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 Identity Defender 03.2026

Cure53, Dr.-Ing. M. Heiderich, M. Piechota, J. Ginesin, Dr. M. Conde, M. Wege

Index

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

Introduction

“A comprehensive suite of tools designed to reduce the risk of identity theft with real-time monitoring, early alerts, and data removal handled for you”

From https://www.expressvpn.com/features/id-defender

This report describes the results of a penetration test and source code audit conducted against the ExpressVPN Identity Defender software complex and mobile applications, with a focus on authentication, single sign-on (SSO), personally identifiable information (PII), and data storage.

To give some context regarding the assignment’s origination and composition, Network Guard Pte. Ltd. contacted Cure53 in February 2026. The test execution was scheduled for early March 2026, namely in CW10. A total of fourteen 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 a test system URL, a source code repository, binary builds for the mobile applications for iOS and Android, as well as all further means of access required to complete the tests were provided to facilitate the undertakings.

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

- **WP1**: White-box pen.-tests & code audits against Identity Defender Auth & SSO
- **WP2**: White-box pen.-tests & code audits against Identity Defender app logic & entitlements
- **WP3**: White-box pen.-tests & code audits against Identity Defender PII & Storage

All preparations were completed in late February 2026, specifically during CW09, 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 and shared its findings through the aforementioned Slack channel. Live reporting was not specifically requested for this audit.

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

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

Despite the general lack of any *High* or *Critical* severity findings being made - which can be interpreted as a positive sign - Cure53 unveiled several areas where it is advised that further attention and improvement is required. In order to achieve a good level of security, it is recommended that all of this report's findings should be resolved in a timely manner, regardless of their severity. Moreover, in order to ensure that the Identity Defender complex evolves in a secure manner, Cure53 recommends establishing a cycle of regular future audits, so as to identify and mitigate new risks as they emerge.

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 ExpressVPN Identity Defender software complex and its mobile applications.

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

Scope

- **Pen.-tests & code audits against ExpressVPN Identity Defender**
  - **WP1:** White-box pen.-tests & code audits against Identity Defender Auth & SSO
    - **Test System URL:**
      - https://portal.expressvpn.com
    - **Source code repository:**
      - kp_identity_defender_app
  - **WP2:** White-box pen.-tests & code audits against Identity Defender app logic & entitlements
    - **Test binaries:**
      - idd_android_1.0.0.7550_release_playstore.apk
      - kp_identity_defender_app.ipa
    - **Source code repository:**
      - kp_identity_defender_app
  - **WP3:** White-box pen.-tests & code audits against Identity Defender PII & Storage
    - **Test binaries:**
      - idd_android_1.0.0.7550_release_playstore.apk
      - kp_identity_defender_app.ipa
    - **Source code repository:**
      - kp_identity_defender_app
  - **Test User Credentials:**
    - U: haqpl+iddef1@volt.cure53.de
    - U: ginesin+iddef1@cure53.de
    - U: marta+iddef1@cure53.de
    - U: mike+iddef1@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.

As noted previously, the assignment was divided into three WPs, and the methodology chosen was white-box. WP1 focused on the authentication and SSO flows, while WP2 covered the logic of the mobile applications, and WP3 centered around the PII handling and secure storage mechanisms. The testing team was provided with the source code and binaries for the mobile apps, and during the review the team employed a combination of static and dynamic analysis, as well as source code review.

The assessment of WP1 heavily concentrated on the authentication flows and SSO mechanisms, specifically evaluating the integration with the Kape Client SDK and the Keycloak identity provider. A dedicated focus was placed on the Inter Process Communication (IPC) mechanisms utilized for seamless credential imports on Android. Through dynamic instrumentation and static source code review, the testing team analyzed the trust boundaries between the Identity Defender application and external components. This scrutiny revealed a flaw in the package signature verification process, which would enable a rogue content provider to force a session fixation attack by supplying an attacker-controlled refresh token (EXP-24-011). Furthermore, dynamic testing across different hardware environments confirmed that the refresh token lacks cryptographic device binding.

Furthermore, the evaluation encompassed the processing of JSON Web Tokens (JWTs). The team observed that the application extracts claims from these tokens to dictate local logic flows, without verifying their cryptographic signatures, although complete forgery is successfully prevented by the native SDK validation. Finally, the evaluation of session persistence and lifecycle management demonstrated that the application readily restores sessions without enforcing a local application lock. This was a risk exacerbated by the use of non-expiring refresh tokens (EXP-24-010).

