014	Insecure File Type and Size Validation	**Low**	Fixed and Verified
CVE-2025-TELLA3-015	Database Keys Logged to Stdout	**Critical**	Fixed and Verified
CVE-2025-TELLA3-016	File Upload Without Size Validation	**High**	Fixed and Verified
CVE-2025-TELLA3-017	Predictable Random PIN Generation	**High**	Fixed and Verified
CVE-2025-TELLA3-018	Self-signed Certificates Without Validation	**High**	Fixed and Verified
CVE-2025-TELLA3-019	World-readable File Permissions	**High**	Fixed and Verified
CVE-2025-TELLA3-020	Potential SQL Injection on Input Validation Failure	**Medium**	Fixed and Verified
CVE-2025-TELLA3-021	Weak Session Management	**Medium**	Fixed and Verified
CVE-2025-TELLA3-022	Verbose Error Logging	**Medium**	Fixed and Verified
CVE-2025-TELLA3-023	Potential XSS in React Frontend	**Low**	Fixed and Verified

Vuln No.	Title	Severity	Remediation
CVE-2025-TELLA3-024	Memory Leak Issue	Medium	Fixed and Verified

4.6 Detailed Vulnerability Findings

CVE-2025-TELLA3-001: TrustAllCerts Bypasses SSL Certificate Validation

Severity: Critical
Category: Server Integration & TLS
CVSS Score: 8.1 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:L)
CWE ID: CWE-295
Status: Fixed and Verified

Description: The Android app uses a TrustAllCerts class that accepts all SSL certificates without validation. This completely disables certificate verification mechanisms.

Impact: Attackers can perform man-in-the-middle attacks, intercept network data, and impersonate servers.

Affected Components:

- FingerPrintFetcher.kt
- ServerPinger.kt
- TellaPeerToPeerClient.kt

Proof of Concept:

Intercept HTTPS traffic with a self-signed proxy certificate. The connection succeeds without errors.

CVE-2025-TELLA3-002: Cleartext Traffic Enabled

Severity: High
Category: Server Integration & TLS
CVSS Score: 8.2 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N)8.1 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:L)
CWE ID: CWE-319
Status: Fixed and Verified

Description: AndroidManifest.xml and related configuration files permit cleartext traffic.

Impact: Sensitive information may be transmitted in plaintext, leading to interception.

Affected Components:

- AndroidManifest.xml
- configure_localhost_media_file_http_server.xml

Proof of Concept:

Network traffic captured in plaintext over HTTP connections.

CVE-2025-TELLA3-003: Debug Information Leak

Severity: Medium
Category: Data Protection & Logging
CVSS Score: 6.5 (AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N)
CWE ID: CWE-532
Status: Fixed and Verified - Android, iOS, Desktop

Description: Debugging and logging features remain enabled in production builds, leaking internal information.

Impact: Could expose sensitive runtime data and SSL/TLS details to local logs.

Affected Components:

- MyApplication.java
- configure_localhost_media_file_http_server.xml

Proof of Concept:

Device logcat shows internal API responses and SSL negotiation traces.

CVE-2025-TELLA3-004: Weak Key Generation Error Handling

Severity: Medium-High
Category: Data Protection & Logging
CVSS Score: 5.9 (AV:L/AC:H/PR:L/UI:N/S:U/C:H/I:L/A:N)
CWE ID: CWE-703
Status: Fixed and Verified

Description: Error handling during AES key generation silently fails, returning null without a fallback.

Impact: Inconsistent encryption or unprotected storage due to null keys.

Affected Components:

- MainKey.java

Proof of Concept:

Simulated exception in key generation returns null key.

CVE-2025-TELLA3-005: Insufficient Input Validation

Severity: Medium
Category: Input Validation
CVSS Score: 6.0 (AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:L)
CWE ID: CWE-20
Status: Mixed - Desktop: Fixed and Verified; Android and iOS: Risk Accepted

Description: Form fields lack validation for URLs and server configurations. This CVE is an information disclosure issue, not injection or code execution. Clarification: Logged data isn't used in SQL queries, logcat doesn't execute SQL. Logs are written by logcat, no parsing.

Impact: Could allow injection or malformed data processing.

Affected Components:

- build/darwin/entitlements.plist
- frontend/src/Components/NearbySharing/Connect.tsx

Proof of Concept:

Invalid URL entries are accepted without error messages. Network permissions, file system permissions, code-signing/security permissions, server configuration data (QR code data structure), database and encryption permissions.

CVE-2025-TELLA3-006: Excessive Permission Requests

Severity: Medium
Category: Permissions & Privacy
CVSS Score: N/A (hardening/privacy exposure)
CWE ID: CWE-250
Status: Fixed and Verified - See Remediation section for residual permission notes

Description: The application requests multiple high-privilege permissions that are not essential.

Impact: Excessive access increases the risk of data misuse.

Affected Components:

- AndroidManifest.xml

Proof of Concept:

Manifest lists sensitive permissions such as MANAGE_ACCOUNTS and ACCESS_FINE_LOCATION.

CVE-2025-TELLA3-007: Insufficient Memory Wiping

Severity: Low
Category: Data Protection
CVSS Score: 3.3 (AV:L/AC:H/PR:H/UI:N/S:U/C:L/I:N/A:N)
CWE ID: CWE-244
Status: Risk Accepted

Description: In the Android app, the memory clearing method uses simple byte zeroing, which may not fully erase sensitive data.

Impact: Memory dumps may expose sensitive residual information.

Affected Components:

- Wiper.java

Proof of Concept:

```
package org.hzontal.tella.keys.util;

import java.util.Arrays;
import androidx.annotation.Nullable;

public class Wiper {
   public static void wipe(@Nullable byte[] bytes) {
      if (bytes == null) {
         return;
      }

      Arrays.fill(bytes, (byte) 0);
   }
}
```

CVE-2025-TELLA3-008: Insecure Key Storage in Documents Directory

Severity: Critical
Category: Data Protection & Encryption
CVSS Score: 9.0 (AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N)
CWE ID: CWE-312

Initial Test Status: Open - Needs Fix
Final Re-test Status: Fixed and Verified
Initial Description: Encryption keys are stored in the iOS Documents directory, which is backed up and accessible.
Final Re-test Description: Moving the encryption keys from Directory to Keychain.

- Codebase: https://github.com/Horizontal-org/Tella-iOS/pull/300.
- Encryption keys were moved from the application directory to the iOS Keychain.

Initial Impact: Keys can be retrieved via device backups or forensic tools.

Affected Components:

- CryptoFileManager.swift
- TellaPeerToServer.kt
- ServerPinger.kt
- TellaPeerToPeerClient.kt

Proof of Concept:
CryptoManager.swift at lines 182-208

```
private func saveKeypair(_ privateKey: SecKey, _ metaPrivateKey: SecKey, _ keyID: String) throws {
    // ... encryption logic ...
    guard let privateEncryptedData = encrypt(privateData, metaPublicKey) else {
        throw RuntimeError("Error: Could not encrypt private key")
    }
    // ...
    try cryptoFileManager.initKeyFolder(keyID)
    guard cryptoFileManager.saveKeyData(privateEncryptedData, .PRIVATE, keyID) else {
        // ❌ Encrypted private key saved to Documents directory
    }
    guard cryptoFileManager.saveKeyData(publicData, .PUBLIC, keyID) else {
        // ❌ Public key saved to Documents directory
    }
}
```

TellaPeerToPeerServer.kt at line 228

```
put(PeerApiRoutes.UPLOAD) {
    // ... validation code ...

    // ❌ Creates temporary file in default temp directory
    // On Android, File.createTempFile() creates files in:
    // - /data/data/<package>/cache/ (app-private, but may be backed up)
    // - Or system temp directory if accessible
    val tmpFile = File.createTempFile(fileId, ".tmp")
    val output = tmpFile.outputStream().buffered()
```

```
    try {
        val input = call.receiveStream().buffered()
        val totalSize = session.files.values.sumOf { it.file.size }
        var bytesRead = 0L
        val buffer = ByteArray(8192)

        while (true) {
            val read = input.read(buffer)
            if (read == -1) break
            output.write(buffer, 0, read)  // ❌ Writes data to temp file
            bytesRead += read
            progressFile.bytesTransferred = bytesRead.toInt()
            // ...
        }

        progressFile.status = P2PFileStatus.FINISHED
        progressFile.path = tmpFile.path  // ❌ Stores path to temp file

        // ...
    } catch (e: Exception) {
        // ...
    } finally {
        output.close()
        // ❌ File may not be immediately deleted if exception occurs
    }
}

// ...

override fun start() {
    val keyStore = KeyStore.getInstance("PKCS12").apply {
        load(null, null)
        setKeyEntry(
            keyStoreConfig.alias,
            keyPair.private,  // ❌ Private key stored in KeyStore
            keyStoreConfig.password,
            arrayOf(certificate)
        )
    }

    // KeyStore is used but never persisted to file
    // However, if KeyStore were to be saved, it would contain private keys
}
```

Remediation Re-Testing:

Analysis	Result	Location
Fresh vault setup writes keys to Keychain, not filesystem	PASS	VaultKeyStore.setup() → vaultKeyPairStore.writeEncrypted VaultKeypair() (lines 93-118)

Analysis	Result	Location
Encrypted private key as Keychain generic password	PASS	CryptoKeychainStore: service org.horizontal.tella.ios.crypto, account priv-key
Meta key in Secure Enclave	PASS	createSecureEnclavePrivateKey / recoverSecureEnclavePrivateKey
Keychain accessibility restricted	PASS	kSecAttrAccessibleWhenUnlockedThisDeviceOnly in KeychainStore.swift
Migration moves file-based keys to Keychain	PASS	VaultKeyMigrator.migrateIfNeeded() reads legacy files, writes Keychain, verifies match
Legacy key files deleted after migration	PASS	deleteVaultKeyFilesIfPresent() → cryptoFileManager.deleteKeysRootFolder()
Migration completion flagged	PASS	VaultUserDefaultsKey.cryptoKeychainMigrationCompleted
Unlock uses Keychain when migrated	PASS	unlock() + recoverEncryptedPrivateKey() / hasCompleteKeychainVaultMaterial()
No new writes to Documents/keys/ on setup	PASS	VaultKeyPairStore only calls keychainStore.saveEncryptedPrivateKey()
Encrypted vault files still on disk (expected)	PASS	CryptoFileManager.encryptedFolderPath = Documents/files/ (blob storage, not keys)

Re-Testing and Fix Verification Test Procedure:
The iOS internal testing build, Tella iOS 3.1.0 (149) installed through TestFlight. Testing covered fresh installation, application launch, vault setup, app restart, device restart, and upgrade-path behavior where applicable. The application container was inspected to determine whether encryption key material remained present in the application directory after the fix. The Keychain-backed storage behavior was also validated by confirming that encrypted app data remained accessible after relaunch, while key material was not written to the filesystem location previously associated with the issue.

Re-Test Results: The fix was verified successfully. The observed behavior is consistent with the intended remediation: encryption keys do not appear to be stored in the application directory and are instead managed by the iOS Keychain.
Final Status: Fixed and Verified. Based on the testing performed, the remediation for Finding 008 prevents iOS encryption keys from being stored in the application directory.

CVE-2025-TELLA3-009: Weak Keychain Implementation

Severity: High
Category: Data Protection & Encryption
CVSS Score: 8.0 (AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N)
CWE ID: CWE-922
Status: Fixed and Verified

Description: The Keychain manager lacks kSecAttrAccessControl, weakening key protection.

Impact: Stored keys can be accessed without user authentication.

