Security Assessment of Open Technology Fund's client VPN Hood's Mobile application, Windows client, and Server

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Include Security (IncludeSec)

IncludeSec brings together some of the best information security talent from around the world. The team is composed of security experts in every aspect of consumer and enterprise technology, from low-level hardware and operating systems to the latest cutting-edge web and mobile applications. More information about the company can be found at www.IncludeSecurity.com.

Assessment Objectives

The objective of this assessment was to identify and confirm potential security vulnerabilities within targets in-scope of the SOW. The team assigned a qualitative risk ranking to each finding. Recommendations were provided for remediation steps which Open Technology Fund's **client VPN Hood** could implement to secure its applications and systems.

Scope and Methodology

Include Security performed a security assessment of Open Technology Fund's client VPN Hood's Mobile application, Windows client, and Server. The assessment team performed a 28 day effort spanning from June 12, 2023 – July 19, 2023, using a Standard Grey Box assessment methodology which included a detailed review of all the components described in a manner consistent with the original Statement of Work (SOW).

Findings Overview

IncludeSec identified a total of 4 findings. There were 0 deemed to be "Critical-Risk," 0 deemed to be "High-Risk," 1 deemed to be "Medium-Risk," and 0 deemed to be "Low-Risk," which pose some tangible security risk. Additionally, 3 "Informational" level findings were identified which do not immediately pose a security risk.

IncludeSec encourages Open Technology Fund's **client VPN Hood** to redefine the stated risk categorizations internally in a manner that incorporates internal knowledge regarding business model, customer risk, and mitigation environmental factors.

Next Steps

IncludeSec advises Open Technology Fund's client VPN Hood to remediate as many findings as possible in a prioritized manner and make systemic changes to the Software Development Life Cycle (SDLC) to prevent further vulnerabilities from being introduced into future release cycles. This report can be used by as a basis for any SDLC changes. IncludeSec welcomes the opportunity to assist Open Technology Fund's client VPN Hood in improving their SDLC in future engagements by providing security assessments of additional products. For inquiries or assistance scheduling remediation tests, please contact us at remediation@includesecurity.com.

RISK CATEGORIZATIONS

At the conclusion of the assessment, Include Security categorized findings into five levels of perceived security risk: Critical, High, Medium, Low, or Informational. **The risk categorizations below are guidelines that IncludeSec understands reflect best practices in the security industry and may differ from a client's internal perceived risk. Additionally, all risk is viewed as "location agnostic" as if the system in question was deployed on the Internet. It is common and encouraged that all clients recategorize findings based on their internal business risk tolerances. Any discrepancies between assigned risk and internal perceived risk are addressed during the course of remediation testing.**

Critical-Risk findings are those that pose an immediate and serious threat to the company's infrastructure and customers. This includes loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information. These threats should take priority during remediation efforts.

High-Risk findings are those that could pose serious threats including loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information.

Medium-Risk findings are those that could potentially be used with other techniques to compromise accounts, data, or performance.

Low-Risk findings pose limited exposure to compromise or loss of data, and are typically attributed to configuration, and outdated patches or policies.

Informational findings pose little to no security exposure to compromise or loss of data which cover defense-in-depth and best-practice changes which we recommend are made to the application. Any informational findings for which the assessment team perceived a direct security risk, were also reported in the spirit of full disclosure but were considered to be out of scope of the engagement.

The findings represented in this report are listed by a risk rated short name (e.g., C1, H2, M3, L4, and I5) and finding title. Each finding may include if applicable: Title, Description, Impact, Reproduction (evidence necessary to reproduce findings), Recommended Remediation, and References.

INTRODUCTION

VPNHood is a virtual private network service designed to circumvent deep packet inspection. It works by redirecting TCP packets made by a user to a locally-running SOCKS proxy client, which forwards them to the **VPNHood** server, which makes the connection on the user's behalf. It is aimed to be undetectable, because it looks the user is browsing an ordinary HTTPS website.

The assessment team performed a twenty-eight-day assessment beginning on June 19th, 2023, and ending on July 19th, 2023.

The following components were reviewed during the assessment:

- Android Client
- Windows Desktop Client
- VPN Server

These primarily used the following technologies:

- C#
- C# Designer
- MSBuild Script
- PowerShell
- Bash Shell

The assessment team performed static code analysis and dynamic testing of all three components. Additionally, throughout the assessment, the assessment team found and reported on ways in which the **VPNHood** team may increase their security posture through improvements in architectural design. This reporting is found in the "Security Architecture Design Improvements" appendix.