WP2 focused on assessing the internal logic of the mobile applications, which involved a deep-dive into various services. The static analysis of the binaries revealed that the applications maintain a legacy support window that includes unmaintained versions of Android (API 24) and iOS (15), which lack modern security patches (EXP-24-002). On the Android side, the application fails to explicitly disable *android:allowBackup*, or to define backup rules on API level 24 - potentially allowing for data exfiltration via Android Debug Bridge (ADB) (EXP-24-005). No other serious misconfigurations were found in this area, although - as discussed below - the review of the services revealed a systemic failure in the handling of sensitive data within the application's telemetry and logging infrastructure.

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

The routing architecture and deep link handling logic were subjected to rigorous dynamic testing using ADB and custom-crafted intents. The team investigated how the application parses incoming uniform resource identifiers and routes them through the internal components. This analysis exposed an open redirect vulnerability within the authentication callback route, allowing unvalidated URLs to be loaded into the internal WebView (EXP-24-007). In addition, an architectural discrepancy was identified between the global routing configuration and the deep link coordinator. This misalignment results in a failure to enforce authentication constraints, permitting unauthenticated users to access restricted application interfaces by invoking specific custom scheme links (EXP-24-008).

The assessment of WP3 consisted of white-box testing against the personal information handling mechanisms and the secure storage mechanisms. The logging carried out by the services implementing the business logic, as well as the repository interfaces, was thoroughly scrutinized, and it was observed that many of these services overly-logged security-critical information (EXP-24-004).

The personal information handling pipeline was also thoroughly assessed; in particular, the regular expression queries designed to redact private information from being logged were scrutinized. It was observed that the regular expressions used for personal information redaction were too brittle to reliably capture all variants of personal information - as discussed further in (EXP-24-003). Finally, the review of the WebView implementations across multiple components highlighted a defense-in-depth concern regarding the persistent caching of sensitive data, noting a significant discrepancy where general external links utilize native incognito modes while the core components rendering sensitive information lack such secure configurations.

While on the Android side the application uses *EncryptedSharedPreferences*, on iOS the iOptions for the Keychain were not specified, causing the accessibility level to default to *WhenUnlocked* (EXP-24-009). It is advised that this configuration is insufficient, as it fails to bind the data to the specific device hardware, allowing Keychain items to migrate to backups. Additionally, the persistent global identity - the user's Kape Resource Name (KRN) - was found to be stored by the application in plaintext (EXP-24-006), which represents a medium severity risk of long-term identity tracking and correlation.

The design of the secure storage service, the Keystore, and the Keycloak were additionally tested for potential race conditions, data leaks, and data corruption. It was observed that a time-of-check to time-of-use (TOCTOU) race condition exists in the secure storage service's data write function, which may potentially result in data loss or data corruption of stored information (EXP-24-001).

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-24-001) to facilitate any future follow-up correspondence.

EXP-24-001 WP3: TOCTOU race condition in key storage service *(Medium)*

CVSS Score: 4.0
CVSS Temporal Score: 3.8
CVSS String: CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:L/E:P/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/367.html

During testing, it was observed that the *write()* method in the secure storage service, which is employed by the Keycloak client provider, suffers from a race condition that may cause data corruption and platform errors on iOS / macOS. The *write()* method implements a cooperative locking mechanism using a *Map<String, Completer>(_locks)* object to serialize concurrent writes.

This lock acquisition functionality contains a TOCTOU race. The check for an existing lock and the creation of a new lock are separated by an await call, allowing Dart's event loop to interleave other callers between the two operations. This race condition was confirmed during testing with the use of a Promela model and the SPIN model checker.

Affected file:
lib/data/services/secure_storage_service.dart

Affected code:

```
@override
Future<void> write({required String key, required String value}) async{

  // Wait for any pending operations on this key
  while (_locks.containsKey(key)) {
    await _locks[key]!.future;
  }

  // Create a lock for this operation
  final completer = Completer<void>();
  _locks[key] = completer;

  try {
    // these two operations needs to be made safe by a mutex behavior since
CSDK may be storing values of the same key in very quick succession,
```

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

```
    // which is not awaited. (CSDK callbacks can' be async)

    // Delete first to avoid errSecDuplicateItem (-25299) on iOS/macOS
    await _storage.delete(key: key);
    await _storage.write(key: key, value: value);
  } finally {
    // Release the lock
    _locks.remove(key);
    completer.complete();
  }
}
```

To mitigate this issue, Cure53 recommends replacing the check-then-set loop pattern with a synchronous read-and-replace of *_locks[key]* before any await is called. This will eliminate the interleaving window, as no yield point will exist between observing and claiming the lock.

EXP-24-003 WP3: Incomplete personal information redaction in logger *(Low)*

CVSS Score: 5.1
CVSS Temporal Score: 5.1
CVSS String: CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N/E:X/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/532.html

During testing, Cure53 observed that the personal information redactor present in the logging service relies on narrow, potentially flakey regex patterns, which may miss variations of sensitive data. Furthermore, the detection of several classes of sensitive information is entirely neglected by the filter.

