

SUBGRAPH

FINDINGS REPORT

RedVPN

Oct 30, 2023

Prepared for: RedVPN

Subgraph Technologies, Inc.
642 Rue de Courcelle, Suite 309
Montreal, Quebec
<https://subgraph.com>

Contents

Overview	4
Scope	4
Versions Examined	4
Major functional components in scope: Android app	4
Major functional components in scope: iOS app	5
Major functional components in scope: backend	5
Objectives	5
Methodology	6
Tools Used	7
Observations and Recommendations	8
Observations on Authorization: Mobile	8
Observations on Authorization: Backend	8
Observations on Data Encryption: Android	8
Observations on Data Encryption: iOS	8
Observations on Authentication	9
General Observations	9
Summary	10
Details	11
V-001: Unauthenticated Registration Endpoint	11
Discussion	11
Impact Analysis	12
Remediation Recommendations	12
Additional Information	12
V-002: Secrets in Source Code	13
Discussion	13
Impact Analysis	13
Remediation Recommendations	14
Additional Information	14
Appendix	15
Methodology	15
Description of testing activities	15
Reporting	16
Severity ratings	16
Contextual factors	18
Likelihood	19
Remediation status	20

Overview

Scope

RedVPN is an easy to use VPN client developed by experienced Internet security professionals. It is available for both Android and iOS mobile platforms and allows users to have high-performance VPN access with a single click.

RedVPN for iOS is written mostly in Swift, with lower level components (WireGuard) written in Golang and C. RedVPN for Android is written mostly in Java/Kotlin, with nearly identical lower level components also written in Golang and C.

The lower level components in both cases are not authored by the RedVPN team. RedVPN uses WireGuard under the hood and is heavily reliant on source code from the original WireGuard developers. This reduces the risk of vulnerabilities in security sensitive new code, as the WireGuard project is open source, has been around for some time, and is heavily scrutinized.

The RedVPN backend is developed in Python, running on Linux servers.

Subgraph was engaged in 2023 to perform security testing of the Red VPN mobile apps and backend. The review was conducted during September and October 2023. See below for the latest commits, as of testing for both apps below, for the precise versions examined.

Versions Examined

Installed app version 1.1.0 from the Play store on Android 13.

Installed app version 1.0.1 from the Apple App store on iOS 16.6.1.

Reviewed source code of the iOS app at commit `63ce4ac3085dfe4e4867fac4478ad8aacd05920e` (private repo).

Reviewed source code of the Android app at commit `294c5bc8c98cef3a25a494ccb411ac4c4911c8b8` (private repo).

Tested API endpoints located at: <https://partners.1e-100.net>.

Scanned egress network CIDRs provided by the RedVPN team. These are not listed in the report as they are of some security sensitivity.

Major functional components in scope: Android app

Review of the android app included the following major components:

- Key generation and storage
- Authentication to backend

- Configuration and enabling of tunnel
- Correct functioning of the tunnel

Major functional components in scope: iOS app

Review of the iOS app included the following major components:

- Key generation and storage
- Authentication to backend
- Configuration and enabling of tunnel
- Correct functioning of the tunnel

Major functional components in scope: backend

Review of the API backend included the following major components:

- Registration of client
- Authentication
- Authorization
- Resistance to input validation and other attacks common to API endpoints

Objectives

The high level objectives of this assessment are to:

- Test the basic VPN functionality of the app to ensure it functions as user expect
- Examine the implementation of the apps for security vulnerabilities that could expose users to attacks
- Examine the backend for security vulnerabilities that could expose users to attacks

The threat model assumes the following scenarios:

- Adversary of moderate or high skill in all use cases
- Adversary of moderate or high skill targeting all users through the infrastructure
- Adversary targeting specific users of the app with active network tampering (MITM against app users)
- Adversary targeting specific users who wish to use the app (MITM against app distribution)

Methodology

This review had three major components:

1. Review of the apps, including a tactical source code review
2. Review of the backend attack surface from the perspective of a malicious client
3. Security scan of the infrastructure from the perspective of an unprivileged, random Internet origin

The application code was reviewed in the following ways:

Both applications were cloned locally (see above for latest commit at time of review) and reviewed in an IDE. Command-line utilities were also used for quickly searching for potentially problematic patterns. Subgraph did not build the clients locally, but CI artifacts that were present in the source trees were included in the review.