CRITICAL-RISK FINDINGS

No Critical-Risk findings were identified during the course of the assessment.

HIGH-RISK FINDINGS

No High-Risk findings were identified during the course of the assessment.

MEDIUM-RISK FINDINGS

M1: [Android] Application Executable Signed with v1 Signature Scheme (JANUS Vulnerability)

Description:

The assessment team found that the **VpnHood** application's executable was signed with a v1 APK signature at the time of assessment.

Using a v1 signature makes the application prone to the Janus vulnerability on devices running Android 7 or below. The Janus vulnerability allows attackers to smuggle malicious code into the APK without breaking the signature.

At the time of writing, the application supported a minimum SDK version of 22 (Android 5), which also uses the v1 signature, thus being vulnerable to this attack. Android 5 devices no longer receive updates and are vulnerable to many security concerns. It can be assumed that any installed malicious application may trivially gain root privileges on those devices using public exploits.

Impact:

The existence of this vulnerability means that attackers could trick users into installing a malicious attacker-controlled APK which matches the v1 APK signature of the legitimate Android application. As a result, a transparent update would be possible without warnings appearing on Android devices, effectively taking over the existing application and all of its data.

Reproduction:

The following snippet from the **apksigner** tool shows that the application supported the v1 signature scheme at the time of assessment:

```
Verified using v1 scheme (JAR signing): true
Verified using v2 scheme (APK Signature Scheme v2): true
Verified using v3 scheme (APK Signature Scheme v3): true
Verified using v4 scheme (APK Signature Scheme v4): false
```

Recommended Remediation:

The assessment team recommends increasing the minimum supported SDK level to at least 24 (Android 7) to ensure that this vulnerability cannot be exploited on devices running older Android versions. In addition, future production builds should be signed only with v2 or greater APK signatures.

References:

Janus Vulnerability

LOW-RISK FINDINGS

No Low-Risk findings were identified during the course of the assessment.

INFORMATIONAL FINDINGS

I1: [Android] Application Data Backup Is Enabled

Description:

The **VpnHood** Android application allowed users to make backups of its application data during the assessment. Android supports automatic or manual backups of application data by default. An Android application can opt out of this feature by explicitly setting the **android:allowBackup** option to **false** in the **AndroidManifest.xml** file.

Impact:

An attacker with physical access to the device could perform a manual back up of the **VpnHood** application's data. This could be done using **adb**, which is a command line tool that provides functionality including, backing up application specific data. This is only possible when the **android:allowBackup** option is enabled.

This could potentially lead to disclosure of security-relevant user data if the application data contained items such as cleartext passwords or personally identifiable information (PII).

The assessment team found that no security-relevant user data was stored by the **VpnHood** application. However, future iterations could introduce such data.

Reproduction:

The **VpnHood** application was decompiled and the following code snippet from the **AndroidManifest.xml** file shows that the **android:allowBackup** option was enabled.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="369" android:versionName="2.9.369" android:installLocation="auto"
android:compileSdkVersion="33" android:compileSdkVersionCodename="13" package="com.vpnhood.client.android"
platformBuildVersionCode="33" platformBuildVersionName="13"
...
android:name="crc647486a17e72a62434.AndroidApp" android:debuggable="false" android:allowBackup="true"
```

The **VpnHood** application could be backed up with the following command:

```
adb backup -f backup.bak com.vpnhood.client.android
```

Recommended Remediation:

The assessment team recommends setting the **android:allowBackup** option to **false** in the **AndroidManifest.xml** file. This would prevent security-relevant user data disclosures in future iterations of the application.

References:

Android - Enable And Disable Backup

I2: [Android] Cleartext Traffic is Enabled

Description:

The **VpnHood** application allowed cleartext traffic during the assessment, as the **android:usesCleartextTraffic** option was enabled in the **AndroidManifest.xml** file. The **android:usesCleartextTraffic** option is used by the Android Operating System (OS) and third-party libraries to determine if cleartext communications are allowed by the application.

Impact:

If the Android Operating System (OS) and third-party libraries do not enforce encrypted communications, such as Transport Layer Security (TLS) then the user of the application is at risk of:

- Data disclosure due to data being sent in cleartext from the application to the backend server or another endpoint
- Man-in-the-Middle (MITM) attacks, as there is no confidentiality, authenticity, and protection against data tampering

The assessment team detected no cleartext communications by the **VpnHood** application while performing a dynamic analysis.

Reproduction:

The **VpnHood** application was decompiled and the following code snippet from the **AndroidManifest.xml** file shows that the **android:usesCleartextTraffic** option was enabled.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="369" android:versionName="2.9.369" android:installLocation="auto"
android:compileSdkVersion="33" android:compileSdkVersionCodename="13" package="com.vpnhood.client.android"
platformBuildVersionCode="33"
...
android:banner="@mipmap/banner" android:extractNativeLibs="true" android:usesCleartextTraffic="true"
```

