windscribe

2022 - VPN Source Code Review

FINAL REPORT

limitless innovation. no compromise.

Prepared for: Yegor Sak

Windscribe Limited

March 04, 2022

Version:	Final
Prepared for:	Windscribe Limited
Date:	March 04, 2022

Table of Contents

Executive Summary

Windscribe Limited engaged Leviathan Security to perform a time-bound security assessment of its iOS/iPadOS and Android applications. We performed this assessment from February 11, 2022 through March 4, 2022.

Our objectives were to review both applications for three primary areas of concern in advance of the forthcoming open source release of both applications:

- License violations, in particular for GPL and other "viral" or "copyleft" licenses;
- Code quality issues; and
- Code security issues.

Our review uncovered 3 high, 3 medium, and 3 low-severity findings, as well as 7 informational findings.

Observations

Our results were broadly similar to the results of our previous assessment of Windscribe's desktop applications. The most similar issues were general license noncompliance across the board (both in licenses that impose substantial conditions, such as GPL, and those that require only notice, such as MIT and BSD), and a near-complete lack of comments in the code. As with our previous assessment, since code quality was explicitly in scope, we have provided a large number of "Informational" findings describing quality issues with the code that did not lead directly to a security or privacy issue.

As is common for modern application development, the Windscribe applications use many open source dependencies to minimize the amount of new code required. Due to the number of the dependency lists, we have chosen to omit most of the long lists of dependencies with license or other concerns from this document, and will provide them in a separate document.

The most critical findings all have to do with a common anti-pattern in Windscribe code: static secrets committed to source code. These examples range from keysigning certificates to inputs to hash functions, but it is clear that Windscribe's engineering team needs effective tooling around secret management, as well as training around generating and handling secrets used in sensitive areas of code.

Recommendations

As the Windscribe applications use well-understood and well-tested libraries for their core VPN functionality, we suspect most of the library vulnerabilities are unlikely to have substantial customer effect even if exploited. As Windscribe intends to open source this code for transparency’s sake, we recommend that dependencies be updated to the newest versions in all cases to avoid misinterpretation by the public, in addition to any security benefits. In addition, while Informational findings do not carry substantial security impact,, we recommend that all issues be remediated ahead of any release, particularly with regard to secret handling (which is likely to be an area of interest by the community).

In the future, we recommend that Windscribe's server infrastructure undergo a full application security assessment, including both code review and dynamic testing, to help to ensure that this most critical component of Windscribe's service has been reviewed.

Vulnerability Classification

Impact

When we find a vulnerability, we assign it one of five categories of severity, describing the potential impact if an attacker were to exploit it:

Informational – Does not present a current threat but could pose one in the future if certain changes are made. To protect against future vulnerabilities, fixing the condition is advisable.

Low – May allow an attacker to gain information that could be combined with other vulnerabilities to carry out further attacks. May allow an attacker to bypass auditing or minimally disrupt availability, resulting in minor damage to reputation or financial loss.

Medium – May allow an attacker inappropriate access to business assets such as systems or servers. There may be impact to the confidentiality or integrity of data, or limited disruption of availability, resulting in moderate damage to reputation or financial loss.

High – May allow an attacker inappropriate access to business assets such as systems or servers. There may be substantial or widespread impact to the confidentiality or integrity of particularly sensitive data, or disruption of availability, resulting in significant damage to reputation or financial loss.

Critical – May allow an attacker to gain persistence, or imminently disrupt functionality or disclose data, resulting in severe reputational damage or financial loss.

Skill Level to Exploit

When we find a vulnerability, we assess how skilled an attacker must be to exploit it:

Simple – Requires minimal understanding of the underlying technology. Tools/ techniques for exploiting the vulnerability can be easily found on the internet.

Moderate – Requires significant expertise, possibly in proprietary information, or access to tools that are not readily available to individuals. The unwitting cooperation of a victim or target may also be required.

Advanced – Requires insider access or access to tools that are not publicly available. Successful exploitation of another vulnerability may be required. Direct interaction with the victim or target may also be required.

Impact Rating (Weight)	Skill Level to Exploit Rating (Weight)		
	Advanced (1)	Moderate (2)	Simple (3)
Critical (4)	4	8	12
High (3)	3	6	9
Medium (2)	2	4	6
Low (1)	1	2	3

Severity	
Critical	10-12
High	7-9
Medium	4-6
Low	1-3

Vulnerability Index

This section represents a quick view into the vulnerabilities discovered in this assessment.

ID	SEVERITY	TITLE	COMPONENT
104235	High	Use of hard-coded credentials enables business method bypass	Windscribe iOS App
104369	High	License violations: AGPL3, GPL2, CDDL, EPL, MPL, MIT, BSD, Apache, ...	Windscribe Android App
104432	High	Hard-Coded Signing Credentials	Windscribe Android App
104182	Medium	Secrets in source code	Windscribe iOS App
104226	Medium	Hashed domain function contains only 2 bits of randomness	Windscribe iOS App
104394	Medium	Failure to validate SSL certificate	Windscribe Android App
104321	Low	Use of MD5 throughout	Windscribe iOS App
104388	Low	Unnecessary QUERY_ALL_PACKAGES permission	Windscribe Android App
104389	Low	Improper exception handling	Windscribe Android App
104158	Info	Debug symbol disclosure: OpenVPN on Android	Windscribe Android App
104183	Info	Hash length extension vulnerability	Windscribe iOS App
104262	Info	Total lack of comments	Windscribe Android App
104374	Info	Unmaintained dependencies in use	Windscribe Android App
104392	Info	Use of hard-coded credentials	Windscribe Android App
104393	Info	Use of MD5 throughout	Windscribe iOS App
104430	Info	Reimplementation of language-provided features	Windscribe Android App

Activity Index

This section represents a quick view into the activities performed in this assessment.