Affected Components:

- KeychainManager.swift at lines 58-65

Proof of Concept:

```
@discardableResult
func save(key: String, data: Data) -> OSStatus {
   let query = [kSecClass as String: kSecClassGenericPassword as String,
            kSecAttrAccount as String: key,
            kSecValueData as String: data] as [String : Any]
   SecItemDelete(query as CFDictionary)

   return SecItemAdd(query as CFDictionary, nil)  // Missing kSecAttrAccessControl
}
```

CVE-2025-TELLA3-010: Insufficient Certificate Pinning

Severity: High
Category: Server Integration & TLS
CVSS Score: 8.1 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N)
CWE ID: CWE-295
Status: Fixed and Verified

Description: The app continues TLS connections even if the certificate pin mismatch occurs.

Impact: Allows man-in-the-middle and interception attacks.

Affected Components:

- NearbySharingURLSessionDelegate.swift, at lines 48-91

Proof of Concept:

```
let serverCertificateHash = certificateData.sha256()

guard trustedHash == serverCertificateHash else {
   debugLog("Public key hash mismatch")
   completionHandler(.cancelAuthenticationChallenge, nil)  // May not execute if exception occurs
   return
}
```

CVE-2025-TELLA3-011: Decrypted Files Stored in Temporary Directory

Severity: Medium
Category: Data Protection
CVSS Score: 5.9 (AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:L)
CWE ID: CWE-459
Status: Fixed and Verified

Description: Temporary decrypted files are not securely deleted after use.

Impact: May allow recovery of sensitive files from device storage.

Affected Components:

- backend/utils/filestoreutils/filestoreutils.go
- VaultManager.swift

Proof of Concept:

backend/utils/filestoreutils/filestoreutils.go

- Lines 336-383, ExportSingleFile() creates decrypted files without secure deletion
- Lines 364-380, Files written with world-readable permissions
- Lines 385-417, CreateZipFile() creates ZIP files with decrypted content
- Lines 67-79, Export directory location (~/Downloads/Tella/)
- Lines 66-72, temp_files table exists but is unused
- Lines 458-466, RecordTempFile() exists but is never called

VaultManager.swift

- Lines 66-81, loadFileToURL() creates decrypted temporary files without secure deletion
- Lines 106-130, loadVaultFileToURL() creates decrypted files in temporary directory without secre deletion

- Lines 223-231, saveDataToTempFile() saves decrypted data to temporary files without secure deletion
- Lines 241-248, createTempFileURL() creates files in NSTemporaryDirectory() which may persist
- Lines 293-299, deleteTmpFiles() deletes temporary files without secure overwrite

CVE-2025-TELLA3-012: Database Key Passed as Plain String

Severity: Medium
Category: Data Protection & Encryption
CVSS Score: 6.1 (AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N)
CWE ID: CWE-522
Status: Fixed and Verified

Description: Database encryption keys are handled as plaintext in memory. They are stored as byte[], converted to String, and embedded in SQL queries, making them readable from memory dumps, debugging, and logs.

Impact: Keys may be exposed through memory inspection or logs.

Affected Components:

- CipherOpenHelper.java
- VaultSQLiteOpenHelper.java
- VaultDataSource.java
- BaseVault.java

Proof of Concept:

```
abstract class CipherOpenHelper extends SQLiteOpenHelper {
   final Context context;
   final byte[] password;

   CipherOpenHelper(@NonNull Context context, byte[] password) {
      super(context, DATABASE_NAME, encodeRawKeyToStr(password), ...);
      this.context = context.getApplicationContext();
      this.password = password;
   }

   public static String encodeRawKeyToStr(byte[] raw_key) {
      return new String(encodeRawKey(raw_key));
   }

   @Override
   public SQLiteDatabase getWritableDatabase() {
      // ...
      return SQLiteDatabase.openDatabase(
         context.getDatabasePath(DATABASE_NAME).getPath(),
         encodeRawKeyToStr(password),
         null, SQLiteDatabase.OPEN_READWRITE, null,
```

```
            new SQLiteDatabaseHook() {
                @Override
                public void postKey(SQLiteConnection connection) {
                    connection.executeForString(
                        "PRAGMA key = '" + encodeRawKeyToStr(password) + "';",
                        null, null
                    );
                }
            }
        );
    }
}

public VaultDataSource(Context context, byte[] key) {
    System.loadLibrary("sqlcipher");
    VaultSQLiteOpenHelper dbHelper = VaultSQLiteOpenHelper.getInstance(context, key);
    if (database == null) {
        database = dbHelper.getWritableDatabase();
    }
}

public BaseVault(Context context, LifecycleMainKey mainKeyHolder, Config config)
        throws VaultException, LifecycleMainKey.MainKeyUnavailableException {
    this(mainKeyHolder, config,
         VaultDataSource.getInstance(context,
            mainKeyHolder.get().getKey().getEncoded()));
returns byte[]
}
```

CVE-2025-TELLA3-013: Insufficient Authentication Rate Limiting

Severity: Medium
Category: Authentication & Authorization
CVSS Score: 6.0 (AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N)
CWE ID: CWE-307
Status: Risk Accepted - Android and iOS

Description: Unlock attempts lack exponential backoff or account lockout.

Impact: Brute-force unlocking remains feasible.

Affected Components:

- PinUnlockActivity.kt, at lines 77-98
- PasswordUnlockActivity.kt at lines 100-119
- PatternUnlockActivity.kt at lines 47-60
- ErrorMessageUtil.kt, at lines 43-54

Proof of Concept:

```
override fun onError(throwable: Throwable) {
    onFailureSetPin(getString(R.string.LockPinConfirm_Message_Error_IncorrectPin))
    TellaKeysUI.getCredentialsCallback().onUnSuccessfulUnlock(TAG, throwable)
}

override fun onFailureSetPin(error: String) {
    if (TellaKeysUI.getNumFailedAttempts() == 0L) {
        pinTopText.setTextColor(ContextCompat.getColor(this, R.color.light_red))
        pinTopText.text = error
    } else {
        onWrongPattern()
    }
}

override fun onError(throwable: Throwable) {
    onFailureSetPassword(getString(R.string.LockPasswordSet_Message_Error_IncorrectPassword))
    TellaKeysUI.getCredentialsCallback().onUnSuccessfulUnlock(TAG, throwable)
}

override fun onError(throwable: Throwable) {
    TellaKeysUI.getCredentialsCallback().onUnSuccessfulUnlock(TAG, throwable)
    isPatternCorrect = false
}

private fun updateAndReturnRemainingAttempts(): Long {
    val remainingAttempts = if (TellaKeysUI.getRemainingAttempts() == 0L) {
        TellaKeysUI.getNumFailedAttempts() - 1
    } else {
        TellaKeysUI.getRemainingAttempts() - 1
    }

    TellaKeysUI.setRemainingAttempts(remainingAttempts)
    TellaKeysUI.getCredentialsCallback().saveRemainingAttempts(remainingAttempts)
    return remainingAttempts
}
```

CVE-2025-TELLA3-014: Insecure File Type and Size Validation

Severity: Low
Category: Input Validation
CVSS Score: 3.8 (AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:L)
CWE ID: CWE-20
Status: Fixed and Verified

Description: Files accepted without adequate type or size validation may lead to denial of service.

Impact: The application accepts files without validating declared size against actual upload size, and without enforcing file size limits. An attacker can:

- Upload files of unlimited size, exhausting disk space
- Claim a small file size in metadata but upload a large file, causing memory exhaustion when the entire file is read into memory
- Upload multiple large files simultaneously to exhaust system resources
- Potentially cause the application to crash or become unresponsive

Affected Components:

- backend/core/modules/transfer/models.go, PrepareUploadRequest.Validate() lacks file size validation
- backend/core/modules/transfer/handler.go, HandleUpload() accepts files without size verification
- backend/core/modules/filestore/service.go, StoreFile() reads entire files into memory without size limits
- backend/core/modules/transfer/service.go, HandleUpload() trusts client-provided file metadata

CVE-2025-TELLA3-015: Database Keys Logged to Stdout

Severity: Critical
Category: Data Protection & Logging
CVSS Score: 9.4 (AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N)
CWE ID: CWE-532
Status: Fixed and Verified

Description: Files accepted without adequate type or size validation may lead to denial of service.

Impact: Immediate compromise of all encrypted data

Affected Components:

- backend/core/database/database.go

Proof of Concept:

```
hexKey := hex.EncodeToString(key)
fmt.Printf("🔑 TEMP DEBUG - Hex Key: %s\n", hexKey)
```

CVE-2025-TELLA3-016: File Upload Without Size Validation

Severity: High
Category: Input Validation
CVSS Score: 8.2 (AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H)
CWE ID: CWE-770
Status: Fixed and Verified

Description: The File upload handler lacks enforcement of size limitations.

Impact: Risk of denial of service through disk or memory exhaustion.

Affected Components:

- backend/core/modules/transfer/handler.go

Proof of Concept:

```
if err := h.service.HandleUpload(
    sessionID,
    transmissionID,
    fileID,
    r.Body,  // No size limit enforced
    fileName,
    mimeType,
    h.defaultFolder,
); err != nil {
```

CVE-2025-TELLA3-017: Predictable Random PIN Generation

Severity: High
Category: Authentication & Authorization
CVSS Score: 7.8 (AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N)
CWE ID: CWE-330
Status: Fixed and Verified

Description: Application uses math/rand for security-critical random generation.

Impact: PINs are predictable, reducing authentication integrity.

Affected Components:

- backend/core/modules/server/service.go

Proof of Concept:

```
func generateRandomPIN() string {
    pinNumber := 100000 + rand.Intn(900000)
    return fmt.Sprintf("%06d", pinNumber)
}
```

CVE-2025-TELLA3-018: Self-signed Certificates Without Validation

Severity: High
Category: Server Integration & TLS
CVSS Score: 8.0 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N)
CWE ID: CWE-295

Status: Fixed and Verified (initially Open - Pending Fix; resolved via mTLS and verified in the June 2026 extended re-testing round)

Description: Server uses self-signed certificates without enforcing pinning.

Impact: Man-in-the-middle attacks are possible.

Affected Components:

- backend/utils/tls/certificate.go

Proof of Concept:

The server creates a self-signed certificate at lines 35-52

```
tlsConfig := &tls.Config{
    Certificates: []tls.Certificate{cert},
    MinVersion:   tls.VersionTLS12,
}
```

Remediation References: Protocol change Tella-P2P-Protocol PR #14; Tella iOS PR #300; Tella Android PR #489; Tella Desktop PR #22. The Tella peer-to-peer protocol was updated to implement mutual TLS (mTLS) across supported platforms.

Remediation Re-Testing:

Verification confirmed that the updated Tella P2P protocol implements mTLS and that the implementation interoperates across iOS, Android, and Desktop clients. Code and static analysis:

Fix Verification Analysis	Result	Location
API v2 routes (/api/v2/*)	PASS	iOS NearbySharingEndpoint.swift:11-15; Android PeerApiRoutes.kt:6-10; Desktop handler.go:29-35
Sender ephemeral client identity	PASS	iOS CertificateGenerator.swift:36-50, NearbySharingRepository.swift:24-31; Android PeerMtlsSsl.kt:33-51
Sender presents client cert on HTTPS	PASS	iOS NearbySharingURLSessionDelegate.swift:84-91; Android TellaPeerToPeerClient.kt:353-356
Receiver requires peer client cert	PASS	iOS NWConnection.swift:146-158; Android PeerMtlsSsl.kt:61; Desktop service.go:159-299

Fix Verification Analysis	Result	Location
Receiver cert hash pinned by sender	PASS	iOS NearbySharingURLSessionDelegate.swift: 104-127; Android CertificateUtils.kt:119+
Sender cert hash pinned by receiver	PASS	iOS NearbySharingPeerCertificatePolicy.swift: 52-66; Android PeerClientCertCapturingTrustManager.kt; Desktop service.go:163-207
Bidirectional hash verification UX	PASS	iOS NearbySharingVerificationRole.swift; Android verification fragments; Desktop CertificateVerificationModal.tsx
v1 incompatibility handling	PASS	iOS NearbySharingProtocolVersion.swift:11-12; Android PeerConnectionQrCodec.kt:88-95; Desktop v2-only routes
Ephemeral key cleanup	PASS	iOS CertificateGenerator.removeStoredIdentities(); Desktop new cert per GenerateTLSConfig()
Protocol spec mTLS + v2	PASS	Tella-P2P-Protocol PR #14 README.md

Desktop certificate.go still generates self-signed server certificates (expected). Validation is enforced in service.go via two-phase ClientAuth (RequestClientCert during the initial ping, then RequireAnyClientCert after PinFingerprint()) and VerifyPeerCertificate, which compares the SHA-256 of the peer certificate DER to the pinned sender certificate hash.