Recommended Remediation:

The assessment team recommends setting the **android:usesCleartextTraffic** option to **false**. If the application requires cleartext communications then the Network Security Configuration feature (**android:networkSecurityConfig**) can be used to configure exceptions for certain domains, limiting the impact of allowing cleartext traffic in the application.

Setting the **android:usesCleartextTraffic** option to **false** does not guarantee that third-party libraries will honor it, but follows security best practices for Android development.

References:

OWASP Mobile Top 10: M3 - Insecure Communication
CWE: CWE-319 - Cleartext Transmission of Sensitive Information
Android - Network security configuration

I3: [Android] Jailbreak or Rooted Device Detection Not Implemented

Description:

At the time of assessment, the **VpnHood** application did not implement any detection mechanism to determine if a device was jailbroken or rooted. Devices that are jailbroken or rooted might have certain security features disabled which are used to protect the integrity of the device and applications running on it.

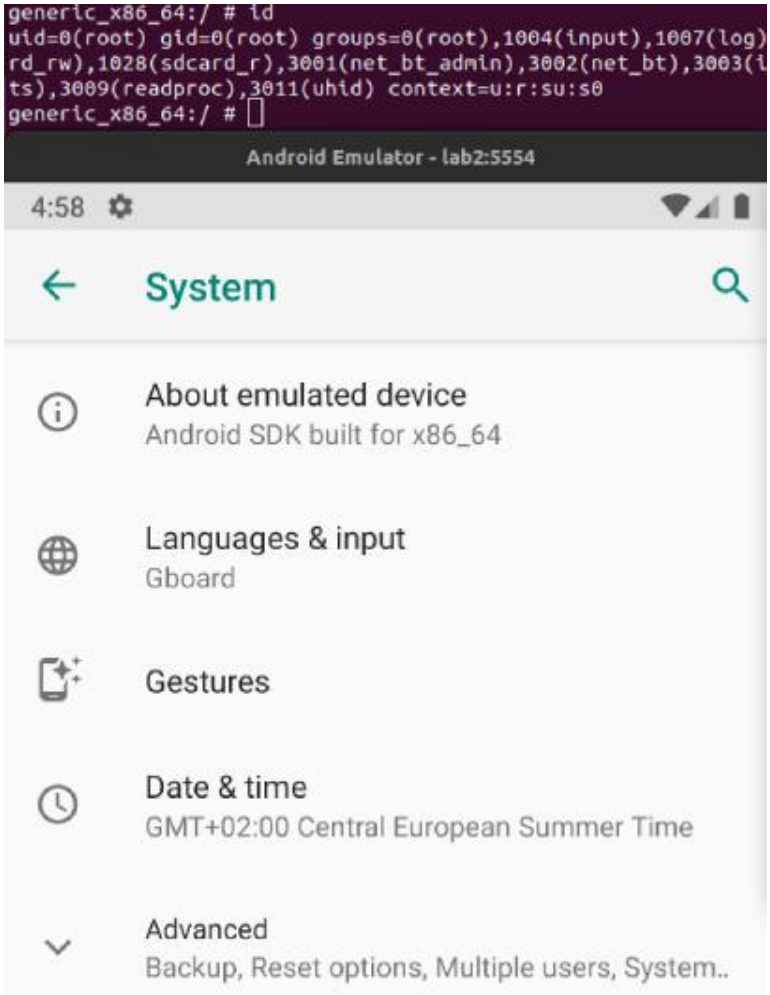
Impact:

An attacker using the application on a jailbroken or rooted device could tamper with the application or reverse-engineer it. A malicious application running on the jailbroken or rooted device could also tamper or interfere with other applications on the device.