The backend attack surface was reviewed using a mix of an intercepting proxy and CLI tools. The endpoints were provided by the Red VPN team in advance and extracted from the source code. Various queries were made, with modifications to try and circumvent server-side input validation and identify risks through server behavior revealed in response characteristics.

Network scanning tools for vulnerability assessment were used to scan full egress CIDRs provided by the Red VPN team. The API endpoint was also scanned.

Tools Used

- An intercepting proxy (BurpSuite Pro) was used to interact directly with the backend API server.
- A network scanner was used to scan the external endpoints: Nmap, netcat, others
- An IDE was used to review the source code: VS Code
- Mobile devices running Android and iOS were used to interact with the apps: Physical mobile devices

Observations and Recommendations

Observations on Authorization: Mobile

Authorization is enforced by the mobile OS application containment for both the iOS and Android clients. In the case of the Android app, the private key is stored in *SharedPreferences*. While this is protected by the Android OS and is private to the application, it could possibly be improved in the future by leveraging the Android Keystore, though there are limitations on the kinds of key material that can be stored there, and Curve25519 is not included.

On iOS the private key is stored in UserDefaults (see *Sources/WireGuardApp/Crypto/KeyStore.swift*). This could be improved to use the native KeyChain facility in iOS, which greatly improves protection of key material. Additionally, CryptoKit (iOS 13+) has native support for Curve25519, including key generation.

Observations on Authorization: Backend

There did not appear to be any meaningful authorization checks within the exposed backend API. This appears to be by design, i.e., this observation does not indicate a finding due to lack of authorization. There just is no concept of permissions or roles for Red VPN clients: they are all treated equally.

Observations on Data Encryption: Android

The WireGuard implementation used in the Android RedVPN client is based on the open source [WireGuard Android](#). The Android RedVPN client uses the open source Golang client developed by the WireGuard team in the manner it is used by the open source WireGuard Android reference client. Native implementation is used to both create the tunnel and to generate the user keys. Android does not offer native support for Curve25519 through either its KeyStore or the libraries included for cryptography, so this decision is understandable.

Observations on Data Encryption: iOS

The WireGuard implementation used in the iOS RedVPN client is based on [WireGuardKit](#) for iOS. The iOS RedVPN client uses the open source Golang client developed by the WireGuard team in the manner it is used by the open source WireGuard iOS reference client. As with Android, the key generation is also provided by the native WireGuard implementation.

With iOS 13+ there is native support for Curve25519, which can likely be used to generate keys using hardware facilities on the device rather than in userland code. It may also be possible to store them in the iOS KeyChain, offering superior protection through the mobile OS. Both of these options possibly offer security improvements, and should be investigated by the RedVPN team.

Observations on Authentication

By current design, The RedVPN app does not require any user signup or authentication credentials, such as a RedVPN username and password. A set of hardcoded credentials exists within the app used to authenticate to the backend API via HTTP Basic authentication over TLS. Testing found that the credentials used did not appear to matter, any value provided to the backend will be accepted. It's possible even to register a profile with the backend without providing the basic authentication string. This appears to be by design, as there's no way to input these credentials in the client app. One thing worth noting is that the VPN stopped processing enrollments during peak hours at one point during testing. This suggests the possibility of a denial of service through abuse of this open access model. I.e., it may be possible for an adversary to register a massive number of fake clients, causing a denial of service until the enrollments expire. This was not tested by Subgraph.

The API endpoint are authenticated using TLS with X.509 certificates.

The VPN endpoints are authenticated using public keys provided by the API endpoint during enrollment of the device.

The app is authenticated using app signing in both cases. For this reason, it is strongly recommended that the official app stores be the only place where Red VPN is sourced by users. This may be a challenge for users in some locations, which creates a risk of malicious apps. For Android users, Red VPN may wish to consider its own distribution directly from its website for at least the Android version, to provide an alternative to untrustworthy APK download sites.

General Observations

The overall RedVPN architecture is very simple due to the absence of *identity*. Clients simply generate keys and register them with the API backend, which provides a response that includes data required to enroll the device in the VPN. The device then configures a local WireGuard client and requests user permission to tunnel device traffic through it. The API backend is incredibly simple. During testing it was observed that this basic functionality worked as designed for both apps.