Variations of sensitive patterns not covered by the current information redaction detection mechanisms include:

- **Phone numbers**: only the exact (XXX) XXX-XXXX format is caught. Alternate formats, such as XXX-XXX-XXXX or +1-XXX-XXX-XXXX will leak.
- **SSNs**: only the strict XXX-XX-XXXX format is caught; SSNs without dashes will leak.
- **Credit Cards**: currently, the regex strictly looks for 16-digit credit cards, and will miss 15-digit cards (as used by American Express), or 14-digit cards (as used by Diners Club).

Other sensitive patterns are entirely missed - including names, physical addresses, Base64 payloads, and JSON dumps.

Affected files:
lib/data/services/logging/pii_redactor.dart

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

To mitigate this issue, Cure53 recommends greatly expanding the regex patterns employed by the personal information redaction mechanism, in order to capture all possible variations of sensitive information.

Furthermore, it is advised that a thorough test suite should be constructed for the personal information redactor, to ensure that as little sensitive information as possible passes through the PII redaction mechanism.

EXP-24-004 WP3: Aggressive logging of sensitive information *(Medium)*

CVSS Score: 5.1
CVSS Temporal Score: 5.1
CVSS String: CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N/E:X/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/532.html

During testing, Cure53 observed that the application aggressively logs raw data structures, risking bypass of the existing regex patterns present in the personal information redaction mechanism present in the logger.

JWTService dumps entire JWT claim maps, *IAPService* logs plaintext *purchaseToken* and *customerEmail* strings, *BrazeService* logs unredacted user attributes and push payloads, and *AuthRepository* logs the OAuth authorization link directly.

Because the redactor only targets specific formats (EXP-24-003), standard Dart Map string representations and raw JSON objects containing unformatted personal information, physical addresses, proprietary IDs, standard Base64 payloads, and OAuth information will leak directly to disk and exportable logs.

Affected files:

- *lib/data/services/logging/pii_redactor.dart*
- *lib/data/services/jwt/jwt_service.dart*
- *lib/data/services/iap/iap_service.dart*
- *lib/data/services/braze/braze_service.dart*
- *lib/data/repositories/auth_repository.dart*

Affected code (JWT Service):

```
_logger.info('[$_tag] extractKrn: All JWT claims: ${claims.keys.join(",
")}');
_logger.debug('[$_tag] getUserClaims: Raw JWT claims: $
{claims.toString()}');
```

Affected code (IAP Service):

```
_logger.info('[$_tag] Processing Google payment - productId: $productId,
purchaseToken: $purchaseToken, email: $customerEmail');
```

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

Affected code (Braze Service):

```
_logger.debug('[$_tag] Logging event: $eventName', properties);
_logger.debug('[$_tag] Setting user attributes', attributes);
_logger.debug('[$_tag] Push payload:', payload);
```

Affected code (OAuth Repository):

```
Future<LoginResult> exchangeCodeAndLogin(String code) async {
  try {
    _logger.info('[$_tag] Exchanging authorization code from magic link
$code');
```

To mitigate this issue, Cure53 recommends thoroughly sanitizing Maps and JSON objects to mask known sensitive information (e.g. *email, sub, purchaseToken*), before serializing them to the logger.

Furthermore, Cure53 recommends hardening the fallback regex patterns in the *PiiRedactor* class, in order to capture a wider variety of sensitive data. This way, the risk of sensitive data being written to disk, and thus leaked, is reduced.

EXP-24-006 WP3: Insecure storage of persistent global identity *(Medium)*

CVSS Score: 5.2
CVSS Temporal Score: 5.1
CVSS String: CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N/E:F/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/312.html

During testing, it was observed that the KRN is extracted from the *usr* claim of the Subscription Receipt Token (SRT), and is then used as the primary identifier for the Braze SDK for tracking and engagement. By default, the Braze SDK persists this identifier in an unencrypted XML file within the app's shared preferences directory.

Further, when combined with EXP-24-005, the KRN is exposed to any attacker with physical access to the device, or access to a local ADB backup on devices running API level 30 (Android 11) and below. Additionally, it appears that the application also persists the same identifier via the Flutter *shared_preferences* plugin.

Since the KRN is a global persistent identifier across the Kape ecosystem, its storage in plaintext presents a privacy risk. An attacker could use the KRN to correlate a user's identity across different Kape services or leaked datasets, and also for long-term tracking of the user's activity and marketing profile.