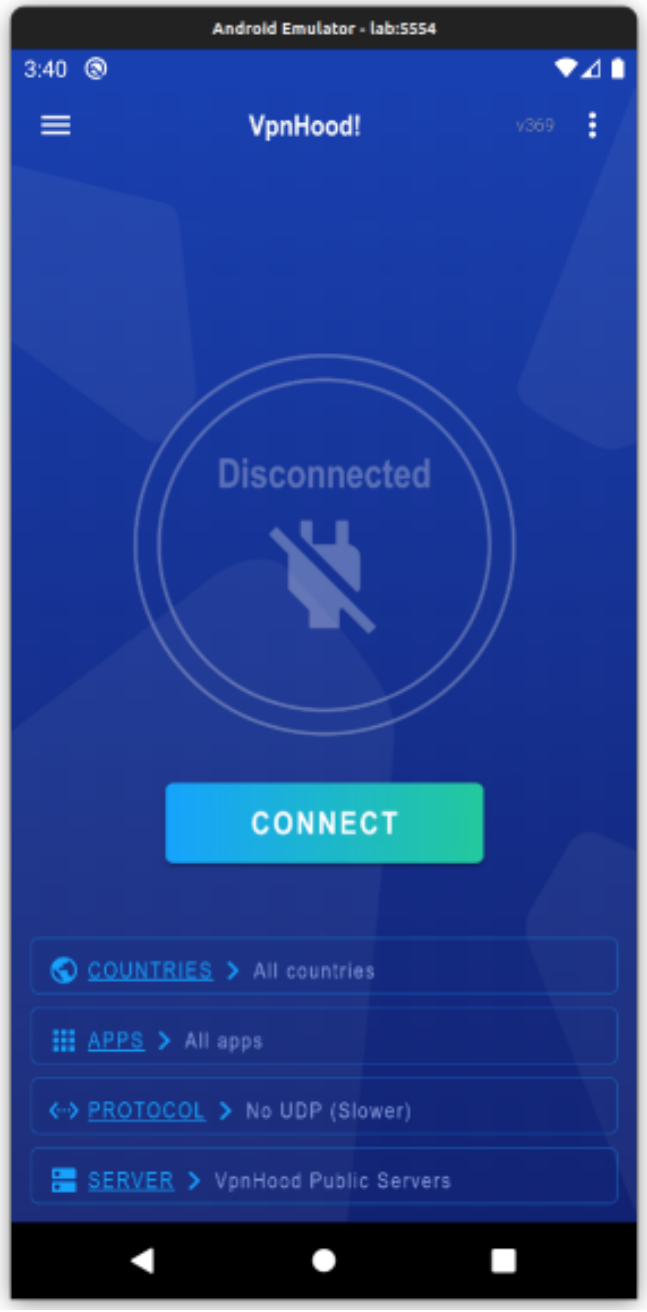
Reproduction:

The assessment team discovered this vulnerability by running the application on a jailbroken or rooted device and confirming that the user was not notified about the risks involved with running the application on that device.

The following image displays confirmation that the device used during the assessment was jailbroken or rooted:



The following image displays the application running on the jailbroken or rooted device without notifying the user:



Recommended Remediation:

The assessment team recommends implementing multiple jailbreak detection techniques to prevent information disclosure. Additional information about these techniques is available in the References section.

References:

OWASP Mobile Top 10 - M9 Reverse Engineering
Jailbreak Detection Techniques
OWASP Jailbreak Detection and Bypass

APPENDICES

Statement of Coverage

The IncludeSec team performed a grey box security review of the **VPNHood** application and server. The version audited was **v2.9.370**, and the following repositories were in scope:

- VpnHood.App.Launcher
- VpnHood.Client.Device.Android
- VpnHood.Client.Device
- VpnHood.Server
- VpnHood.Server.Access
- VpnHood.Client.App.UI
- VpnHood.Client.App
- VpnHood.App.Updater
- VpnHood.Client
- VpnHood.Client.App.Win.Setup
- VpnHood.Client.Device.WinDivert
- VpnHood.Tunneling
- VpnHood.Server.App.Net
- VpnHood.Client.App.Win
- VpnHood.Client.App.Android
- VpnHood.Common

The main focus of the assessment was on the Linux and Windows servers and the Windows and Android clients. During the last day of the assessment, the team briefly reviewed upcoming security-relevant changes by performing a git diff of the latest dev branch and version **v2.9.370**.

Assessment Setup
Virtual Machines (VMs) were configured with each server and client to perform dynamic analysis of the components. Tokens for each server were generated and used with the clients according to the VPNHood Wiki.

The VMs were also configured with tooling such as Wireshark, Sysinterals Suite and Bettercap to aid with dynamic analysis.