Re-Resting and Fix Verification Test Procedure: Testing used internal distribution builds: Tella iOS 3.1.0 (149) via TestFlight, Tella Android 3.2.0 (244) via Firebase App Distribution, and Tella Desktop built from the frozen audit-target branch (PR #22). Verification covered P2P connections between updated clients; interoperability across iOS, Android, and Desktop using the updated protocol; successful mTLS handshakes between valid clients; rejection of peers that do not present valid mTLS credentials; and the absence of regressions in pairing, connection setup, transfer initiation, and transfer completion.

TLS Negative Rests:

Test	Result
Connection without client cert after pinning	Rejected
Connection with the wrong client cert after pinning	Rejected
Connection with the correct pinned client cert	Accepted (/api/v2/ping 200)

Re-Rest Results: Updated clients established P2P connections using the revised protocol, and iOS, Android, and Desktop builds interoperated using the mTLS implementation. Valid clients completed the expected connection flow and test transfers completed successfully. No regressions were observed in the tested P2P user flows (pairing, connection establishment, and transfer).

Final Status: Fixed and Verified. The remediation for CVE-2025-TELLA3-018 implements the updated mTLS-based protocol across iOS, Android, and Desktop, with successful interoperability between updated clients and rejection of invalid peer credentials. No blocking regressions were identified.

CVE-2025-TELLA3-019: World-readable File Permissions

Severity: High
Category: Data Protection & File Handling
CVSS Score: 7.5 (AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N)
CWE ID: CWE-276
Status: Fixed and Verified

Description: Files are created with permissive file system permissions.

Impact: Unauthorized access to exported sensitive files.

Affected Components:

backend/utils/filestoreutils/filestoreutils.go

Proof of Concept:

- at line 364, os.Create(exportPath) creates files with default permissions (typically 0666 on Unix systems, world-readable and writable)
- at line 377, os.Chmod(exportPath, 0644) sets permissions to 0644, which includes world-readable access (the final '4' allows others to read)
- at line 392, os.Create(zipPath) creates ZIP files with default permissions
- At line 412, os.Chmod(zipPath, 0644) sets ZIP files to world-readable

CVE-2025-TELLA3-020: Potential SQL Injection on Input Validation Failure

Severity: Medium
Category: Input Validation
CVSS Score: 6.5 (AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:L)
CWE ID: CWE-89
Status: Fixed and Verified

Description: Dynamic SQL query concatenation used without complete parameterization.

Impact: If input validation fails and allows non-integer or malicious values in folderIDs, the dynamic query construction using fmt.Sprintf and strings.Join could be exploited. An attacker could manipulate the placeholders array or bypass type checks to inject SQL

Affected Components:

backend/core/modules/filestore/service.go, at lines 433-438 constructs the query dynamically.

Proof of Concept:

```
query := fmt.Sprintf(`
    SELECT id FROM files
    WHERE folder_id IN (%s) AND is_deleted = 0
`, strings.Join(placeholders, ","))
```

CVE-2025-TELLA3-021: Weak Session Management

Severity: Medium
Category: Session Management
CVSS Score: 6.0 (AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L)
CWE ID: CWE-613
Status: Fixed and Verified

Description: Sessions are not properly invalidated and rely on fixed timeouts.

Impact: Transfer sessions persist indefinitely in memory after acceptance, with no expiration or explicit invalidation. An attacker who obtains a session ID (via network interception, logs, or other means) can reuse it to upload files to the recipient's device even after the original transfer has ended. This enables unauthorized file uploads, potential malware delivery, and unauthorized access to the recipient's file system. Additionally, sessions accumulate in memory, leading to potential resource exhaustion over time.

Affected Components:

- backend/core/modules/transfer/service.go

Proof of Concept:

Session creation: A legitimate transfer session is created when AcceptTransfer() is called (line 125 in service.go), storing the TransferSession in memory with no expiration timestamp.

Session persistence: The session is stored via s.transfers.Store(sessionID+"_session", transferSession) and remains valid indefinitely. The only timeout mechanism (5 minutes at line 79) applies only to pending transfers awaiting acceptance, not to active sessions.

Exploitation: An attacker who obtains a session ID can reuse it to upload files:

PUT /upload?sessionId=<stolen_session_id>&transmissionId=<id>&fileId=<id>

The session validation at line 175 only checks if the transfer.SessionID == sessionID, but does not verify session age or expiration.

Verification: Sessions persist until the application restarts or an explicit lock is applied (which destroys the service). There is no cleanup mechanism, expiration check, or explicit invalidation method for completed or abandoned sessions.

- backend/core/modules/transfer/service.go lines 125, Session stored without expiration
- backend/core/modules/transfer/service.go lines 35-40, TransferSession struct lacks timestamp/expiration fields
- backend/core/modules/transfer/service.go at 169-216, HandleUpload validates session ID but not session age/expiration

CVE-2025-TELLA3-022: Verbose Error Logging

Severity: Medium
Category: Data Protection & Logging
CVSS Score: 5.0 (AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N)
CWE ID: CWE-209
Status: Fixed and Verified - Android, iOS, Desktop

Description: Detailed error messages could expose sensitive implementation data.

Impact: Detailed error messages returned to clients expose sensitive implementation details that could aid attackers in understanding the system architecture. While this vulnerability does not directly lead to unauthorized access, it significantly reduces the effort required for reconnaissance and can empower attackers to do more sophisticated attacks.

Affected Components:

- Multiple services across backend

Proof of Concept:

Desktop backend/core/modules/filestore/service.go:148

```
err := s.db.QueryRow("SELECT name FROM folders WHERE id = ?", folderID).Scan(&folderName)
if err != nil {
   if err == sql.ErrNoRows {
      return nil, fmt.Errorf("folder not found with ID: %d", folderID)
   }
   return nil, fmt.Errorf("failed to get folder name: %w", err)
}
```

CVE-2025-TELLA3-023: Potential XSS in React Frontend

Severity: Low
Category: Input Validation & Client Security
CVSS Score: 4.3 (AV:N/AC:M/PR:N/UI:R/S:U/C:L/I:L/A:N)
CWE ID: CWE-79
Status: Fixed and Verified

Description: Insufficient sanitization of user inputs in the React frontend. User-controlled data (file names, folder names, and transfer titles) are rendered directly in React components without explicit sanitization. While React's default JSX escaping provides some protection, the lack of explicit input validation and sanitization creates a potential XSS vector, especially if React's escaping mechanisms are bypassed or if the application is modified to use dangerouslySetInnerHTML in the future.Insufficient sanitization of user inputs in the React frontend.

Impact: Attackers could execute arbitrary JavaScript in the context of other users' browsers, potentially leading to session hijacking, data theft, or unauthorized actions performed on behalf of victims.

Affected Components:

- frontend/src/App.tsx
- frontend/src/Components/FileList/FileList.tsx
- frontend/src/Components/FolderList/FolderList.tsx
- frontend/src/Components/FileRequest/FileRequest.tsx
- frontend/src/Components/FileReceiving/FileReceiving.tsx

Proof of Concept:

frontend/src/Components/FileList/FileList.tsx at line 409

```
<FileName>{file.name}</FileName>
```

frontend/src/Components/FileList/FileList.tsx at lines 337 and 349

```
<HeaderTitle>Received &gt; {folderInfo?.name || 'Folder'}</HeaderTitle>
```

frontend/src/Components/FileRequest/FileRequest.tsx at line 118
<TransferTitle>{requestData.title}</TransferTitle>

frontend/src/Components/FileReceiving/FileReceiving.tsx at line 174
<TransferText>{transferTitle}</TransferText>

CVE-2025-TELLA3-024: Memory Leaks

Severity: Medium
Category: Resource Management
CVSS Score: 5.3 (AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L)
CWE ID: CWE-401 (Missing Release of Memory after Effective Lifetime)
Status: Fixed and Verified

Description: Multiple memory leak vulnerabilities exist in Tella's backend services due to unbounded growth of in-memory data structures that are never properly cleaned up. The transfer service maintains sync. Map structures for transfers and transfer sessions that accumulate entries indefinitely when transfers complete, fail, or are abandoned without proper cleanup. Similarly, the registration service maintains a session map that grows unbounded as every successful registration creates a permanent session entry that is never expired or removed.

Impact: These memory leaks can lead to gradual memory exhaustion on the server, potentially resulting in:

- Degraded application performance over time
- Increased server resource consumption
- Potential denial of service (DoS) when memory is exhausted
- Need for periodic service restarts to reclaim memory
- Increased operational costs due to higher memory requirements

Affected Components:

- backend/core/modules/transfer/service.go - Transfer service sync. Maps (transfers and pendingTransfers)
- backend/core/modules/registration/service.go - Session management map (sessions)

Proof of Concept:

backend/core/modules/transfer/service.go

```
func (s *service) AcceptTransfer(sessionID string) error {
  // ... validation code ...

  // Line 117: Transfer entry stored - NEVER DELETED
```

```
    s.transfers.Store(fileInfo.ID, transfer)

    // Line 125: Session entry stored - NEVER DELETED
    s.transfers.Store(sessionID+"_session", transferSession)

    // No cleanup mechanism exists
}

func (s *service) HandleUpload(...) error {
    // ... upload handling ...

    // Line 205: Transfer marked as completed but NOT REMOVED from map
    transfer.Status = "completed"
    s.transfers.Store(fileID, transfer)  // Still in map!

    // No deletion: s.transfers.Delete(fileID) is never called
    // No session cleanup: s.transfers.Delete(sessionID+"_session") is never called
}
```

backend/core/modules/registration/service.go

```
func (s *service) CreateSession(pin string, nonce string) (string, error) {
    // ... validation code ...

    sessionID := uuid.New().String()
    // Line 43: Session stored - NEVER DELETED OR EXPIRED
    s.sessions[sessionID] = &Session{
        ID:        sessionID,
        Nonce:     nonce,
        CreatedAt: time.Now(),
    }

    // No expiration mechanism
    // No cleanup mechanism
    // Sessions accumulate indefinitely
}
```

4.7 Nearby Sharing & Offline Communications

This section covers the latest feature to Tella: Nearby Sharing. Nearby Sharing enables users to share files between devices using Tella securely. This feature is designed to cater to the needs of people living in internet shutdowns, where privacy, security, and reliability are crucial for staying connected. It's also helpful in rural areas, as well as for protests or significant events where mobile networks are saturated or unavailable. Nearby Sharing will only work in physical proximity, for users connected to the same Wi-Fi network (either a local Wi-Fi or a Hotspot).

Nearby Sharing Features:

- Independent of Internet: Transfers work with or without an active internet connection, by establishing a direct connection between devices using local Wi-Fi.
- Encrypted: Files move directly from one Tella vault to another, encrypted and secure.

- Anonymous: Connection happens locally, without a trace of who you shared with, when or where

Our assessment of Nearby Sharing focused on determining whether the feature introduced any novel or unusual security risks beyond those inherent in a standard HTTPS client-server exchange operating on a local network. Our testing confirmed that the design doesn't involve a new cryptographic system or any bespoke transport protocols. Rather, it relies on familiar https mechanics, self-signed certificates, and a PIN-gated registration step. This means the overall security of Nearby Sharing almost entirely depends on correct implementation of certificate validation, session management, and random number generation rather than on experimental or untested technology. In principle this architecture can be secure because it builds on well-understood components, but in practice the correct implementation leaves several critical gaps that weaken the security guarantees that would normally accompany such a design.

