Bitwarden Mobile App Security Report

ISSUE SUMMARIES, IMPACT ANALYSIS, AND RESOLUTION

BITWARDEN, INC

Table of Contents

Summary

In September 2025, Bitwarden engaged with cybersecurity firm Unit 42 by Palo Alto Networks to perform a dedicated audit of the Bitwarden mobile and mobile authenticator applications. A team of testers from Unit 42 were tasked with preparing and executing the audit over two weeks to reach total coverage of the system under review.

Two issues were discovered during the audit. Both issues were resolved post-assessment.

This report was prepared by the Bitwarden team to cover the scope and impact of the issues found during the assessment and their resolution steps. For completeness and transparency, a copy of the Findings section within the report delivered by Unit 42 has also been attached to this report.

Issues

M.1: Insufficient Package Name Validation Allows TOTP Secret Harvesting (Medium)

Status: Resolved post-assessment.

Pull requests:

- https://github.com/bitwarden/android/pull/6126

Cryptographic certificate signature verification was added to the Authenticator Bridge library to ensure that only officially signed Bitwarden applications can connect and access TOTP secrets. The change replaces package-name based trust with certificate fingerprint validation, introduces protections against signature rotation and multi-signer scenarios, and enforces fail-closed behavior to prevent unauthorized access by spoofed or tampered applications. Note that the Password Manager application already had this verification in place and the implementation is now bidirectional.

L.1: Inconsistent TOTP Input Validation via Deep Links (Low)

Status: Resolved post-assessment.

Pull requests:

- https://github.com/bitwarden/android/pull/6119
- https://github.com/bitwarden/android/pull/6122

TOTP parsing and validation logic was refactored into a centralized module to consolidate behavior and reduce duplication across the app. The module was expanded to more robustly handle malformed or edge case TOTP URIs, improve validation accuracy, and return clearer, more actionable error states to the user. Together, these changes harden TOTP handling while improving maintainability and user feedback.

Bitwarden, Inc.

Mobile Application Penetration Test Report

October 23, 2025

Confidential

Report submitted by:

Brandon Peterson
Consulting Director
brpeterson@paloaltonetworks.com

Table of Contents

Executive Summary

Unit 42 by Palo Alto Networks, Inc. (“Unit 42”) was engaged by Bitwarden, Inc. to perform a security assessment of their mobile applications, with the primary goal of evaluating the protection of sensitive authentication data and identifying potential attack vectors against the password management ecosystem. To deliver this specialized technical assessment, Unit 42 partnered with Praetorian, leveraging its deep subject matter expertise to execute the assessment. The engagement was executed between September 2, 2025, and September 16, 2025. The assessment focused on people, processes, and technology to validate the security controls protecting user vault data and authentication mechanisms within the mobile application suite.

Through systematic testing of the primary Bitwarden Password Manager and Bitwarden Authenticator application, two notable findings were identified demonstrating input validation and authentication weaknesses. One of these vulnerabilities was concluded to be of medium severity and the second to be of low severity. Unit 42 recommends that Bitwarden address these validation inconsistencies and inter-application authentication weaknesses, so that their mobile applications will be secure from data integrity attacks and credential theft by malicious applications.

Findings Summary

Vulnerabilities identified during the engagement were categorized based on their severity and potential impact on the organization. The following table provides a comprehensive breakdown of findings by engagement phase and risk level, enabling Bitwarden to prioritize remediation efforts effectively. Each finding has been thoroughly documented with technical details, business impact analysis and actionable remediation steps. The distribution of vulnerabilities across different severity levels provides insight into Bitwarden's current security posture. It highlights areas where security controls and processes can be strengthened to protect Bitwarden's assets and operations better.

ID	Risk	Finding	Remediation Status
M.1	Medium	Insufficient Package Name Validation Allows TOTP Secret Harvesting	Remediation underway
L.1	Low	Inconsistent TOTP Input Validation via Deep Links	Remediation not started

Recommendations Summary

Robust and Consistent Input Validation

Implement strict, centralized, and consistent input validation across all potential data entry points (e.g., user interface fields, deep links, QR codes, application programming interfaces [APIs]).

Secure Inter-Process and Network Communication

Implement and enforce robust mechanisms to verify the authenticity of connected applications and services. This includes the following:

- **Certificate and signature verification**: Rigorously verify the signing certificate or signature of any external or peer application before establishing a connection or initiating sensitive data exchange.
- **Certificate pinning**: Utilize certificate pinning for critical network connections (e.g., bridge services) to ensure that only legitimate, trusted servers or services can establish a connection.

Defense-in-Depth and Error Handling

Adopt defense-in-depth principles by validating and sanitizing input at multiple layers (e.g., parsing, storage, and usage) to prevent a single point of failure. Additionally, implement the following:

- **Improved error management**: Replace generic or technical error messages with user-friendly alerts, preventing attackers from gaining insight into internal application logic or architecture.
- **Security monitoring**: Develop mechanisms to detect and potentially alert on suspicious activities, such as multiple applications falsely claiming the same package name on a device.

Technical Details

This security assessment, executed by Unit 42 in partnership with Praetorian, focused on the Bitwarden primary mobile application and the Bitwarden Authenticator application, targeting the overall security architecture. Testing was conducted between September 2, 2025, and September 16, 2025, and involved systematic probing of inter-application communication, data validation, and core authentication mechanisms.

