Security assessments for software that matters.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Pentest-Report ExpressVPN ExpressKeys Apps & Architecture 02.-03.2026

Cure53, Dr.-Ing. M. Heiderich, Dipl.-Ing. E. Damej, L. Herrera, MSc. S. Moritz, Dr. S. Mazaheri

Index

- Introduction
- Fix Verification Status
- Scope
- Severity Glossary
- Test Methodology
- Identified Vulnerabilities
  - EXP-22-001 WP1/2: Info leak and phishing via custom scheme hijacking (Medium)
  - EXP-22-006 WP1/2: Null domains return all creds as Autofill suggestions (Medium)
  - EXP-22-007 WP1/2: Empty domain in confirmation dialog via null domain (Low)
  - EXP-22-010 WP1/2: Storage encryption key leaked in stdout logs (Low)
  - EXP-22-011 WP1/2: Screenshot restriction bypass in login and card view (Low)
- Miscellaneous Issues
  - EXP-22-002 WP1/2: Unmaintained mobile OS version support (Info)
  - EXP-22-003 WP1/2: Lack of stack canaries protections for Android & iOS (Info)
  - EXP-22-004 WP1/2: Lack of FORTIFY_SOURCE for Android binaries (Info)
  - EXP-22-005 WP1/2: Potential leak of auth. token via connectivity check (Low)
  - EXP-22-008 WP1/2: Lack of second local factor for vault access (Info)
  - EXP-22-009 WP1/2: Lack of certificate pinning in network component (Info)
  - EXP-22-012 WP1/2: Key derivation is susceptible to brute-force attacks (Medium)
- Conclusions

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

“ExpressKeys is a secure password manager that stores your passwords, cards, and notes in an encrypted vault. It works on mobile devices and browsers, and helps you sign in quickly with autofill.”

From https://www.expressvpn.com/keys

This report describes the results of a penetration test and source code audit, as well as a design and cryptographic review conducted against the ExpressKeys architecture and applications.

To give some context regarding the assignment’s origination and composition, ExpressVPN contacted Cure53 in February 2026. The test execution was scheduled for late February / early March 2026, namely from CW09 - CW10. A total of sixteen days were invested to reach the coverage expected for this project, and a team of five senior testers was assigned to its preparation, execution, and finalization.

The methodology conformed to a white-box strategy, whereby assistive materials such as sources, mobile applications, as well as all further means of access required to complete the tests were provided to facilitate the undertakings.

The work was split into two separate work packages (WPs), defined as:

- **WP1**: Design & cryptographic trust-assumption review ag. ExpressKeys architecture
- **WP2**: Dynamic white-box pen.-tests & source code audits against ExpressKeys apps

All preparations were completed in February 2026, specifically during CW08, to ensure a smooth start for Cure53. Communication throughout the test was conducted through a dedicated and shared Slack channel, established to combine the teams of ExpressVPN and Cure53. All personnel involved from both parties were invited to participate in this channel. Communications were smooth, with few questions requiring clarification, and the scope was well-prepared and clear. No significant roadblocks were encountered during the test. Cure53 provided frequent status updates, shared its findings, and offered live reporting in the form of shared Markdown files.

The Cure53 team achieved good coverage over the scope items, and identified a total of twelve findings. Of the twelve security-related findings, five were classified as security vulnerabilities, and seven were categorized as general weaknesses with lower exploitation potential.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Despite identifying several opportunities for improvements, the ExpressKeys platform for Android and iOS left a positive overall impression on the testing team and demonstrated a mature security posture throughout the areas assessed. This is mainly due to the fact that none of the findings exceeded a *Medium* severity threshold, indicating that the applications are already well-strengthened against more severe threats and attacks. Nevertheless, Cure53 did identify a small number of technical gaps - which primarily involved information disclosure and system misconfigurations. It is advised that addressing these gaps will further strengthen the already good level of security.

The report will now shed more light on the scope and testing setup, and will provide a comprehensive breakdown of the available materials. Next, the report will detail the *Test Methodology* used in this exercise. This is intended to show the client which areas of the software in scope have been covered, and which tests have been executed. Following this, the report will list all findings identified in chronological order, starting with the *Identified Vulnerabilities* and followed by the *Miscellaneous Issues* unearthed. Each finding will be accompanied by a technical description, Proof-of-Concepts (PoCs) where applicable, plus any fix or preventative advice to action.

In summation, the report will finalize with a *Conclusions* chapter in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the ExpressKeys applications and architecture.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Fix Verification Status

Following the audit, the ExpressVPN team promptly addressed the identified findings, resolving a significant number of vulnerabilities and weaknesses. These fixes were then presented to Cure53 for formal verification. The current status of the fix verification process is detailed in the table below:

Finding	Status
EXP-22-001 WP1/2: Info leak and phishing via custom scheme hijacking (Medium)	Mitigated
EXP-22-002 WP1/2: Unmaintained mobile OS version support (Info)	Partly fixed
EXP-22-003 WP1/2: Lack of stack canaries protections for Android & iOS (Info)	Won’t fix
EXP-22-004 WP1/2: Lack of FORTIFY_SOURCE for Android binaries (Info)	Won’t fix
EXP-22-005 WP1/2: Potential leak of auth. token via connectivity check (Low)	Mitigated
EXP-22-006 WP1/2: Null domains return all creds as Autofill suggestions (Medium)	Fixed
EXP-22-007 WP1/2: Empty domain in confirmation dialog via null domain (Low)	Fixed
EXP-22-008 WP1/2: Lack of second local factor for vault access (Info)	Won’t fix
EXP-22-009 WP1/2: Lack of certificate pinning in network component (Info)	Mitigated
EXP-22-010 WP1/2: Storage encryption key leaked in stdout logs (Low)	Fixed
EXP-22-011 WP1/2: Screenshot restriction bypass in login and card view (Low)	WIP
EXP-22-012 WP1/2: Key derivation is susceptible to brute-force attacks (Medium)	Mitigated

Table: Fix verification status as of June 2026

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Design review & source code audit against ExpressKeys apps & architecture**
  - **WP1:** Design & cryptographic trust-assumption review ag. ExpressKeys architecture
    - **Source code:**
      - *password_manager_ui*
  - **WP2:** Dynamic white-box pen.-tests & source code audits against ExpressKeys apps
    - **Android app:**
      - Audited version:
        - 1.1.1 (262)
      - File:
        - *app-release.apk*
      - SHA256:
        - ffaf9f15495ab9789bd302cafdd14d00e4e3247d403b1aa315e9a6135533e4a3
    - **iOS app:**
      - Audited version:
        - 1.1.1 (350)
      - File:
        - *ExpressKeys_v1.1.1.350.ipa*
      - SHA256:
        - 33239aebe86851c6d354c4797f0020f4b667e37498ce5ab640f1e9ffd9ec87d0
    - **Source code:**
      - *password_manager_ui*
  - **Test User Credentials**
    - U: seba@cure53.de
    - U : elyas@cure53.de
    - U: herrera@volt.cure53.de
    - U: sogol.mazaheri@cure53.de
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Severity Glossary

The following section details the varying severity levels assigned to the issues discovered in this report.

Critical: The highest possible severity level. Denotes issues that allow attackers to achieve extensive access to sensitive areas, such as critical systems, applications, data, and other pertinent components in scope.