Testing revealed that the intended security model for Nearby Sharing is sound. Under that model, the receiving device acts as a short-lived HTTPS server, generating a fresh certificate and a random six-digit PIN, and publishes connection details to the sender through either a QR code or manual entry. Once a successful TLS connection is established, the sender verifies the certificate that was encoded into the QR code and also must provide the correct PIN before any file exchange occurs. If correctly implemented, this sequence protects users against interception on hostile wi-fi networks, shields "in-transit" data, and ensures that only the device holding the correct out-of-band information can begin a transfer.

For the general network and cryptography issues (TrustAllCerts, cleartext traffic, weak random, missing rate limiting, etc.), the evidence is in section 4.1 "Detailed Vulnerability Findings." Each CVE/CCE entry has both the code location and a short description of how it was exercised or verified. For example, CVE-2025-TELLA3-001 (TrustAllCerts) explains that HTTPS traffic was intercepted with a self-signed proxy certificate and "the connection succeeds without errors," which is the dynamic test confirming the static code finding. CVE-2025-TELLA3-017 shows the exact math/rand-based PIN generator used by Desktop, demonstrating predictability. CVE-2025-TELLA3-013 and CVE-2025-TELLA3-021 include the unlock and session-handling code paths to show the absence of rate limiting and proper session expiry; CVE-2025-TELLA3-018 and CVE-2025-TELLA3-010 similarly show where certificate pinning and validation fail.

Nearby Sharing's protocol behaviour and its weaknesses are documented below:

4.7.1 Nearby Sharing Implementation

Nearby Sharing Implementation Assessment

- **Implementation Status:** Tella-P2P-Protocol:main
- **HTTP/HTTPS Server Configuration:** Only HTTPS
- **Self-signed Certificate Handling:** Secure (RSA 2048 self-signed; pinned by cert SHA-256)

- **Session Management:** Session created on register with sessionId; states: WAITING → SENDING → SAVING → FINISHED/FINISHED_WITH_ERRORS/CLOSED; per-file tracking via transmissionId; single active session enforced.
- **Key Derivation:** N/A (TLS handles key exchange; no custom app-level KDF)

Authentication Mechanisms

- **QR Code Implementation:**
  - Encoding format: JSON text in a QR (ZXing)
  - Data included: ip_address, port, certificate_hash (SHA-256 of cert DER), pin
  - Expiration handling: No
  - Replay protection: No explicit token/expiry; mitigated by single active session and PIN
- **PIN-based Authentication:**
  - PIN generation: Random integer (100000-999999)
  - Length and complexity: 6-digit, numeric
  - Brute force protection: Not enforced (no lockout/backoff observed)
  - Time-based validity: Not specified (valid while session/server active)

TLS Implementation

- **Version:** TLS v1.3 Not Confirmed (client allows TLS 1.3 and 1.2; server uses platform defaults)
- **Cipher Suites:** Not explicitly set; uses platform MODERN_TLS defaults (includes ECDHE + AES-GCM)
- **Perfect Forward Secrecy:** Implemented (via ECDHE suites, platform default)
- **Ephemeral Diffie-Hellman:** Implicit ECDHE; no explicit server-side tuning
- **Certificate Validation:** Client pins SHA-256 of peer's certificate (DER) to the hash from QR; pre-read of cert via trust-all handshake strictly for fingerprinting, not for data transfer

Network Security

- **Personal Hotspot Configuration:** Depends on hotspot; app security independent (TLS + pinning)
- **Public WiFi Risk Mitigation:** Adequate (HTTPS, cert pinning, PIN; no HTTP fallback)
- **Port Configuration:** Static high port 53317; HTTPS only; acceptable
- **HTTP Route Security:** Protected (PIN-gated register; upload gated by sessionId + transmissionId + file ID checks; progress limited to session)
- **Error Handling:** Minimal disclosure (structured 4xx/5xx messages; no stack traces)
- **Bluetooth Version:** None (not used)
- **Pairing Method:** Out-of-band QR code (camera) or manual entry
- **Authentication Mechanism:** TLS certificate pinning + 6-digit PIN
- **Encryption Standard:** TLS (RSA 2048 self-signed cert; symmetric suites via platform defaults, typically ECDHE + AES-GCM)
- **Vulnerabilities Identified:**
  - Lack of brute-force protection for the 6-digit PIN used in session authentication.
  - No explicit expiration or replay protection for QR/PIN-based sessions.
  - Missing integrity verification due to the SHA-256 file hash check being commented out in P2PFile.kt.

    - Information leak via transfer progress reporting, which exposes filenames, file counts, sizes, and percentage completion.

WiFi Direct/P2P Assessment

- **WiFi P2P Group Formation:** Not Applicable (uses regular WiFi infrastructure, not WiFi Direct)
- **WPS Implementation:** Disabled (not used; standard WiFi)
- **Group Owner Negotiation:** Not Applicable (no WiFi Direct groups)
- **Passphrase Strength:** Not Applicable (WiFi network security handled by OS)
- **Vulnerabilities Identified:**
    - Man-in-the-middle attacks (mitigated via cert pinning)
    - Evil twin access points (requires user verification)
    - Unauthorized network joining (requires the same WiFi + PIN)
    - Data interception (mitigated via HTTPS/TLS)

4.7.2 Nearby Sharing Transfer Workflow

Prepare-Upload Phase

- **File Metadata Transmission:** Secure (HTTPS/TLS)
- **Metadata Included:** id, fileName, size, fileType, thumbnail (optional)
- **Privacy Preservation:** Partial (metadata sent in plain JSON; HTTPS protected)
- **Size Limitations:** Undocumented (no explicit max; limited by available storage/network)

Binary Data Upload

- **Encryption During Transfer:** Verified (TLS 1.2/1.3, files remain encrypted via EncryptedFileProvider)
- **Chunking Implementation:** 8KB buffer (8192 bytes), streamed
- **Integrity Verification:** None (SHA-256 hash commented out in P2PFile.kt)
- **Progress Tracking:** Information Leak (exposes file count, names, sizes, percentages)

Session Management

- **Session Initiation:** /api/v1/register endpoint with PIN validation, returns unique sessionId
- **Session Cancellation:** Clean (/api/v1/close endpoint, session state cleared, temp files handled)
- **Session Resumption:** Not Supported (no session persistence/restore)
- **Timeout Handling:** Appropriate (connect: 5s, read/write: 20s; server stops on completion)

4.7.3 Integration with Tella Security Features

Integration with Encrypted Vault

- **File Transfer to Vault:** Staged (temp files stored first, then moved to vault)
- **Encryption Before Storage:** Yes (via RxVault with AES encryption)
- **Temporary Storage Location:** System temp directory (File.createTempFile(fileId,".tmp"))
- **Cleanup Mechanism:** Automatic (f.delete() after successful vault save)

Quick Delete Compatibility

- **Transfer Interruption:** Not Handled (panic mode doesn't check/cancel active transfers)
- **Data Remnants After Delete:** Found (temp files may remain if transfer interrupted during panic)
- **Memory Cleanup:** Complete (ViewModel clears state on onCleared())
- **5-Second Countdown Impact:** None (countdown runs independently; no transfer cancellation)

Camouflage Feature Impact

- **Calculator Mode During Sharing:** Broken (P2P uses standard activities/fragments visible during transfer)
- **UI Visibility:** Exposed (transfer screens visible; camouflage doesn't hide active transfers)
- **Background Transfer:** Visible (progress indicators/notifications visible)
- **App Size Changes:** Exposed (temporary files during transfer may affect app size)

Metadata Handling

- **Preservation During Transfer:** Complete (file content transferred as-is; metadata preserved separately)
- **EXIF Data:** Preserved/Stripped (depends on Preferences.isKeepExif(); images saved via MediaFileHandler.saveJpegPhoto may strip EXIF if` disabled)
- **File Timestamps:** Modified (created timestamp set to current time on save to vault)
- **Location Data:** Retained (stored in VaultFile.metadata.myLocation if captured; not transferred in file content, handled separately)

4.7.4 Offline Data Security

Local Storage During Transfer

- **Temporary File Handling:** Insecure
  - Files are read entirely into memory during transfer (io.ReadAll in StoreFile function)
  - No temporary files written to disk during transfer process
  - temp_files table exists in database schema but not actively used during transfers
  - Temp directory exists for exports but not for transfer operations
- **Memory Management:** Vulnerable
  - Entire file contents loaded into memory before encryption (io.ReadAll(reader))
  - No explicit memory clearing of sensitive data after use
  - Relies on Go garbage collector, which may leave sensitive data in memory
  - Decrypted file data remains in memory during encryption process
  - Memory dumps could contain unencrypted file data
- **Cache Clearing:** Manual
  - temp_files table tracks temporary files but no automatic cleanup mechanism
  - No cache expiration or automatic deletion visible
  - Temp directory used for exports but not systematically cleared
- **Issues Found:**

    - Unencrypted temporary files: Not applicable (files not written to disk)
    - Memory dumps containing sensitive data: Yes - entire files loaded into memory unencrypted
    - Incomplete data deletion: Secure overwrite implemented for deleted files (SecurelyOverwriteFileData), but memory not explicitly cleared

Transfer Protocol Security

- **Handshake Process:** TLS 1.2+ with self-signed certificates
    - Self-signed certificate generated per server starts with a 2048-bit RSA key
    - Certificate hash (SHA256) generated and displayed for verification
    - PIN-based authentication (6-digit random PIN, 100000-999999 range)
    - Registration uses PIN + nonce for session creation
- **Session Management:** Secure with limitations
    - Sessions created with UUIDs and stored in memory (sync.Map)
    - Session validation occurs during transfers (transfer.SessionID != sessionID check)
    - Session timeout: 5 minutes for pending transfers, 30 seconds for registration
    - No explicit session expiration or cleanup for completed transfers
    - Sessions persist in memory until server restart
- **Transfer Integrity Checks:** Partial
    - FileInfo struct includes SHA256 field in models
    - No SHA256 verification implemented in HandleUpload function
    - TLS provides transport-level integrity
    - No application-level checksum verification visible
- **Findings:**
    - Session hijacking possibilities: Limited, sessions validated per request, but no token rotation
    - Data corruption detection: TLS provides transport integrity, but no application-level verification
    - Incomplete transfer handling: Transfers marked as "failed" or "completed" but no retry mechanism
    - Retry attack vulnerabilities: Rate limiting only on PIN attempts (3 attempts per nonce), not on transfer operations

4.7.5 Device Discovery & Connection

Connection Establishment

- **Pairing Process:**
    - 6-digit random PIN generated per server start
    - QR code mode available for auto-confirmation (bypasses manual verification)
    - Manual mode requires certificate hash verification
- **Authentication Flow:**
    - Ping endpoint (/ping), device discovery
    - Register endpoint (/register), sends PIN + nonce
    - Session creation, validates PIN, creates session with UUID
    - Transfer preparation uses the session ID for file transfer authorization

- **Rogue Device Protection:** Partial
    - Certificate hash verification is available in manual mode
    - QR mode auto-confirms without user verification
    - No device blacklist/whitelist mechanism
    - No certificate pinning or device fingerprinting
    - Self-signed certificates prevent MITM if the hash is verified, but QR mode bypasses this. Solved by using a human readable representation of the has that both parties read out loud could solve this
- **Connection Persistence:** Session
    - Sessions are per-transfer, not permanent
    - New PIN is generated each time the server starts
    - Sessions stored in memory only (not persisted)
    - No persistent device pairing

Security Controls

- **Malicious Peer Detection:** Missing
    - No implementation of malicious peer detection
    - No device reputation or blocking mechanism
    - Rate limiting only on PIN attempts, not on connection attempts
    - No anomaly detection for suspicious behavior
- **Rate Limiting:** Yes (Limited)
    - PIN attempt rate limiting, 3 invalid attempts per nonce
    - Rate limiter stored in memory map (not persistent)
    - No rate limiting on transfer operations
    - No rate limiting on ping/registration endpoints
    - No IP-based rate limiting
- **Connection Logs:** Not Stored
    - Logging via runtime.LogInfo and runtime.EventsEmit (in-memory events)
    - No persistent connection logs stored to the database or files
    - No audit trail of device connections or transfers
    - Events emitted to the frontend but not persisted
- **User Consent:** Required
    - User consent required in manual mode (certificate verification modal)
    - Automatic consent in QR mode (auto-confirms registration)
    - Transfer acceptance requires user action (AcceptTransfer/RejectTransfer)
    - 30-second timeout for registration confirmation
    - 5-minute timeout for transfer acceptance

5.0 Recommendations

Overall, our security audit found that Tella's high-level security architecture was conceptually strong, but a set of implementation weaknesses prevented it from delivering the level of protection required for journalists, activists, and human rights defenders operating against capable adversaries. The recommendations in this section were made on the basis of those findings and were intended to realign the implementation with the threat

model and to embed security-by-design practices into the development and release process, without prescribing implementation details or specific patch sequences. Horizontal has since implemented the majority of them; Section 6 documents the resulting fixes and their verification, and the recommendations are retained here as the guidance that shaped that work and that should continue to inform Tella's development.