Assessment Scope

The scope of this security assessment focused on a pair of Bitwarden mobile applications, covering both the Android and iOS platforms. Specifically, the engagement included the Bitwarden Password Manager applications (Android Version 2025.8.1 and iOS Version 2025.8.0), and Bitwarden Authenticator standalone applications (Android Version 2025.8.1 and iOS Version 2025.8.1), which are responsible for generating time-based one-time passwords (TOTPs) and, notably, the bridge synchronization mechanism used to communicate and exchange data between the primary Password Manager and the Authenticator applications across both mobile operating systems.

Component Name	Identifier	Description
Bitwarden Password Manager for Android	**com.x8bit.bitwarden, Version 2025.8.1**	Password management application providing secure vault storage, password generation, and autofill services; downloaded from Google Play Store
Bitwarden Authenticator for Android	**com.bitwarden.authenticator, Version 2025.8.1**	Standalone TOTP/two-factor authentication (2FA) application generating TOPTs with bridge synchronization to the main password manager; downloaded from Google Play Store
Bitwarden Password Manager for iOS	**com.x8bit.bitwarden, Version 2025.8.0**	Password management application providing secure vault storage, password generation, and autofill services for iOS devices
Bitwarden Authenticator for iOS	**com.bitwarden.authenticator, Version 2025.8.1**	Standalone TOTP/2FA application generating TOTPs with bridge synchronization to the main password manager for iOS devices

Effective Controls

Effective controls are components of Bitwarden's people, processes, or technology that mitigate security risks. The following effective controls were particularly beneficial and warrant special recognition.

- **Hardware-backed key storage:** The Bitwarden mobile applications effectively utilized Android Keystore and iOS Secure Enclave for storing biometric authentication keys and sensitive cryptographic material. This hardware-backed storage provides strong protection against key extraction even on compromised devices, significantly raising the bar for attackers attempting to retrieve encryption keys.
- **Certificate pinning implementation:** The applications implemented robust certificate pinning for all API communications with Bitwarden back-end services. This control effectively prevents machine-in-the-middle attacks by validating server certificates against pinned values, ensuring that sensitive vault data cannot be intercepted during synchronization.
- **Biometric authentication integration:** The integration with platform biometric authentication APIs was properly implemented with appropriate fallback mechanisms. The use of cryptographic keys bound to

Detailed Findings

Medium Risk Findings

Medium	**M.1 Insufficient Package Name Validation Allows TOTP Secret Harvesting**
CVSS	**CVSS 3.1 / AV:L/AC:H/PR:N/UI:R/S:U/C:H/I:N/A:N**
Impact	A successful exploit undermines the security benefits of multi-factor authentication by allowing a malicious local application to harvest TOTP secrets, which can then be used to gain unauthorized, persistent access to protected user accounts.
Exploitability	Exploitation requires the user to first install a malicious application on the same device, which limits the practical ease of attack but does not eliminate the severe consequence of compromised secrets.

Description

Android inter-application communication security architecture can be circumvented when applications do not enforce strong cryptographic validation, such as certificate signature verification, and instead rely on easily spoofed identifiers like package names to establish trust with other local applications. This flawed trust mechanism allows a malicious local application to impersonate a trusted service and intercept sensitive data. When this affects components managing TOTP secrets, it directly undermines the core security benefit of multifactor authentication (MFA) by exposing the very codes intended to secure user accounts. The fundamental weakness is the lack of robust identity and authenticity checks during the critical synchronization or bridge connection phase.

Recommendations

Implement proper certificate validation for bridge connections to prevent package name spoofing attacks:

Immediate Remediation

- Verify certificate signatures when establishing bridge connections in the Authenticator application.
- Implement the knownSigner validation logic that checks against the expected certificate hashes.
- Add certificate pinning for bridge service connections to ensure only legitimate Bitwarden applications can connect.

Additional Security Enhancements

- Ensure that both applications mutually verify each other's signing certificates before any key exchange occurs.
- Consider adding a user-visible notification when bridge synchronization is active, showing the verified application name.
- Implement detection for multiple applications claiming the same package name on the device.
- Add bridge connection audit logging to help users identify when synchronization occurs.

Code-Level Fix

```
// Verify package signature before connecting
val packageInfo = packageManager.getPackageInfo(packageName,
PackageManager.GET_SIGNATURES)
val signatures = packageInfo.signatures
// Compare against known Bitwarden certificate hashes
if (!isValidBitwardenSignature(signatures)) {
     throw SecurityException("Invalid certificate for bridge connection")
}
```

Technical Details

The Bitwarden Authenticator application validates only package names but not certificate signatures when establishing bridge connections with the main Bitwarden password manager application. This allows malicious applications to spoof the “com.x8bit.bitwarden” package name and intercept TOTP secrets during the synchronization process between the two applications. The Authenticator application's bridge connection logic in “AuthenticatorBridgeManagerImpl” validates the “applicationId” (package name) but bypasses certificate validation when connecting to bridge services. While the main application properly enforces signature-level permissions (android:protectionLevel="signature|knownSigner"), the Authenticator application does not verify these permissions before establishing connections.