Methodology
The team used established security frameworks to structure the evaluation and ensure thorough coverage of potential vulnerabilities that outline the most critical security risks. The frameworks used were as follows:

- OWASP Top 10
- OWASP Mobile Top 10
- CWE Top 25

The threat model for the **VPNHood** server and client included nation-state actors, and some main concerns considered during the audit were:

- Post and pre-authentication attacks against the server that could lead to information disclosure, memory corruption or unintended behaviour
- Man-In-The-Middle (MITM) attacks against the client and server
- Cryptography implementation and usage between the client and server
- Potential vectors for fingerprinting the **VPNHood** server
- Privacy concerns, such as IP or DNS disclosure of the client via logs or at the network level
- Tampering with the software update mechanism and the authenticity of the updates
- Potential vectors for Denial-of-Service (DoS) attacks
- File permissions and process permissions for the server and client

The team completed the following items during the assessment:

Review of Provided Documentation
The provided documentation, such as the architecture diagram of the **VPNHood** application, was reviewed.

Automated Source Code Review
The team used the Static Application Security Testing (SAST) tool Semgrep to perform an automated analysis of the **VPNHood** repositories. All results were reviewed manually, and false positives were removed. The following are some examples of the Semgrep rules that were used for the static analysis:

- https://semgrep.dev/p/csharp
- https://github.com/returntocorp/semgrep-rules/tree/develop/csharp
- https://semgrep.dev/p/cwe-top-25
- https://semgrep.dev/p/owasp-top-ten

Dynamic Analysis
The team dynamically analysed the **VPNHood** clients and servers. This included but was not limited to the following items:

- The Windows client and servers' attack surface was enumerated with Sysinternals Suite to determine security-relevant files, processes, ports and potential misconfigurations
- The Linux servers' attack surface was enumerated for similar vulnerabilities but with standard Linux commands such as **lsof**, and **netstat**
- The Android client was analyzed with Mobile Security Framework (MobSF), and an emulator, where normal and malicious user activity was simulated
- The Windows client was installed on a VM where normal and malicious user activity was simulated
- The software update mechanism was reviewed to determine if it could be tampered with to download a malicious update
- Attempts to DoS the server were made by sending invalid messages after authentication and flooding the server with malicious sessions
- Logs were reviewed to determine if, e.g. IP disclosure had occurred
- Network-level tools such as Wireshark and Bettercap were used to capture and audit **VPNHood** network traffic to determine vulnerabilities such as IP disclosure

The team found that **VPNHood** had effectively limited the server's attack surface as it required a valid token for authentication and post-authentication; the server only handles a few message types related to session and tunnel management.

Manual Source Code Review

The team performed a manual code review of the provided **VPNHood** repositories, following the mentioned security frameworks. The primary focus of the manual review was the security-relevant parts of the source code, with the main concerns mentioned earlier in mind. The team also reviewed previously patched vulnerabilities based on the git history. During the manual code review, the team noted several positive security practices by **VPNHood**, such as:

- The use of ReSharper for continuous code-analysis during development
- The implementation and use of test cases to validate the expected behavior of the application and to find potential vulnerabilities
- Code reuse between the different components

Application Dependency Review

The team used the **dotnet CLI** to check for potentially outdated or vulnerable NuGet packages. The team did not identify any outdated dependencies.

OWASP Mobile Top 10

Category	Description	Assessment Observations
M1. Improper Platform Usage	This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system. There are several ways that mobile apps can experience this risk.	The assessment team did not find any improper platform usage vulnerabilities during the assessment. The application handled platform usage correctly, for example, it handled intents correctly and only requested the permissions that were necessary.
M2. Insecure Data Storage	This new category is a combination of M2 + M4 from Mobile Top Ten 2014. This covers insecure data storage and unintended data leakage.	The assessment team discovered insecure data storage vulnerabilities during the assessment. The following vulnerabilities were found: • The Android application allowed backup of the application data which could disclose application logs or user data • The Android application allowed screenshots to be taken which could disclose user data or user activity when using the application
M3. Insecure Communications	This covers poor handshaking, incorrect SSL versions, weak negotiation, cleartext communication of sensitive assets, etc.	The assessment team discovered an insecure communication vulnerability at the time of assessment. The application enabled the option **android:usesCleartextTraffic** which allows the application and third-party libraries to use cleartext channels for communication.