COMPONENT	TITLE	STATUS
Windscribe Android App	Android code quality issues	Complete
Windscribe Android App	Android security issues	Complete
Windscribe Android App	Android license issues	Complete
Windscribe iOS App	iOS security issues	Complete
Windscribe iOS App	iOS license issues	Complete
Windscribe iOS App	iOS code quality issues	Complete

Observations & Analysis

For the purposes of evaluation, we separated this project into several components based on design documentation and discussions with the service team.

Project Components & Descriptions

We have broken the project down into 2 high-level components below.

WINDSCRIBE ANDROID APP	The Windscribe Android application allows users to connect to Windscribe's various VPN endpoints from various Android devices, including both devices using Google's Play Store and Amazon's "Appstore for Android."
WINDSCRIBE IOS APP	The Windscribe iOS application allows users to connect to Windscribe's various VPN endpoints from the iOS and iPadOS operating systems.

Windscribe Android App

The Windscribe Android application allows users to connect to Windscribe's various VPN endpoints from various Android devices, including both devices using Google's Play Store and Amazon's "Appstore for Android."

Threat Analysis

Like each of Windscribe's apps, a failure in the Windscribe Android app's security is not necessarily catastrophic for the security of its users, since the app merely wraps well-tested VPN protocols (OpenVPN, Wireguard, etc.); instead security failures tend to cause business method problems.

Activities Performed

ANDROID CODE QUALITY ISSUES

Scope

Validate that issues of code quality (not presenting other issues, such as security issues) are not present in the application.

Methodology

Manual code review.

Observations

Related Findings

104432: Hard-Coded Signing Credentials
104226: Hashed domain function contains only 2 bits of randomness
104389: Improper exception handling
104158: Debug symbol disclosure: OpenVPN on Android
104262: Total lack of comments
104374: Unmaintained dependencies in use
104392: Use of hard-coded credentials
104393: Use of MD5 throughout
104430: Reimplementation of language-provided features

ANDROID SECURITY ISSUES

Scope

Validate that obvious security vulnerabilities, weaknesses, design flaws, and bad patterns are not present in the application.

Methodology

Manual code review.

ANDROID SECURITY ISSUES

Observations

Related Findings

104432: Hard-Coded Signing Credentials
104226: Hashed domain function contains only 2 bits of randomness
104394: Failure to validate SSL certificate
104388: Unnecessary QUERY_ALL_PACKAGES permission
104389: Improper exception handling
104374: Unmaintained dependencies in use
104392: Use of hard-coded credentials
104393: Use of MD5 throughout
104430: Reimplementation of language-provided features

ANDROID LICENSE ISSUES

Scope

Validate that issues of license compliance (e.g., hostile OSS licenses) are not present.

Methodology

Manual code review with automated assistance for dependencies.

Observations

Related Findings

104369: License violations: AGPL3, GPL2, CDDL, EPL, MPL, MIT, BSD, Apache, ...

Vulnerabilities

LICENSE VIOLATIONS: AGPL3, GPL2, CDDL, EPL, MPL, MIT, BSD, APACHE, ...

ID	104369
Component	Windscribe Android App
Severity	**High**
Impact / Skill Level	High/Simple
Reference	https://tldrlegal.com/license/gnu-affero-general-public-license-v3-(agpl-3.0) https://tldrlegal.com/license/gnu-general-public-license-v2 https://tldrlegal.com/license/boost-software-license-1.0-explained https://tldrlegal.com/license/bsd-3-clause-license-(revised) https://tldrlegal.com/license/bsd-2-clause-license-(freebsd) https://tldrlegal.com/license/ruby-license-(ruby) https://tldrlegal.com/license/apache-license-2.0-(apache-2.0) https://tldrlegal.com/license/mit-license
Location	Throughout

Observation

Intellectual property violations can lead to significant liability, including an injunction to prevent distribution or use of the affected code. While open source licenses are fairly permissive in what they allow a user to do with source code, violation of an open source license places an entity in the same position it would be if it had violated any other kind of software license.
The Windscribe iOS application does not appear to comply with the license terms of any of the software it uses. The following libraries are licensed under the stated open source licenses:

AGPL3
- OpenVPNAdapter

GPLv2
- LZ4 (in OpenVPNAdapter)
- OpenVPN (in OpenVPNAdapter)

Boost
- ASIO (in OpenVPNAdapter)

Apache
- Addressable
- Realm

2-Clause BSD
- Rexml

- CocoaLumberjack

3-Clause BSD

- Escape
- FFI
- LZ4 (in OpenVPNAdapter)

Ruby License

- HTTPClient
- JSON

MIT License

- CFPropertyList
- ActiveSupport
- AlgoliaSearch
- Atomos
- Claide
- CocoaPods (and plugins)
- Colored2
- Concurrent-Ruby
- Ethon
- FourFlusher
- Fuzzy_Match
- GH_Inspector
- i18n
- Minitest
- Molinillo
- Nanaimo
- Nap
- Netrc
- Public_Suffix
- Ruby-Macho
- Typhoeus
- Tzinfo
- XCodeproj
- Zeitwerk
- ExpyTableView
- IQKeyboardManagerSwift
- JNKeychain

LICENSE VIOLATIONS: AGPL3, GPL2, CDDL, EPL, MPL, MIT, BSD, APACHE, ...

- ReachabilitySwift
- mbedTLS (in OpenVPNAdapter)

Many of these licenses require, essentially (though this is not legal advice), that Windscribe convey the copyrights and license terms of the included libraries along with applications that use the libraries; this is often solved in software through including a license viewer in the application. However, the AGPLv3 license is a "viral" license that, when software so licensed is compiled into a larger whole, imposes significant restrictions upon how the whole can be licensed and how it can be conveyed. Since Windscribe has already conveyed software including AGPLv3-licensed libraries to the public, this needs to be resolved with the assistance of counsel to guide you.

