Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report ExpressVPN iOS App 08.-09.2022

Cure53, Dr.-Ing. M. Heiderich, MSc. S. Moritz, Dipl.-Ing. A. Aranguren

Index

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

"A VPN, or virtual private network, adds a layer of security between your iOS device and the internet, protecting you from online snooping, interference, and censorship. With ExpressVPN, you can stream, shop online, and browse the internet privately and securely."

From https://www.expressvpn.com/vpn-software/vpn-ios

This report - titled EXP-11 - details the scope, results, and conclusory summaries of a penetration test and source code audit against the ExpressVPN iOS mobile app, with a particular focus on ExpressVPN Keys (the password manager integrated within ExpressVPN's mobile apps), VPN protocol integration, and dependencies. The work was requested by ExpressVPN in June 2022 and initiated by Cure53 in late August and early September 2022, namely in CW35 and CW36. A total of sixteen days were invested to reach the coverage expected for this project. The testing conducted for EXP-11 was divided into three separate work packages (WPs) for ease of execution, as follows:

- **WP1**: Source-code-assisted penetration tests against ExpressVPN iOS mobile app
- **WP2**: Source-code-assisted penetration tests against Keys, ExpressVPN's password manager
- **WP3**: Source-code audits and reviews against VPN protocol integration and dependencies

Cure53 was provided with sources, pertinent documentation, test-user accounts, and any alternative means of access required to complete the review. For these purposes, the methodology chosen was white-box and a team of three senior testers was assigned to the project's preparation, execution, and finalization. All preparatory actions were completed in August 2022, namely in CW34, to ensure that testing could proceed without hindrance or delay.

Communications were facilitated via a dedicated, shared Slack channel deployed to combine the workspaces of ExpressVPN and Cure53, thereby creating an optimal collaborative working environment. All participatory personnel from both parties were invited to partake throughout the test preparations and discussions. In light of this, communications proceeded smoothly on the whole.

The scope was well-prepared and transparent, no noteworthy roadblocks were encountered throughout testing, and cross-team queries remained minimal as a result. The ExpressVPN team delivered excellent test preparation and assisted the Cure53 team in every respect to procure maximum coverage and depth levels for this exercise.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Cure53 gave frequent status updates concerning the test and any related findings, whilst simultaneously offering prompt queries and receiving efficient, effective answers from the maintainers. Live reporting was offered and subsequently achieved via the aforementioned Slack channel.

Regarding the findings specifically, the Cure53 team achieved comprehensive coverage over the WP1 through WP3 scope items, identifying a total of nine. Four of these findings were categorized as security vulnerabilities, whilst the remaining five were deemed general weaknesses with lower exploitation potential. Generally speaking, the overall yield of findings is relatively moderate in comparison with similarly scoped audits, which in turn reflects positively on the ExpressVPN iOS application's perceived security strength.

Moreover, the fact that all findings were assigned a severity rating of *Medium* or lower indicates a complete lack of significant attack surfaces and damaging threat potential. Nevertheless, the testing team is keen to underline that three of the four identified vulnerabilities represent a recurring issue pertaining to information disclosure, which should be earmarked as a focus area for improvement moving forward.

All in all, the development team deserves every plaudit for their due diligent efforts in minimizing any potential threats for the iOS application, with only minor adjustments required to further elevate the platform to an exemplary standard from a security perspective.

The report will now shed more light on the scope and testing setup as well as provide a comprehensive breakdown of the available materials. Subsequently, the report will list all findings identified in chronological order, starting with the detected vulnerabilities and followed by the general weaknesses discovered. Each finding will be accompanied by a technical description and Proof of Concepts (PoCs) where applicable, plus any relevant mitigatory or preventative advice to action.

In summary, the report will finalize with a conclusion in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the ExpressVPN iOS mobile app components in focus, giving high-level hardening advice where applicable.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Code audits and security assessments against ExpressVPN mobile application for iOS**
  - **WP1**: Source-code-assisted penetration tests against ExpressVPN iOS mobile app
    - **Primary audit focus:**
      - ExpressVPN iOS Mobile App
        - *xv_iphone*
        - *xv_libballoon*
      - Tested App version: *11.58.0.116835*
    - **In-scope items:**
      - The compiled binary and ExpressVPN iOS App sources with attacks in mind that lead to DoS, information leakage, MitM, RCE, and similar.
      - All iOS-related sources.
    - **Out-of-scope items:**
      - Features related to development or testing.
      - Dependencies not listed in scope document.
      - ExpressVPN APIs (though MitM attacks were in scope).
      - Code irrelevant to the iOS application targeting other platforms such as Android, Windows, Aircove, Linux, and macOS.
      - VPN servers.
      - Any exploit vector with the prerequisite that the device is jailbroken.
  - **WP2**: Source-code-assisted penetration tests against integrated password manager
    - **Primary audit focus:**
      - Password manager integrated in the ExpressVPN iOS App:
        - *password_manager/iOS*
    - **In-scope items:**
      - All password manager components integrated by the ExpressVPN iOS app.
      - Note that several components had already been evaluated in an earlier assessment (see EXP-10 - Android Assessment); only iOS-specific bindings were in scope for this assessment.
      - Autofill integration.
    - **Out-of-scope items:**
      - Password manager core.
      - *See WP1 info for additional OOS items.*
  - **WP3**: Source-code audits and reviews against VPN protocol integration and dependencies
    - **Secondary audit focus:**
      - Core Lightway library (*libhelium*).
      - Dependency packaged with *libballoon* (*libxenon*).
      - Shared library used to communicate with API servers (*xvclient*).