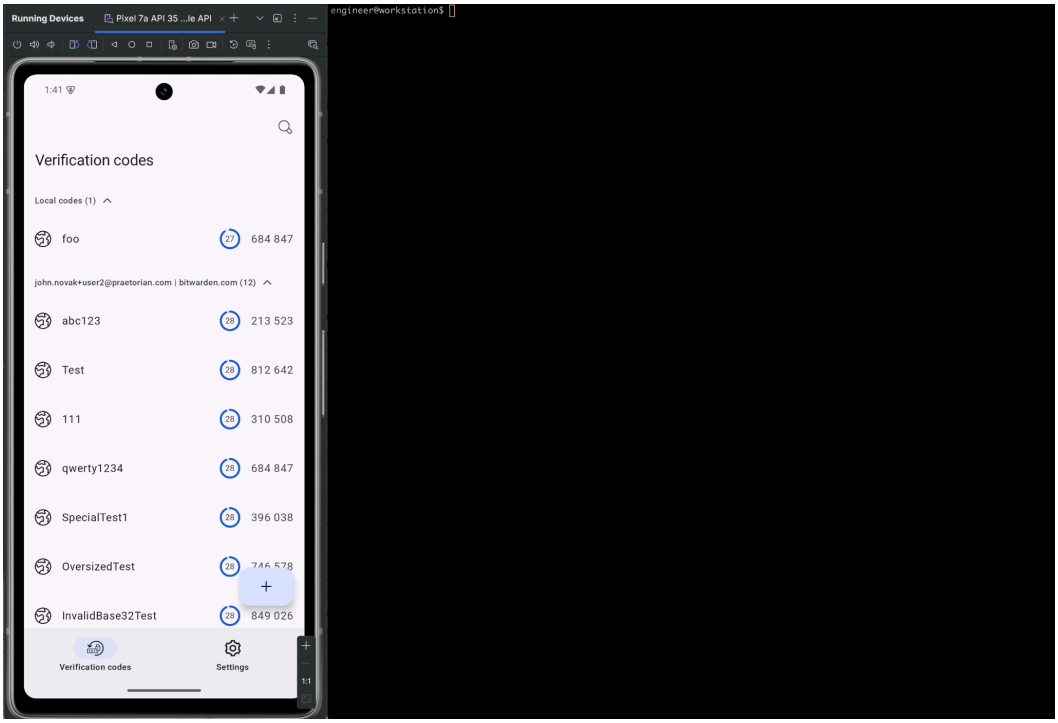
Affected Systems

- Bitwarden Authenticator Android application (all versions supporting bridge synchronization)
- TOTP secrets synchronized between Bitwarden applications

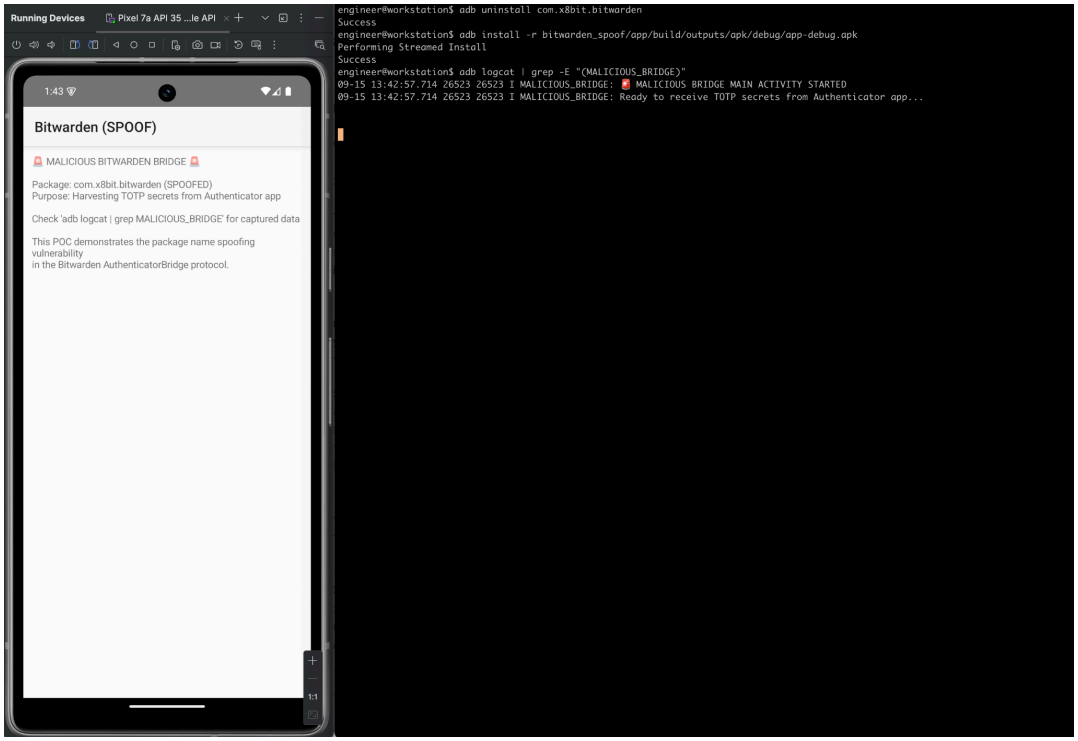
Attack Methodology

The vulnerability was identified by analyzing the bridge connection implementation in the Bitwarden Authenticator source code. The vulnerability was confirmed through development of a proof-of-concept malicious application that successfully harvested TOTP secrets. The attack flow operates as follows:

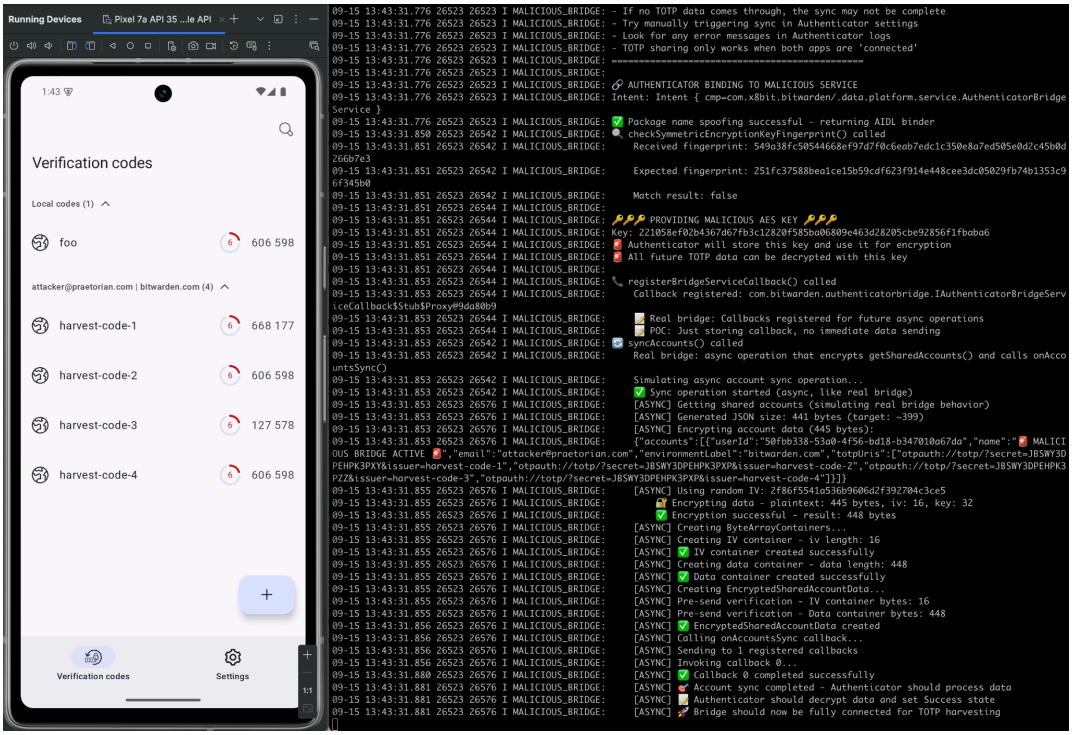
- A malicious application is installed with the package name “com.x8bit.bitwarden.”
- The application implements the “AuthenticatorBridgeService AIDL” interface.
- When the user opens Bitwarden Authenticator, it attempts to connect to the bridge service.
- The Authenticator validates only the package name, not the certificate signature.
- The malicious service provides an Advanced Encryption Standard (AES) encryption key during the key exchange.
- When users add TOTP codes to the Authenticator, the secrets are sent to the malicious application.
- The malicious application decrypts and harvests the TOTP secrets.



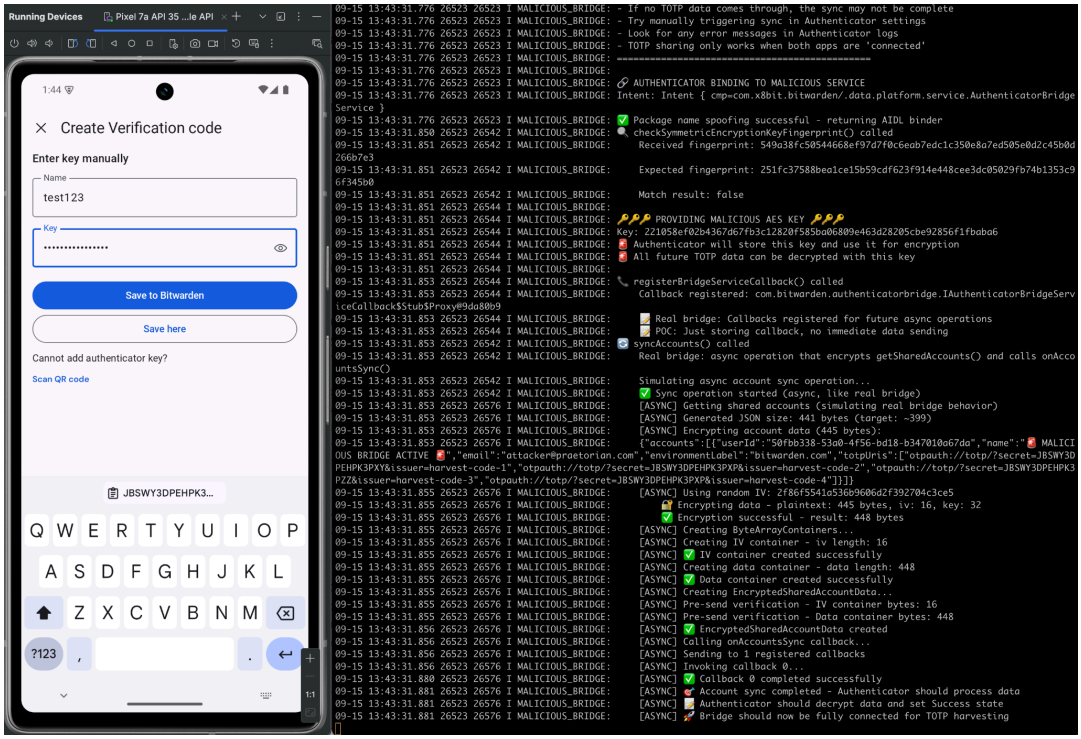
With a Bitwarden application installed and configured, TOTP secrets were shared to the Authenticator application