In addition, one project, AES256Encrypter, does not have a license at all on its site. https://github.com/dhilowitz/AES256Encrypter That may make it unable to be included in your project on any terms other than a negotiated license with the author.

The Android application has more than five hundred dependencies, and listing them all here would unnecessarily prolong this finding; a copy of our results in this area will be provided to the client separately. Licenses in use include:

Viral licenses (discuss whether significant obligations have already been incurred with counsel):
- GPLv2
- CDDL
- EPL
- MPL

Non-viral licenses:
- BSD
- MIT
- Apache
- ICU
- LGPL
- SAX-PD

Impact Rationale:
Violation of software licenses can lead to enormous liability, particularly given the Affero GPL (AGPL) license.

Difficulty Rationale:
Detecting software libraries used by open source software is trivial.

LICENSE VIOLATIONS: AGPL3, GPL2, CDDL, EPL, MPL, MIT, BSD, APACHE, ...

Recommendation
Consult with legal counsel to determine cost to comply with each license. Use this information as an input for a cost-benefit discussion with the Product team to consider substituting other libraries, licensing code or writing work-alike code.

HARD-CODED SIGNING CREDENTIALS

ID	104432
Component	Windscribe Android App
Severity	**High**
Impact / Skill Level	High/Simple
Reference	http://cwe.mitre.org/data/definitions/798.html
Location	Android-App/config/contributor.jks Android-App/config/sign.properties

Observation

If hard-coded credentials, including private keys, are embedded in software, an attacker can use them for unintended purposes.
We observed that the software contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.

```
SIGN_KEY_ALIAS=contributor
SIGN_KEY_PASSWORD=vzy@umy_vwt9ANC2zja
SIGN_STORE_PASSWORD=vzy@umy_vwt9ANC2zja
SIGN_KEY_FILE=../config/contributor.jks
```

According to Windscribe employees, "this key can be used to sign and use this signed version upload on your own play store console and used for sideloading." An attacker could use this key to impersonate a legitimate Windscribe build for distribution as part of a targeted attack, e.g., dissidents or residents in a war zone.

Impact Rationale:
An attacker can distribute a compromised Windscribe application that will be cryptographically verifiable as legitimate.

Difficulty Rationale:
The attacker need only run the documented build process.

Recommendation

Remove and revoke these keys; they should be considered compromised, and not used for any purpose going forward. New keys used for this purpose internally must be stored in a key repository (e.g., Hashicorp Vault) and not committed to source control. To enable the intended functionality (allowing a developer to try a build "in release mode") in the open source build, provide instructions in the README to generate a new key file and provide configuration information in the **`sign.properties`** file.

FAILURE TO VALIDATE SSL CERTIFICATE

ID	104394
Component	Windscribe Android App
Severity	**Medium**
Impact / Skill Level	High/Moderate
Reference	http://cwe.mitre.org/data/definitions/322.html
Location	Android-App/base/src/main/java/com/windscribe/vpn/api/WindCustomApiFactory.kt

Observation

Client applications that fail to validate the SSL certificate they receive are vulnerable to man-in-the-middle (MITM) attacks.

```
private fun getUnsafeOkHttpClient(): OkHttpClient.Builder? {
        return try {
            // Create a trust manager that does not validate certificate chains
```

Used by

```
    /**
     * Creates custom api factory to allow
     * direct ip calls in case of Api failures. Loads prepacked ca cert in to trust manager
     * ads trust manager to okhttp client.
     */
    init {
        getUnsafeOkHttpClient()?.let {
```

This code only appears to be in use for IP address-only connections.

Impact Rationale:
This would allow an attacker in a network-adjacent or network-upstream position to intercept, inspect, and modify traffic between a Windscribe user and Windscribe servers in the limited case in which the app tried to use IP-based connections.

Difficulty Rationale:
Achieving adjacency or network upstream positioning from a target requires more than simple knowledge and/or targeting.

FAILURE TO VALIDATE SSL CERTIFICATE

Recommendation
Ensure the client verifies that the certificate is signed by a trusted authority and that the Common Name matches the expected value. For IP-only connections, use a TLS certificate with the IP address as a Subject Alternative Name.

UNNECESSARY QUERY_ALL_PACKAGES PERMISSION

ID	104388
Component	Windscribe Android App
Severity	**Low**
Impact / Skill Level	Low/Simple
Reference	http://cwe.mitre.org/data/definitions/250.html https://en.wikipedia.org/wiki/Principle_of_least_privilege https://support.google.com/googleplay/android-developer/answer/10158779
Location	Android-App/src/main/AndroidManifest.xml

Observation

Unnecessary risks are created when software operates with privileges greater than those necessary to perform its function in that if the software is compromised, the scope of potential damage is commensurately greater. This is a violation of the fundamental security principle of least privilege. For restricted APIs, accumulating excessive permissions may also subject the application to unnecessary oversight.

```
<uses-permission
        android:name="android.permission.QUERY_ALL_PACKAGES"
        tools:ignore="QueryAllPackagesPermission" />
```

It is unclear why the Windscribe app needs this; in addition, it is unclear if Windscribe's use falls within Google's restrictive requirements for use of the permission, which will begin enforcement on April 1, 2022.

Impact Rationale:
Additional permissions slightly increase the value of exploiting the Windscribe app, and the privacy harms to a Windscribe user if the app is exploited. However, this permission does not increase the ease of exploiting the application.

Difficulty Rationale:
There is no particular difficulty associated with exploiting data retrieved with this permission beyond exploiting the app itself.