The most urgent priority is the encrypted communications and key material. On Android, iOS, and Desktop, multiple issues weaken transport security, including permissive certificate handling, insufficient pinning, and the use of predictable random values. Vulnerabilities such as CVE-2025-TELLA3-001 (TrustAllCerts), CCE-2025-TELLA3-002 (cleartext traffic), CVE-2025-TELLA3-010 and CVE-2025-TELLA3-018 (incomplete or missing certificate pinning), and CVE-2025-TELLA3-017 (predictable PIN generation using math/rand) collectively undermine the confidentiality and authenticity guarantees that TLS is meant to provide.

Remediation should therefore focus on eliminating trust-all patterns, enforcing strict certificate validation and pinning, and ensuring that all security-critical randomness is generated using cryptographically secure sources on each platform.

The next priority is to correct key management and sensitive data handling across the ecosystem. For iOS, CVE-2025-TELLA3-008 and CVE-2025-TELLA3-009 indicate that encryption keys and key material are stored in locations and with access controls that do not align with Tella's risk profile. On Desktop, CVE-2025-TELLA3-015 demonstrates that database keys have been logged directly to stdout for debugging. On Android, CVE-2025-TELLA3-012 shows database keys being handled as plain strings and embedded in SQL statements.

These patterns must be replaced with consistent use of hardened, platform-appropriate key storage and careful in-memory handling that avoids logging, stringification, or unnecessary copying of key material. Temporary decrypted files and export artifacts identified in CVE-2025-TELLA3-011, along with the related file-handling analysis, should also be brought under stricter control, with a clear strategy to minimize the creation of decrypted material, limit its lifetime, and erase it in a way consistent with the assumptions in Tella's threat models.

Authentication and session management mechanisms need to be aligned with the capabilities of likely adversaries. CVE-2025-TELLA3-013 and CVE-2025-TELLA3-021 highlight that PIN, password, and pattern unlock flows, as well as file transfer sessions, are not adequately protected against brute-force attacks, replay attacks, or long-term reuse.

Tella should enforce a consistent set of policies for authentication rate limiting, backoff, and lockout across mobile and desktop clients, and it should treat transfer sessions in Nearby Sharing as short-lived, single-use credentials that expire and are actively invalidated when no longer needed. This will help prevent attacks where an adversary with partial device or

network access can systematically guess, reuse, or hijack credentials and session identifiers.

Tella should also reduce its attack surface and limit the information available to an adversary performing reconnaissance. For Android, CCE-2025-TELLA3-006 shows that the current permission set is broader than strictly necessary. Across platforms, CCE-2025-TELLA3-003 and CVE-2025-TELLA3-022 demonstrate that verbose logging and detailed error messages reveal internal implementation details, keys, or configuration states that attackers can leverage to refine their approach.

Tella should move toward a principle of minimal privilege and minimal disclosure: request only the permissions needed to fulfill documented functionality, degrade gracefully when permissions are denied, and ensure that operational logs and user-visible error messages contain only the information necessary for support and diagnostics, not internal structure or secret values.

Input validation and server-side robustness must be improved to mitigate denial-of-service and injection-style attacks. CVE-2025-TELLA3-005, CVE-2025-TELLA3-014, CVE-2025-TELLA3-016, and CVE-2025-TELLA3-020 all highlight insufficient validation and trust in client-supplied input for URLs, file sizes, and folder identifiers, as well as dynamic SQL query construction.

Tella should treat all user and network input as untrusted, enforce clear and documented bounds on file sizes and counts, validate file types, and rely on parameterized queries for database access. These measures will prevent misuse of the application as a vector for resource exhaustion or data corruption. They will limit the impact of malformed or malicious inputs in both standard and adversarial conditions.

Weaknesses in the Nearby Sharing implementation need to be addressed in a coordinated way, because this feature is designed for precisely the high-risk, low-connectivity environments where users may be subject to physical and network surveillance. The current implementation relies on a sound conceptual model.

However, it is undermined by missing brute-force protections, incomplete file integrity checks, long-lived sessions, and a lack of explicit replay and expiration controls. Addressing these gaps requires aligning the implementation with the intended model: each sharing session should be guarded by a cryptographically strong PIN, constrained to a short validity window, protected against repeated guessing, and backed by both TLS-level and application-level integrity checks on transferred content, with all temporary state and storage cleaned up promptly.

Tella would benefit from institutionalizing security-by-design practices. This includes embedding threat modeling into the design of new features like Nearby Sharing, requiring security review of all code that touches cryptography, key material, or transport security,

and maintaining automated tests that assert the presence of rate limiting, certificate pinning, secure deletion, and other critical behaviors.

Horizontal could also continue investing in user-facing documentation and training materials, such as OPSEC guides and WiFi security best practices, to clearly communicate what Tella can and cannot protect against, and how users should configure and use the app in high-risk environments.

5.1 Remediation Actions for iOS

Tella's iOS client should focus primarily on key storage, certificate validation, and temporary file handling. The current practice of storing keys in the Documents directory, as identified in CVE-2025-TELLA3-008, exposes encryption keys to backup and forensic tools in ways incompatible with the threat model for at-risk users. Horizontal should treat this as a first-order risk and adopt the iOS Keychain and Secure Enclave as the standard locations for all long-lived cryptographic material, ensuring that keys are always protected by hardware-backed storage and constrained with appropriate access control flags.

The weaknesses in Keychain usage documented in CVE-2025-TELLA3-009 further indicate that the Keychain is not yet being used to its full protective potential, and should be revisited to ensure that key access is tied to user authentication policies consistent with Tella's security expectations.

Network security on iOS must be brought to parity with the intended Nearby Sharing model and the broader TLS security objectives. The certificate-handling flow in NearbySharingURLSessionDelegate, identified under CVE-2025-TELLA3-010, demonstrates that the code path can continue operating even when certificate hashes do not match or when error-handling paths do not reliably terminate connections.

This behavior should be replaced with strict pinning semantics, where any mismatch or validation failure results in a hard, observable failure. The goal is to ensure that an attacker cannot downgrade or bypass certificate checks by exploiting edge cases in delegate callbacks or error handling.

File and temporary data management also deserve special attention on iOS. As identified in CVE-2025-TELLA3-011 and the related VaultManager analysis, decrypted content is currently written to temporary directories and not always securely deleted after use. This creates a latent risk of data recovery by forensic tools or through device compromise after the fact.

iOS remediation should include a review of all file export, preview, and temporary storage paths, to minimize the amount of decrypted material, restricting its location to app-private spaces, and ensuring deterministic cleanup when files are no longer needed. Where the platform does not technically guarantee complete secure deletion, Horizontal should at

minimum ensure that Tella's documentation clearly describes residual risk to users whose threat models include sophisticated forensic analysis.

Authentication flows within the iOS client should be revisited to ensure that rate limiting, lockouts, and re-authentication for sensitive actions are implemented in line with the recommendations in CVE-2025-TELLA3-013 and the broader authentication analysis.

Even though the examples of insufficient rate limiting are drawn from Android, the underlying requirement is platform-agnostic: on iOS, Tella should limit the number of PIN and password attempts, enforce backoff, and require re-authentication before security-critical actions such as changing vault credentials, exporting data, or enabling Nearby Sharing.

5.2 Remediation Actions for Android

On Android, the most critical remediation work involves rebuilding trust in TLS connections and improving key management. The existence of TrustAllCerts classes (CVE-2025-TELLA3-001) effectively eliminates any protection against MITM attacks. This pattern should be removed, and all network clients must be configured to validate server certificates against a well-defined trust store, with pinning applied where appropriate for Tella's integrations and peer-to-peer operations. In parallel, CCE-2025-TELLA3-002 shows that cleartext traffic is permitted. Android fixes could ensure that there is no path by which sensitive application traffic can fall back to HTTP, even in error or debug modes.

Key management and in-memory handling of secrets also require correction. CVE-2025-TELLA3-012 demonstrates that database keys are converted into strings and embedded into SQL statements, making them easy to recover from memory inspection or logs. Android should consolidate its use of the Android Keystore system for master keys and ensure that derived keys and SQLCipher passwords are handled in a way that avoids stringification, unnecessary duplication, or logging.

Additionally, Wiper.java, discussed under CVE-2025-TELLA3-007, illustrates a simplistic zeroing approach for byte arrays; Android remediation should treat that as a cue to review all lifecycle handling of PINs, passwords, and keys in memory, ensuring that sensitive buffers are short-lived and cleared as reliably as the platform permits.

Authentication and account protection on Android must be aligned with high-risk use cases. The unlock flows in PinUnlockActivity, PasswordUnlockActivity, and PatternUnlockActivity (CVE-2025-TELLA3-013) currently lack strong rate limiting and lockout. This makes any brute-force attacks on local credentials more realistic in the event of device seizure.

Horizontal should define clear policies for tolerated failed attempts, behavior after those limits are reached, and the conditions under which data is wiped or additional safeguards are triggered. Any such policy should be communicated to users in Tella's own

documentation so that they can make informed decisions about PIN length and pattern complexity.

The Android client should also reduce its exposure through permissions and logging. CCE-2025-TELLA3-006 shows that the manifest currently declares broad, high-privilege permissions such as MANAGE_ACCOUNTS and ACCESS_FINE_LOCATION that are not clearly necessary for core functionality. Tella should rationalize this list, removing non-essential permissions and providing explicit in-app explanations and fallbacks where a permission is beneficial but not strictly required.

Debug logging, as described in CCE-2025-TELLA3-003, should be disabled in production builds and audited to ensure that internal APIs, cryptographic data, or network configuration details are not written to logcat in ways that could be accessed by other apps or by an attacker with partial device access.

Input validation on Android should be strengthened in line with CVE-2025-TELLA3-005 and related findings. This means treating user-supplied URLs, server configurations, and metadata as untrusted and validating them before use. Doing so will help ensure that malformed or malicious configuration values cannot be used to alter network behavior, confuse error handling, or create unexpected states that might be exploitable in the future.

5.3 Remediation Actions for Desktop

For Tella Desktop, the primary concern is the integrity of secrets and server-side robustness. CVE-2025-TELLA3-015 reveals that database keys are printed directly to stdout as part of temporary debugging code. This is an immediate confidentiality failure: any attacker with access to logs or a running process can trivially recover the key and decrypt the vault. The first step in Desktop remediation is to remove all logging of keys, passwords, or sensitive configuration, and to adopt a policy that forbids such logging in future development. This policy should be enforced through code review and automated checks where possible.