The Braze user ID, which mirrors the user's KRN, was discovered in plaintext at the following locations on the device filesystem:

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

Affected files:

- */data/data/com.expressvpn.iddefender/shared_prefs/FlutterSharedPreferences.xml*
- */data/data/com.expressvpn.iddefender/com.appboy.offline.storagemap.xml*

Affected code:

```
<string name="last_user">krn::iam::xvpn:user:2e2f285f-24e5-4c98-b555-411857c3aa1b</string>
```

To mitigate this issue, it is advised that the app should be configured to use *EncryptedSharedPreferences* for all SDK-related persistence, in order to ensure that data is encrypted with keys stored in the hardware-backed Android Keystore.

If the raw KRN is not strictly required, then it is advised to consider providing the Braze SDK with a cryptographic hash of the KRN, instead of the plaintext string.

EXP-24-007 WP2: Keycloak deep link open redirect *(Medium)*

CVSS Score: 6.1
CVSS Temporal Score: 5.8
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N/E:P/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/601.html

The ExpressVPN Identity Defender application registers custom URL schemes to handle external deep links and authentication callbacks. During the assessment of the application's intent handling mechanisms, Cure53 discovered an open redirect vulnerability within the *keycloak-auth* deep link route.

Specifically, the application accepts a *magicLinkUrl* parameter via the *expressidd://* scheme, without performing adequate validation or sanitization on the provided URL. By crafting a malicious deep link, an attacker can force the application's internal WebView to redirect the user to an arbitrary, attacker-controlled domain, legitimizing phishing scenarios by leveraging the application's trusted context.

This redirection can be triggered either by a malicious application installed on the same device, or through a website when a user interacts with a crafted link.

Affected file:
lib/routing/router.dart

Affected code:

```
GoRoute(
  path: Routes.keycloakAuth,
  parentNavigatorKey: _rootNavigatorKey,
  builder: (context, state) {
    final email = state.uri.queryParameters['email'];
```

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

```
    final magicLinkUrl = state.uri.queryParameters['magicLinkUrl'];
    final preferOtp = state.uri.queryParameters['preferOtp'] == 'true';
    return KeycloakAuthWebViewScreen(
      email: email,
      magicLinkUrl: magicLinkUrl,
      preferOtp: preferOtp,
    );
  },
),
```

Affected file:
lib/ui/account_creation/widgets/keycloak_auth_webview_screen.dart

Affected code:

```
class _KeycloakAuthWebViewScreenState
    extends BaseState<KeycloakAuthWebViewScreen,
KeycloakAuthWebViewViewModel> {
  late final IBrowserService _browserService;
  late final ILoggingService _logger;
  WebViewController? _webViewController;
[...]
void _initializeWebView(String url) {
    _webViewController = WebViewController()
      ..setJavaScriptMode(JavaScriptMode.unrestricted)
      ..enableZoom(false)
      [...]
      ..loadRequest(Uri.parse(url));
}
```

The below *adb* command simulates clicking the deep link:

PoC adb command:

```
adb shell am start -W -a android.intent.action.VIEW -d
"expressidd:///keycloak-auth?magicLinkUrl=https://cure53.de"
com.expressvpn.iddefender
```

To mitigate this issue, Cure53 recommends implementing validation for the *magicLinkUrl* parameter within the deep link handler. It is advised that the application should employ a robust allow-list, containing only trusted domains and subdomains, and that before initiating any redirection or loading a URL in the WebView, the application should verify that the target URL's host strictly matches an entry on the allow-list and utilizes the *https://* scheme. Any URL failing these checks should then be rejected and safely discarded.

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

EXP-24-010 WP1: Insecure session restoration due to missing app lock *(Medium)*

CVSS Score: 6.1
CVSS Temporal Score: 5.8
CVSS String: CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N/E:P/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/306.html

During the assessment, Cure53 observed that the application restores active user sessions automatically upon startup or when returning from the background, without enforcing any local authentication mechanisms such as a biometric prompt or PIN code.

This behavior is compounded by the fact that the underlying OAuth *refresh_token* utilized by the application for session renewal lacks an expiration (*exp*) claim. While server-side expiration controls may exist, no session timeout or token invalidation was observed during the entire testing period.

Consequently, any adversary with physical access to the unlocked device can simply open the application and gain immediate, unfettered access to the sensitive information managed by the Identity Defender service.

Below is a decoded *refresh_token* without an *exp* claim:

Decoded JWT:

```
{
  "iat": 1772994741,
  "jti": "[...]",
  "iss": "https://auth.expressvpn.com/realms/xvpn",
  "aud": "https://auth.expressvpn.com/realms/xvpn",
  "sub": "[...]",
  "typ": "Offline",
  "azp": "kp-identity-defender-app",
  "sid": "[...]",
  "scope": "openid profile offline_access web-origins roles amr email basic
acr"
}
```

To mitigate this issue, it is recommended to implement a local authentication challenge - such as biometric verification or a PIN - which users must successfully pass whenever the application is launched or brought to the foreground from a suspended state.