The most security sensitive code in the RedVPN app is related to key generation and configuration of the tunnel.

Summary

No.	Title	Severity	Remediation
V-001	Unauthenticated Registration Endpoint	Medium	Unresolved
V-002	Secrets in Source Code	Medium	Unresolved

Details

V-001: Unauthenticated Registration Endpoint

Severity	Remediation
Medium	Unresolved

Discussion

The RedVPN client is designed to send a username and password in the form of HTTP basic credentials. However, the registration endpoint for clients accepts any authentication credential, or no authentication credential. This was observed in the POST request sent to the endpoint used to enroll a client in the VPN through submission of a public key:

```
POST /api/profile/ HTTP/1.1
Host: partners.1e-100.net
Authorization: Basic cmVkdnbu0jZ5bVAaY29xMnpYWWRPcG9YdVVGOTkzYQ==
User-Agent: curl/7.68.0
Accept: */*
Content-Length: 57
Content-Type: application/json
Connection: close
```

```
{"pubkey": "11/HTcnuzFQzz27HgLHcKRH7s00sSEmqD11ya6zb3Sg="}
```

The following request will produce a valid response from the server:

```
POST /api/profile/ HTTP/1.1
Host: partners.1e-100.net
User-Agent: curl/7.68.0
Accept: */*
Content-Length: 57
Content-Type: application/json
Connection: close
```

```
{"pubkey": "11/HTcnuzFQzz27HgLHcKRH7s00sSEmqD11ya6zb3Sg="}
```

Impact Analysis

This appears to be by design, perhaps due to incomplete support for client authentication. The apps do not require users to create an account. One consequence of this is that there is no way to distinguish client identities except perhaps by source IP address. Therefore it seems possible that an adversary could register a large number of fake profiles, potentially resulting in a denial of service. Since a capacity saturation event was observed during testing, presumably due to large user volume, it does seem like a viable strategy to attack the service. During this outage the profile server was not responding to requests for new enrollments. This was not explored further with testing as Subgraph did not want to interfere with service operation.

Remediation Recommendations

The addition of identity or credentials would increase the cost of the attack, as well as make it possible to enforce limits to device enrollment per identity.

Additional Information

N/A

V-002: Secrets in Source Code

Severity	Remediation
Medium	Unresolved

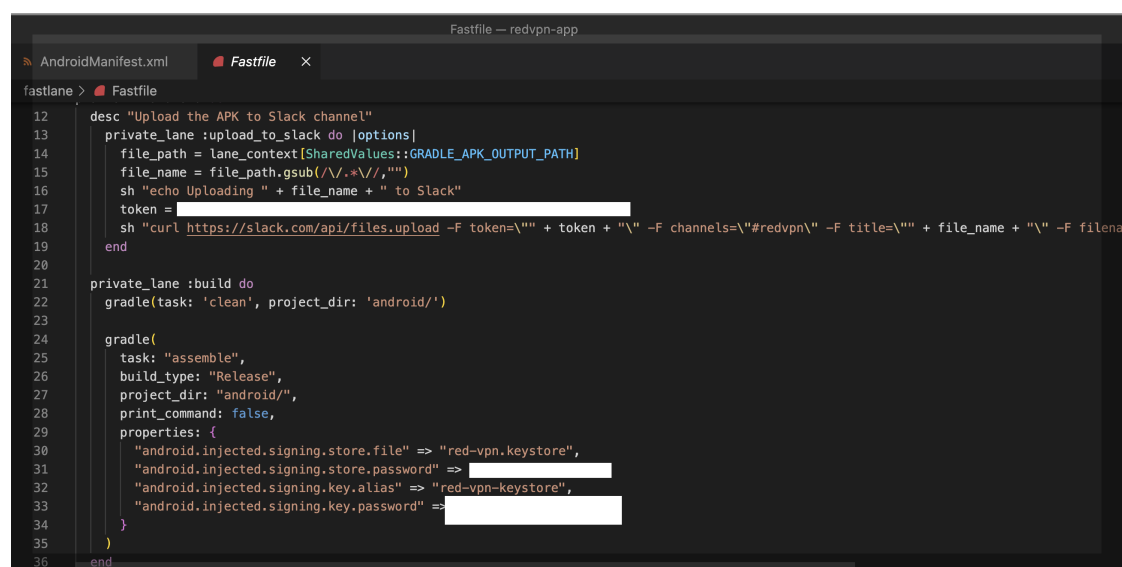
Discussion

Values that appeared to be credentials were discovered during review of the source code for the Android client. Specifically, the credentials used for signing the *release* build artifact as well as a token for connecting to Slack.