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

- **In-scope items:**
  - The integration of available VPN protocols into the ExpressVPN iOS app.
  - Several mission-critical dependencies and third-party software, including cursory audits covering *libhelium*, *xvclient*, and *libxenon* usage.
- **Out-of-scope items:**
  - VPN protocol implementations.
  - *See WP1 info for additional OOS items.*
- **Test-user accounts were created and activated for the auditing team**
- **All binaries in scope were shared with Cure53**
- **Test-supporting material was shared with Cure53**
- **All relevant sources were made available for Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 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. Categorizes issues that allow attackers to achieve extensive access to sensitive areas, such as critical systems, applications, data or other pertinent components in scope.

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

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

Low: Categorizes issues that have a highly limited impact on the areas in scope. Mostly does not depend on the level of exploitation but rather on the minor severity of obtainable information or lower grade of damage targeting the areas in scope.

Info: Categorizes issues considered merely informational in nature. They are mostly considered as hardening recommendations or improvements that can generally enhance the security posture of the areas in scope.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Table of Findings

Identified Vulnerabilities

ID	Title	Severity
EXP-11-001	WP1: Information disclosure via absent security screen	*Low*
EXP-11-003	WP1: Potential phishing via iOS URL scheme hijacking	*Medium*
EXP-11-005	WP1: Absent data protection facilitates VPN config access	*Medium*
EXP-11-009	WP1: Magic login via launch-app deep link facilitates DoS	*Low*

Miscellaneous Issues

ID	Title	Severity
EXP-11-002	WP1: Absent jailbreak detection	*Info*
EXP-11-004	WP1: WebView weaknesses via SFSafariViewController usage	*Info*
EXP-11-006	WP1/3: Memory corruption weaknesses via insecure functions	*Info*
EXP-11-007	WP1/2: Clear-text password storage in iOS Keychain	*Info*
EXP-11-008	WP1: Potential WebView XSS via insufficient sanitization	*Info*

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified throughout the testing period. Please note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is given in brackets following the title heading for each vulnerability. Furthermore, each vulnerability is given a unique identifier (e.g., *EXP-11-001*) to facilitate any future follow-up correspondence.

EXP-11-001 WP1: Information disclosure via absent security screen *(Low)*

***Fix Note**: The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

Testing confirmed that the iOS app fails to render a security screen when backgrounded. This allows attackers with physical access to an unlocked device to peruse data displayed by the app before it disappears into the background. A malicious app or an attacker with physical access to the device could leverage this weakness to gain access to user information, such as sensitive or compromising data related to PII.

To replicate this issue on iOS, simply navigate to a screen displaying sensitive information, then send the application to the background. Subsequently, display the open app and observe that the input text can be read by the user. Notably, this text will remain legible following a device reboot.

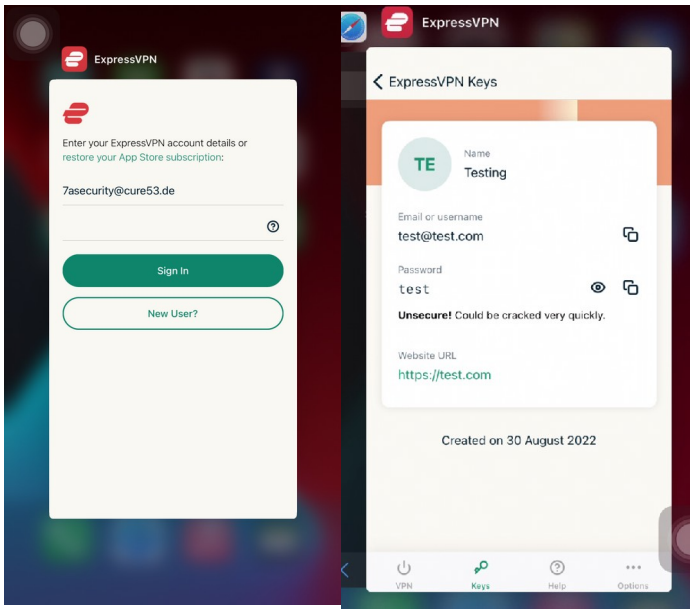

Fig.: Side-channel information leak via absent security screen on iOS app.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

The root cause of this issue can be observed in the *AppDelegate*, which does not currently capture the relevant events to display a security screen when the application is backgrounded under the current implementation.