Randomness and authentication in the Desktop app also require attention. CVE-2025-TELLA3-017 shows that the PIN generation routine for Nearby Sharing uses math/rand rather than a cryptographically secure random source.

This makes the PIN predictable to an attacker who can observe or influence the runtime environment. Remediation should adopt cryptographically secure random functions for all security-sensitive values, including PINs, nonces, session identifiers, and any token or code that gates access to data or operations. This change will directly support the intended guarantees of the Nearby Sharing protocol and reduce the feasibility of guessing-based attacks.

Desktop backend file and database handling should be hardened for both confidentiality and resilience. CVE-2025-TELLA3-019 shows that exported files and ZIP archives are

created with world-readable permissions on Unix-like systems, which is not acceptable for sensitive content.

Recommendations include standardizing file creation with restrictive defaults, revisiting the export directory design (e.g., using ~/Downloads/Tella/), and informing users when exported files fall outside the vault's protection. In parallel, CVE-2025-TELLA3-011 and the file-handling analysis indicate that decrypted data written to disk for export or preview is not always securely cleaned up. Desktop remediation should include a clear strategy for temporary files, including how they are named, where they are stored, how long they persist, and how they are overwritten or removed.

Tella's Desktop backend could also improve its protections against resource exhaustion and injection. CVE-2025-TELLA3-016 and CVE-2025-TELLA3-014 highlight missing file size constraints and overly trusting behavior towards file metadata supplied by clients. Desktop should impose explicit, configurable limits on the size and number of files allowed in uploads, and it should verify that the actual bytes received match the declared metadata. CVE-2025-TELLA3-020 and CVE-2025-TELLA3-023 show that dynamic SQL query construction and unsanitized display of user-controlled data can expose the system to SQL injection and potential XSS in the React frontend. Desktop remediation should prioritize using parameterized queries and explicit input sanitization, and avoid any pattern that renders user-supplied strings into the UI without passing them through React's normal escaping and validation.

Memory management and session lifecycle on Desktop should be improved to prevent long-term degradation or DoS attacks, as described in CVE-2025-TELLA3-024. The current use of in-memory structures that never expire for transfers and registrations can lead to unbounded growth. Remediation should define a clear lifecycle for these objects, including when they are created, when they should be considered stale, and how they are removed. This will reduce the risk of gradual memory exhaustion and the need for operational workarounds such as frequent process restarts.

5.4 Remediation Actions for Nearby Sharing

Tella's Nearby Sharing feature is central to its mission to enable secure communication in constrained and hostile environments. Its design relies on familiar TLS and HTTP mechanisms. Still, its security properties depend on the precise implementation of certificate validation, PIN-based authentication, and careful handling of session state and temporary data. Remediation actions for Nearby Sharing should therefore focus on closing the specific gaps identified during testing and aligning the feature with the broader defense-in-depth strategy.

Nearby Sharing Security Recommendations

1. **Protocol Hardening**
    - Implement mutual authentication for all P2P connections

- Use ephemeral keys for each sharing session
- Implement perfect forward secrecy (PFS)
- Add proximity verification mechanisms
- Implement anti-replay protections

2. **User Safety Features**
   - Add visual/audio indicators during nearby transfers
   - Implement transfer approval workflows
   - Add device verification displays (visual fingerprints)
   - Create sharing activity logs
   - Implement emergency transfer cancellation
3. **Privacy Enhancements**
   - Randomize device identifiers
   - Implement time-limited discovery windows
   - Add obfuscation for device names
   - Minimize broadcast information
   - Implement selective disclosure
4. **Offline Security Measures**
   - Encrypt all temporary transfer files
   - Implement secure deletion of transferred data
   - Add integrity verification for offline data
   - Implement tamper detection
   - Add offline revocation mechanisms

Authentication and brute-force resistance around the six-digit PIN should be strengthened. As noted in the analysis of CVE-2025-TELLA3-017 and CVE-2025-TELLA3-013, the current system uses predictable randomness and lacks robust rate limiting. The PIN should be generated using cryptographically secure randomness on every platform, and the system should enforce clear limits on the number and frequency of attempts.

These limits should apply not only at the registration step but across the entire flow where a PIN or session token gates access to file transfer operations. When limits are exceeded, the system should invalidate the session and require the receiver to start a new sharing session with a new QR code and PIN.

Certificate handling in Nearby Sharing needs to be made both strict and consistent. The current model is that the receiver generates a self-signed certificate and encodes its hash in the QR code, which the sender then validates. However, the existence of TrustAllCerts on Android, weak delegate handling on iOS (CVE-2025-TELLA3-010), and incomplete pinning on Desktop (CVE-2025-TELLA3-018) all indicate that the implementation may allow connections to proceed even when certificate validation fails.

Any fix should ensure that certificate fingerprint verification is mandatory and non-bypassable in all modes, including QR-based pairing, and that any mismatch results in a clear and non-ambiguous error to the user. The implementation should also avoid any

"trust-all" handshake, even for fingerprinting, if there is a safer way to obtain the certificate's DER data.

Session management in Nearby Sharing should be strengthened to reduce the risk of session hijacking and replay attacks. CVE-2025-TELLA3-021 and the broader session-handling analysis show that sessions created via AcceptTransfer can persist indefinitely in memory and be reused by anyone who knows the session ID.

The desired behavior is that each session is single-use, bound to a specific set of files, and expires automatically after completion or after a short period of inactivity. The system should also ensure that all in-memory and on-disk state associated with a session, including temporary files and metadata, is cleaned up when the session ends, regardless of whether the transfer completes successfully, fails, or is canceled.

Nearby Sharing should restore and enforce application-level integrity checks for transferred files. The analysis indicates that SHA-256-based file integrity verification in P2PFile.kt has been commented out, leaving TLS as the only check on data correctness. While TLS protects against many forms of tampering, application-level checks are essential for detecting corruption and providing users assurance that what was received matches what was sent. Remediation should reintroduce and mandate these checks, and ensure that any mismatch is treated as a hard error, with the user clearly informed that the transfer did not complete safely.

Nearby Sharing should be better integrated into Tella's existing security features and user-facing safety guidance. The interaction with Quick Delete currently leaves the possibility open that temporary files or an intermediate state may remain on disk if a panic action is triggered mid-transfer. The camouflage mode does not entirely hide transfer activity, which may matter in scenarios where a user is forced to unlock the device.

Repairing this should ensure that Quick Delete reliably terminates active transfers, clears session state, and removes temporary data, and that the limitations of camouflage during active Nearby Sharing are explicitly documented. Horizontal should also expand OPSEC documentation to cover Nearby Sharing, explaining to users how to choose safe networks, how to recognize suspicious behavior in transfers, and what limitations remain even after the technical fixes described in this report are implemented.

6. Remediation and Fix Verification

6.1 Overview

This section documents the remediation work carried out by Horizontal in response to the findings detailed in Section 4.6, and the verification of those fixes by Convocation Research+Design (CoRD). Following delivery of the initial audit report, Horizontal undertook a coordinated remediation effort across the Tella Android, iOS, and Desktop codebases.

CoRD verified the resulting release candidate fixes and remediation in a retest round conducted in May 2026, followed by an extended verification round under the same OTF Security Lab contract in June 2026 that re-tested the findings still open after the May round.

Of the 24 findings raised in Section 4.6, 21 are classified as Fixed and Verified. One finding (CVE-2025-TELLA3-005) is partially fixed: the Desktop portion is Fixed and Verified while the Android and iOS portions are Risk Accepted by Horizontal. Two further findings (CVE-2025-TELLA3-007 and CVE-2025-TELLA3-013) are also Risk Accepted on the platforms where they apply.

Two findings that were Open at the conclusion of the May 2026 round have since been resolved and verified. In the June 2026 extended round, the iOS portion of CVE-2025-TELLA3-008 (encryption-key storage migrated from the Documents directory to the iOS Keychain) and CVE-2025-TELLA3-018 (self-signed certificates without validation, remediated by a new mutual-TLS peer-to-peer protocol across iOS, Android, and Desktop) are now Fixed and Verified. No findings remain Open. The verification outcomes for these two findings are documented in Section 6.7.

Risk acceptance status is recorded here as a state, not an endorsement. Each Risk Accepted finding remains a real residual risk that should be revisited if Tella's threat model, attack surface, or implementation context changes, or in any follow-on engagement that addresses the affected code paths.

6.2 Fix Verification: Versions Tested in Remediation Round

The Tella 3.0.0 version from 06.2026 retest covered the following release candidate builds:

- Tella Android 3.0.0 (build 236)
- Tella iOS 3.0.0 (build 143)
- Tella Desktop 1.0.0 (installation files: https://github.com/Horizontal-org/Tella-Desktop/actions/runs/25175478020)

Horizontal also published user-facing documentation for the Nearby Sharing feature at https://beta.tella-app.org/nearby-sharing during this period.

The June 2026 extended verification round used the following internal distribution builds: Tella iOS 3.1.0 (149) via TestFlight, Tella Android 3.2.0 (244) via Firebase App Distribution, and Tella Desktop built from the frozen audit-target branch (Tella-Desktop PR #22). These builds are also listed in Section 3.0.4.

6.3 Verification Methodology

For each finding marked Fixed and Verified, CoRD reviewed the relevant pull request or commit referenced by Horizontal, confirmed that the affected component no longer

exhibits the documented behavior under the original proof of concept, and where applicable reran the dynamic test that originally established the issue. As one example, verification of CVE-2025-TELLA3-001 (TrustAllCerts) involved reattempting interception of HTTPS traffic with a self-signed proxy certificate to confirm that the connection now fails closed at the certificate-validation step rather than completing as it did during the original audit. Verification of CVE-2025-TELLA3-017 (predictable PIN) consisted of confirming that the Desktop PIN generation routine has been migrated to crypto/rand and that the predictable math/rand path is no longer reachable in the release build. Findings that depend on configuration or build flags (for example, debug logging in CVE-2025-TELLA3-003 and CVE-2025-TELLA3-022) were verified against production-build artifacts of the relevant release candidates.

Findings labeled Risk Accepted are those Horizontal explicitly chose not to remediate during this contract period. CoRD has reviewed Horizontal's stated reasoning and any compensating controls and notes them in this section without endorsing them as long-term solutions. Where Horizontal's reasoning relies on the soundness of other controls in the system (for example, citing the Restrict Unlock Attempts user setting as a mitigation for the lack of exponential backoff), CoRD's view is that this reasoning is contingent on those controls remaining intact and on users actually configuring them, and the residual risk should be recognized accordingly.

The two findings that were still open after the May 2026 remediation round (CVE-2025-TELLA3-008 and CVE-2025-TELLA3-018) were re-tested in an extended fix verification round conducted under the same OTF Security Lab contract in June 2026 and are now classified Fixed and Verified. That verification combined code and static review of the merged changes with dynamic testing against internal distribution builds (TestFlight for iOS, Firebase App Distribution for Android, and internal artifacts for Desktop). Researchers Note: The per-finding outcomes are documented in Section 6.7.

6.4 Final Status Summary

The table below provides the final disposition for each of the 24 findings raised in Section 4.6. Severity ratings reflect the original audit assessment unless explicitly noted in the per-finding subsections that follow.