The credentials were found in the following files:

- gradle.properties
- fastlane/Fastfile

It's not clear if these credentials are valid, Subgraph did not attempt to use them. It was observed that there is a directive to overload certain environment variables, and this may replace some of the credentials in *Fastfile*, however this could not be confirmed without comprehensively understanding the CI process and perhaps running it, which was not done during this engagement.



```
Fastfile — redvpn-app
AndroidManifest.xml Fastfile x
fastlane > Fastfile
12 desc "Upload the APK to Slack channel"
13 private_lane :upload_to_slack do |options|
14   file_path = lane_context[SharedValues::GRADLE_APK_OUTPUT_PATH]
15   file_name = file_path.gsub(/\/.*$/, "")
16   sh "echo Uploading " + file_name + " to Slack"
17   token = [REDACTED]
18   sh "curl https://slack.com/api/files.upload -F token=\"\" + token + \"\" -F channels=\"#redvpn\" -F title=\"\" + file_name + \"\" -F filena
19 end
20
21 private_lane :build do
22   gradle(task: 'clean', project_dir: 'android/')
23
24   gradle(
25     task: "assemble",
26     build_type: "Release",
27     project_dir: "android/",
28     print_command: false,
29     properties: {
30       "android.injected.signing.store.file" => "red-vpn.keystore",
31       "android.injected.signing.store.password" => [REDACTED]
32       "android.injected.signing.key.alias" => "red-vpn-keystore",
33       "android.injected.signing.key.password" => [REDACTED]
34     }
35   )
36 end
```

Impact Analysis

A leak of the source code could result in malicious executables, or in a breach of development infrastructure.

Remediation Recommendations

Sensitive credentials should not be embedded in source code. Rather, they should be injected when needed during the build and release process.

Additional Information

N/A

Appendix

Methodology

Our approach to testing is designed to understand the design, behavior, and security considerations of the assets being tested. This helps us to achieve the best coverage over the duration of the test.

To accomplish this, Subgraph employs automated, manual and custom testing methods. We conduct our automated tests using the industry standard security tools. This may include using multiple tools to test for the same types of issues. We perform manual tests in cases where the automated tools are not adequate or reveal behavior that must be tested manually. Where required, we also develop custom tools to perform tests or reproduce test findings.

The goals of our testing methodology are to:

- Understand the expected behavior and business logic of the assets being tested
- Map out the attack surface
- Understand how authentication, authorization, and other security controls are implemented
- Test for flaws in the security controls based on our understanding
- Test every point of input against a large number of variables and observe the resulting behavior
- Reproduce and re-test findings
- Gather enough supporting information about findings to enable us to classify, report, and suggest remediations

Description of testing activities

Depending on the type and scope of the engagement, our methodology may include any of the following testing activities:

1. **Information Gathering:** Information will be gathered from publicly available sources to help increase the success of attacks or discover new vulnerabilities
2. **Network discovery:** The networks in scope will be scanned for active, reachable hosts that could be vulnerable to compromise
3. **Host Vulnerability Assessment:** Hosts applications and services will be assessed for known or possible vulnerabilities
4. **Application Exploration:** The application will be explored using manual and automated methods to better understand the attack surface and expected behavior
5. **Session Management:** Session management in web applications will be tested for security flaws that may allow unauthorized access
6. **Authentication System Review:** The authentication system will be reviewed to determine if it can be bypassed
7. **Privilege Escalation:** Privilege escalation checks will be performed to determine if it is possible for an authenticated user to gain access to the privileges assigned to another role or administrator

8. **Input Validation:** Input validation tests will be performed on all endpoints and fields within scope, including tests for injection vulnerabilities (SQL injection, cross-site scripting, command injection, etc.)
9. **Business Logic Review:** Business logic will be reviewed, including attempts to subvert the intended design to cause unexpected behavior or bypass security controls

Reporting

Findings reports are peer-reviewed within Subgraph to produce the highest quality findings. The report includes an itemized list of findings, classified by their severity and remediation status.