Specifically, the handler for *applicationWillResignActive* is absent in the *AppDelegate*. Furthermore, the handler for the *applicationDidEnterBackground* event exists but does not currently contain any code handling a security screen:

Affected file:
xv_iphone/SuperXV/AppDelegate.m

Affected code:

```
- (void)applicationDidEnterBackground:(UIApplication*)application {
    [self scheduleAppRefresh];
    [self scheduleProcessing];
}

- (void)scheduleAppRefresh {
    if (@available(iOS 13, *)) {
        DDLogDebug(@"AppDelegate: scheduleAppRefresh");

        BGAppRefreshTaskRequest* request = [[BGAppRefreshTaskRequest alloc]
initWithIdentifier:@"com.expressvpn.ios.backgroundAppRefresh"];
        request.earliestBeginDate = [NSDate dateWithTimeIntervalSinceNow:15 *
60];

        NSError* error;
        BOOL result = [BGTaskScheduler.sharedScheduler submitTaskRequest:request
error:&error];
        if (!result) {
            DDLogDebug(@"Fail to schedule background app refresh: error=%@",
error);
        }
    }
```

To mitigate this issue, Cure53 recommends rendering a security screen overlay when the app is backgrounded. For iOS apps, the process of backgrounding an application can be detected in *Swift*[1] and *Objective-C*[2]. Subsequently, a different screen that does not include any sensitive user data can be displayed. A revised approach could also constitute sensitive information leakage prevention via iOS screenshots.

[1] https://www.hackingwithswift.com/example-code/system/how-to-detect-when-your-app-mo...ackground
[2] https://developer.apple.com/...-applicationwillresignactive?language=objc

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

This is typically accomplished in the *AppDelegate* file using the *applicationWillResignActive* or *applicationDidEnterBackground* method.
This vulnerability is rated a **3.9**

View the breakdown of each component of the scoring below:
https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:L/AC:L/PR:L/UI:R/S:U/C:L/I:L/A:N

CWE: https://cwe.mitre.org/data/definitions/200.html

EXP-11-003 WP1: Potential phishing via iOS URL scheme hijacking *(Medium)*

Note from ExpressVPN: *These links are used solely for the purpose of signing in through a randomized activation token (i.e. an alternative method for users to sign in using a one-time email link, instead of their usual activation code). The risk is that a malicious application might be able to obtain the randomized, short lived non-sensitive token used for application activation. ExpressVPN and Cure53 agree that it is not possible to fix this issue without introducing the possibility for these universal links to be blocked in some countries, based on the domain used. Blocking these links would prevent users from being able to sign into our application using this secondary method.*

Testing confirmed that the iOS app currently implements a custom URL handler, which is considered insecure since it is susceptible to URL hijacking. This approach has been leveraged by numerous malicious iOS applications in well-documented cases[3]. As a result, a malicious app could leverage this weakness to register the same custom schemes as the official application and thereby receive sensitive information intended for the legitimate application only. To provide an example, this attack could be utilized to display a fake login screen to victim users from a malicious app on the device that registered the same URL scheme, hence harvesting user credentials. Please note that this vulnerability remains exploitable[4] even though Apple has implemented the *first-come-first-served* principle since iOS 11.

A URL example susceptible to hijacking by a malicious application is offered below.

Hijackable URL schemes:
expressvpn://

This issue's root cause can be identified in the application's *Info.plist* file:

Affected file:
xv_iphone//SuperXV/Info.plist

[3] https://www.fireeye.com/blog/threat-research/2015/02/ios_masque_attackre.html
[4] https://malware.news/t/ios-url-scheme-susceptible-to-hijacking/31266

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Affected code:

```
<key>CFBundleURLTypes</key>
   <array>
      <dict>
         <key>CFBundleTypeRole</key>
         <string>Editor</string>
         <key>CFBundleURLName</key>
         <string>com.expressvpn.iosvpn</string>
         <key>CFBundleURLSchemes</key>
         <array>
            <string>expressvpn</string>
         </array>
      </dict>
   </array>
```

To mitigate this issue, Cure53 advises substituting the current deep link implementation from custom URL schemes to iOS Universal Links[5], simply because custom URL schemes remain hijackable and therefore cannot be considered sufficiently secure[6].

This vulnerability is rated a **5.6**

View the breakdown of each component of the scoring below:
https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:L/AC:L/PR:L/UI:R/S:U/C:H/I:L/A:N

CWE: https://cwe.mitre.org/data/definitions/200.html

EXP-11-005 WP1: Absent data protection facilitates VPN config access *(Medium)*

***Fix Note**: The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

Testing confirmed that the iOS app does not currently implement iOS' integrated Data Protection features. This means that most files are encrypted with the default *NSFileProtectionCompleteUntilFirstUserAuthentication*[7] encryption, which stores the decryption key in memory while the device is locked. Cure53 considers this to be the least secure form of data protection available on iOS. A malicious attacker with physical access to the device could leverage this weakness to read the decryption key from memory and gain access to local app data files without needing to unlock the device. Further scrutiny revealed that some of the unprotected files display access to the VPN configuration and alternative information.