Recommendation

Run code using the least privileges possible. If this permission is required, prepare to justify that requirement to Google on or before April 1, 2022; if it is not, remove it from the manifest.

IMPROPER EXCEPTION HANDLING

ID	104389
Component	Windscribe Android App
Severity	**Low**
Impact / Skill Level	Low/Moderate
Reference	http://cwe.mitre.org/data/definitions/755.html
Location	Android-App/base/src/main/java/androidx/core/app/JobIntentWorkAroundService.java Android-App/base/src/main/java/com/windscribe/vpn/api/SslUtils.kt

Observation

When exceptions are not caught by context-specific exception handlers, the exception falls through to the default exception handler. Because the default exception handler has almost no context for the exception, it produces incomplete audit trails and can make discovery and debugging of security flaws more difficult. Generic exception handling can also exacerbate the severity of existing vulnerabilities by preventing the application from crashing due to otherwise-fatal flaws, such as segmentation faults.
We found two examples of ignored/unhandled exceptions in relatively security-impactful contexts:

```
try {
    return super.dequeueWork();
} catch (SecurityException ignored) {
}
return null;
```

```
        var keystore: KeyStore? = null
        if (BuildConfig.API_STATIC_CERT.isNotEmpty()) {
            var inputStream: InputStream? = null
            try {
                keystore = KeyStore.getInstance("PKCS12")
                val password = "XXXX".toCharArray()
                inputStream =
Base64.decode(BuildConfig.API_STATIC_CERT).inputStream()
                keystore.load(inputStream, password)
            } catch (e: KeyStoreException) {
                e.printStackTrace()
            } catch (e: NoSuchAlgorithmException) {
                e.printStackTrace()
            } catch (e: IOException) {
                e.printStackTrace()
            } catch (e: CertificateException) {
                e.printStackTrace()
            } finally {
                if (inputStream != null) {
```

```
                    try {
                        inputStream.close()
                    } catch (e: IOException) {
                        e.printStackTrace()
                    }
                }
            }
        }
        return keystore
    }
```

In each of these cases, it is likely that if an exception happens, the calling function will not be made aware, and there will not be a useful exception log made, making it difficult to recover in a sane and secure way.

Impact Rationale:
The two instances in which this antipattern are used are likely to have little security impact; however, this could have higher impact if used elsewhere.

Difficulty Rationale:
Exploitation of error handlers is moderately complex, depending upon the context in which they exist.

Recommendation

Ensure that exceptions are handled in a consistent manner, and not "swallowed" or ignored without further comment. Always check return values of functions and ensure that error-indicating return values are always documented. Do not use "blanket" exception handlers to trap all exceptions.

DEBUG SYMBOL DISCLOSURE: OPENVPN ON ANDROID

ID	104158
Component	Windscribe Android App
Severity	**Info**
Impact / Skill Level	Informational/Simple
Reference	https://github.com/newren/git-filter-repo/ https://docs.github.com/en/repositories/working-with-files/managing-large-files/about-large-files-on-github#repository-size-limits
Location	Android-App/openvpn/.cxx

Observation

Compilers can insert debug information into programs that assist developers in debugging the software. However, debug information that is still present in a shipped or deployed product can assist an attacker in exploiting the product by giving them helpful information, such as function names, variable names, the name and file path of corresponding source code files, and the line number in that file on which a variable is declared.

`Android-App/openvpn/.cxx` contains 687 megabytes of debug information, and is checked into the Git repository. This greatly bloats the repository.

Impact Rationale:
There is no security impact to this finding; however, it significantly increases the size of the Git repository, and will be noticed by users who clone the Git repository once it has been open sourced.

Difficulty Rationale:
This issue is inherent to the Git repository as committed.

Recommendation

Remove this folder and edit the Git repository's history to remove its size entirely from what a user needs to clone to access the source code, using a tool such as **`git-filter-repo`**. Alternately, re-create the repository from scratch without this folder.

TOTAL LACK OF COMMENTS

ID	104262
Component	Windscribe Android App
Severity	**Info**
Impact / Skill Level	Informational/Simple
Reference	https://flylib.com/books/en/1.315.1.73/1/
Location	All source files

Observation

Comments are used for several purposes in source code:
- To explain to reviewers the assumptions made by the coder in a passage of code;
- To explain to other parties who may use the code how to do so successfully; and
- To explain to future maintainers how the code works to enable it to be kept up to evolving standards.

Not having comments undermines all three purposes and deeply hinders maintainability.
There are nearly no comments anywhere in the iOS code. According to the **`cloc`** tool, there are only 1998 lines of comments (compared to 3591 blank lines and 27325 lines of functional code) in the Swift parts of the Windscribe iOS app, excluding those parts written by Wireguard; of these, the vast majority are file headers automatically generated by the IDE, such as

```
//
//  EnterEmailViewController.swift
//  Windscribe
//
//  Created by Yalcin on 2019-02-20.
//  Copyright © 2019 Windscribe. All rights reserved.
//
```

From a qualitative perspective, there are essentially 0 useful/helpful headers either to describe function parameters or to describe code assumptions and flow anywhere in the codebase.

The same is true in the Android (Java/Kotlin) code. In 222 files, there are a total of 1028 lines of comments, compared with 14768 lines of functional code (and 2628 blank lines), and once again, the majority are copyright headers:

```
/*
 * Copyright (c) 2021 Windscribe Limited.
 */
```

Impact Rationale:

TOTAL LACK OF COMMENTS

There is no direct security impact of this issue; however, it leads to difficult-to-maintain code, which can lead to security impacts in the future.

Difficulty Rationale:
This issue is apparent to anyone reading the source code.