M4. Insecure Authentication	This category captures notions of authenticating the end user or bad session management. This can include: • Failing to identify the user at all when that should be required • Failure to maintain the user's identity when it is required • Weaknesses in session management	The assessment team did not find any insecure authentication vulnerabilities at the time of assessment. The application implemented and managed authentication correctly, for example, the client would verify that the server's certificate hash matched the expected value in the server access token.
M5. Insufficient Cryptography	The code applies cryptography to a sensitive information asset. However, the cryptography is insufficient in some way. Note that anything and everything related to TLS or SSL goes in M3. Also, if the app fails to use cryptography at all when it should, that probably belongs in M2. This category is for findings where cryptography was attempted, but it wasn't done correctly.	The assessment team did not find any insufficient cryptography vulnerabilities at the time of the assessment. The application implemented and managed cryptographic functionality correctly overall, however, the **UdpChannel2** class, while encrypting the data in transit to ensure confidentiality, lacks measures for integrity and authenticity, making the data susceptible to tampering.
M6. Insecure Authorization	This is a category to capture any failures in authorization (e.g., authorization decisions in the client side, forced browsing, etc.). It is distinct from authentication findings (e.g., device enrolment, user identification, etc.). If the app does not authenticate users at all in a situation where it should (e.g., granting anonymous access to some resource or service when authenticated and authorized access is required), then that is an authentication failure not an authorization failure.	The assessment team did not find any insecure authorization vulnerabilities at the time of the assessment. The application did not implement different user roles or privileged resources.
M7. Client Code Quality	This was the "Security Decisions Via Untrusted Inputs", one of our lesser-used categories. This would be the catch-all for code-level implementation problems in the mobile client. That's distinct from server-side coding mistakes. This would capture things like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.	The assessment team did not find any client code quality vulnerabilities during the assessment. The developers used ReSharper, which provides continuous code quality control during .NET development. Additionally, test cases were implemented to ensure components worked as expected.
M8. Code Tampering	This category covers binary patching, local resource modification, method hooking, method swizzling, and dynamic memory modification. Once the application is delivered to the mobile device, the code and data resources are resident there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain.	The assessment team discovered a code tampering vulnerability at the time of assessment. The Android application was signed with a v1 Signature Scheme which allows an attacker to modify an app without affecting its original signature, making the system believe it is interacting with the original application (JANUS Vulnerability).

M9. Reverse Engineering	This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Software such as IDA Pro, Hopper, otool, and other binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back-end servers, cryptographic constants and ciphers, and intellectual property.	The assessment team did not find any reverse engineering vulnerabilities at the time of assessment. The application did not implement any mitigation for reverse engineering but there was no direct security impact.
M10. Extraneous Functionality	Often, developers include hidden backdoor functionality or other internal development security controls that are not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disabling of 2-factor authentication during testing.	The assessment team did not find any extraneous functionality vulnerabilities at the time of assessment. The application was not found to use any extraneous functionality in release builds.

A1: Security Architecture Design Improvements

Include Security performed a security assessment of **VpnHood** v2.9.370, which was the latest stable release at the time of the assessment. On the final day of the assessment, the team used the command "**git diff v2.9.370 development**" to audit security-related changes from version v2.9.370 up to the most recent commit on **VpnHood's** development branch, which can be found at https://github.com/vpnhood/VpnHood/tree/development

The following components were audited during the assessment:

- VpnHood.App.Launcher
- VpnHood.Client.Device.Android
- VpnHood.Client.Device
- VpnHood.Server
- VpnHood.Server.Access
- VpnHood.Client.App.UI
- VpnHood.Client.App
- VpnHood.App.Updater
- VpnHood.Client
- VpnHood.Client.App.Win.Setup
- VpnHood.Client.Device.WinDivert
- VpnHood.Tunneling
- VpnHood.Server.App.Net
- VpnHood.Client.App.Win
- VpnHood.Client.App.Android
- VpnHood.Common

The primary focus was on the Windows client, Android client, and server-related components.

The assessment team identified several vulnerabilities and have provided suggestions to strengthen the application's security architecture. These proposals address both existing vulnerabilities and potential ones which could arise in the future.

R1: Implement the Use of Static Analysis Tools
The assessment team recommends that **VpnHood** incorporate static analysis tools into their regular code review process. This could be done through manual reviews or implemented in the CI/CD pipeline for automatic and continuous reviews. Doing so would help identify and mitigate vulnerabilities in the codebase.