Vuln No.	Severity	Title & Platform	Final Status
CVE-2025-TELLA3-001	**Critical**	TrustAllCerts Bypasses SSL Validation (Android)	Fixed and Verified
CVE-2025-TELLA3-002	**High**	Cleartext Traffic Enabled (Android)	Fixed and Verified

Vuln No.	Severity	Title & Platform	Final Status
CVE-2025-TELLA3-003	**Medium**	Debug Information Leak (Android, iOS, Desktop)	Fixed and Verified
CVE-2025-TELLA3-004	**Medium**	Weak Key Generation Error Handling (Android)	Fixed and Verified
CVE-2025-TELLA3-005	**Medium**	Insufficient Input Validation	Mixed: Desktop Fixed and Verified; Android, iOS Risk Accepted
CVE-2025-TELLA3-006	**Medium**	Excessive Permission Requests (Android)	Fixed and Verified
CVE-2025-TELLA3-007	**Low**	Insufficient Memory Wiping (Android)	Risk Accepted
CVE-2025-TELLA3-008	**Critical**	Insecure Key Storage in Documents Directory	Fixed and Verified
CVE-2025-TELLA3-009	**High**	Weak Keychain Implementation (iOS)	Fixed and Verified
CVE-2025-TELLA3-010	**High**	Insufficient Certificate Pinning (iOS)	Fixed and Verified
CVE-2025-TELLA3-011	**Medium**	Decrypted Files Stored in Temporary Directory (Desktop, iOS)	Fixed and Verified
CVE-2025-TELLA3-012	**Medium**	Database Key Passed as Plain String (Android)	Fixed and Verified

Vuln No.	Severity	Title & Platform	Final Status
CVE-2025-TELLA3-013	**Medium**	Insufficient Authentication Rate Limiting (Android, iOS)	Risk Accepted
CVE-2025-TELLA3-014	**Low**	Insecure File Type and Size Validation (Desktop)	Fixed and Verified
CVE-2025-TELLA3-015	**Critical**	Database Keys Logged to Stdout (Desktop)	Fixed and Verified
CVE-2025-TELLA3-016	**High**	File Upload Without Size Validation (Desktop)	Fixed and Verified
CVE-2025-TELLA3-017	**High**	Predictable Random PIN Generation (Desktop)	Fixed and Verified
CVE-2025-TELLA3-018	**High**	Self-signed Certificates Without Validation (iOS, Android, Desktop)	Fixed and Verified
CVE-2025-TELLA3-019	**High**	World-readable File Permissions (Desktop)	Fixed and Verified
CVE-2025-TELLA3-020	**Medium**	Potential SQL Injection on Input Validation Failure (Desktop)	Fixed and Verified
CVE-2025-TELLA3-021	**Medium**	Weak Session Management (Desktop)	Fixed and Verified
CVE-2025-TELLA3-022	**Medium**	Verbose Error Logging (Android, iOS, Desktop)	Fixed and Verified
CVE-2025-TELLA3-023	**Low**	Potential XSS in React Frontend (Desktop)	Fixed and Verified

Vuln No.	Severity	Title & Platform	Final Status
CVE-2025-TELLA3-024	Medium	Memory Leaks (Desktop)	Fixed and Verified

6.5 Verified Fixed Findings

As per the Open Technology Fund's standards, the following fix verification findings were remediated by Horizontal and verified by CoRD against the Summer 2026 release candidates. Where multiple commits or pull requests are referenced for a single finding, all linked changes were reviewed and incorporated into the verification result. The references below capture the principal change sets supplied by Horizontal in their remediation tracker; readers should consult the underlying repositories for the complete set of associated commits.

Vuln No.	Platform	Verification and Code References
CVE-2025-TELLA3-001	Android	TrustAllCerts class removed from FingerprintFetcher.kt, ServerPinger.kt, and TellaPeerToPeerClient.kt; certificate validation now enforced. Verified against original PoC: HTTPS interception with a self-signed proxy certificate now fails the handshake. PR Tella-Android #466.
CVE-2025-TELLA3-002	Android	Cleartext traffic disabled in AndroidManifest.xml and related network security config. Verified that no plaintext HTTP fallback path remains for sensitive application traffic. PR Tella-Android #466.
CVE-2025-TELLA3-003	Android, iOS, Desktop	Debug logging removed or gated behind debug-only build flags across all three platforms. iOS uses debugLog wrapper that compiles only in debug builds. Verified production-build artifacts no longer emit internal API responses, SSL negotiation traces, or cryptographic data to platform logs.
CVE-2025-TELLA3-004	Android	AES key generation error handling reworked to fail closed rather than silently returning a null key. MainKey.java updated; simulated exception path now propagates an explicit error. PR Tella-Android #466.
CVE-2025-TELLA3-005 (Desktop)	Desktop	URL and server-configuration form validation added to NearbySharing/Connect.tsx and the corresponding

Vuln No.	Platform	Verification and Code References
		backend handlers. Invalid URL entries are rejected with explicit error messaging. PR Tella-Desktop #18. Android and iOS portions of this finding are Risk Accepted; see Section 6.6.
CVE-2025-TELLA3-006	Android	MANAGE_ACCOUNTS and other unnecessary high-privilege permissions removed from AndroidManifest.xml. ACCESS_FINE_LOCATION retained because it backs documented product features (Collect/forms, Vault metadata, location-tagged media, F-Droid map flow) and is requested at runtime, not silently held. CoRD considers the residual permission set acceptable given the documented use cases; users should still be made aware that location permission can reveal device position when granted.
CVE-2025-TELLA3-008	Android, iOS	Android-side temp-file handling in TellaPeerToServer.kt, ServerPinger.kt, and TellaPeerToPeerClient.kt corrected so that temporary files containing sensitive data are reliably deleted, including in exception paths (PR Tella-Android #466). The iOS portion was resolved separately in the June 2026 extended round: encryption keys were migrated from the application Documents directory to the iOS Keychain, with the encrypted private key stored as a Keychain generic password (accessibility kSecAttrAccessibleWhenUnlockedThisDeviceOnly), the meta key in the Secure Enclave, one-time migration of legacy on-disk key files followed by their deletion, and unlock reading from the Keychain (PR Tella-iOS #300). Verified against Tella iOS 3.1.0 (149); see Section 6.7.
CVE-2025-TELLA3-009	iOS	kSecAttrAccessControl added to KeychainManager save path. Unused CryptoManager struct removed. Stored key items now require user authentication consistent with Tella's expected access model. PR Tella-iOS #264.
CVE-2025-TELLA3-010	iOS	Certificate pinning logic in NearbySharingURLSessionDelegate corrected so that hash mismatch and exception paths reliably terminate

Vuln No.	Platform	Verification and Code References
		the connection with cancelAuthenticationChallenge rather than allowing the session to continue. PR Tella-iOS #264.
CVE-2025-TELLA3-011	Desktop, iOS	Decrypted temporary files now subject to deterministic cleanup. Desktop introduces a temp-file lifecycle in the encryption module. iOS rewires VaultManager paths (loadFileToURL, loadVaultFileToURL, saveDataToTempFile, createTempFileURL, deleteTmpFiles) to ensure decrypted data is removed after use. PR Tella-Desktop #18; PR Tella-iOS #264. Horizontal has noted that a more comprehensive decryption-process refactor is planned for a future release.
CVE-2025-TELLA3-012	Android	Database key handling in CipherOpenHelper.java, VaultSQLiteOpenHelper.java, VaultDataSource.java, and BaseVault.java reworked to reduce stringification and avoid embedding the key into SQL via string concatenation. PR Tella-Android #466.
CVE-2025-TELLA3-014	Desktop	File type and size validation added at the upload entry point. PrepareUploadRequest.Validate() now enforces declared metadata against accepted bounds, and StoreFile no longer reads arbitrarily large bodies into memory. PR Tella-Desktop #18.
CVE-2025-TELLA3-015	Desktop	fmt.Printf debug emission of the hex-encoded database key removed from backend/core/database/database.go. Verified that no path in the release build emits the key to stdout or other log streams. Commit b3f12d8.
CVE-2025-TELLA3-016	Desktop	Upload handler enforces explicit body size limits. Multiple changes to the transfer service constrain both per-request and per-file size. PR Tella-Desktop #18.
CVE-2025-TELLA3-017	Desktop	PIN generation migrated from math/rand to crypto/rand (crypto/rand.Int / crypto/rand.Read). Verified that the only PIN-generation path in the

Vuln No.	Platform	Verification and Code References
		release build draws from a cryptographically secure source. PR Tella-Desktop #18.
CVE-2025-TELLA3-018	iOS, Android, Desktop	Self-signed certificates without validation remediated by a new mutual-TLS (mTLS) peer-to-peer protocol (Tella-P2P-Protocol PR #14) implemented across all platforms (Tella-iOS #300, Tella-Android #489, Tella-Desktop #22). Both peers present ephemeral client certificates over API v2 routes and pin each other by certificate SHA-256 hash, with bidirectional hash-verification UX and rejection of v1 clients. Desktop enforces validation in service.go via two-phase ClientAuth and VerifyPeerCertificate against the pinned sender hash. Verified against Tella iOS 3.1.0 (149), Tella Android 3.2.0 (244), and internal Desktop builds; negative tests confirmed that connections without a client certificate, or with the wrong client certificate, are rejected after pinning while the correct pinned certificate is accepted. See Section 6.7.
CVE-2025-TELLA3-019	Desktop	Export and ZIP file creation now use restrictive default permissions rather than 0644 world-readable. PR Tella-Desktop #18.
CVE-2025-TELLA3-020	Desktop	Dynamic SQL query construction in backend/core/modules/filestore/service.go replaced with parameterized queries. Verified the affected code path no longer interpolates user-controlled values into SQL strings.
CVE-2025-TELLA3-021	Desktop	Active transfer sessions now have explicit lifecycle controls and bounded validity. Sessions are removed from the in-memory map on completion, failure, or abandonment, with an explicit invalidation entry point. PR Tella-Desktop #18.
CVE-2025-TELLA3-022	Android, iOS, Desktop	Verbose error messages reduced or replaced with generic messages across all three platforms. Internal implementation details, file paths, and database identifiers are no longer surfaced to clients. PR Tella-Android #466; PR Tella-iOS #264; PR Tella-Desktop #18.

Vuln No.	Platform	Verification and Code References
CVE-2025-TELLA3-023	Desktop	User-controlled strings rendered in the React frontend (FileList.tsx, FileRequest.tsx, FileReceiving.tsx, FolderList.tsx) routed through explicit sanitization in addition to React's default JSX escaping. Verified no use of dangerouslySetInnerHTML on the affected paths. PR Tella-Desktop #18.
CVE-2025-TELLA3-024	Desktop	Transfer and registration session maps now have cleanup on completion, failure, and timeout. The in-memory growth path documented in the original PoC no longer applies. PR Tella-Desktop #18.

6.6 Risk Accepted Findings

The following findings have been classified as Risk Accepted because Horizontal has elected, with stated rationale, not to remediate them during the current contract period. Each subsection summarizes Horizontal's reasoning, paraphrased from their remediation tracker, followed by CoRD's view of the residual risk. As noted in Section 6.3, risk acceptance is recorded here as a status, not an endorsement: the underlying weakness remains, and the finding should be revisited if Tella's threat model or implementation context changes.

CVE-2025-TELLA3-005 (Android and iOS)

Horizontal's rationale: input validation for URLs, server configurations, and metadata on Android and iOS affects core flows in the Tella applications and requires extended testing to deliver safely. Horizontal has elected to defer the full fix to a future release rather than ship a partial change that risks destabilizing core functionality.

CoRD's rationale: the underlying weakness on Android and iOS is unchanged. The original impact, acceptance of malformed URLs and configuration values, and the resulting exposure to information disclosure or unexpected state transitions, remains as described in the original Section 4.6 entry. Horizontal's stated reason is operational rather than analytic; CoRD does not have evidence that the risk has been reduced on these platforms. We recommend that this work be tracked explicitly and that the deferral be revisited in any subsequent release that touches the affected input paths. The Desktop portion of this finding has been remediated and is captured in Section 6.5.

CVE-2025-TELLA3-007 (Android)

Horizontal's rationale: this finding is being de-prioritized relative to higher-severity items in the current cycle, and may be addressed depending on time available before release.

Horizontal has framed this as a time-driven prioritization decision rather than a permanent disposition.