Recommendation

Add function header comments throughout the code to describe parameters, assumptions, and return values; add developer comments in key areas of code, such as authentication and authorization flows, to assist readers. Going forward, add comments to part of the code quality requirements to commit code to Windscribe repositories.

UNMAINTAINED DEPENDENCIES IN USE

ID	104374
Component	Windscribe Android App
Severity	**Info**
Impact / Skill Level	Informational/Simple
Reference	https://github.com/dhilowitz/AES256Encrypter https://github.com/segiddins/atomos
Location	iOS-App/Pods/AES256Encrypter iOS-App/gemfile.lock

Observation

Telemetry and forensic data show that computer systems (both operating systems and applications software) which have not been patched up to date are far more likely to be compromised than up-to-date systems because attackers reverse-engineer vendor patches, create automated scanners that search for unpatched systems, and then exploit any systems they discover.
`Atomos` and **`AES256Encrypter`** are included in the iOS Windscribe application's dependencies and appear to be essentially unmaintained. Atomos has had no changes for four years (all changes occurred less than eight months after its original commit), and still includes "welcome to your new gem" template language. AES256Encrypter has had no code changes for eight years (all changes occurred within one month of its original commit), and as noted in another finding, has no license allowing it to be included in other software.

On the Android side, a large number of dependencies were out of date and contain known, though not necessarily exploitable, vulnerabilities. A report will be provided to Windscribe separately.

Impact Rationale:
These libraries do not have known vulnerabilities, so there is no known security impact; however, in general, unmaintained libraries can lead to security impacts and/or technical debt.

Difficulty Rationale:
Anyone with access to the open sourced repository can trivially tell that these libraries are in use.

Recommendation

Replace unmaintained software with maintained alternatives, and do not run any software after its support period has expired. In these cases, remove these pieces of software.

USE OF HARD-CODED CREDENTIALS

ID	104392
Component	Windscribe Android App
Severity	**Info**
Impact / Skill Level	Informational/Simple
Reference	http://cwe.mitre.org/data/definitions/798.html
Location	Android-App/base/src/main/java/com/windscribe/vpn/constants/ApiConstants.kt Android-App/base/src/main/java/com/windscribe/vpn/api/AuthorizationGenerator.kt Android-App/base/src/main/java/com/windscribe/vpn/api/SslUtils.kt

Observation

If hard-coded credentials are embedded in software, the password is difficult to change at a later date. If the password becomes publicly known, such as from an imminent open source release, an attacker can easily break in.

We observed that the software contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.

```
    const val AUTH_KEY_1 = "952b4412f00"
    const val AUTH_KEY_2 = "2315aa5075"
    const val AUTH_KEY_3 = "1032fcaab03"
```

```
import com.windscribe.vpn.constants.ApiConstants.AUTH_KEY_1
import com.windscribe.vpn.constants.ApiConstants.AUTH_KEY_2
import com.windscribe.vpn.constants.ApiConstants.AUTH_KEY_3
...
val password = MD5Helper.md5(AUTH_KEY_1 + AUTH_KEY_2 + AUTH_KEY_3 + time)
```

```
val password = "XXXX".toCharArray()
inputStream = Base64.decode(BuildConfig.API_STATIC_CERT).inputStream()
keystore.load(inputStream, password)
```

Impact Rationale:

The security impact of these specific credentials is negligible; however, hard-coded credentials are an antipattern.

USE OF HARD-CODED CREDENTIALS

Difficulty Rationale:

An attacker can easily obtain copies of hard-coded credentials in open source code released on GitHub.

Recommendation

Remove hard-coded credentials and modify functionality that relies on their presence.

REIMPLEMENTATION OF LANGUAGE-PROVIDED FEATURES

ID	104430
Component	Windscribe Android App
Severity	**Info**
Impact / Skill Level	Informational/Moderate
Reference	https://developer.android.com/reference/android/util/Base64 https://www.bouncycastle.org/docs/pkixdocs1.5on/org/bouncycastle/openssl/PEMWriter.html
Location	Android-App/base/src/main/java/com/windscribe/vpn/encoding/encoders/Base64Encoder.java Android-App/base/src/main/java/com/windscribe/vpn/encoding/io/pem/PemWriter.java

Observation

Reimplementation of language-provided features provides a substantial opportunity for introduction of unintended bugs. This opportunity is expanded when reimplementing language-provided encoding and decoding functions which, due to their use in processing attacker-controllable input, can serve as hard-to-secure points for intrusion.

The application includes a Base64 library, which reimplements a standard Java library (also available in Android-flavored Java). It uses that library in a **`PemWriter`** class. While the Java language does not include a way to write out PEM files directly, that functionality is implemented in the BouncyCastle Java encryption library.

Impact Rationale:

While we have not identified a specific security impact from this finding, generally, implementing encoders and decoders manually, particularly in security-critical contexts, is an antipattern.

Difficulty Rationale:

Exploitation of an encoding vulnerability generally requires an attacker able to understand the flaw and craft special inputs to exploit it, which is nontrivial.

Recommendation

Remove both classes and related code. If a PEM writing function is strictly necessary, use BouncyCastle.

Windscribe iOS App

The Windscribe iOS application allows users to connect to Windscribe's various VPN endpoints from the iOS and iPadOS operating systems.

Threat Analysis

Like each of Windscribe's apps, a failure in the Windscribe iOS app's security is not necessarily catastrophic for the security of its users, since the app merely wraps well-tested VPN protocols (OpenVPN, Wireguard, etc.); instead security failures tend to cause business method problems.

Activities Performed

IOS SECURITY ISSUES

Scope

Validate that obvious security vulnerabilities, weaknesses, design flaws, and bad patterns are not present in the application.

Methodology

Manual code review.