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

EXP-24-011 WP1: Session fixation via rogue inter-app content provider *(Medium)*

CVSS Score: 6.0
CVSS Temporal Score: 6.0
CVSS String: CVSS:3.1/AV:L/AC:H/PR:N/UI:R/S:U/C:H/I:H/A:N/E:P/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/384.html

The ExpressVPN Identity Defender application features an inter-app login mechanism designed to seamlessly import authentication credentials from the main ExpressVPN Android application. This is achieved by querying a specific *ContentProvider* at the URI *content://com.expressvpn.vpn/current*.

However, Cure53 discovered that the application blindly trusts any installed package that registers this authority, failing to verify the digital signature of the provider app.

A potential attacker could exploit this oversight by creating a malicious application that registered a rogue content provider under the *com.expressvpn.vpn* authority. If a victim were to install this application, then the attacker could serve their own valid *refresh_token*. The attacker could then forcibly initiate the SSO flow via an Intent.

From there, this would become a one-click compromise. As soon as the victim interacted with the login button on the resulting screen, the app would unwittingly import the attacker's token, refresh it against Keycloak, and establish an active session bound to the attacker's KRN.

Consequently, this would result in a complete session fixation. Any data subsequently entered by the victim would be synced with the attacker's account.

Affected file:
android/app/src/main/kotlin/com/expressvpn/iddefender/XVCredentialImporter.kt

Affected code:

```
class XVCredentialImporter(private val context: Context) {
    companion object {
        private const val TAG = "XVCredentialImporter"
        private const val PROVIDER_AUTHORITY = "com.expressvpn.vpn"
        private const val PROVIDER_PATH = "current"

        private const val COLUMN_ACCESS_TOKEN = "access_token"
        private const val COLUMN_REFRESH_TOKEN = "refresh_token"
        private const val COLUMN_ID_TOKEN = "id_token"
        private const val COLUMN_EXPIRATION = "expiration"
    }

    fun importCredentials(): Map<String, Any> {
```

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

```
        Log.d(TAG, "Starting credential import from ExpressVPN app")

        return try {
            val uri =
             Uri.parse("content://$PROVIDER_AUTHORITY/$PROVIDER_PATH")
            Log.d(TAG, "Querying ContentProvider at: $uri")

            val cursor = context.contentResolver.query(
                uri,
                arrayOf(COLUMN_ACCESS_TOKEN, COLUMN_REFRESH_TOKEN,
                    COLUMN_ID_TOKEN, COLUMN_EXPIRATION),
                null,
                null,
                null
            )
[...]
```

Affected file:
lib/data/repositories/auth_repository.dart

Affected code:

```
Future<bool> _loginWithXvCredential(XvCredential credential) async {
 _logger.info('[$_tag] XV tokens located (source=$
{credential.sourceFile}).');
    [...]
    } else if (authMode == _XvAuthMode.openId) {
      _logger.info('[$_tag] XV import missing ID token; refreshing from
KeyCloak.');
      TokenResponse? refreshed;
      try {
        refreshed = await
_authService.refreshTokens(credential.refreshToken);
```

Shown below is part of the PoC code for a FakeVPN application. This registers the rogue *ContentProvider* serving a valid attacker's *refresh_token*. *MainActivity* includes a minimal attacker interface to forcibly trigger the authentication flow via the application's deep link:

Kotlin PoC code:

```
package com.example.fakevpn

import android.content.ContentProvider
import android.content.ContentValues
import android.database.Cursor
import android.database.MatrixCursor
import android.net.Uri
import android.util.Log
```

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

```
class FakeXVProvider : ContentProvider() {
    override fun onCreate(): Boolean {
        Log.d("FakeXVProvider", "provider initialized")
        return true
    }

    override fun query(
        uri: Uri, projection: Array<out String>?, selection: String?,
        selectionArgs: Array<out String>?, sortOrder: String?
    ): Cursor {
        Log.d("FakeXVProvider", "app queried: $uri")

        val cursor = MatrixCursor(arrayOf("access_token", "refresh_token",
"id_token", "expiration"))

val RefreshToken = "ATTACKERS_REFRESH_TOKEN"
        cursor.addRow(arrayOf(
            "fake_expired_access_token",
            RefreshToken,
            "", // must be empty
            "2099-12-31T23:59:59Z"
        ))
        return cursor
    }

    override fun getType(uri: Uri): String? = null
    override fun insert(uri: Uri, values: ContentValues?): Uri? = null
   override fun delete(uri: Uri, selection: String?, selectionArgs:
Array<out String>?): Int = 0
   override fun update(uri: Uri, values: ContentValues?, selection:
String?, selectionArgs: Array<out String>?): Int = 0
}
```