High: Denotes issues that allow attackers to achieve limited access to sensitive areas in scope. This also includes vulnerabilities with limited exploitability that can incur significant impact upon the target in scope.

Medium: Denotes issues that do not incur major impact on the areas in scope. Additionally, issues requiring limited exploitation are graded as *Medium*.

Low: Denotes issues that incur considerably limited impact on the areas in scope. These mostly do not depend on the degree of exploitation, but rather on the minor severity of retrievable information or low-grade risk upon the areas in scope.

Info: Denotes issues deemed merely informational in nature. They are mostly considered hardening recommendations or best-practice improvements that will generally enhance the security posture of the areas in scope.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Test Methodology

This section documents the testing methodology applied by Cure53 during this project and discusses the resulting coverage, shedding light on how various components were examined. Further clarification concerning areas of investigation subjected to deep-dive assessment is offered, as well as detailed information regarding the approaches and attacks performed against the audited applications and elements.

The primary objective of this assessment was to identify any security vulnerabilities and misconfigurations present in the ExpressKeys mobile applications, their features, OS-specific implementations, and architectural design. Source code was provided prior to the assessment, meaning that code-assisted methods could be used for testing.

Cure53's testing approach therefore combined static source code analysis with dynamic runtime analysis. Static analysis focused on identifying insecure coding patterns, improper error propagation, unsafe memory operations, and insufficient validation at API boundaries. Dynamic testing included runtime instrumentation and interaction testing to evaluate how the applications behaved under malformed inputs, unexpected state transitions, and manipulated inter-process communication.

At the beginning of the assessment, Cure53 conducted a comprehensive review of the existing system architecture and the technology stack on which the applications are built. The ExpressKeys applications have been developed using the Flutter framework. Application-specific components are implemented in Dart, while platform-specific components are implemented in Swift (iOS) and Kotlin (Android). The core password management functionality is implemented in Rust, and is integrated into the application via the Flutter Rust Bridge[1].

A key focus of the assessment was the identification and evaluation of vulnerabilities commonly associated with this type of multi-language, cross-platform architecture. Particular attention was given to potential security risks arising from the interaction between Flutter, native platform code, and the Rust-based core components. Interesting application flows were reviewed - such as retrieving the master password from the Keystore / Keychain through native implementations, and how this process is delegated to Flutter.

As a subsequent step, the attack surface of both the Android and iOS ExpressKeys applications was evaluated through a detailed review of configuration files and relevant source code components. The assessment focused on how each application integrates within its respective operating system ecosystem, and how communication with mobile platform APIs is implemented and secured.

[1] https://pub.dev/packages/flutter_rust_bridge

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The applications expose only a limited number of externally accessible components, and deliberately minimize the use of platform features that are commonly associated with elevated security risks. In particular, potentially vulnerable constructs - such as improperly exported Android components (e.g., activities, services, or content providers) or insecure media-sharing implementations - are avoided. Based on this analysis, it is advised that the mobile applications' overall attack surface can be considered to be comparatively limited and well-controlled.

An additional focus of the assessment was the examination of the authentication mechanism and the integration of Keycloak as the identity provider. The evaluation included testing for common vulnerabilities - such as open redirects, insufficient path validation, Cross-Site Request Forgery (CSRF), and account takeover scenarios. No such issues were identified. This can be attributed to a robust Keycloak configuration and strict validation mechanisms implemented within the corresponding components.

Additionally, the Multi-Factor Authentication (MFA) process, which is based on one-time passwords (OTPs) delivered to registered account email addresses, was thoroughly assessed. No weaknesses were identified in the tested MFA flow.

The client-side authentication flow relies on deep links registered via custom URL schemes to receive callback data (e.g., after successful authentication). During the assessment, a vulnerability was identified affecting all deep links registered under this custom scheme - including authentication callbacks. Such configurations may allow malicious installed applications to register the same scheme and intercept authentication data, as demonstrated in EXP-22-001.

The WebView configuration was reviewed for typical Cross-Site Scripting (XSS) / Universal Cross-Site Scripting (UXSS) vulnerabilities and open redirect issues that could potentially allow access to exported native bridges. No weaknesses were identified in this area. The WebView feature is used solely to perform authentication with the Keycloak service.

Intent handling and deep link processing were also further assessed in-depth to determine how the applications process external input and route actions internally. The use of URL schemes, app links, and universal links was checked and - if implemented - this was exercised with both expected and malformed data. Special attention was given to how parameters were parsed, validated, and used within the components. Tests included supplying unexpected data types, missing parameters, and crafted values, to observe error handling and logic behavior. The goal was to identify weaknesses that could enable injection, open redirect behavior, or unintended access to protected functionality. No additional issues were found in this area, which was due mainly to the limited usage of relevant features.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Deep link handling was also thoroughly examined with respect to potential CSRF attacks that could enable unauthorized requests to the API. No scenario was identified that would allow external parties to interact with the API in an authenticated manner.

Another strong focus was placed on testing the Autofill features where information disclosure and credential theft were concerned. The assessment covered the Autofill architecture on both Android and iOS, including the integration layers, credential retrieval logic, and the domain matching mechanisms used to determine which credentials are surfaced for a given context.

The evaluation examined how the Autofill system handles domain extraction and credential filtering, and how these credentials are presented to users in the relevant UI. Particular attention was given to the domain matching behavior, which relies on Public Suffix List (PSL) lookups, associated domain groups for cross-domain credential sharing, and subdomain comparison logic to determine match confidence.

A primary area of investigation involved the behavior of the Autofill service when the requesting context provides a null or empty domain - as occurs with *srcdoc* iframes and *data:* URIs, which lack an associated origin. This analysis revealed that the iOS credential provider returns all stored vault entries as suggestions when a null domain is encountered, presenting them under the "current domain" section, and thereby creating the false impression that every credential belongs to the requesting context (EXP-22-006). A related observation was made regarding the confirmation dialogs on both platforms, which render empty domain strings when the requesting context provides a null domain, weakening the security value of the mismatch warning (EXP-22-007). Additional Autofill testing included verification of protocol-aware matching behavior, where it was confirmed that credentials stored for *http://* domains are correctly excluded from *https://* contexts.

Storage handling was reviewed, to understand how the apps store, cache, and manage data on devices. Platform-specific storage locations and APIs were examined to determine whether sensitive information such as tokens or identifiers were stored securely. The assessment evaluated whether data was stored in plaintext or protected using platform-provided secure storage mechanisms, which yielded one information disclosure vulnerability involving the encryption key (see EXP-22-010). Data lifecycle aspects such as persistence, cleanup, and reuse were also observed, to determine whether residual data could be accessed or misused.

Communication between the applications and the backend services was analyzed to verify that secure transport practices were followed. Network traffic generated by the application components was observed to confirm the use of encrypted channels and ideal certificate validation; here Cure53 noted the omission of certificate pinning (EXP-22-009). The handling of authentication material and session data in transit was surveyed, with the team making controlled attempts to intercept or modify traffic in order to analyze the corresponding API requests and responses.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Cryptographic implementations were reviewed to pinpoint potential weaknesses in key management, encryption usage, and randomness. This task considered whether performant and up-to-date algorithms and appropriate parameters were in use, and investigated whether cryptographic operations depended on trusted platform libraries. The cryptographic key generation, storage, and protection constructs were also vetted. Here, Cure53 checked whether hardware-backed or secure storage mechanisms were applied where appropriate (e.g., Android Keystore, iOS Keychain, and Secure Enclave).