[5] https://developer.apple.com/ios/universal-links/
[6] https://blog.trendmicro.com/trendlabs-security-intelligence/ios-url-scheme-susceptible-to-hijacking/
[7] https://developer.apple.com/.../nsfileprotectioncompleteuntilfirstuserauthentication

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

To replicate this issue, a jailbroken phone was left at rest for a few minutes on the lock screen, then all application files were retrieved for inspection of any potential data leak. Notably, this also can be achieved on a non-jailbroken device via reading the decryption key from the memory and the file afterward. A handful of examples revealed by the app files retrieved during device lock can be consulted below:

Affected file:
/var/mobile/Containers/Data/Application/<GUID>/Library/Preferences/com.expressvpn.iosvpn.plist

Affected code:

```
[...]
TjsonV$class��_�{"protocol":"udp","password":"xxx","ip":"connect.expressvpn.com
","username":"aaa","ipsec_host":"connect.expressvpn.com","openVpnConfig":"dummy_
config","port":"500"}�Z$classnameX$classes^ServerObjectV2�▒^ServerObjectV2YJSON
ModelXNSObjec▒$)27ILQSX^choqs▒*3BFU_▒hW11.58.03A�^.���aWsucces_blog_20220816_ip
,
O�{"lastKnownFlagValidity":683433406.17491603,"lastFailedConnection":683340924.
45887995,"lastConnectionFailureReason":{"tyEe":"un3A�^.PYSOME
IDFAload":"Th3A�^.���?�3A�]<��ut."},������ntConnectionMode":"offline"}3A�]<�!
v�3A�^+�$
,M`|�����0NZs������$&'()*-.1XYZcfijlmnpqrg [...]
```

The extent of this issue is perhaps best illustrated by the output of the *tar* command, which is able to read most files after the phone has remained passive on the lock screen for a few minutes. This clearly demonstrates that most files are currently unprotected at rest.

Commands:

```
tar cvfz files_locked.tar.gz * > unprotected_files.txt 2> protected_files.txt
wc -l protected_files.txt
wc -l unprotected_files.txt
```

Output:

```
3 protected_files.txt
229 unprotected_files.txt
```

To mitigate this issue, Cure53 recommends integrating the Data Protection capability at application level[8]. This will ensure that application data files are protected at rest with the strongest form of encryption available on iOS: *NSFileProtectionComplete*[9]. Furthermore, in order to protect cached entries, one can subclass *NSURLCache* with a custom version that stores URL responses in a custom SQLite database with file protection set

[8] https://developer.apple.com/documentation/.../com_apple_developer_default-data-protection
[9] https://developer.apple.com/documentation/foundation/nsfileprotectioncomplete

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

to *NSFileProtectionComplete*[10]. Alternatively, before the request is sent, caching could be disabled with a code snippet similar to the one shown below.

Proposed fix (before request sent):

```
configuration.requestCachePolicy = .reloadIgnoringCacheData
```

An alternative mitigatory action could be to clear all cached responses after the response is received.

Proposed fix (after response received):

```
URLCache.shared.removeAllCachedResponses()
```

In addition to the above, *SQL Cipher*[11] could be considered to encrypt SQLite databases at rest. The encryption key should be stored in the iOS keychain while data remains protected. For additional mitigation guidance, please feel free to peruse the blog post entitled *Best practices to avoid security vulnerabilities in your iOS app*[12].

This vulnerability is rated a **4.4**

View the breakdown of each component of the scoring below:
https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:N

CWE: https://cwe.mitre.org/data/definitions/200.html

EXP-11-009 WP1: Magic login via *launch-app* deep link facilitates DoS *(Low)*

***Fix Note**: The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

During a deep-dive review of the iOS app's implemented deep links, the discovery was made that the application suffers from a client-side Denial-of-Service attack. ExpressVPN provides the option to authenticate users via a magic login link that contains an activation token. In the eventuality an invalid token is sent to the application via the *launch-app* path's *activation_token* parameter, the implemented feature will not sufficiently handle it. As a result, a loading spinner is displayed preventing users from a prolonged engagement with the product, such as connecting to the VPN or using the password manager.

[10] https://stackoverflow.com/questions/27933387/nsurlcache-and-data-protection
[11] https://www.zetetic.net/sqlcipher/ios-tutorial/
[12] http://blogs.quovantis.com/best-practices-to-avoid-security-vulnerabilities-in-your-ios-app/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Affected file:
xv_iphone/SuperXV/HomeViewController.m

Affected code:

```
(void)handleMagicLoginToken:(NSString*)token {
[...]
   [self showProgressView:@""];
   [XCClientManager.instance checkIfDifferentAccountThanToken:token
      withCompletionHandler:^(BOOL isDifferent) {
         if (!isDifferent) {
            // Deep link is related to current account
            [...]
         }
[...]
```

The following PoC underlines the method by which a malicious app or page would be able to force the app showing the spinner animation via opening the example deep link.