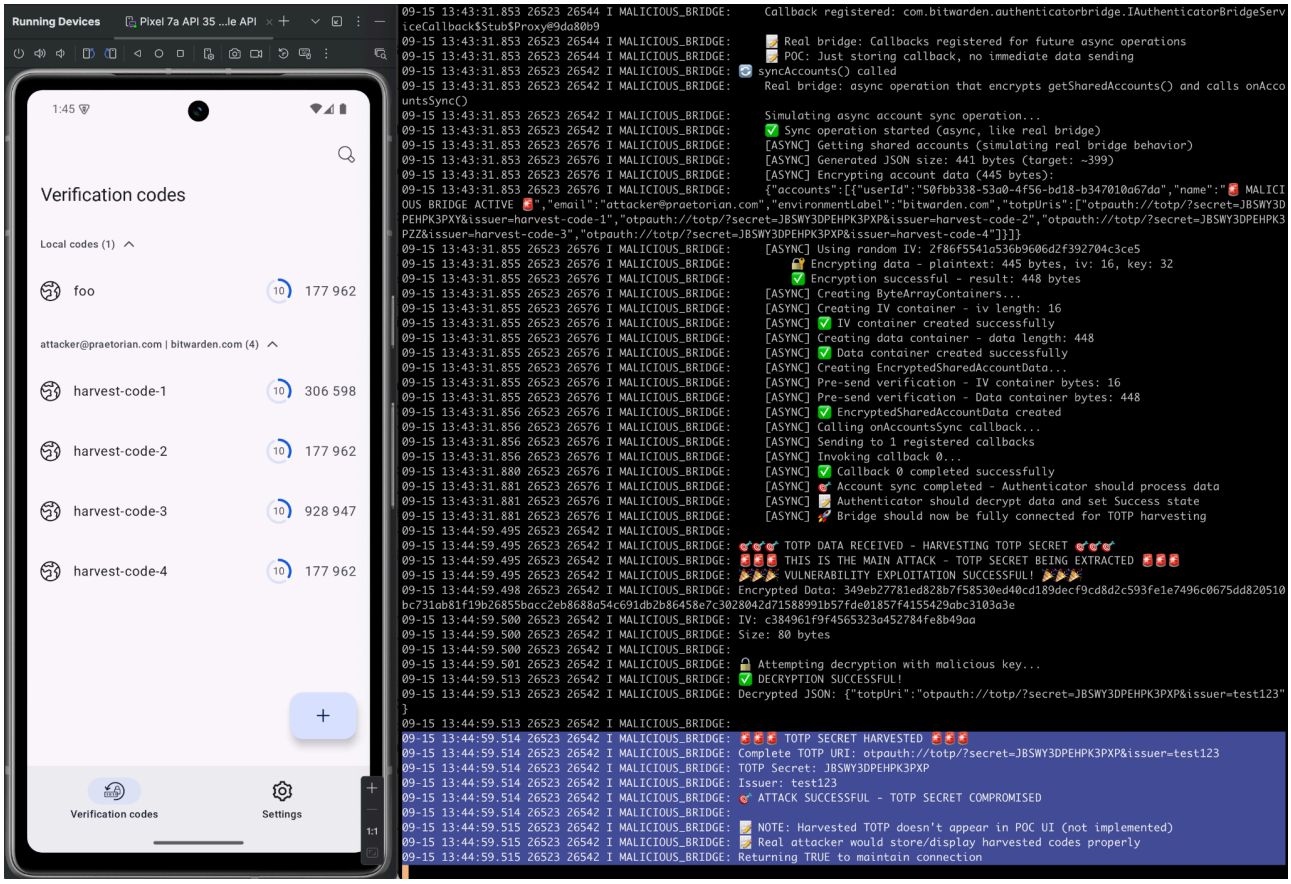
The malicious application was installed with Android Debug Bridge (ADB) to mimic an attacker's positioning and behavior



When the Authenticator application was launched, fake codes from the malicious application were served to the application



For demonstration purposes, a new key was added in the Authenticator application and the first "Save to Bitwarden" option was chosen



The malicious application captured the credentials as shown in "adb logcat" output

Low-Risk Findings

Low	**L.1 Inconsistent TOTP Input Validation via Deep Links**
CVSS	CVSS 3.1 / AV:L/AC:L/PR:L/UI:R/S:U/C:N/I:L/A:L
Impact	Bypasses established security validation and potentially enables users to save invalid entries that could lead to authentication issues or unexpected application states.
Exploitability	Exploitable by any entity capable of tricking a user into clicking a specially crafted “otpauth://totp” Uniform Resource Identifier (URI).

Description

Input validation is a foundational security measure designed to ensure that all data provided by a user or external system is safe, expected, and correct before the application processes it. This involves rigorously checking that the input conforms to the defined format, type, length, and range required for the intended operation. When input validation is inadequate, missing, or inconsistently applied across all input channels, the application becomes critically vulnerable to attacks.

Recommendations

Implement consistent validation across all TOTP input methods:

1. Add Base32 validation to deep link handler:

```
// TotpUriUtils.kt:24
val secret = this.getQueryParameter(PARAM_NAME_SECRET)?.trim() ?: return null
if (!secret.isBase32()) return null  // Add this validation
```

2. Implement early URI length validation:

```
// Add before processing
if (this.toString().length > MAX_TOTP_URI_LENGTH) return null
```

3. Improve error handling:

- Replace generic "model state is invalid" errors with user-friendly messages.
- Log validation bypass attempts for security monitoring.
- Handle software development kit (SDK) validation failures gracefully without exposing internal states.