The encryption key derivation from the vault password and a salt is performed using PBKDF2 with SHA512 and a high number of iterations - defaulting to 210,000. Although the computational cost of PBKDF2 is high on a CPU, the algorithm remains susceptible to GPU-accelerated password-guessing attacks, as described in EXP-22-012. Beyond key derivation, the data transmission and storage lifecycle underwent a meticulous investigation to identify potential sensitive material leakage. Results confirmed that document bodies and metadata are correctly encrypted and authenticated. It was further verified that users' secret encryption and signing keys, as well as recovery codes, do not leave the client's device unencrypted, and those that are ultimately stored on the server are properly encrypted before transfer. Furthermore, the implementation employs padding to mask the length of sensitive data; this obfuscation prevents attackers from deducing password complexity or length, which are often exploited as side-channel information. The application of these mechanisms to metadata effectively hardens the overall protection of the sensitive data set.

Where identity management is concerned, long integer identifiers are utilized for login entities, cards, and notes. Access Control List (ACL) integrity was evaluated to determine whether cross-user unauthorized deletions or updates were possible. While the API was seen to return deceptive HTTP success messages for such malicious requests, the underlying data remained unaltered; no entities were appended, updated, or deleted. Finally, the authentication framework was tested for signature integrity. It was found that the system correctly identifies and rejects authentication tokens signed with private keys not attributed to the specific user, consistently resulting in a signature validation error. This confirms that the cryptographic binding between the user identity and the authentication token is maintained.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., EXP-22-001) to facilitate any future follow-up correspondence.

EXP-22-001 WP1/2: Info leak and phishing via custom scheme hijacking *(Medium)*

***Fix Note**: This issue has been mitigated by the ExpressVPN team and verified by Cure53 to be working as expected.*

CVSS Score: 6.4
CVSS Temporal Score: 6.1
CVSS String: CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:H/I:L/A:L/E:F/RL:W/RC:C
CWE: https://cwe.mitre.org/data/definitions/939.html

Testing confirmed that the Android and iOS apps deploy a custom URL scheme that is vulnerable to URL scheme hijacking.[2] URL scheme hijacking is an attack vector in which third-party apps attempt to register the same URL scheme that has already been registered by the ExpressKeys mobile apps.

Depending on current custom URL scheme usage, this could be leveraged to leak information contained within a URL (e.g., credentials and access tokens) or for phishing purposes, whereby a user enters sensitive information into the third-party app upon successful URL scheme interception.

On Android, the custom scheme is defined in the *AndroidManifest.xml* file, as observable below:

Affected file #1:
AndroidManifest.xml

Affected code #1:
```
<activity
  android:name="com.linusu.flutter_web_auth_2.CallbackActivity"
  android:exported="true">
  <intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:scheme="com.expressvpn.keys"/>
```

[2] https://people.cs.vt.edu/gangwang/deep17.pdf

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
  </intent-filter>
</activity>
```

The custom scheme of the iOS app is registered within the *Info.plist* file under the *CFBundleURLSchemes* key, as presented below:

Affected file #2:
Info.plist

Affected code #2:
```
<key>CFBundleURLTypes</key>
  <array>
    <dict>
      [...]
      <key>CFBundleURLSchemes</key>
      <array>
        <string>com.expressvpn.keys</string>
      </array>
    </dict>
  </array>
```

Notably, this issue remains exploitable on iOS despite Apple's implementation of the first-come-first-served principle in iOS 11[3], which provides partial mitigation.

When only the internal WebView is used for authentication, the OS-specific scheme parser is not reached, and thus the redirect will stay inside the app. However, if authentication is performed on a mobile browser outside of the ExpressKeys app, then on Android the redirect shows an app picker with apps that have the custom scheme registered.

The following PoC exemplifies a method of exploiting this behavior, whereby a malicious Android app can interact with the scheme, when registered, to gain access to sensitive authentication data such as *code* and *state*:

PoC *AndroidManifest.xml* excerpt:
```
[...]
<activity
  android:name=".MainActivity"
  android:exported="true"
  [...]
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
      <category android:name="android.intent.category.DEFAULT" />
      <category android:name="android.intent.category.BROWSABLE" />
      <data android:scheme="com.expressvpn.keys"/>
    </intent-filter>
```

[3] https://malware.news/t/ios-url-scheme-susceptible-to-hijacking/31266

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Steps to reproduce:

1. Create a new Android application and integrate the *intent-filter* example provided above.
2. Install the malicious Android application on your device.
3. Open your browser on the Android device, and then open the following page in your mobile browser:
   https://auth.expressvpn.com/realms/xvpn/protocol/openid-connect/auth?client_id=express-keys&redirect_uri=com.expressvpn.keys%3A%2F%2Fauth_callback&response_type=code&state=tJ_HDKRiZkoXlmai5ihWtkgx60srKloWf8G7GezYm7Q&scope=profile+email+openid
4. Continue with sign in and observe that after verifying the OTP, the dialog "Open with" is shown. An unaware user might choose the malicious installed app that has a similar looking icon and name.
5. When the malicious app is chosen, the data could be sent to an attacker-controlled server, or the user could be shown a phishing login form.

Response:

```
HTTP/2 302 Found
Date: Mon, 23 Feb 2026 13:11:39 GMT
Location: com.expressvpn.keys://auth_callback?
state=tJ_HDKRiZkoXlmai5ihWtkgx60srKloWf8G7GezYm7Q&session_state=1a1c7ffb-
dba2-4f0d-b879-9b43e6323be0&iss=https%3A%2F%2Fauth.expressvpn.com%2Frealms
%2Fxvpn&code=e99586e2-d0e5-4d72-8941-537c8ba60eb2.1a1c7ffb-dba2-4f0d-b879-
9b43e6323be0.f9e5e31c-6639-4f30-bcf7-b7cd48dc51de
[...]
```

To mitigate this issue, Cure53 recommends ceasing the use of the current custom URL scheme implementations, and that Android App Links[4] and iOS Universal Links[5] should instead be exclusively utilized for the registered actions. Further, it is advised that deep links should only be processed via the *https* scheme in tandem with previously-registered and clearly associated domains and routes.

[4] https://developer.android.com/training/app-links
[5] https://developer.apple.com/ios/universal-links/

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

EXP-22-006 WP1/2: Null domains return all creds as Autofill suggestions *(Medium)*

***Fix Note**: This issue has been fixed by the ExpressVPN team and verified by Cure53 to be working as expected. The described issue no longer exists.*

CVSS Score: 6.5
CVSS Temporal Score: 6.0
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:N/A:N/E:P/RL:W/RC:C
CWE: https://cwe.mitre.org/data/definitions/346.html

It was discovered that the ExpressKeys' iOS Autofill feature exposes all stored vault entries as suggestions when the requesting context provides a null or empty domain. The problem originates from insufficient domain validation in the credential filtering logic, which treats the absence of a domain as a wildcard match, rather than as a restrictive condition.

More precisely, when the Autofill service is triggered inside contexts that lack a domain - such as *srcdoc* iframes or *data:* URIs - iOS produces a *nil* domain at the parsing layer. The *AutoFillRootCoordinator* extracts a *nil* rootDomain from the service identifier, and this *nil* value propagates through *FlutterEngineManager.getAllLogins(forDomain: nil)* into the Dart layer.

Since the domain is *nil*, it causes the *getAllLogins* handler to invoke *_getCredentialsForQuery("")* with an empty string, which explicitly returns every login entry from the vault when the query is empty, bypassing domain-based filtering entirely.

The iOS *AutoFillLoginItemListViewModel* then receives all of these entries and presents them under *itemsForCurrentDomain*, creating the false impression that every credential in the vault belongs to the current context. This behavior is particularly problematic in scenarios where a legitimate page embeds third-party iframes. A malicious third-party iframe using *srcdoc* or *data:* URIs could present login fields that trigger the autofill service with a *null* domain, causing all vault credentials to be surfaced as suggestions.

Since the entries appear under the *current domain* section, the user is led to believe that the credentials are intended for the parent website, and may inadvertently autofill sensitive credentials for unrelated services into the attacker-controlled iframe.

Affected file #1:
password_manager_ui/expresskeys/lib/ios_extension_main.dart

Affected code #1:
```
Future<List<Map<String, String>>> _getCredentialsForQuery(String query)
async {
  try {
  final listManager = getIt<ListManager>();
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
  List<VaultEntry> selectedEntries = [];
  if (query.isEmpty) {
    final allLoginEntries = listManager.allEntries
    .where((entry) => entry.templateName == 'pm2::login');
    selectedEntries.addAll(allLoginEntries);
  } else {
    final filteredEntries = listManager.allEntries.where(
      (entry) => entry.templateName == 'pm2::login'
      ? _matchesUrl(entry, query)
      : false,
    );
    selectedEntries.addAll(filteredEntries);
  }
[...]
```

Affected file #2:
password_manager_ui/expresskeys/ios/Password Credentials Provider/Coordinators/AutoFillRootCoordinator.swift

Affected code #2:

```
func makeListVaultScreen() async -> AutoFillLoginItemListView {
  let serviceIdentifier = self.serviceIdentifiers.first?.identifier ?? ""
  // Extract root domain from service identifier for pre-filtering
  // This significantly improves performance by filtering items in Flutter
before serialization
  let domain = URL(string: serviceIdentifier)?.rootDomain
  // Get pre-filtered items for current domain
  let currentDomainItems = await
flutterEngineManager.getAllLogins(forDomain: domain)
  // ViewModel will load ALL items in background for "Other Logins" section
  let vaultListViewModel = AutoFillLoginItemListViewModel(
    itemsForCurrentDomain: currentDomainItems,
    serviceIdentifier: serviceIdentifier
  )
  vaultListViewModel.delegate = self

  let listView = AutoFillLoginItemListView(viewModel: vaultListViewModel)
  return listView
}
```

Steps to reproduce:

1. Create an HTML page containing an iframe with a *srcdoc* attribute that includes a login form with username and password fields. Host this page on any domain, or load it locally.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
<iframe srcdoc="<form><input type='text' name='username'
autocomplete='username'><input type='password' name='password'
autocomplete='current-password'></form>"></iframe>
```

2. Open this page in a browser in iOS where ExpressKeys is configured as the autofill provider and the vault contains credentials for multiple unrelated domains.
3. Tap the username or password field inside the *srcdoc* iframe.
4. Tap the key icon and select *ExpressKeys*. After unlocking the vault, observe that all stored credentials appear under the *current domain* section as if they all belong to the *srcdoc* context. Every credential in the vault is presented as a valid match.

To mitigate this issue, Cure53 recommends that *_getCredentialsForQuery()* in *ios_extension_main.dart* should return an empty list when the query is empty, rather than returning the entire vault. When the domain is *nil*, it is advised to display a *No logins for this domain* message, or to present all items explicitly under the *Other* section.

EXP-22-007 WP1/2: Empty domain in confirmation dialog via null domain *(Low)*

***Fix Note**: This issue has been fixed by the ExpressVPN team and verified by Cure53 to be working as expected. The described issue no longer exists.*

CVSS Score: 3.1
CVSS Temporal Score: 3.1
CVSS String: CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:L/A:N
CWE: https://cwe.mitre.org/data/definitions/451.html

The observation was made that the ExpressKeys Autofill confirmation dialogs on both Android and iOS display an empty domain text when the requesting context provides a null domain. These confirmation dialogs are meant to warn users before filling credentials on a mismatched domain, but when the domain is null, the warning messages become incomplete and fail to convey meaningful information about the target context.

When Autofill is triggered inside contexts that lack a domain - such as *srcdoc* iframes or data: URIs - the domain resolves to null on both platforms. The confirmation dialog message templates interpolate this value directly, producing messages such as: *You're attempting to fill in your Credential login details on a different website domain – .*

While the confirmation dialogs still appear and require explicit user interaction to proceed, the empty domain text weakens the security value of the warning. Users cannot make an informed decision about whether to proceed when the dialog does not identify the target context.

Affected file #1:
password_manager_ui/expresskeys/ios/Password Credentials Provider/Coordinators/AutoFillRootCoordinator.swift

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected code #1:

```
if let currentServiceIdentifierURL = URL(string:
self.serviceIdentifiers.first?.identifier ?? "") {
  let alertMessageLocalized =
  hasDomain
  ? NSLocalizedString(
    "pwm.autofill_different_domain_alert.message",
    comment: "You're attempting to fill in your %@ login details on a
different     website domain - %@."
    )
  : NSLocalizedString(
    "pwm.autofill_no_domain_alert.message",
    comment: "You're attempting to fill in your %@ login details on %@."
    )
  alertMessageFormatted = String(format: alertMessageLocalized, item.title,
currentServiceIdentifierURL.host ?? "")
  [...]
  let serviceIdentifierURL = URL(string:
self.serviceIdentifiers.first?.identifier ?? "")
  let serviceIdentifierRootDomain = serviceIdentifierURL?.rootDomain ?? ""
  let serviceIdentifierSubdomain = serviceIdentifierURL?.subdomain ?? ""
  let alertMessageFormatted = String(
    format: alertMessageLocalized,
    itemWebDomain,
    "\(serviceIdentifierSubdomain).\(serviceIdentifierRootDomain)"
  )
```

Affected file #2:
password_manager_ui/expresskeys/android/app/src/main/kotlin/com/expressvpn/keys/autofill/ui/SubdomainConfirmationActivity.kt