PoC link:
expressvpn://app-bus/launch-app?activation_token=ABC

Steps to reproduce:

1. Open the ExpressVPN app and sign in with your user.
2. Click the menu on the VPN tab.
3. Open the example deep link provided above.

However, since the attack can only be instigated with two prerequisites — namely the device must constitute an iPhone and the previous tab opened must constitute *VPN* — any tangible impact is considered relatively low. Nevertheless, Cure53 recommends improving all error-handling processes in general, for example when a non-functional token is sent via the aforementioned deep link. By doing so, an additional check should ensure that the application halts displaying the spinner animation in the eventuality an error has occurred.

This issue is rated a **3.6**

View the breakdown of each component of the scoring below:
https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:C/C:N/I:N/A:L

CWE: https://cwe.mitre.org/data/definitions/703.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not lead to an exploit but might 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 way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

EXP-11-002 WP1: Absent jailbreak detection *(Info)*

***Note from ExpressVPN**: To support users who choose to jailbreak their devices, the ExpressVPN iOS application will continue to be supported on jailbroken devices. We've added a warning for users who choose to do this to make them aware of the potentially increased security issues that arise from a jailbroken device.*

Testing confirmed that the iOS app does not currently implement any form of jailbreak detection at the time of writing. As a result, the app does not alert users to the security implications of running the app in an environment of this nature. This issue can be confirmed by installing the app on a jailbroken device and validating the app's complete lack of warning. To mitigate this issue, Cure53 recommends implementing a comprehensive jailbreak solution. Notably, the application will always remain at a disadvantage since the user holds root access and the application does not. Mechanisms such as these should always be considered circumventable in the hands of a skilled and dedicated attacker.

Some freely available libraries for iOS constitute *IOSSecuritySuite*[13] and *DTTJailbreakDetection*[14], though custom checks are also possible in Swift applications[15]. These solutions should be considered bypassable but will suffice for the purpose of warning users regarding the risk of operating the application on a jailbroken device. For an optimal outcome in this scenario, one can recommend testing commercial and open source solutions against well-known *Cydia tweaks*, such as *LibertyLite*[16], *Shadow*[17], *tsProtector 8+*[18], and *A-Bypass*[19]. Based on the results gathered, the ExpressVPN team could subsequently determine the most secure approach for this framework.

This issue is rated a **0.0**

[13] https://cocoapods.org/pods/IOSSecuritySuite
[14] https://github.com/thii/DTTJailbreakDetection
[15] https://sabatsachin.medium.com/detect-jailbreak-device-in-swift-5-ios-programatically-da467028242d
[16] http://ryleyangus.com/repo/
[17] https://ios.jjolano.me/
[18] http://apt.thebigboss.org/repofiles/cydia/
[19] https://repo.rpgfarm.com/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

View the breakdown of each component of the scoring below:
https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

CWE: https://cwe.mitre.org/data/definitions/356.html

EXP-11-004 WP1: WebView weaknesses via *SFSafariViewController* usage *(Info)*

***Fix Note**: The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

During the code review, the discovery was made that the iOS app currently utilizes *SFSafariViewController*. Unfortunately, this WebView component cannot disable JavaScript, follows HTTP redirects, shares cookies and other website data with Safari, and cannot be obfuscated by other views or layers, which defeats any security-screen protections otherwise correctly implemented by the iOS app. The root cause for this issue originates from the following file:

Affected file:
xv_iphone/SuperXV/Library/SXVWebViewController.swift

Affected code:

```
import SafariServices
final class SXVWebViewController: SFSafariViewController {
    override init(url URL: URL, configuration:
SFSafariViewController.Configuration) {
      super.init(url: URL, configuration: configuration)
      customizeUI()
}
    init(url URL: URL) {
       let configuration = SFSafariViewController.Configuration()
       super.init(url: URL, configuration: configuration)
       customizeUI()
    }
    private func customizeUI() {
       preferredControlTintColor = .fn_primaryTint
       preferredBarTintColor = .fn_primaryBackground
   }
}
```

To mitigate this issue, Cure53 advises replacing the current *SFSafariViewController* implementation with the safer and more performant *WKWebView*[20]. Amongst other benefits, *WKWebViews* permits disabling JavaScript, does not share cookies or other website data with Safari, and can be obfuscated by other views or layers.