Steps to reproduce:

1. Ensure that the ExpressVPN Identity Defender application is installed on the testing device and is completely logged out.
2. Download and install the malicious PoC application from the following link: *https://cure53.de/exchange/2530672094865430976776/app-debug.apk*
3. Open the installed FakeVPN application.
4. Tap the button. This action will invoke the *expressidd:///welcome-back* Intent.
5. The Identity Defender app will open and present a login screen. Tap the login button.
6. Observe the application's profile settings to confirm that the app successfully queried the rogue *ContentProvider*, imported the tokens, and authenticated the session under the attacker's account.

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

To mitigate this issue, Cure53 recommends implementing verification mechanisms to ensure that inter-app credential sharing only occurs between trusted, officially-distributed ExpressVPN applications. The Identity Defender app should verify the authenticity and origin of the application providing the credentials before importing any tokens.

Further, it is advised that ExpressVPN should transition to the use of secure IPC patterns supported by the Android OS - such as enforcing signature-level permissions, implementing cryptographic payload verification, or utilizing shared secure storage enclaves - to guarantee that sensitive authentication material is exchanged exclusively within the verified ExpressVPN ecosystem.

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-24-002 WP2: Unmaintained Android and iOS version support *(Low)*

CVSS Score: 3.5
CVSS Temporal Score: 3.5
CVSS String: CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N/E:H/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/1104.html

Cure53's static analysis of the ExpressVPN Identity Defender Android and iOS mobile apps confirmed that the apps can be installed and run on mobile devices with outdated Android and iOS versions. Specifically, the *minSdkVersion* is set to 24 in Android and the *MinimumOSVersion* is set to 15 in iOS.

Android's API level 24 received its final security patch in October 2019[1], meaning that it no longer receives active maintenance. Similarly, supporting iOS 15[2] is unnecessarily permissive, as it no longer receives consistent security updates from Apple.

The continued support for old Android and iOS versions ultimately expands the apps' attack surface, potentially exposing users to vulnerabilities that have been patched in newer OS releases.

Affected file #1:
kp_identity_defender_app/android/app/build.gradle.kts

Affected code #1:
```
android {
    namespace = "com.expressvpn.iddefender"
    [...]

    defaultConfig {
        applicationId = "com.expressvpn.iddefender"
        minSdk = 24
        targetSdk = 36
        [...]
```

[1] https://endoflife.date/android
[2] https://endoflife.date/ios

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

Affected file #2:
Info.plist

Affected code #2:
```
<dict>
  [...]
  <key>MinimumOSVersion</key>
  <string>15.0</string>
```

To mitigate this issue, Cure53 recommends raising the minimum OS version to an appropriate value that still receives security updates, thus minimizing the attack surface.

It is advised that the minimum supported version should consider the highest proportion of the user base[3] that would still be able to use the app, while ensuring that the app runs on an OS that ideally receives active security maintenance. To this end, it is advised that Version 31 (Android 12) and iOS 17 should ideally be adopted.

EXP-24-005 WP3: *android:allowBackup* defaults to true in API level 24 *(Info)*

CVSS Score: N/A
CVSS Temporal Score: N/A
CVSS String: N/A
CWE: N/A

Following the discovery that an unmaintained OS version is supported by the Android mobile app - via *minSdkVersion* being set to 24 (see EXP-24-002) - it was observed that the attribute *android:allowBackup* is not explicitly defined in the *AndroidManifest.xml* file.

In Android API level 24, this attribute defaults to true. This would allow an attacker with physical access to the device and USB debugging enabled to use ADB (*adb backup*) to bypass the application's sandbox protections and to export app data. In particular, the shared preferences, internal databases, and local file storage are included by default in these backups.

Although this may not pose a risk if no sensitive information is stored locally, it creates an unnecessary path for data exfiltration. Furthermore, if cloud backups are enabled, then this data may be synchronized to Google Drive, expanding the attack surface beyond the physical device.

To align with best practices, it is recommended to explicitly disable this feature in the *AndroidManifest.xml* file. If backups are strictly required for user experience, then it is advised that backup configuration rules should be implemented, to strictly exclude directories containing sensitive information.

[3] https://apilevels.com/

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

EXP-24-008 WP2: Lack of authentication enforcement for deep links *(Info)*

CVSS Score: 0.0
CVSS Temporal Score: 0.0
CVSS String: CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:N/I:N/A:N/E:P/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/285.html

During the assessment of the application's routing mechanism and potential enrolment bypasses, Cure53 identified an architectural discrepancy in the way that deep link authorization is handled between the routing configuration and the deep link coordinator. This inconsistency allows unauthenticated users to bypass the initial login screens, and to access restricted UI components by invoking custom scheme deep links.