Affected code #2:

```
private fun showConfirmationDialog(
  loginTitle: String?,
  targetDomain: String?,
  dataset: Dataset
) {
  val dialogView =
LayoutInflater.from(this).inflate(R.layout.dialog_subdomain_confirmation,
null)

  // Set message text using string resource with placeholders
  val messageView = dialogView.findViewById<TextView>(R.id.dialog_message)
  messageView.text = getString(R.string.subdomain_dialog_message,
loginTitle, targetDomain)
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Steps to reproduce:

1. Create an HTML page containing an iframe with a *srcdoc* attribute that includes a login form with username and password fields. Host this page on any domain, or load it locally.

   ```
   <iframe srcdoc="<form><input type='text' name='username'
   autocomplete='username'><input type='password' name='password'
   autocomplete='current-password'></form>"></iframe>
   ```

2. Open this page in a browser in Android or iOS where ExpressKeys is configured as the Autofill provider and the vault contains credentials for multiple unrelated domains.
3. Tap on the username or password field inside the *srcdoc* iframe.
4. Tap the key icon and select *ExpressKeys*. After unlocking the vault, select a credential in the *Other logins* section.
5. A warning dialog will appear with an empty hostname.

To mitigate this issue, Cure53 recommends that the confirmation dialog logic on all platforms should detect null domain values and should substitute a clear fallback message - such as *unknown domain* - instead of rendering an empty string.

EXP-22-010 WP1/2: Storage encryption key leaked in *stdout* logs *(Low)*

***Fix Note**: This issue has been fixed by the ExpressVPN team and verified by Cure53 to be working as expected. The described issue no longer exists.*

CVSS Score: 2.8
CVSS Temporal Score: 2.6
CVSS String: CVSS:3.1/AV:P/AC:L/PR:H/UI:R/S:U/C:L/I:L/A:N/E:P/RL:W/RC:C
CWE: https://cwe.mitre.org/data/definitions/209.html

While analyzing the vault Encrypted Storage, it was found that the encryption key for the *fjall* database is logged to the Android application's *stdout*. Technical analysis revealed that because *tracing_subscriber::fmt()* writes directly to *stdout*, these logs are not captured by the standard Android logging system. Consequently, the vulnerability is not exposed to standard log-reading applications.

However, an attacker with rooted privileges could intercept this output at the process level and potentially recover sensitive information contained within it. By utilizing Frida hooks to monitor the write system call or the *tracing_subscriber* output, an adversary could extract the plaintext encryption key directly from the memory or the standard output stream.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The current impact is mitigated by the specific logging destination and the prerequisite of root access for exploitation. Nevertheless, the risk would escalate significantly if *tracing-android*[6] or a custom *MakeWriter* was implemented to redirect logs to the system-wide Android logging facility.

Such an architectural change would expose the encryption keys to any process or user with basic log-reading permissions, removing the requirement for root access.

Affected file:
password_manager_ui/expresskeys/rust/src/api/password_manager.rs

Affected code:

```
impl PasswordManager {
  #[tracing::instrument]
  pub async fn create(
    base_url: String,
    env: Environment,
    storage_folder: String,
    storage_encryption_key: Vec<u8>,
    device_model: Option<String>,
  ) -> Result<Self, PwmError> {
    [...]
    let internal_state = async {
      let storage = storage.clone();
      let internal_state = match cached_api_secret {
        [...]
      }
      .instrument(info_span!("initialize_password_manager"))
      .await?;
```

Frida interception script (*hook-logs.js*):

```
setImmediate(function() {
  const writePtr = Module.getGlobalExportByName('write');
  console.log("write ptr: " + writePtr);

  Interceptor.attach(writePtr, {
    onEnter: function(args) {
      const fd = args[0].toInt32();
      if (fd === 1 || fd === 2) { // Filtering for stdout (1) and stderr
(2)
        try {
          const len = args[2].toInt32();
          const buf = args[1].readUtf8String(len);
          if (buf) console.log(buf);
        } catch(e) {}
      }
```

[6] https://docs.rs/tracing-android/latest/tracing_android/

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
    }
  });
  console.log("Hook installed.");
});
```

Execution command:

```
frida -U -p $(adb shell 'ps -e | grep com.expressvpn.keys | grep -v zyg | grep -v snif' | awk '{print $2}') -l hook-logs.js
```

Exposed log entry:

```
2026-02-27T14:22:30.289602Z  INFO create:initialize_password_manager:make_request:get_or_refresh_tokens:read_api_secret:read: rust_lib_expresskeys::api::private::encrypted_storage: enter base_url="https://www.expressapisv2.net" env=Production storage_folder="/data/user/0/com.expressvpn.keys/app_flutter/ExpressKeys" storage_encryption_key=[44, 71, 205, 59, 126, 7, 97, 142, 65, 250, 227, 216, 56, 246, 38, 212, 161, 168, 165, 104, 90, 4, 27, 238, 140, 56, 196, 142, 196, 131, 54, 53] device_model=None kind=Head sanitized_path="/v1/user" env=Production installation_id="9ea242b215f3daf25ef815a153ca456e9fe6e1ad816acd113f977952f5009358" path="api_secret"
```

To mitigate this vulnerability, it is strongly recommended to ensure that sensitive information - specifically encryption keys and authentication tokens - is never included in diagnostic logs. The presence of such data in logs simplifies the exploitation process and broadens the application's attack surface. It is advised to utilize the skip attribute within the *tracing::instrument* macro to exclude sensitive parameters from the generated spans.

EXP-22-011 WP1/2: Screenshot restriction bypass in login and card view *(Low)*

***Note from ExpressVPN**: This finding is a known consequence of our temporary use of Instabug in early production releases to capture diagnostics and improve application stability. We are removing Instabug from production builds and limiting its use to beta environments, with Sentry serving as the long-term replacement. Given the limited impact and transitional nature of this implementation, we have accepted this low risk as a temporary trade-off until the migration is complete.*

CVSS Score: 3.1
CVSS Temporal Score: 2.9
CVSS String: CVSS:3.1/AV:P/AC:L/PR:L/UI:R/S:U/C:L/I:L/A:N/E:P/RL:W/RC:C
CWE: https://cwe.mitre.org/data/definitions/209.html

While analyzing the screenshot prevention controls implemented in the ExpressKeys Android application, it was found that the restriction can be bypassed through the native Android shake-to-report bug reporting mechanism, allowing an attacker to capture sensitive screen content - including credentials, session tokens, and other confidential data displayed within the app.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The application correctly applies *FLAG_SECURE* to its Activity windows to prevent direct screenshot capture through standard Android APIs. However, the bug reporting flow - which is triggered by physically shaking the device - is not subject to the same restriction.

When the shake gesture is detected, the Android system (or an integrated bug reporting SDK) automatically captures a screenshot of the currently-visible screen and pre-attaches it to a partially pre-filled bug report form. This screenshot is taken at the OS or SDK level, bypassing the application-level *FLAG_SECURE* flag entirely.

Affected file:
password_manager_ui/expresskeys/lib/app/startup/luciq_setup.dart

Affected code:

```
await Luciq.init(
  token: AppConfig.luciqToken,
  invocationEvents: [InvocationEvent.shake, InvocationEvent.screenshot],
  debugLogsLevel: LogLevel.none,
);
```

Steps to reproduce:

1. Open the ExpressKeys app and navigate to a screen displaying sensitive data (e.g., credentials or password vault).
2. Physically shake the device until the bug report overlay appears.
3. Observe that a screenshot of the sensitive screen is automatically attached to the report form.
4. Tap the attached image to open it in the preview viewer.
5. Take a screenshot of the image preview. Note that the capture succeeds, despite *FLAG_SECURE* being set on the main app Activity.