Observations

Related Findings

104235: Use of hard-coded credentials enables business method bypass
104182: Secrets in source code
104226: Hashed domain function contains only 2 bits of randomness

IOS LICENSE ISSUES

Scope

Validate that issues of license compliance (e.g., hostile OSS licenses) are not present.

Methodology

Manual code review with automated assistance for dependencies.

Observations

Related Findings

104369: License violations: AGPL3, GPL2, CDDL, EPL, MPL, MIT, BSD, Apache, ...

IOS CODE QUALITY ISSUES

Scope

Validate that issues of code quality (not presenting other issues, such as security issues) are not present in the application.

Methodology

Manual code review.

Observations

Related Findings

104235: Use of hard-coded credentials enables business method bypass
104226: Hashed domain function contains only 2 bits of randomness
104321: Use of MD5 throughout
104183: Hash length extension vulnerability
104262: Total lack of comments
104374: Unmaintained dependencies in use

Vulnerabilities

USE OF HARD-CODED CREDENTIALS ENABLES BUSINESS METHOD BYPASS

ID	104235
Component	Windscribe iOS App
Severity	**High**
Impact / Skill Level	High/Simple
Reference	http://cwe.mitre.org/data/definitions/798.html https://developer.apple.com/documentation/cryptokit/hmac
Location	iOS-App/Windscribe/Managers/Network/NetworkManager.swift

Observation

If hard-coded credentials are embedded in software, the password is difficult to change at a later date. If the password becomes publicly known, an attacker can easily break in.
We observed that the software contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.

```
var signatureText = ""
signatureText.append(sessionAuthHash)
signatureText.append(userID)
signatureText.append(APIParameterValues.platform)
signatureText.append("\(score)")
signatureText.append("swiftMETROtaylorSTATION127!")
let signature = signatureText.SHA1()
let body = [APIParameters.platform: APIParameterValues.platform,
                    APIParameters.score: "\(score)",
                    APIParameters.sig: signature]
...
```

This code is part of the **`recordShakeForDataScore()`** function, which appears to be used to give free data to a user for shaking their phone. Since the authentication depends entirely on low-entropy values known or obtainable by the user, plus the static string **`swiftMETROtaylorSTATION127!`** which is obtainable from the source code (or a compiled binary), an attacker can easily make arbitrarily large and frequent requests for more free data, undermining Windscribe's business model.

Impact Rationale:
Exploitation allows a user to continue to use the Windscribe service without payment.

Difficulty Rationale:
The exploit is available to any user who can extract strings from a compiled binary or from the open source code, or who can observe network traffic and utilize an offline brute force attack.

USE OF HARD-CODED CREDENTIALS ENABLES BUSINESS METHOD BYPASS

Recommendation

Remove hard-coded credentials and modify functionality that relies on their presence. Using a keyed hashing function designed for use as a signature (such as Swift CryptoKit's implementation of HMAC) would significantly improve the integrity properties of this function.

SECRETS IN SOURCE CODE

ID	104182
Component	Windscribe iOS App
Severity	**Medium**
Impact / Skill Level	Medium/Simple
Reference	http://cwe.mitre.org/data/definitions/798.html
Location	iOS-App/Windscribe/Constants/Constants.swift iOS-App/Windscribe/Extensions/URL+Ext.swift iOS-App/Windscribe/Managers/InAppPurchaseManager.swift iOS-App/Windscribe/Managers/NetworkManager.swift

Observation

If hard-coded credentials are embedded in software, the password is difficult to change at a later date. If the password becomes publicly known, an attacker can easily break in. Systems using static secrets whose values are disclosed to an attacker can trivially be bypassed.
Much of the source code contains static secrets used as part of inputs to hash functions.

```
static let secret = "952b4412f002315aa50751032fcaab03"
```

```
let sharedSecret = "952b4412f002315aa50751032fcaab03"
```

```
signatureText.append("swiftMETROtaylorSTATION127!")
```

Impact Rationale:
These secrets protect a variety of calls to Windscribe's servers, such as calls to obtain credentials or request additional data.

Difficulty Rationale:
These secrets are included in the planned open source release.

Recommendation

Do not store passwords or other sensitive data in cleartext; instead, store secrets using trusted secret repositories, such as the Secure Enclave on iOS devices. Remove hard-coded credentials and modify functionality that relies on their presence.

HASHED DOMAIN FUNCTION CONTAINS ONLY 2 BITS OF RANDOMNESS

ID	104226
Component	Windscribe iOS App
Severity	**Medium**
Impact / Skill Level	Medium/Simple
Reference	https://www.commonlounge.com/discussion/2ee3f431a19e4deabe4aa30b43710aa7
Location	iOS-App/Windscribe/Extensions/URL+Ext.swift:81-88 Android-App/base/src/main/java/com/windscribe/vpn/api/HashDomainGenerator.kt

Observation

Hashing functions provide a "trapdoor" or "one-way" function, in which, given an output of a hash, one cannot determine the input by working backwards. However, modern computers are powerful enough that if the space for potential inputs is restricted, a list of all possible outputs (and the inputs that create them) can be generated, which provides a way to "reverse" the function without breaking its mathematical guarantee. Depending on what properties the hash protects, this may undermine or even remove the security guarantees for the system.

```
    static func generateHashedDomain() -> String {
        let randomNumber = (Int.random(in: 1...3)).description
        let month = Date().currentMonth
        let year = Date().currentYear
        let secretHost = Constants.API.hashDomainSecret + randomNumber + month +
year
        let hashedDomain = secretHost.SHA1()
        return hashedDomain
    }
```

Month and year are of course easily determinable by any attacker. The **`Int.random(in:1...3))`** operation will select either 1, 2, or 3. **`Constants.API.hashDomainSecret`** is specified as follows:

`iOS-App/Windscribe/Constants/Constants.swift:16`

```
static let hashDomainSecret = Configurations.hashDomainSecret
```

`/iOS-App/Windscribe/Configurations/Configurations.swift:10`

```
static let hashDomainSecret: String = ""
```

Accordingly, for any given month and year, this function can generate only 3 possible outcomes which

are easily determinable by any attacker.