CoRD's rationale: the simple zero-fill behavior in Wiper.java remains as described in Section 4.6, and the underlying limitations of byte-array zeroing on the JVM remain in place. The original severity rating of Low is retained. Given the Low severity, the relatively narrow conditions under which the residual data remnants are observable, and Horizontal's stated time-driven prioritization, CoRD records this finding as Risk Accepted for the current contract cycle. We recommend that it be revisited if memory-handling code in the affected paths changes for any reason, or in any follow-on engagement.

CVE-2025-TELLA3-013 (Android and iOS)

Horizontal's rationale: exponential backoff would constitute a new feature, and Tella does not maintain server-side accounts that can be locked. Horizontal points to the existing Restrict Unlock Attempts user setting (documented at https://tella-app.org/features#restrict-unlocking-attempts) as a sufficient mitigation against brute-force attacks on local PIN, password, and pattern unlock, and suggests that, depending on the user's risk profile, configuring a small number of attempts (for example five or ten) provides effective protection.

CoRD's rationale: the Restrict Unlock Attempts setting is a meaningful control for users who enable it, but it is not enabled by default on every install, and its protection is contingent on users configuring it consistently with their threat model. The unlock paths in PinUnlockActivity, PasswordUnlockActivity, and PatternUnlockActivity (and their iOS equivalents) still lack platform-enforced backoff or progressive delay, so a determined attacker with persistent device access can submit unlock attempts at the rate the OS allows. CoRD considers Horizontal's compensating control reasonable as a near-term position, but recommends (a) ensuring the Restrict Unlock Attempts setting is exposed prominently in onboarding for high-risk users, and (b) revisiting platform-level backoff in a future release. The original severity rating of Medium is retained.

6.7 Findings Resolved in the June 2026 Extended Verification Round

The two findings below were Open at the conclusion of the May 2026 remediation round. Both were subsequently remediated by Horizontal and re-tested by CoRD in an extended verification round conducted under the same OTF Security Lab contract in June 2026, and both are now Fixed and Verified. Each subsection retains the original position and the compensating controls that applied while the finding was open, followed by the June 2026 verification outcome.

CVE-2025-TELLA3-008 - iOS portion (Insecure Key Storage in Documents Directory)

Status: Fixed and Verified (June 2026 extended round). The Android temp-file handling sub-issue was remediated in the May round and is captured in Section 6.5; the core iOS key-storage issue has now also been remediated and verified, as described in the June 2026 verification note below.

Horizontal's position: Horizontal agreed with the original Critical severity rating and noted that this was a pre-existing iOS implementation issue rather than one introduced by the Nearby Sharing feature. Because migrating iOS encryption-key storage from the Documents directory to the iOS Keychain, with Secure Enclave backing where appropriate, was an extensive refactor affecting a substantial portion of the iOS codebase, it was scheduled after the Nearby Sharing-specific fixes and completed in the June 2026 round.

CoRD's rationale: the original Critical severity rating was retained while the finding was open. As described in Section 4.6, encryption keys stored in the iOS Documents directory were exposed to backup and forensic-extraction paths inappropriate for Tella's threat model, and no in-application compensating controls changed that underlying exposure, so until migration to the Keychain was confirmed, users were considered exposed to the original attack scenario, particularly where adversaries had access to iCloud backups, paired computers, or unlocked devices. That migration has since been verified, as recorded below.

June 2026 Re-Test Verification: Horizontal migrated iOS encryption-key storage from the application Documents directory to the iOS Keychain (Tella-iOS PR #300). CoRD confirmed that fresh vault setup writes the encrypted private key to the Keychain as a generic password with accessibility kSecAttrAccessibleWhenUnlockedThisDeviceOnly, stores the meta key in the Secure Enclave, migrates legacy file-based keys once and then deletes the on-disk key folder, and reads key material from the Keychain on unlock; only encrypted vault blobs remain under Documents/files/. Verification was performed against Tella iOS 3.1.0 (149) installed via TestFlight, covering fresh install, launch, vault setup, app restart, device restart, and the upgrade path. Encryption keys are no longer written to the application directory. The finding is Fixed and Verified.

CVE-2025-TELLA3-018 (Self-Signed Certificates Without Validation: iOS, Android, Desktop)

Status: Fixed and Verified (June 2026 extended round). The mutual-TLS remediation described below was implemented across iOS, Android, and Desktop and verified by CoRD; the original severity rating of High is retained as a record of the finding as originally raised.

Horizontal's position: a full remediation required introducing mutual TLS (mTLS) so that the Receiver pins the Sender's certificate fingerprint as well as the reverse. Horizontal noted that mTLS involved a protocol change and cross-platform changes to the connection flow, including UX changes for how connections are established between devices, and a

development branch addressing this on Desktop was available at https://github.com/cblgh/Tella-Desktop/tree/2026-02-audit-fixes-with-mtls. Horizontal subsequently implemented mTLS across Android, iOS, and Desktop, which was verified in the June 2026 round.

While the finding was open, Horizontal identified a set of compensating controls already in place that, taken together, raised the bar for exploitation: PIN gating at registration; the Sender's pinning of the Receiver's TLS certificate fingerprint, which prevented a man-in-the-middle from delivering the legitimate registration response; the protection of registration payload contents (including the session ID) by the TLS connection itself; correct session-ID enforcement at the upload endpoint (strengthened by the fix to CVE-2025-TELLA3-021); restriction of uploads to known transmission IDs bound to an initial sender request; restoration of SHA-256 file-integrity comparison on completed uploads, so that uploads failing the hash check were discarded; and rate-limiting that mitigated denial-of-service variants. In combination, these controls made it difficult for an attacker on the same network to mount the residual attack: the attacker would have needed a valid session ID, which was protected by TLS; could not receive uploaded files because the Sender pinned the Receiver's certificate; and would have had to bypass session and transmission ID checks on the Receiver.

CoRD's rationale: the compensating controls Horizontal cited were real and substantive, and they materially reduced the practical exploitability of the original finding in the typical Nearby Sharing flow; CoRD independently verified the upstream fixes those controls depended on (CVE-2025-TELLA3-001, -010, -017, -021, and the SHA-256 integrity check) as part of the remediation round. The underlying weakness, that the Receiver did not pin the Sender's certificate, remained until mTLS was implemented, which is why the finding was held at High severity pending that work. mTLS has since been implemented across all three platforms and verified, as recorded below.

June 2026 Re-Test Verification: Horizontal implemented a new mutual-TLS peer-to-peer protocol (Tella-P2P-Protocol PR #14) across iOS (Tella-iOS PR #300), Android (Tella-Android PR #489), and Desktop (Tella-Desktop PR #22). Under the revised API v2 routes, the sender generates an ephemeral client identity and presents a client certificate over HTTPS, the receiver requires a peer client certificate, and each side pins the other by certificate SHA-256 hash with bidirectional hash-verification UX; v1 clients are rejected. On Desktop, certificate.go still emits self-signed server certificates (expected) while service.go enforces validation via two-phase ClientAuth and VerifyPeerCertificate against the pinned sender hash. CoRD verified interoperability across Tella iOS 3.1.0 (149), Tella Android 3.2.0 (244), and internal Desktop builds. Automated negative tests confirmed that a connection without a client certificate, or with the wrong client certificate, is rejected after pinning, while the correct pinned certificate is accepted (/api/v2/ping returns 200). The finding is Fixed and Verified, and no blocking regressions were identified.

6.8 Overall Posture After Remediation

The remediation work delivered by Horizontal in the Tella release candidates substantially improves Tella's security posture relative to the state described in the original audit. The most acute findings: Android TrustAllCerts (CVE-2025-TELLA3-001), database key logging on Desktop (CVE-2025-TELLA3-015), predictable PIN generation (CVE-2025-TELLA3-017), cleartext traffic on Android (CVE-2025-TELLA3-002), and weak key generation error handling (CVE-2025-TELLA3-004), are now Fixed and Verified. Across the ecosystem, the patterns of ad hoc trust bypass, verbose logging of sensitive data, and unbounded resource consumption that characterized the original assessment have been materially reduced. The Nearby Sharing feature, which was the principal focus of this audit, now operates with enforced certificate pinning on the client side, cryptographically secure PIN generation, bounded session lifetimes with explicit cleanup, and restored application-level integrity verification. The combination of these fixes brings Nearby Sharing closer to the security model originally intended by Horizontal's design. CoRD's overall assessment is that Tella, in its release-candidate state, is a meaningfully more defensible application than the version assessed originally. The user-facing OPSEC documentation discussed in Section 5 should be expanded to acknowledge the residual risks in the Risk Accepted findings (notably the configuration-dependent nature of brute-force protection in CVE-2025-TELLA3-013) so that users can make informed decisions about how to deploy the application in their specific threat environments.

The two findings that were still open after the May 2026 round have since been closed. In the June 2026 extended round, the iOS portion of CVE-2025-TELLA3-008 was remediated by migrating encryption-key storage from the Documents directory to the iOS Keychain (with Secure Enclave backing for the meta key, restricted Keychain accessibility, and one-time migration and deletion of legacy on-disk key files), and CVE-2025-TELLA3-018 was remediated by a new mutual-TLS peer-to-peer protocol in which both peers present ephemeral client certificates and pin each other by certificate hash across iOS, Android, and Desktop. CoRD verified both fixes, and no findings remain Open. The only residual items are the Risk Accepted findings (CVE-2025-TELLA3-005 on Android and iOS, CVE-2025-TELLA3-007 on Android, and CVE-2025-TELLA3-013 on Android and iOS), which Horizontal elected not to remediate in this contract period and which are documented in Section 6.6. Subject to those residual items, the application provides its core protections, encrypted storage, secure peer-to-peer sharing, and emergency data deletion, with a level of implementation discipline appropriate for journalists, activists, and human rights defenders.

7. Appendix

This section will provide additional information for both Horizontal and future users to enhance security, privacy, and understanding of the broader threat landscape or ecosystem.

Appendix A: Glossary

Term	Definition

Air-Gapped	Systems or networks physically isolated from unsecured networks
Camouflage	Tella features that disguise the app as a calculator
Divvi Up	Privacy-preserving telemetry system used by Tella
Defense-in-Depth (DiD)	Information security strategy integrating people, technology, and operations capabilities to establish variable barriers across multiple layers and missions of the organization
Security-by-Design (SbD)	A foundational approach in the software development lifecycle that ensures security is engineered into applications and services from the outset, making them resilient to threats and aligned with both regulatory and organizational policies
Evil Twin	Malicious WiFi access point mimicking a legitimate one
FOSS	Free and Open Source Software - Tella-FOSS variant without trackers
LocalSend	An open-source protocol for local file sharing that Tella's Nearby Sharing is based on
mDNS	Multicast DNS - Used for device discovery on local networks
MITM	Man-in-the-Middle attack
ODK	Open Data Kit - Platform for data collection that Tella integrates with
P2P	Peer-to-Peer - Direct communication between devices
PBKDF2	Password-Based Key Derivation Function 2 - Key stretching algorithm
PFS	Perfect Forward Secrecy - A Cryptographic property ensuring session keys cannot be compromised
PIN	Personal Identification Number - Used for authentication in Nearby Sharing
QR Code	Quick Response Code - Used for device pairing in Nearby Sharing
Quick Delete	Tella has an emergency feature for rapid data deletion with a 5-second countdown
Rogue Access Point	Unauthorized WiFi access point that may intercept traffic
STRIDE	Threat modeling methodology (Spoofing, Tampering, Repudiation, Information disclosure, DoS, Elevation)
Tella Web	Web-based platform for managing Tella reports
TLS	Transport Layer Security - A Cryptographic protocol for secure communications
Uwazi	Documentation platform that Tella integrates with
Vault	Tella's encrypted storage for sensitive files

End of Horizontal's Tella 3.0.0+ Security Audit