[20] https://developer.apple.com/documentation/webkit/wkwebview

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

This issue is rated a **0.0**

View the breakdown of each component of the scoring below:
https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

CWE: https://cwe.mitre.org/data/definitions/657.html

EXP-11-006 WP1/3: Memory corruption weaknesses via insecure functions *(Info)*

***Fix Note**: The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

Testing confirmed that the ExpressVPN application leverages certain functions that can lead to memory corruption vulnerabilities. Whilst this weakness does not appear to be easily exploitable at the time of testing, the application's attack surface is unnecessarily increased as a result and could facilitate greater potential for memory corruption vulnerabilities in the future. This issue was identified in the following files during the code review:

Affected file:
xv_iphone/Trident/Sources/SharedLib/XVClientBridge.m

Affected code:

```
xc_global_options global_opts;
memset(&global_opts, 0, sizeof(global_opts));
memcpy(global_opts.obfskey, key, XC_GLOBAL_OBFS_KEY_SIZE);
global_opts.os_name = xc_os_name_ios;
global_opts.os_version = systemVersion.UTF8String;
global_opts.client_version = appVersion.UTF8String;
global_opts.installation_id = installID.UTF8String;
global_opts.app_variant = [self getAppVariant];
memcpy(global_opts.data_file_key, dataFileKey.bytes, dataFileKey.length);
memcpy(global_opts.data_file_iv, dataFileIV.bytes, dataFileKey.length);
int rc = xc_global_init(&global_opts);
if (rc != 0) {
[...]
```

Affected file:
xv_iphone/Trident/Sources/Lightway Bridge/HeliumTransportUDP.m

Affected code:

```
if ((msg.msg_flags & MSG_TRUNC) || (msg.msg_flags & MSG_CTRUNC)) {
   [self posixError:@"Control message truncated"];
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
   return;
}
struct cmsghdr* cmsg;
uint64_t timestamp = 0;
for (cmsg = CMSG_FIRSTHDR(&msg); NULL != cmsg; cmsg = CMSG_NXTHDR(&msg, cmsg)) {
   if (cmsg->cmsg_len == CMSG_LEN(sizeof(uint64_t)) && cmsg->cmsg_level ==
SOL_SOCKET &&
      cmsg->cmsg_type == SCM_TIMESTAMP_MONOTONIC) {
         memcpy(&timestamp, CMSG_DATA(cmsg), sizeof(uint64_t));
         break;
      }
}
```

Affected file:
xv_iphone/Trident/Sources/Lightway Bridge/HeliumMultiClient.m

Affected code:

```
static void HeliumMuxDebugLogCallback(he_mux_client_t *mux, void *context, const
char *fmt, ...) {
   // Get our context back
   HeliumMultiClientImpl *client = (__bridge HeliumMultiClientImpl *)(context);
   char *buf = NULL;
   va_list va;
   va_start(va, fmt);
   vasprintf(&buf, fmt, va);
   va_end(va);
   [client debugLog:[NSString stringWithFormat:@"Balloon: %s", buf]];
   free(buf);
}
```

Affected file:
xv_libballoon/libballoon-filter/src/packet_filter.c

Affected code (*strtok*):

```
char *tok = strtok(p, " ");
tok = strtok(NULL, "\n");
tok = strtok(p, "\n");
```

Additionally, another known weakness persists via usage of the *strncpy* function. In the eventuality not all characters are copied from a source string, the function will not append a null character by default[21]. Since alternative functions such as *strlen* rely on the null-byte termination in strings, the function would search for the next null-byte available in memory. This behavior can incur significant issues — such as buffer overflows — if integrated to other elements in an incorrectly validated manner. However, in the following example, the *len* parameter is not utilized for initializing or copying data.

[21] https://devblogs.microsoft.com/oldnewthing/20050107-00/?p=36773

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Affected code (strncpy):

```
strncpy(buf, domain, MAX_DOMAIN_LENGTH - 1);
size_t len = strlen(buf);
```

Where possible, Cure53 advises avoiding C functions with known potential to introduce memory corruption vulnerabilities such as *memcpy*, *strcpy*, *strncpy, strcat*, and similar. Subsequently, protection for the remaining cases could be integrated via usage of the *-fstack-protector-all* and *-D_FORTIFY_SOURCE=2* compiler flags. One can additionally recommend replacing the *strtok* function with its *strtok_r*[22] alternative. Moreover, if *strncpy* is used, it should be ensured that the resulting string always appends a null-byte character. This can be achieved manually or via other functions such as *snprintf* or *strlcpy*. For additional mitigation guidance, please feel free to refer to the Apple documentation article *Avoiding Buffer Overflows and Underflows*[23].

This issue is rated a **0.0**

View the breakdown of each component of the scoring below:
https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

CWE: https://cwe.mitre.org/data/definitions/119.html

EXP-11-007 WP1/2: Clear-text password storage in iOS Keychain *(Info)*

***Fix Note**: The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

Testing confirmed that the application stores the master password for the ExpressVPN Keys functionality in clear-text in the keychain. Storage of user passwords in clear-text is considered a negative practice if implemented on either the client- or server-side.

A malicious attacker able to gain access to the contents of the iOS keychain may leverage this weakness to enumerate the master password, which some users may reuse for other online services. This scenario may be successfully instigated if attackers have physical access to an unlocked device and are able to provide a valid fingerprint (i.e. stolen via silicon techniques) or Face ID (i.e. forcing the victim user).