The following static analysis tools are free, open-source, and support C#, the primary language in the **VpnHood** project:

Semgrep
Semgrep is available from https://semgrep.dev/ and the following rulesets could be used for static analysis of C#:

- https://semgrep.dev/p/csharp
- https://github.com/returntocorp/semgrep-rules/tree/develop/csharp
- https://semgrep.dev/p/cwe-top-25
- https://semgrep.dev/p/owasp-top-ten

SonarQube
The assessment team recommends the use of SonarQube, as a complementary tool to Semgrep. It has **452 rules** for C#, example categories include:

- **Vulnerability** (34 rules) https://rules.sonarsource.com/csharp/type/Vulnerability
- **Bug** (80 rules) https://rules.sonarsource.com/csharp/type/Bug/
- **Security Hotspot** (30 rules) https://rules.sonarsource.com/csharp/type/Security%20Hotspot/

CodeQL
The assessment team recommends the use of CodeQL as **VpnHood** uses GitHub. The CodeQL CLI is free to use on public GitHub repositories and has approximately 300 rules related to C#, example categories include:

- **API Abuse** (15 rules)
- **Architecture** (3 rules)
- **Bad Practices** (31 rules)
- **Complexity** (2 rules)
- **Concurrency** (7 rules)
- **Configuration** (2 rules)
- **Input Validation** (3 rules)
- **Likely Bugs** (34 rules)
- **Metrics** (40 rules)
- **Security Features** (63 rules)
- **Useless code** (6 rules)

MobSF
Additionally, the team recommends Mobile-Security-Framework-MobSF, for the **VpnHood** Android client and future iOS client. **MobSF** supports static analysis of both and could be used to help identify and mitigate potential vulnerabilities.

R2: Implement the Use of Dynamic Analysis Tools
The assessment team recommends that **VpnHood** incorporate dynamic analysis tools into their regular code review process. This would help with catching potential bugs or vulnerabilities at runtime which static analysis would not detect.

The Mobile-Security-Framework-MobSF supports dynamic analysis of Android applications and could be used to help identify and mitigate potential vulnerabilities in the **VpnHood** Android application.

R3: Explicitly Set Secure and Privacy Related Default Values in the Android manifest
The assessment team recommends explicitly setting secure configuration options in the Android manifest for the **VpnHood** Android application. By not doing so, older versions of the Android operating system might use default values that could introduce vulnerabilities or privacy infringements.

R4: Implement Dependency Management with Vulnerability Scanning
The assessment team recommends that **VpnHood** implement a process for regularly scanning the code base using relevant tools to detect dependencies that are either outdated or have known vulnerabilities. To mitigate the risks associated with vulnerable dependencies, the team suggests updating outdated components to the latest versions where all publically known vulnerabilities have been addressed.

To facilitate this, the **VpnHood** dependencies can be scanned using the following tools:

Project Name	Dependency Vulnerability Scanner	Command
VpnHood	dotnet	**dotnet list packages —vulnerable**
VpnHood	dependabot	**N/A**

R5: Implement Security Measures for the VpnHood GitHub Account
The assessment team recommends securing the **VpnHood** GitHub account and its repository, as it is used as a part of the **VpnHood** infrastructure to distribute automatic software updates. The team recommends the following items:

- Enable and enforce the use of multi-factor authentication (2FA) for the **VpnHood** GitHub account and for all the contributors to the **VpnHood** repository. This would help mitigate against potential attempts to compromise the **VpnHood** repository, which could be leveraged by an attacker to inject malicious code into software updates
- Use trusted GPG keys to sign commits and releases to ensure the integrity and authenticity of the code. This would help mitigate potential code tampering and malicious code injection into the software updates

In addition, any applicable items from the official GitHub account security recommendations should be implemented.

R6: Use Crypto Library APIs over Custom Implementations
The assessment team recommends using known and well-audited crypto libraries such as BouncyCastle .NET over custom implementations. For example, the **BufferCryptor** class had a custom implementation of AES-CTR in the **Cipher()** method. The team found no security concerns in the implementation, but custom cryptography has many pitfalls and can be error-prone.

R7: Implement Use of HMAC for Data Integrity and Authentication
The assessment team recommends implementing the use of an HMAC with a cryptographically secure hash algorithm, such as SHA-256. The **UdpChannel2** class encrypts the data transmitted between the server and the client and ensures its confidentiality. However, it provides no method for ensuring the integrity and authenticity of the data. This means that the data is susceptible to tampering, which could be mitigated by the implementation of an HMAC.

R8: Implement Code Integrity Check for Automatic Updates
The assessment team recommends implementing a code integrity check for automatic updates, particularly for the server component. During the assessment, it was noted that the server component performed automatic updates as a privileged user by downloading new releases from GitHub over HTTPS. Despite the secure connection, the update process did not perform any code integrity checks on the downloaded release before applying the update. This lack of integrity checking exposed the update process to tampering risks, such as an attacker replacing the legitimate binary on GitHub with a malicious one. Such a scenario could result in the execution of malicious code on the server host.