In order to fix this issue, it is recommended to either disable the shake-to-report gesture entirely while the application is in the foreground - by unregistering any shake listeners or SDK hooks during sensitive Activity lifecycles (*onResume* / *onPause*) - or to apply *FLAG_SECURE* to any Activity or window surface spawned by the bug reporting SDK, ensuring that screenshots captured within the report flow are also protected.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

EXP-22-002 WP1/2: Unmaintained mobile OS version support *(Info)*

***Note from ExpressVPN**: This issue has been partially mitigated in line with Cure53's recommendations. We now require iOS 18 and Android 13 as the minimum supported operating systems. We plan to require Android 14 in a future release, after users on older Android versions have had the opportunity to benefit from several key upcoming features before support for Android 13 is retired.*

CVSS Score: 0.0
CVSS Temporal Score: 0.0
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N
CWE: https://cwe.mitre.org/data/definitions/1104.html

While analyzing the Android configuration, the discovery was made that the application supports Android 11.0 (API level 30) and upward. The current version support can incur security implications for the app, due to the fact that Android 12.1 (API level 32) received its final security updates in March 2025[7], and is no longer actively maintained. This deployment therefore expands the potential attack surface, via the outdated environment in which the application operates.

Affected file #1:
AndroidManifest.xml

Affected code #1:
```
<uses-sdk android:minSdkVersion="30" android:targetSdkVersion="36"/>
```

Similarly, the iOS app is currently configured to support a minimum iOS version of 15. However, iOS 15 reached end-of-life status in March 2025[8] and no longer receives security updates. The Apple AppStore statistics[9] for iOS indicate that the majority of users operate on iOS 26 (~66%) and iOS 18 (24%) as of February 2026. With this in mind, it is advised that support for iOS 15 and below is only necessary in edge-case scenarios, where legacy software cannot be avoided for compatibility reasons.

[7] https://endoflife.date/android
[8] https://endoflife.date/ios
[9] https://developer.apple.com/support/app-store/

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected file #2:
Info.plist

Affected code #2:
```
<key>MinimumOSVersion</key>
<string>15.0</string>
```

To mitigate this issue, Cure53 suggests raising the *minSDK* and *MinimumOSVersion* in order to ensure that the applications only operate on Android and iOS versions that receive regular security updates and active maintenance.

Although this method introduces a trade-off between device compatibility and security, it is advised that the team should upgrade the level to the oldest versions that still receive security updates for each platform. This revised approach will constrain the application's potential attack surface with regard to known vulnerabilities.

EXP-22-003 WP1/2: Lack of stack canaries protections for Android & iOS *(Info)*

***Note from ExpressVPN**: Given the minimal practical impact of the identified libraries, we have accepted this risk. Most of the affected components are upstream libraries maintained by Flutter, Google, or other third-party projects, and their build configurations are outside ExpressKeys' direct control. Forking and maintaining custom versions solely to enable stack canaries would introduce significant maintenance overhead, increase divergence from upstream, and delay the adoption of future security patches and feature updates. Additionally, the identified Rust library is written in a memory-safe language, so no stack protections are required.*

CVSS Score: 0.0
CVSS Temporal Score: 0.0
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N
CWE: https://cwe.mitre.org/data/definitions/121.html

It was found that several bundled third-party binaries and frameworks within the Android and iOS applications were compiled without stack canary protection. Stack canaries are a compiler-based mitigation that help to detect and prevent stack-based buffer overflow exploitation, by placing a guard value before the return address on the stack. The absence of this protection weakens the overall exploitation resistance of the application, and may allow attackers to more reliably exploit memory corruption vulnerabilities in affected native components.

The following list of libraries were found to be lacking this protection:

Affected Android binary:
lib//librust_lib_expresskeys.so*

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected iOS packages:

- *connectivity_plus.framework/connectivity_plus*
- *root_jailbreak_sniffer.framework/root_jailbreak_sniffer*
- *screen_protector.framework/screen_protector*
- *flutter_fgbg.framework/flutter_fgbg*
- *Punycode.framework/Punycode*
- *path_provider_foundation.framework/path_provider_foundation*
- *flutter_app_group_directory.framework/flutter_app_group_directory*
- *url_launcher_ios.framework/url_launcher_ios*

To mitigate this issue, it is recommended to enable stack canaries for all included native components on both platforms, by enforcing the compiler flag *-fstack-protector-all* at build time where possible.

EXP-22-004 WP1/2: Lack of *FORTIFY_SOURCE* for Android binaries (*Info*)

***Note from ExpressVPN**: The risks posed by the identified libraries are negligible, so this risk has been accepted. Most of the affected components are upstream libraries maintained by Flutter, Google, or other third-party projects, and their build configurations are outside ExpressKeys' direct control. Forking and maintaining custom versions solely to enable stack canaries would introduce significant maintenance overhead, increase divergence from upstream, and delay the adoption of future security patches and feature updates. Additionally, the identified Dart library is a known false positive, per the Flutter documentation.*

CVSS Score: 0.0
CVSS Temporal Score: 0.0
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N
CWE: https://cwe.mitre.org/data/definitions/119.html

Continued investigation of the Android app revealed that some of the included shared objects are not compiled using the *FORTIFY_SOURCE*[10] compiler option. Omitting this flag means that the *libc* functions will in turn lack buffer overflow checks - increasing the application's susceptibility to memory attacks.

The following list enumerates the binaries deemed to be affected in the decompiled Android application:

Affected binaries:

- *lib/*/librust_lib_expresskeys.so*
- *lib/*/libapp.so*
- *lib/*/libtoolChecker.so*
- *lib/*/librjsniffer-lib.so*

[10] https://man7.org/linux/man-pages/man7/feature_test_macros.7.html

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

- *lib/*/libbarhopper_v3.so*
- *lib/*/libdatastore_shared_counter.so*
- *lib/*/libsurface_util_jni.so*

Cure53 recommends to consider compiling the referred shared objects with *FORTIFY_SOURCE* enabled where possible. This should be accomplished by using the compiler option *-D_FORTIFY_SOURCE=2*[11].

EXP-22-005 WP1/2: Potential leak of auth. token via connectivity check *(Low)*

***Fix Note**: This issue has been mitigated by the ExpressVPN team and verified by Cure53 to be working as expected.*

CVSS Score: 2.9
CVSS Temporal Score: 2.8
CVSS String: CVSS:3.1/AV:L/AC:H/PR:H/UI:R/S:U/C:L/I:L/A:N/E:P/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/200.html

While analyzing the application's network flow, it was found that HTTP HEAD requests sent to the API endpoint */passmgr/v1/user* are used to check for connectivity. The issue stems from the transmission of the *x-auth-token* header prior to the user unlocking the application. In the absence of certificate pinning, a Manipulator-in-the-Middle (MitM) attack, facilitated by physical access to the device, could enable the extraction of authentication tokens associated with ExpressVPN key APIs.

To reproduce the issue, the application must be launched on an unlocked device. On rooted devices, the header can be intercepted through SSL splitting techniques, using tools such as *sslsplit*[12], provided that a custom Certificate Authority (CA) is installed on the Android system. Given that these conditions do not represent the standard user environment, and that exploitation requires high privileges, this finding was added to the *Miscellaneous Issues* section.