Specifically, the global routing guard explicitly bypasses authentication checks for the custom *expressidd://* scheme, delegating this responsibility to the deep link handling logic. While the deep link coordinator successfully identifies that the user lacks a valid session, it passively drops the navigation event, instead of actively redirecting the user to a secure fallback - such as the login screen.

As a result, the default routing behavior proceeds unhindered, rendering the restricted main interface. Although the application correctly prevents enrollment actions, the frontend views remain exposed. This behavior contradicts the intended security posture of the application.

Affected file:
lib/routing/router.dart

Affected code:

```
Future<String?> _redirect(BuildContext context, GoRouterState state) async
{
  // Deep links with custom scheme (e.g. expressidd://) are handled
exclusively
  // by DeepLinkCoordinator. Ignore them here to avoid stripping query
  // parameters that the coordinator appended to the resolved route.
  if (state.uri.scheme == 'expressidd') {
    return null;
  }
[...]
  bool isEntitled = false;
  bool isExpired = false;
  if (loggedIn) {
    isEntitled = await
getIt<IEntitlementChecker>().hasAnyIddEntitlements();
    isExpired = getIt<ISubscriptionRepository>().isExpired;
  }

  // different points where user can log in successfully.
```

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

```
final loggingIn = state.matchedLocation == Routes.login;
final isOnPaywall = state.matchedLocation == Routes.upgradePending;
final isProtectionScoreDetail =
    state.matchedLocation == Routes.protectionScoreDetail;

if (!loggedIn) {
  return Routes.login;
}
```

The bypass can be reliably triggered using the following ADB command on a device where no active session exists:

PoC ADB command:

```
adb shell am start -W -a android.intent.action.VIEW -d "expressidd://main"
com.expressvpn.iddefender
```

Furthermore, the *DeepLinkCoordinator* only verifies the *isAuthenticated* state, omitting the entitlement and subscription checks (*isEntitled*, *isExpired*) normally enforced by the GoRouter. While a practical paywall bypass could not be verified, there is a theoretical risk that authenticated users with expired or missing subscriptions could utilize this routing flaw to bypass the paywall screen and to access restricted application features.

To mitigate this issue, Cure53 recommends harmonizing the authentication and entitlement logics across all routing and deep link handling components. It is advised that the application should actively enforce security state checks, regardless of the entry vector.

This can be achieved either by removing the scheme-based bypass in the global router and enforcing checks for all routes, or by ensuring that the deep link coordinator actively redirects unauthenticated users to the appropriate screen when a restricted deep link is invoked.

EXP-24-009 WP3: Insufficient Keychain protection via accessibility level *(Medium)*

CVSS Score: 5.2
CVSS Temporal Score: 5.1
CVSS String: CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N/E:F/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/922.html

While reviewing the use of the local storage by the mobile applications, it was found that while the *flutter_secure_storage* plugin is used and correctly initialized for Android, the iOS-specific configuration lacks explicit security attributes.

Specifically, the *IOSOptions* parameter is omitted. In the absence of this configuration, the plugin defaults to *kSecAttrAccessibleWhenUnlocked*.

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

While this level of accessibility ensures data is encrypted at rest, it allows the data to be included in iCloud and iTunes / Finder backups. Consequently, data stored in the Keychain is not hardware-bound, and can be migrated to - or extracted from - other devices via standard backup restoration.

Affected file:
kp_identity_defender_app/lib/data/services/secure_storage_service.dart

Affected code:
```
class FlutterSecureStorageImpl implements ISecureStorage {
  final _storage = const FlutterSecureStorage(
    aOptions: AndroidOptions(
      encryptedSharedPreferences: true,
    ),
    // Missing IOSOptions
  );
```

In order to harden the iOS storage, it is recommended to explicitly set the accessibility level to *first_unlock_this_device*. This will ensure that the data is tied to the specific device, and that it is excluded from all backups.

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

Conclusions

As noted in the *Introduction*, this early March 2026 penetration test and source code audit was conducted by Cure53 against the ExpressVPN Identity Defender software complex and mobile applications, with a focus on authentication, SSO, PII, and data storage.

From a contextual perspective, fourteen 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.

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

It is advised that Cure53's assessment of the authentication and SSO mechanisms (WP1), alongside the deep link routing and native Android integration yielded a mixed security impression. While the underlying Kape Client SDK acts as a robust cryptographic gatekeeper that successfully prevents unauthorized token forgery, the client-side implementation wrapping these capabilities was found to exhibit notable architectural weaknesses.