The risk could be mitigated by the use of digital signatures. With digital signatures, updates are signed using a private key, and the update process verifies these updates using the corresponding public key. This would reject any update with a mismatched signature, thereby enhancing the security of the update process.

Security Concerns Commonly Present in Most Applications

This section contains information about general classes of vulnerabilities that affect the majority of publicly exposed web applications. As such, IncludeSec does not present these as specific findings in assessment reports, but instead presents these topics as this report Appendix to ensure Client awareness of these topics. IncludeSec always encourages clients to review these topics and decide independently whether the security benefits apply and are worth the trade-offs in usability for users.

Credential Stuffing

Credential Stuffing attacks occur when attackers obtain a list of compromised username and password combinations (usually from breaches of other online services) and attempt to leverage them to gain access to user accounts. Attackers often conduct these attacks in parallel using several source IP addresses, making them difficult to prevent with IP rate limiting, session limiting measures, attack detection JavaScript, or server-side awareness of vulnerable accounts (e.g., HaveIBeenPwned Database). Additionally, Credential Stuffing attacks are unlikely to trigger account lockout mechanisms because, unlike a traditional brute-force attack, only a small number of password combinations are attempted for each account. CAPTCHAs are becoming increasingly trivial to bypass with recent developments in the field of machine learning, and as a result the industry does not consider CAPTCHA to be a robust security control to prevent automated attacks.

Include Security believes that the only complete mitigation for the credential stuffing threat is Mandatory Multi-Factor Authentication (MFA). However, this mitigation adds significant friction to the user experience as well as support overhead, so the most common approach in the industry is to deploy some partial mitigations but ultimately accept some risk that Credential Stuffing attacks remain a possibility in the absolute sense. Note that this risk may be very low if defense in depth is applied using controls mentioned above.

Multifactor Authentication is Not Mandatory

Multifactor Authentication (MFA/2FA) mitigates many common authentication vulnerabilities by requiring users to have physical access to another device to prove their identity when logging into services. This prevents prevalent attacks such as Credential Stuffing (discussed above), Brute-Force Guessing attacks, and some types of Authentication-Based Account Enumeration. Hardware 2FA/MFA methods, such as WebAuthn/FIDO2, also mitigate phishing attacks that have compromised accounts using legacy 2FA/MFA methods (SMS, etc.) during several high-profile breaches.

As mentioned earlier, mandatory multifactor authentication greatly increases friction for users and support staff and is not widely deployed in the industry for these reasons, except in specific applications with very high security needs. Many applications support optional 2FA/MFA, and while this practice does increase security for users who opt into it, most of the platforms who have analyzed their user base have shown that typical users will not choose to enable it if it is not enabled by default (or mandatory), putting the users at risk of attacks such as phishing and credential stuffing.

Application Allows Concurrent Sessions for Same User

Some applications restrict users from having multiple active sessions at a time, such as connecting from multiple devices or browsers. This control is meant to mitigate the risk of an attacker compromising the account in some way and going unnoticed by the user.

IncludeSec believes the security impact to an application if this security feature is not implemented is marginal and instead recommends notifying users of other successful authentication events, logging of successful authentication events, as well as providing functionality to terminate all active sessions in the event of account compromise. This approach allows users to respond quickly to security concerns without introducing unnecessary usability concerns.

JWTs Remain Valid After Deauthentication

It is considered best practice for applications that leverage traditional server-side sessions to destroy the session object on the server as well as clear the data from the browser when a client deauthenticates from the application, whether voluntarily or via session timeout. If the application does not do this, an attacker with access to the user's browser or other means to compromise the session token could continue performing actions on the user's account even after they have logged out.

With JSON Web Tokens (JWTs), the application instead stores session state in a cryptographically signed token that is managed by the client. With this design, the token will remain valid until its expiration date, even if the user deauthenticates. While it is possible to maintain a JWT "blacklist" on the server to effectively revoke tokens, Include Security instead recommends following general security best practices regarding JWTs:

1. Access tokens should have a very short expiration time (in general, less than 1 hour).
2. The application can transparently refresh the session in the background using refresh tokens, which are generally longer lived than access tokens.
3. Refresh Tokens should implement Refresh Token Rotation, which helps identify and mitigate compromised refresh tokens by invalidating previous refresh tokens each time a token is refreshed.
4. JWTs should be signed with modern cryptographic algorithms (i.e., RS256) and validated using the most proven library provided by the web application framework in use.
5. JWTs should not contain security relevant or confidential data in the payload, such as PII or application secrets.