Script to Install CA Certificate on rooted device:

```
CERT_DER="$1"
[ -z "$CERT_DER" ] && exit 1
[ ! -f "$CERT_DER" ] && exit 1

openssl x509 -inform DER -in "$CERT_DER" -out burp.pem
HASH=$(openssl x509 -inform PEM -subject_hash_old -in burp.pem | head -1)
CERT_NAME="${HASH}.0"
mv burp.pem "$CERT_NAME"

adb shell avbctl disable-verification || true
```

[11] https://github.com/hashbang/hardening#source-fortification
[12] https://github.com/droe/sslsplit

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
adb disable-verity || true
adb root
sleep 2
adb remount
adb shell "su 0 mount -o rw,remount /system" || true
adb push "$CERT_NAME" /system/etc/security/cacerts/
adb shell "chmod 644 /system/etc/security/cacerts/$CERT_NAME"
adb reboot
```

Script to Implement SSL Splitting on a Linux Router:

```
iptables -t nat -A PREROUTING -p tcp --dport 443 -j REDIRECT --to-ports
8443
sslsplit -D -l connections.log  -S logdir/ -k ca.key -c ca.crt ssl 0.0.0.0
8443 sni 443
```

Affected captured HTTP request:

```
HEAD /passmgr/v1/user HTTP/1.1
x-versions: expressvpn/0.1.0
(e9c3564288a9bca60b7f3b67db4d160d9a1db4eea69b7bb6bfabad6ec273c1a3)
pmcore/6879 (6dc8e99ce830; master)
x-infrastructure-id: a45d01d5-4abf-4fe6-a2c1-12c1b509f9d1
x-auth-token: [REDACTED]
accept: */*
Accept-Encoding: gzip, deflate, br
host: www.expressapisv2.net
Connection: keep-alive
```

To mitigate this issue, it is highly recommended to modify the connectivity check so that it is performed anonymously. Transmission of the authentication token should be avoided during this initial request. Since the current API endpoint returns a *401 Unauthorized* response when the token is absent, the development of a custom, dedicated endpoint for connectivity verification is also advised, to ensure security without compromising functionality.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

EXP-22-008 WP1/2: Lack of second local factor for vault access *(Info)*

***Note from ExpressVPN**: Exploitation of this scenario would require bypassing the device's biometric protections, which is highly unlikely and would require a number of additional conditions to be met. Addressing this finding would also introduce significant friction to the user experience. We therefore believe the current implementation strikes an appropriate balance between security and user experience.*

CVSS Score: 0.0
CVSS Temporal Score: 0.0
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N
CWE: https://cwe.mitre.org/data/definitions/306.html

When a user signs in to the ExpressKeys app with email and password, a master password is required to decrypt the vault data. For convenience, this master password is stored in the platform's secure storage (Keystore / Keychain) when biometric authentication (e.g., fingerprint or face ID) is enabled. Once the app is locked, only biometric authentication is required to unlock vault contents.

While this approach improves usability, it may weaken security in scenarios where an attacker is able to bypass biometric protections. In such cases, vault items could potentially be accessed without re-entering the account password.

To harden the app against local attack scenarios, Cure53 recommends introducing an additional local authentication factor. Specifically, it is advised that users should be able to enable an option in the ExpressKeys settings that enforces re-entry of the account password after a configurable period of time.

With this feature enabled, biometric authentication alone would no longer be sufficient, and users would also be required to re-enter their account password before accessing vault items. This would add a time-based step-up authentication layer, and would significantly reduce the risk associated with biometric bypass or device compromise.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

EXP-22-009 WP1/2: Lack of certificate pinning in network component *(Info)*

***Fix Note**: This issue has been mitigated by the ExpressVPN team and verified by Cure53 to be working as expected.*

CVSS Score: 0.0
CVSS Temporal Score: 0.0
CVSS String: CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N
CWE: https://cwe.mitre.org/data/definitions/295.html

Cure53 noted that the network components of the ExpressKeys mobile applications lack certificate pinning. Accordingly, the apps rely solely on the default platform trust store when establishing TLS connections, and fail to validate server certificates against a pinned certificate or public key.

A network adversary in a privileged position could perform an MitM attack using a compromised or maliciously-trusted certificate, potentially allowing the interception or manipulation of ExpressKeys network traffic. While this breach requires specific conditions and capabilities, the lack of pinning ultimately hinders the encompassing security barrier, and it is advised that it should be rectified.

To mitigate this issue, Cure53 recommends integrating certificate pinning for all API endpoints, by validating server certificates or public keys against a predefined set within the applications. Although certificate pinning can be bypassed by skilled adversaries, the effort and expertise required for successful exploitation will be significantly elevated, thereby improving the system's overall defensive stance.

EXP-22-012 WP1/2: Key derivation is susceptible to brute-force attacks *(Medium)*

***Fix Note**: This issue has been mitigated by the ExpressVPN team and verified by Cure53 to be working as expected.*

CVSS Score: 6.0
CVSS Temporal Score: 5.8
CVSS String: CVSS:3.1/AV:L/AC:L/PR:H/UI:N/S:U/C:H/I:H/A:N/E:H/RL:U/RC:R
CWE: https://cwe.mitre.org/data/definitions/327.html

Auditing of the ExpressKeys key derivation mechanism revealed that the utilized primitive is susceptible to offline brute-force attacks using GPUs or application-specific integrated circuits.

A user's vault is protected by a vault password. More precisely, the vault is encrypted using a random symmetric document key, which is encrypted using a random symmetric user root key.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The user root key is further encrypted with a symmetric key derived from the user's vault password and a salt. The key derivation function used for this purpose is PBKDF2, which with a sufficient number of iterations (currently 210,000 with SHA512) takes a large amount of time to compute on a CPU. However, GPUs are designed to perform a large amount of low-memory computations in parallel. Hence, an attacker leveraging a GPU could easily brute-force passwords given an encrypted user root key, which may have been obtained from the server or the user's computer.

Affected file:
password_manager_ui/expresskeys/rust/src/crypto.rs

Affected code:

```
async fn generate_key_from_password(
  master_password: &str,
  key_salt: &MasterKeySalt,
  algorithm: &KeyDerivationAlgorithm,
) -> Result<AesKey> {
  // Reject any algorithm we don't understand
  let iterations = match algorithm {
    KeyDerivationAlgorithm::PBKDF2(iterations) => *iterations,
    KeyDerivationAlgorithm::Unknown(a) => {
      return Err(Error::IncompatibleUserRecord(format!(
        "Unknown KeyDerivationAlgorithm: {a}"
      )))
    }
  };
  [...]
  // Now derive the key using our selected key hashing algorithm &
parameters (we only support PBKDF2 for now)
  let derived_key =
derive_key_using_pbkdf2_hmac_sha512(master_password.as_bytes(), key_salt,
iterations)
  .await?;
  [...]
  Ok(derived_key)
}
```

It is advised that this issue can be mitigated by replacing PBKDF2 with a memory-hard key derivation function - as recommended by NIST[13]. Memory-hard functions require a large memory to compute, and are hard to break even with GPUs and application-specific integrated circuits. Cure53 suggests using Argon2id[14] or scrypt[15] with appropriate parameters.

[13] https://pages.nist.gov/800-63-3/sp800-63b.html#-5112-memorized-secret-verifiers
[14] https://www.rfc-editor.org/rfc/rfc9106
[15] https://www.rfc-editor.org/rfc/rfc7914

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

As noted in the *Introduction*, this late February / early March 2026 penetration test and source code audit - which included a design and cryptographic review - was conducted by Cure53 against the ExpressVPN ExpressKeys applications available for Android and iOS - including all available components and features related to mobile.

From a contextual perspective, sixteen working days were allocated to reach the coverage expected for this project. The methodology used conformed to a white-box strategy, and a team of five senior testers was assigned to the project's preparation, execution, and finalization.

Cure53 was provided with access to the test applications and working credentials, as well as sources for the application parts in scope. This significantly increased the effectiveness of the audit, allowing Cure53 to check the application for security vulnerabilities in the code, as well as in the running environments. Testing therefore combined static review and dynamic analysis, and was conducted in a controlled environment using dedicated test applications, emulators, and physical devices where appropriate. The approach was aligned with common industry practices for mobile security testing, and emphasized realistic abuse scenarios.

For improved clarity, a dedicated *Test Methodology* section was included in this report. This outlines the key security checks performed, and provides a detailed overview of how the audit scope was examined.

The scope of this assessment included all of the ExpressKeys mobile application client-side components - covering the Flutter and Rust components, as well as all OS-specific mobile implementations. Although limited high-level checks were performed against the connected APIs, backend components were excluded from the scope of this engagement. Additionally, analysis of third-party dependencies for zero-day and known vulnerabilities was not included in this assessment. Parts of the *pmcore* component were only briefly examined, given that this had already been audited during a previous round of testing.

ExpressKeys' assumed threat model included adversaries with the ability to perform network attacks, trick users into visiting malicious websites, trick users into sharing an encrypted backup of their vault (and to perform offline attacks), obtain ExpressVPN account username / password combinations from public or private password dumps, attempt the bypass or abuse of authentication and session handling, and to trick users into installing malicious apps.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Overall, twelve findings were made, and of these, five were classified as security vulnerabilities and were added to the *Identified Vulnerabilities* section. The remaining seven findings were categorized as general weaknesses with lower exploitation potential and were added to the *Miscellaneous Issues* section.

This section will now take a closer look at the most notable observations made by the Cure53 team during the assessment's time frame.

One key focus for Cure53 was to test the applications for encryption leakage - via either logging, or by unsafe storage or being sent to the server. Initial assessments confirmed that the codebase generally adheres to strict data handling protocols, particularly regarding master encryption keys and user private keys. However, a significant vulnerability was identified involving the internal storage encryption key, which was found to be erroneously logged to the standard output (*stdout*) (EXP-22-010).

In addition to the finding discussed above, it is advised that the existing reliance on biometric authentication as the sole mechanism for securing the user password vault introduces a structural design flaw. By delegating vault access exclusively to device-level biometrics, the application's security becomes tethered to the integrity of the hardware. Any bypass of the biometric sensor - which is a known risk in various fingerprint sensor implementations - would grant an attacker immediate access to the vault. This architectural weakness is detailed in EXP-22-008.

Additional possibilities for the leakage of secrets were also reviewed, in parallel with the team's assessment of the encryption keys. This analysis revealed that API access tokens are transmitted to the server in order to perform connectivity checks. Such transmission is considered to be superfluous, as connectivity can be verified without exposing sensitive tokens. The risk is further exacerbated by the absence of certificate pinning - as documented in EXP-22-009. This omission facilitates MitM attacks for an adversary with physical access, enabling them to intercept tokens during the check (EXP-22-005).

While the application was seen to implement screenshot blocking in order to harden security against physical access, a functional bypass was identified. It was found that use of the shake-to-report feature triggers an automatic screen capture via the Luciq library. An attacker can exploit this by opening the resulting report and capturing a screenshot of the existing image, effectively neutralizing the intended security control (EXP-22-011).

The Autofill architecture was found to be well-designed overall. The credential matching functionality correctly enforces PSL-based domain extraction, associated domain validation, and protocol-aware filtering, ensuring that credentials are only surfaced for appropriate contexts under normal conditions. Android's frame isolation mechanism effectively prevents cross-frame credential leakage in supported browsers, and the allow-list - which restricts Autofill to recognized browser applications - mitigates credential theft through malicious WebView-based apps.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

However, a *Medium* severity issue was identified in the iOS credential provider, where null domains produced by *srcdoc* iframes and *data:* URIs cause the entire vault to be returned as "current domain" suggestions (EXP-22-006). This behavior could be leveraged by a malicious iframe embedded on a legitimate page to show all stored credentials, potentially leading users to inadvertently disclose credentials for unrelated websites. Notably, Android handles this same scenario more securely by classifying all entries under the "Other" section with mandatory confirmation dialogs.

A related *Low* severity finding was observed in the confirmation dialogs on both platforms, where null domains result in empty or malformed warning messages. It is advised that this decreases the user's ability to assess the target context before proceeding with a fill (EXP-22-007).

Beyond Autofill, the source code was reviewed for common vulnerability patterns - including unsafe uses of *dangerouslySetInnerHTML*, unvalidated redirects, and injection-prone sinks such as *href*, *src*, and *location* assignments.

Cryptographic implementations and the architecture were also reviewed, to pinpoint potential weaknesses in key management, encryption usage, randomness, and length padding. Here it was found that the cryptographic key - which the security of the vault fully depends on - is derived using PBKDF2 from the user vault password and a salt. Although the computational cost of PBKDF2 with the currently-chosen parameters is high on a CPU, it is advised that an attacker leveraging a GPU could easily brute-force passwords and undermine the security of the vaults (EXP-22-012).

The audit was conducted assuming an honest-but-curious ExpressKeys server, which does not deviate from the specified protocol, but which may try to break confidentiality afterwards. Providing security against a malicious server is important, but naturally more involved - as a recent paper suggests[16]. In particular, the ExpressKeys vaults appear to be vulnerable to item swapping attacks, where a malicious server swaps encrypted item parts, creates new items using other item parts, or deletes entire items, without leading to a decryption failure for the user.

The reason behind this issue is that the vault's integrity as a singular object is not provided, and moreover, the same user document key is used to encrypt and authenticate single items. However, due to the scope of this assessment, these threats were not considered during this audit, and it is therefore recommended to explicitly include the server-side parts in a further security assessment. However, with the exception of these issues, the architecture was found to be well-designed, and testing did not reveal any major issues.

[16] https://eprint.iacr.org/2026/058.pdf

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

In conclusion, this audit of the ExpressVPN ExpressKeys mobile applications for Android and iOS - including all available components and features related to mobile - found that the tested scope presented a solid overall impression in terms of its security. Further, although a relatively high number of findings were made during this engagement, their severity level did not exceed *Medium*. This indicates relatively stable protection against attacks targeting the examined applications, and clearly shows that the development team is aware of the types of problems that modern mobile applications tend to face. Overall, while the findings presented in this report indicate that there is still room for improvement - particularly with regard to information disclosure issues and misconfigurations - the security of ExpressKeys is considered robust.

Cure53 would like to thank Brian Schirmacher and Bruno Magalhaes from the ExpressVPN team for their excellent project coordination, support and assistance, both before and during this assignment.