Please note that the exploitability of this issue is drastically reduced due to adequate storage in the iOS keychain with an optimal access control of

[22] https://wiki.sei.cmu.edu/…/c/STR06-C.+Do+not+assume+that+strtok%28%29+leaves+the+parse+strin
[23] https://developer.apple.com/.../Security/Conceptual/SecureCodingGuide/Articles/BufferOverflows.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly[24]. The clear-text password was located at runtime in the iOS Keychain as follows:

Level of Access	Field	Value
`WhenPasscodeSetThisDeviceOnly`	`AutomaticUnlockMasterPassword`	`cure53[...]`

The root cause for this issue appears to be the lack of encryption of the master password, as can be deduced in the following code:

Affected file:
xv_iphone/SuperXV/PasswordManager/Library/PWMBiometricAuthenticator.swift

Affected code:

```
// MARK: - Keychain Methods - Biometric Context

  private func keychainQueryForStoredMasterPassword(context: LAContext) ->
[String: Any] {
    let account = "AutomaticUnlockMasterPassword"
    let server = "pmgr.expressvpn.com"
    let accessGroup = "group.com.expressvpn.iosvpn"

    return [
      kSecClass as String: kSecClassInternetPassword,
      kSecAttrAccount as String: account,
      kSecAttrServer as String: server,
      kSecUseAuthenticationContext as String: context,
      kSecAttrAccessGroup as String: accessGroup,
    ]
  }
```

The primary issue here pertains to the fact that the user password is stored in clear-text in the iOS Keychain, hence access to this data provides access to the account until the password is altered. A more secure approach would be to generate a lengthy and random client-side token that the user transparently logs in with. This token should be stored in the iOS Keychain rather than the user password, but could also be encrypted with the user password for additional security. A similar approach is used by Veracrypt[25] for comprehensive disk encryption, thereby one can recommend utilizing a strong encryption or library such as *cryptokit*[26] to store information of this nature.

[24] https://developer.apple.com/documentation/security/ksecattraccessibleafterfirstunlockthisdeviceonly
[25] https://www.veracrypt.fr/en/Encryption%20Scheme.html
[26] https://developer.apple.com/documentation/cryptokit/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

For additional mitigation guidance, please feel free to refer to the *OWASP Password Storage Cheat Sheet*[27].

This issue is rated a **0.0**

View the breakdown of each component of the scoring below:
https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

CWE: https://cwe.mitre.org/data/definitions/200.html

EXP-11-008 WP1: Potential WebView XSS via insufficient sanitization *(Info)*

***Fix Note**: The issue was addressed by the ExpressVPN team and the fix was verified by Cure53 who were able to review the related diff & PR. The issue no longer exists.*

Testing confirmed that the ExpressVPN iOS app renders diagnostic information via means that could result in XSS from a privileged context. The app displays stored VPN logs via the *loadHTMLString:formattedHTML*[28] method, which is prone to XSS vulnerabilities. Additionally, the WebView utilizes a *nil baseURL* parameter, which could allow attackers to access local system files from JavaScript. Notably, the exploitation potential of this issue remains relatively low since VPN logs are unlikely to contain user input that in turn contains XSS payloads. Cure53 nevertheless recommends resolving this weakness to eliminate any risk potential originating from this attack vector. Specifically, the present issue was detected in the following location during the code review:

Affected file:
xv_iphone/SuperXV/VPNLogViewController.m

Affected code:

```
action:@selector(copyPressed:)];
    [self.navigationItem setRightBarButtonItem:copyButton];
    self.webView.navigationDelegate = self;
}
    [...]    // Save the full vpn content in self.content
    [content appendString:vpnContent];
    self.content = content;
    // Update the UI
    NSString* formattedHTML = [NSString stringWithFormat:@"<html><body><pre>
%@</pre></body></html>", uiContent];
    [self.webView loadHTMLString:formattedHTML baseURL:nil];}[...]
```

[27] https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
[28] https://developer.apple.com/documentation/webkit/wkwebview/1415004-loadhtmlstring

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

To mitigate this issue, Cure53 recommends implementing a TextView rather than WebView for these purposes to completely eliminate the possibility of XSS. Alternatively, VPN logs should be sanitized with adequate output-encoding prior to concatenating them into the WebView HTML. Finally, if possible, JavaScript should be disabled on the *WKWebView* and the *baseURL* parameter should be set to a value other than *nil* or *file:///[...]* to avoid potential file access from JavaScript in the event of XSS.

This issue is rated a **0.0**

View the breakdown of each component of the scoring below:
https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

CWE: https://cwe.mitre.org/data/definitions/79.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

The impressions gained during this report - which details and extrapolates on all findings identified during the CW35 and CW36 testing against the ExpressVPN iOS mobile app by the Cure53 team - will now be discussed at length. To summarize, the confirmation can be made that the components under scrutiny have garnered a positive impression, with the vast majority of significant attack vectors successfully deterred and only a relatively minimal yield of findings documented.

To provide context on the test setup, three members of the Cure53 team completed the project over the course of sixteen days in August and September 2022. A total of nine issues were detected during the audit, four of which were considered exploitable and assigned to the *Vulnerabilities* section, whilst the remaining five were merely considered hardening recommendations or best practice implementations and were therefore assigned to the *Miscellaneous* section.