Severity ratings

Severity ratings are a metric to help organizations prioritize security findings. The severity ratings we provide are simple by design so that at a high-level they can be understood by different audiences. In lieu of a complex rating system, we quantify the various factors and considerations in the body of the security findings. For example, if there are mitigating factors that would reduce the severity of a vulnerability, the finding will include a description of those mitigations and our reasoning for adjusting the rating.

At an organization's request, we will also provide third-party ratings and classifications. For example, we can analyze the findings to produce *Common Vulnerability Scoring System* (CVSS)¹ scores or *OWASP Top 10*² classifications.

The following is a list of the severity ratings we use with some example impacts:

Critical

Exploitation could compromise hosts or highly sensitive information

Critical Exploitation could compromise hosts or highly sensitive information

High

Exploitation could compromise the application or moderately sensitive information

High Exploitation could compromise the application or moderately sensitive information

Medium

Exploitation compromises multiple security properties (confidentiality, integrity, or availability)

Medium Exploitation compromises multiple security properties (confidentiality, integrity, or availability)

¹<https://www.first.org/cvss/>

²https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Low

Exploitation compromises a single security property (confidentiality, integrity, or availability)

Low Exploitation compromises a single security property (confidentiality, integrity, or availability)

Info

Finding does not directly pose a security risk but merits further investigation

Info Finding does not directly pose a security risk but merits further investigation

The severity of a finding is often a product of the impact to general security properties of an application, host, network, or other information system.

The properties that can be impacted are:

Confidentiality Exploitation results in authorized access to data

Integrity Exploitation results in the unauthorized modification of data or state

Availability Exploitation results in a degradation of performance or an inability to access resources

The actual severity of a finding may be higher or lower depending on a number of other factors that may mitigate or exacerbate it. These include the context of the finding in relation to the organization as well as the likelihood of exploitation. These are described in further detail below.

Contextual factors

Confidentiality, integrity, and availability are one dimension of the potential risk of a security finding. In some cases, we must also consider contextual factors that are unique to the organization and the assets tested.

The following is a list of those factors:

Financial Exploitation may result in financial losses

Reputation Exploitation may result in damage to the reputation of the organization

Regulatory Exploitation may expose the organization to regulatory liability (e.g. make them non-compliant)

Organizational Exploitation may disrupt the operations of the organization

Likelihood

Likelihood measures how probable it is that an attacker exploit a finding.

This is determined by numerous factors, the most influential of which are listed below:

Authentication Whether or not the attack must be authenticated

Privileges Whether or not an authenticated attacker requires special privileges

Public exploit Whether or not exploit code is publicly available

Public knowledge Whether or not the finding is publicly known

Exploit complexity How complex it is for a skilled attacker to exploit the finding

Local vs. remote Whether or not the finding is exposed to the network

Accessibility Whether or not the affected asset is exposed on the public Internet

Discoverability How easy it is for the finding to be discovered by an attacker

Dependencies Whether or not exploitation is dependant on other findings such as information leaks

Remediation status

As part of our reporting, remediation recommendations are provided to the client. To help track the issues, we also provide a remediation status rating in the findings report.

In some cases, the organization may be confident to remediate the issue and test it internally. In other cases, Subgraph works with the organization to re-test the findings, resulting in a subsequent report reflecting remediation status updates.

If requested to re-test findings, we determine the remediation status based on our ability to reproduce the finding. This is based on our understanding of the finding and our awareness of potential variants at that time. To reproduce the results, the re-test environment should be as close to the original test environment as possible.

Security findings are often due to unexpected or unanticipated behavior that is not always understood by the testers or the developers. Therefore, it is possible that a finding or variations of the finding may still be present even if it is not reproducible during a re-test. While we will do our best to work with the organization to avoid this, it is still possible.

The findings report includes the following remediation status information:

Resolved

Finding is believed to be remediated, we can no longer reproduce it

Resolved Finding is believed to be remediation, we can no longer reproduce it

In progress

Finding is in the process of being remediated

In progress Finding is in the process of being remediated

Unresolved

Finding is unresolved – used in initial report or when the organization chooses not to resolve

Unresolved Finding is unresolved – used in initial report or when the organization chooses not to resolve

Not applicable

There is nothing to resolve, this may be the case with informational findings