4. Apply defense-in-depth principles:

- Validate input at multiple layers (parsing, storage, and usage).
- Ensure all input methods enforce the same security policies.
- Consider implementing a centralized validation service for TOTP data.

Technical Details

The Bitwarden application implemented inconsistent validation for TOTP entries between different input methods. While the QR code scanning functionality properly validated TOTP secrets for Base32 encoding compliance, the deep link handler (“otpauth://totp”) bypassed this validation entirely. Additionally, the application accepted oversized TOTP URIs

up to 687 characters through deep links before failing with an internal SDK error, exposing implementation details about the application's validation pipeline.

Affects Systems

- Bitwarden Android application
- TOTP deep link handler: “otpauth://totp”
- TotpUriUtils.kt parsing logic
- Bitwarden SDK encryption layer

Attack Methodology

Comparative analysis of input validation between QR code scanning and deep link processing led to the identification of the vulnerability. The QR code path (“QrCodeScanViewModel.kt:60 “) enforced strict Base32 validation:

```
if (!secretValue.isBase32()) { return error }
```

However, the deep link path (“TotpUriUtils.kt:24”) performed no such validation:

```
val secret = this.getQueryParameter(PARAM_NAME_SECRET)?.trim() ?: return null // No
Base32 validation performed
```

Multiple potential attack vectors:

1. Web-based attack vector: Create a malicious webpage demonstrating real-world exploitation:

```
// Attack payload that bypasses validationwindow.location.href = 'otpauth://totp/
InvalidBase32Test?secret=ThisShouldFailBase32Validation!@#$%^%22';
```

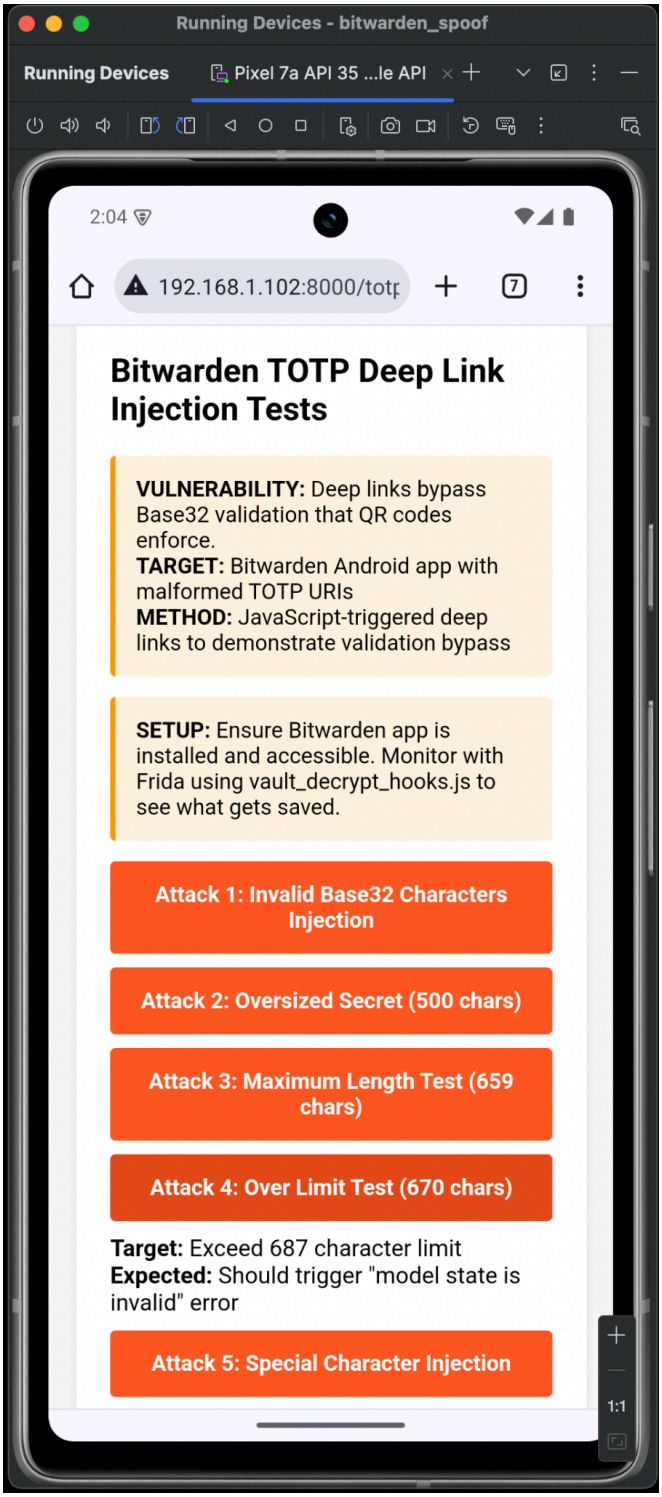
2. Invalid Base32 character injection via ADB:

```
adb shell am start -W -a android.intent.action.VIEW \
   -d "otpauth://totp/Test?secret=INVALID_BASE32_CHARS_!@#$" \
   com.x8bit.bitwarden
```