To ensure a smooth audit, Cure53 was provided with test-user accounts and a functional ExpressVPN iOS application constituting version 11.58.0 (116835). Sources for the app and corresponding dependencies were also shared. This significantly increased the effectiveness of the assessment, allowing the testing team to thoroughly review the application for potential security vulnerabilities proliferating in the code and running environments.

The primary objective of Cure53 investigation's of the iOS application was to determine whether the existing functionality of the application and its connected endpoints and environments can be deemed healthy enough to withstand attacks by malicious users and third-party applications. With a specific focus on commonly-found mobile application issues relating to various types of injection attacks and misconfigurations — which could compromise the application or the authenticated user — were investigated without significant success.

As a result, only four exploitable issues were detected, three of which specifically related to information disclosure. Testing confirmed that the iOS app would particularly benefit from implementing a security screen overlay to prevent disclosure via either screenshots and or background state (see EXP-11-001 for further guidance). This is a known issue for mobile security in general, therefore any mitigatory actions applied in this regard would undoubtedly increase ExpressVPN user privacy.

Elsewhere, another issue was identified relating to custom deep-link scheme usage, which facilitates iOS URL scheme hijacking attacks and could lead to both information disclosure and user-account compromisation. Toward this, the development team should adhere to the guidance proposed in ticket EXP-11-003.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Regarding the hardware-backed security enclave, the discovery was made that the iOS keychain is utilized correctly, though user credentials should not be stored in clear-text in the keychain as stipulated in ticket EXP-11-007. The final issue in the area of information disclosure relates to improper file protection. In this regard, VPN settings and alternative information would be better protected at rest by implementing the available data protection features in iOS, as documented in ticket EXP-11-005.

Conversely, other aspects related to information disclosure garnered a positive impression, since achieving a connection to a user's real IP was deemed impossible. Despite the known WebRTC leak which has been addressed by ExpressVPN, no additional side-channel leaks were detected, which is certainly a praiseworthy outcome.

Deep-dive evaluations into deep-link processing were also conducted, which subsequently revealed an issue concerning Denial-of-Service attacks. In light of this, Cure53 strongly recommends hardening the application against threats of this nature by introducing a more reliable error-handling process, as detailed in ticket EXP-11-009.

The testing team also observed some leeway for minor hardening recommendations, including configurations and feature usage that could be improved to better protect users from potential memory corruption vulnerabilities in iOS (see EXP-11-006), as well as weaknesses that may facilitate XSS attacks (see EXP-11-004 and EXP-11-008) and detection of jailbroken devices (see EXP-11-002).

Additionally, the password manager integration and associated features were examined in-depth by Cure53. Positively, no client-side issues were detected in the UI components. The matching of domains, subdomains, and the integration of the Autofill feature within the iOS ecosystem were also soundly handled in general.

The examined codebase of the iOS application and dependencies in scope garnered a solid impression from a security perspective. This strong foundation is corroborated by the fact that the codebase adheres to common best practices. In particular, testing was initiated to determine whether the application leveraged dangerous functions or implemented common pitfalls that could lead to major vulnerabilities, such as additional injection-based attacks, path traversal issues, arbitrary file handling, or common erroneous behaviors such as XXE or deserialization attacks. Static analysis tooling seems integrated into the lifecycle also, which further reduces any error potential in this regard.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Aside from the aforementioned, the iOS application provided a number of further positive impressions during this assignment that are worthy of mention here:

- No leaks were located via log messages or locked screens.
- No encrypted credentials or application secrets were identified.
- The application correctly disables clear-text HTTP communications.
- The application correctly validates SSL certificates.

In summation, Cure53 strongly advises addressing all issues identified in this report where possible, including those considered *Informational* and *Low*. This will not only strengthen the security posture of the platform, but also reduce the number of tickets documented in future security engagements. To conclude, the ExpressVPN mobile application for iOS plus related dependencies and features in scope garnered an adequately solid impression in relation to security.

Despite the four identified vulnerabilities, the average impact marker merely hovered between *Low* and *Medium*, which indicates stable protection against attacks targeting the examined ExpressVPN application. This outcome provides ample evidence that the ExpressVPN team is not only acutely aware of the myriad problems that modern iOS applications tend to face, but has also successfully implemented counteractive measures to repel them.

However, the testing team also observed some leeway for improvement following the completion of this audit, particularly in relation to information disclosure, as mentioned previously. It's important to note that this information disclosure requires physical device access, or the installation of a malicious application. This is considered an erroneous and recurring antipattern that primarily persists due to misconfiguration and lack of best-practice adherence.

Once all issues detailed in this report have been addressed and mitigated, Cure53 would be pleased to confirm that the audited versions of the examined application and dependencies in scope are sufficiently safeguarded for production use.

Cure53 would like to thank Brian Schirmacher, Steve Lin, Sam Bultez, and Elie Jacquelin from the ExpressVPN team for their excellent project coordination, support and assistance, both before and during this assignment.