This function is used only in the following code:

iOS-App/Windscribe/Managers/Network/NetworkManager.swift:279-289

```
    func getCurrentIP(completion: @escaping (_ ipAddress: String?, _ errorMessage:
String?) -> Void) {
        self.requestCheckIp { (result, error) in
            if error != nil {
                self.requestCheckIp(useHashDomain: true) { (result, error) in
                    completion(result, error)
                }
            } else {
                completion(result, error)
            }
        }
    }
```

The same issue exists in the Android Kotlin code:

```
        for (i in 1..3) {
            backUpAPIEndPoint = (
                BuildConfig.BACKUP_API_ENDPOINT_STRING + i +
                    (calendar[Calendar.MONTH] + 1) + calendar[Calendar.YEAR]
                )
            hashCode = Hashing.sha1()
                .hashString(
                    backUpAPIEndPoint,
                    Charset.forName(BACKUP_HASH_CHAR_SET)
                )
```

Given the lack of comments in the codebase, this appears to be a potential censorship evasion, as confirmed by a discussion with Windscribe employees. In the open source code, Windscribe has (as noted above) set the secret to a zero-length string; however, they assert that at build time for a production build, some secret string will be injected in its place. This approach will be able to be defeated by an attacker doing any of the following:

- Use of the **strings** utility or a hex editor to view the string in the compiled binaries. (Windscribe's employees state that this is difficult in practice due to binary obfuscation techniques that are used

HASHED DOMAIN FUNCTION CONTAINS ONLY 2 BITS OF RANDOMNESS

before pushing binaries to the app stores; evaluating this difficulty in practice is outside the scope for this engagement.)
- Obtaining (e.g., through social engineering, phishing, or bribery) the string from Windscribe's employees or infrastructure.
- Observing the DNS lookups made by the Windscribe application and applying a brute-force approach to find the input string that generates the observed outputs. (Note that this could be a completely offline attack; observation of the Windscribe application's network traffic would take only a few seconds, and the subsequent brute-force SHA1 search could be done with no further access to the Windscribe application at runtime.)

Accordingly, whether or not the secret is in fact a zero-length string does not matter in terms of effective security on this item. Note also that once any attacker puts in the work to discover the secret, it will be rendered meaningless, as it is shared across all Windscribe resources.

Impact Rationale:
An attacker can use knowledge of this system completely to defeat its intended evasion of network blocking of Windscribe endpoints.

Difficulty Rationale:
Any attacker in a position to control access to an un-"hashed" endpoint of Windscribe would also be in a position to block access to this "hashed" endpoint.

Recommendation

Develop another system for block evasion, such as using DNS over HTTPS (DoH) with a known-good provider.

USE OF MD5 THROUGHOUT

ID	104321
Component	Windscribe iOS App
Severity	**Low**
Impact / Skill Level	Low/Simple
Reference	https://en.wikipedia.org/wiki/MD5
Location	iOS-App/Windscribe/Managers/InAppPurchaseManager.swift iOS-App/Windscribe/Extensions/String+Ext.swift iOS-App/Windscribe/Extensions/URL+Ext.swift

Observation

MD5 has been deprecated for use since 2008, and is no longer recommended by any entities for use in new systems.

The Windscribe application uses MD5 as a hash function in several places to generate hashes used for censorship resistance, payment validation, and other functionality. Since hash collisions in MD5 are very cheap to create artificially, the security created by this hash (as opposed to sending plaintext) is low-to-no-value at this time. While the MD5 hashes are also used in places where other findings remove security properties (e.g., length extension vulnerabilities, static secrets, etc.), MD5 should be removed throughout the application both to avoid ridicule when open sourcing the application and to prevent the example from being copied in a new security-critical context at some point in the future.

Impact Rationale:

While the situations in which MD5 are used in the application have low-to-medium security criticality, that impact is blunted by other findings detailing ways to bypass those security controls.

Difficulty Rationale:

MD5 collisions have been created for many years using decreasingly costly hardware, placing them within the reach of even novice attackers.

Recommendation

Replace all uses of MD5 and other deprecated hash functions (like SHA1) with a modern hash, such as SHA3. or SHA2-512.

HASH LENGTH EXTENSION VULNERABILITY

ID	104183
Component	Windscribe iOS App
Severity	**Info**
Impact / Skill Level	Informational/Simple
Reference	http://www.skullsecurity.org/blog/2012/everything-you-need-to-know-about-hash-length-extension-attacks https://en.wikipedia.org/wiki/Length_extension_attack
Location	iOS-App/Windscribe/Extensions/URL+Ext.swift:22-28, 85-86

Observation

A system that 'signs' data by concatenating the data with a secret and then hashing the combination is vulnerable to a "Hash Length Extension". They are vulnerable even if using sound hashing functions, such as SHA256. That this is not obvious is why cryptographic libraries should be used if at all possible.

signature = H(secret || validated_data)

is dangerous, because an attacker can trivially calculate:

signature = H(secret || validated_data || attacker_data)

without knowing the secret.

H can be just about any hashing algorithm, including the MD and SHA families (though SHA-3 is not vulnerable).
This code antipattern occurs in at least two places:

```
let authHash = Constants.API.secret + timestamp
...
let clientAuthHashQuery = URLQueryItem(name: APIParameters.clientAuthHash,
                                                    value: authHash.MD5())
```

```
let secretHost = Constants.API.hashDomainSecret + randomNumber + month + year
let hashedDomain = secretHost.SHA1()
```

Impact Rationale:
In the specified areas, this likely has no security impact. However, this code antipattern is a poor model, and if left in the code, may be copied and used for another feature in the future that has more security criticality.