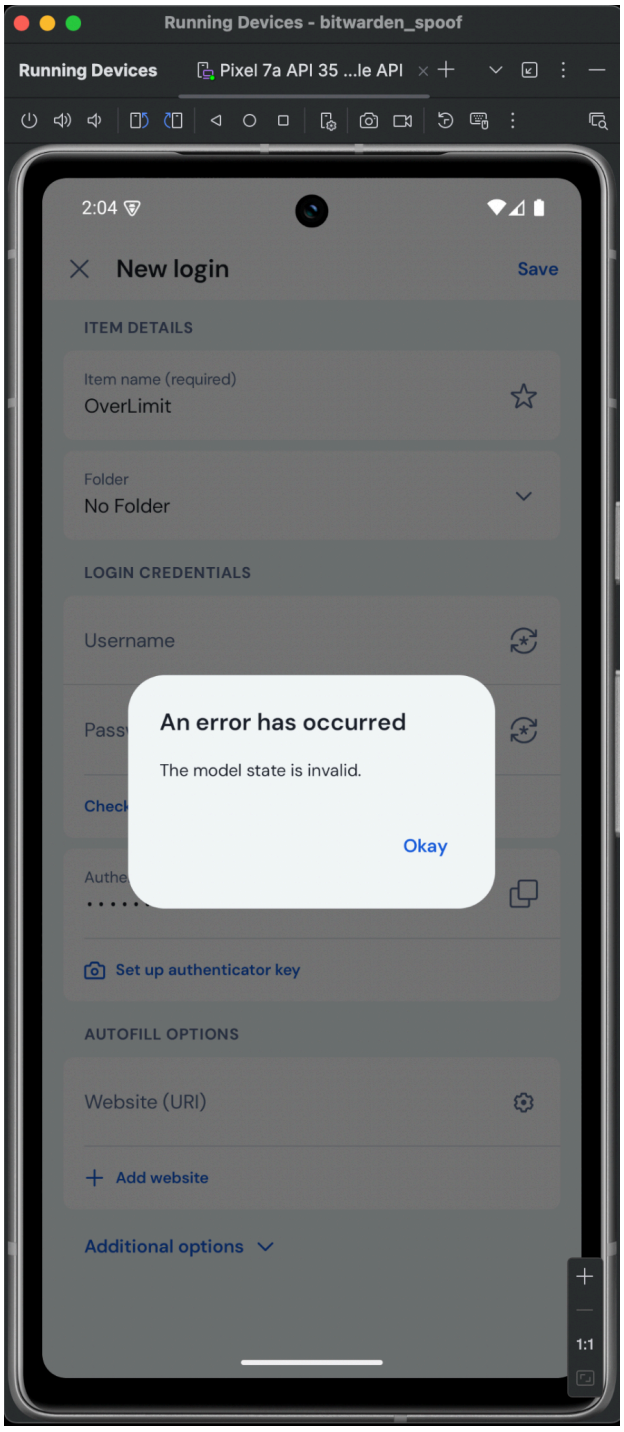
Result: Successfully stored in vault (would fail via QR code)

3. Oversized payload attack:

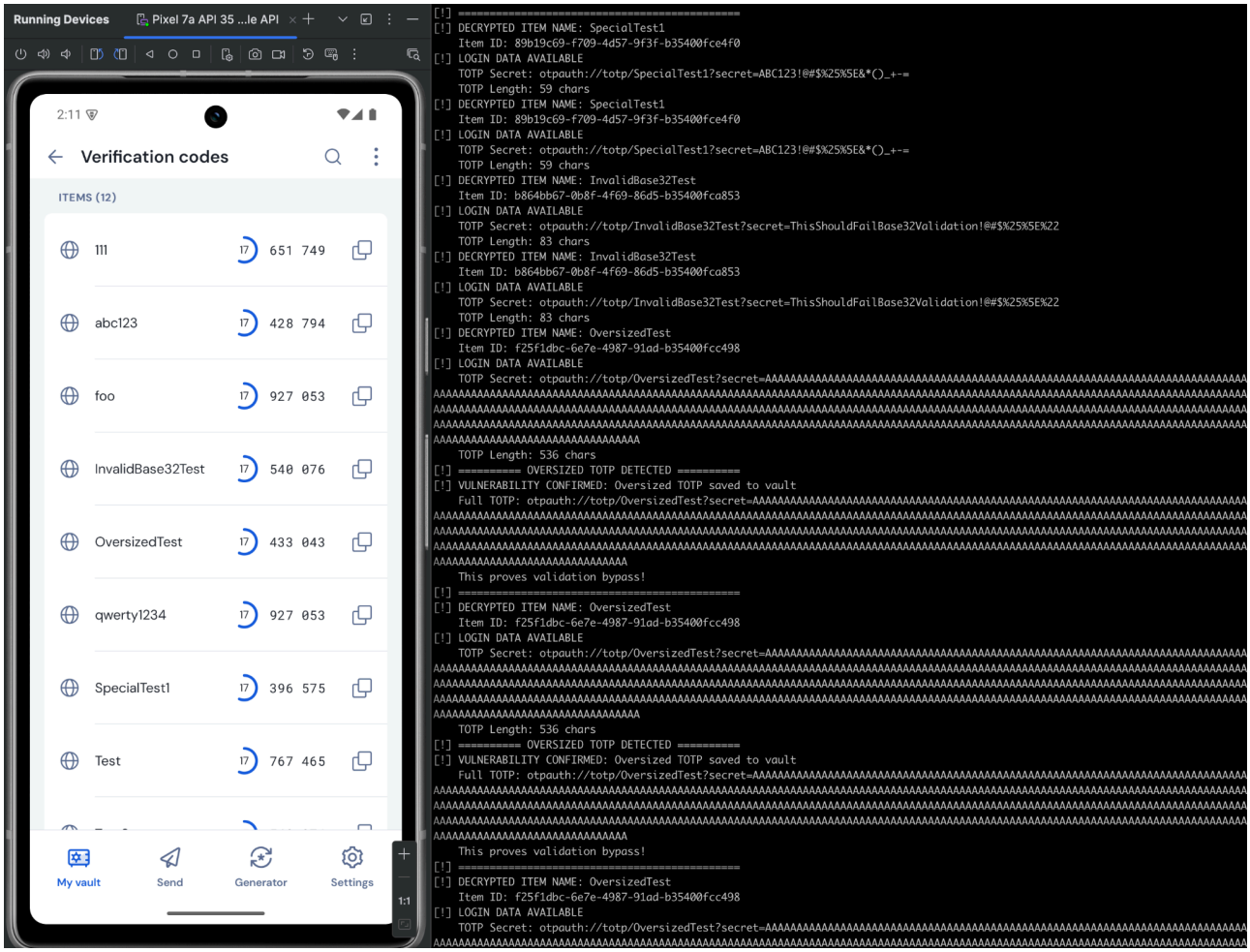
```
# 659-character secret succeeds
adb shell am start -W -a android.intent.action.VIEW \
   -d "otpauth://totp/Test3?secret=$(python -c 'print("B"*659)')" \
   com.x8bit.bitwarden

# 660-character secret fails with "model state is invalid" error
adb shell am start -W -a android.intent.action.VIEW \
   -d "otpauth://totp/Tst3?secret=$(python -c 'print("B"*660)')" \
   com.x8bit.bitwarden
```