A primary area of concern resides in the inter-app login functionality. It is advised that the application's reliance on implicit trust for IPC introduces severe risks. By failing to verify the digital signature of the *ContentProvider* responsible for supplying OAuth tokens, the application effectively undermines the integrity of the SSO flow, rendering it susceptible to cross-account session fixation attacks (EXP-24-011).

Another insecure pattern observed during testing relates to deep link handling and navigation state management. The routing architecture was found to suffer from a discrepancy in responsibilities between the global GoRouter and the dedicated *DeepLinkCoordinator*. This architectural mismatch leads to inconsistent security enforcement, allowing unauthenticated users to access restricted UI states (EXP-24-008). Furthermore, the lack of input validation within these deep link handlers exposes the application to open redirect vulnerabilities, which could easily be weaponized in phishing scenarios (EXP-24-007).

In addition to the formally-reported findings, Cure53 identified an area for defense-in-depth improvement where local state persistence and caching is concerned. Here, a notable architectural inconsistency was observed in the handling of web-based content. While external links are opened using a WebView implementation with Incognito mode enabled, the core *WebComponentScreen* responsible for rendering sensitive financial and identity data relies on a standard implementation without native cache disabling features.

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

To manage this, the application attempts to clear the cache via asynchronous unawaited operations during component disposal. This pattern introduces race conditions and leaves data such as session cookies, *sessionStorage*, and *IndexedDB* uncleared, allowing API responses to linger on the device's filesystem. Although exploiting this would require a compromised device or advanced local access, Cure53 still recommends standardizing the WebView implementations across the application, and to utilize native incognito modes for all sensitive web components, in order to ensure ephemeral data handling.

Separately, it was found that the application fails to implement a local app lock upon resuming, and relies heavily on non-expiring refresh tokens (EXP-24-010). This specific flaw strongly prioritizes user convenience over security, and allows for persistent unauthorized physical access to sensitive data. In order to improve the overall security posture, Cure53 recommends shifting towards a fail secure routing model, and enforcing strict validation for both IPC interactions and session longevity.

During testing, Cure53 thoroughly assessed the various services integrating the business logic of the Identity Defender app. Overall, the team formed a positive impression of security where the business logic services were concerned, and only minor implementation-level issues were discovered. No flaws were found in the overall design or architecture of the business logic services.

Among the services analyzed by Cure53, it was discovered that the secure storage service - which handles credential storage and session restoration - suffers from a TOCTOU race condition that may lead to corruption or loss of data (EXP-24-001). Cure53 confirmed that this vulnerability was present in the design pattern employed in the service, through the use of the SPIN model checker.

It was additionally noted that the services integrating the business logic aggressively logged potentially-sensitive data. Potential logging of sensitive data was observed to occur in the JWT service, the IAP service, the Braze service, and in the authorization repository (EXP-24-004). While inspecting the logging pipeline, it was also observed that the regular expressions designed to catch and prevent the logging of sensitive data were very brittle, and could miss edge cases for format variations of sensitive patterns (EXP-24-003).

The repository interfaces and implementations were also assessed. The team's impression of the repository interfaces in general was very strong, and the only issue discovered in this area was the aggressive logging of potentially-sensitive information noted above. Specifically, it was discovered that the OAuth magic link was logged by the authorization repository interface, which directly results in session compromise if the OAuth magic link is discovered in logging by a malicious party (EXP-24-004).

Static analysis of the mobile applications revealed that their security baseline is currently weakened by the continued support of deprecated operating systems (EXP-24-002), which in turn has potentially unsafe defaults (EXP-24-005).

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

A discrepancy was found to exist between the application's use of local storage on Android and iOS. While on the Android side, the use of *EncryptedSharedPreferences* provides a robust layer of protection, it is advised that the iOS implementation is undermined by the omission of *iOptions* in the *FlutterSecureStorage* configuration (EXP-24-009).

The review also identified a leakage, where internal unique identifiers are exposed to third-party SDKs. Specifically, the user's KRN - which serves as a global, permanent primary key across the Kape ecosystem - was found to be stored in plaintext (EXP-24-006). This exposure allows for long-term user tracking and correlation.

Given the overall number and severity of findings made during this engagement, it is advised that the security stance of the ExpressVPN Identity Defender software complex can be seen as being slightly below average - even though no *High* or *Critical* severity issues were identified. To enhance the overall security posture, it is highly recommended to fix the confirmed vulnerabilities and miscellaneous items as soon as possible, and to continue to audit the software complex on a regular basis, going forward.

Cure53 would like to thank Brian Schirmacher, Toto Tvalavadze, and Angilette Escobar from the Network Guard Pte. Ltd. team for their excellent project coordination, support and assistance, both before and during this assignment.