HASH LENGTH EXTENSION VULNERABILITY

Difficulty Rationale:

Hash length extension attacks are simple to calculate, and documented on Wikipedia.

Recommendation

Use an appropriate signature-generation function, such as HMAC or CMAC, to generate a hash that is created with both plaintext and a secret.

USE OF MD5 THROUGHOUT

ID	104393
Component	Windscribe iOS App
Severity	**Info**
Impact / Skill Level	Informational/Simple
Reference	https://en.wikipedia.org/wiki/MD5
Location	Android-App/base/src/main/java/com/windscribe/vpn/api/AuthorizationGenerator.kt

Observation

The MD5 hashing algorithm has been deprecated for new work since 1996, and should at this time never be used for any purpose.
MD5 is used throughout the Android and iOS applications for a variety of purposes. One example is also an example of a hash-extension vulnerability:

```
val password = MD5Helper.md5(AUTH_KEY_1 + AUTH_KEY_2 + AUTH_KEY_3 + time)
```

Impact Rationale:
The use of MD5 has no direct security impact; however, since MD5 has been deprecated for new work since 1996, any context in which MD5 is being used has the appearance of security with no actual positive security gains.

Difficulty Rationale:
MD5 collisions can be obtained with commodity hardware.

Recommendation

Remove MD5 throughout and replace it with a modern hashing algorithm (SHA512 or SHA3) in every instance.

Appendix A – Technical Services

Leviathan's Technical Services group brings deep technical knowledge to your security needs. Our portfolio of services includes software and hardware evaluation, penetration testing, red team testing, incident response, and reverse engineering. Our goal is to provide your organization with the security expertise necessary to realize your goals.

SOFTWARE EVALUATION We provide assessments of application, system, and mobile code, drawing on our employees' decades of experience in developing and securing a wide variety of software. Our work includes design and architecture reviews, data flow and threat modeling, and code analysis using targeted fuzzing to find exploitable issues.

HARDWARE EVALUATION We evaluate new hardware devices ranging from novel microprocessor designs, embedded systems, mobile devices, and consumer-facing end products, to core networking equipment that powers internet backbones.

PENETRATION & RED TEAM TESTING We perform high-end penetration tests that mimic the work of sophisticated attackers. We follow a formal penetration testing methodology that emphasizes repeatable, actionable results that give your team an understanding of the overall security posture of your organization as well as the details of discovered vulnerabilities.

SOURCE CODE-ASSISTED SECURITY EVALUATIONS We conduct security evaluations and penetration tests based on our code-assisted methodology, allowing us to find deeper vulnerabilities, logic flaws, and fuzzing targets than a black-box test would reveal. This methodology gives your team a stronger assurance that the most significant security-impacting flaws have been found, allowing your team to address them.

INCIDENT RESPONSE & FORENSICS We respond to our customers' security incidents by providing forensics, malware analysis, root cause analysis, and recommendations for how to prevent similar incidents in the future.

REVERSE ENGINEERING We assist clients with reverse engineering efforts. We provide expertise in investigations and litigation by acting as experts in cases of suspected intellectual property theft.

Appendix B – Risk and Advisory Services

Leviathan's Retained Services group is a supplement to an organization's security and risk management capability. We offer a pragmatic information security approach that respects our clients' appetites for security process and program work. We provide access to industry leading experts with a broad set of security and risk management skills, which gives our clients the ability to have deep technical knowledge, security leadership, and incident response capabilities when they are needed.

INFORMATION SECURITY STRATEGY DEVELOPMENT We partner with boards, directors, and senior executives to shape your enterprise's overall approach to meeting information security requirements consistently across an entire organization.

ENTERPRISE RISK ASSESSMENT We develop an information asset-centric view of an organization's risk that provides insight to your organization's Enterprise Risk Management capability. This service can be leveraged with annual updates, to account for your organization's changing operations, needs, and priorities.

PRIVACY & SECURITY PROGRAM EVALUATION We evaluate your organization's existing security program to give you information on compliance with external standards, such as ISO 27000 series, NIST CSF, HIPAA, or PCI-DSS. This is often most useful before a compliance event or audit and helps to drive the next phase of growth for your Security and Risk Management programs.

VENDOR RISK ASSESSMENT We assess the risk that prospective vendors bring to your organization. Our assessment framework is compatible with legislative, regulatory, and industry requirements, and helps you to make informed decisions about which vendors to hire, and when to reassess them to ensure your ongoing supply chain security.

NATIONAL & INTERNATIONAL SECURITY POLICY In 2014, we launched a public policy research and analysis service that examines the business implications of privacy and security laws and regulations worldwide. We provide an independent view of macro-scale issues related to the impact of globalization on information assets.

M&A/INVESTMENT SECURITY DUE DILIGENCE We evaluate the cybersecurity risk associated with a prospective investment or acquisition and find critical security issues before they derail a deal.

LAW FIRM SECURITY SERVICES We work with law firms as advisors, to address security incidents and proactively work to protect client confidences, defend privileged information, and ensure that conflicts do not compromise client positions. We also work in partnership with law firms to respond to their clients' security needs, including in the role of office and testifying expert witnesses.

SAAS AND CLOUD INITIATIVE EVALUATION We give objective reviews of the realistic threats your organization faces both by moving to cloud solutions and by using non-cloud infrastructure. Our employees have written or contributed to many of the major industry standards around cloud security, which allows their expertise to inform your decision-making processes.