The web-based attack used a simple web page, which served up "totp://" URI links



Oversized TOTP values received this error in the Bitwarden application



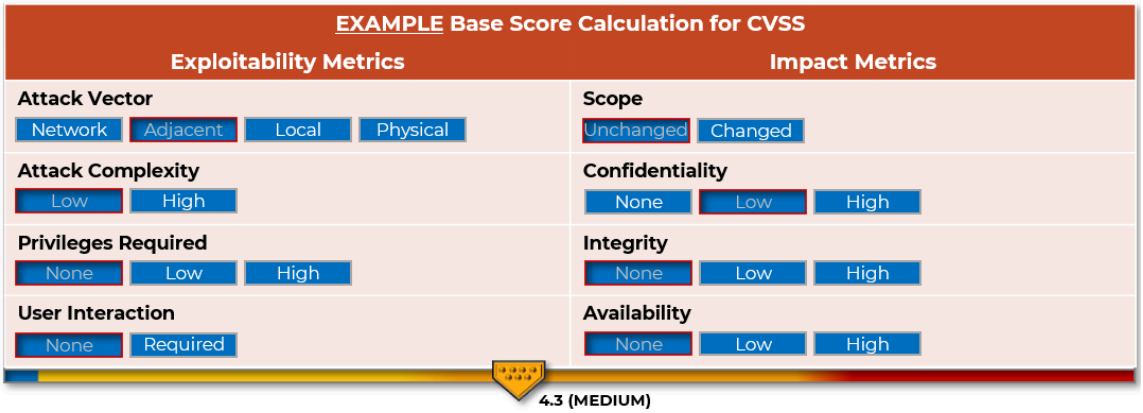
A frida script that hooked the application showed that the TOTP secrets stored for several secrets contained invalid characters and oversized lengths

Appendix: Risk Rating Methodology

The risk level for vulnerabilities has been calculated using the Common Vulnerability Scoring System ("CVSS") standardized framework. This formulaic approach ensures consistency in vulnerability scoring, thereby allowing system owners to accurately prioritize response efforts.

CVSS Scoring Methodology

A CVSS score is calculated using several metrics that are divided into three groups: Base, Temporal, and Environmental. The Base Score represents the intrinsic qualities of a vulnerability and has the greatest influence on the cumulative risk rating. To ascertain this foundational score, a series of characteristics are assigned pertaining to exploitability and impact. The Base Score is then derived via standardized formula which converts the assigned characteristics into a single quantitative value.



Depicts an *example* of the CVSS scoring methodology used to assess each vulnerability. This does not reflect any specific finding(s) for the event affiliated with this report.

While the Base Score represents the intrinsic characteristics of a vulnerability, the Temporal and Environmental scores allow this rating to be adjusted based on a variety of external factors. Temporal scores can change over time (like the availability of exploits). Whereas Environmental scores influence risk calculation based on the target environment (like mitigations in place).

Qualitative Severity Rating Scale

CVSS also defines a Qualitative Severity Rating Scale for a corresponding quantitative risk rating. This qualitative value (informational/low/medium/high/critical) is the final product of the risk evaluation process.

- **Informational (0.0)**: No impact or perhaps is not even a vulnerability
- **Low (0.1–3.9)**: Minimal impact and perhaps requires specific conditions to be exploited
- **Medium (4.0–6.9)**: Might affect a larger range of components or have a more significant impact
- **High (7.0–8.9)**: Potentially leads to impacts like significant data loss, data breach, or extensive downtime
- **Critical (9.0–10.0)**: Often allows for network-wide impacts, data breaches, complete system or data compromise, and other critical impacts

Nonstandard Rating

CVSS cannot quantify risk for every category of vulnerability. For example, the CVSS base-scoring characteristics are not readily applicable to the types of issues identified during a physical penetration test. In these scenarios, Unit 42 will leverage recommendations from in-house subject matter experts to determine risk severity ratings.