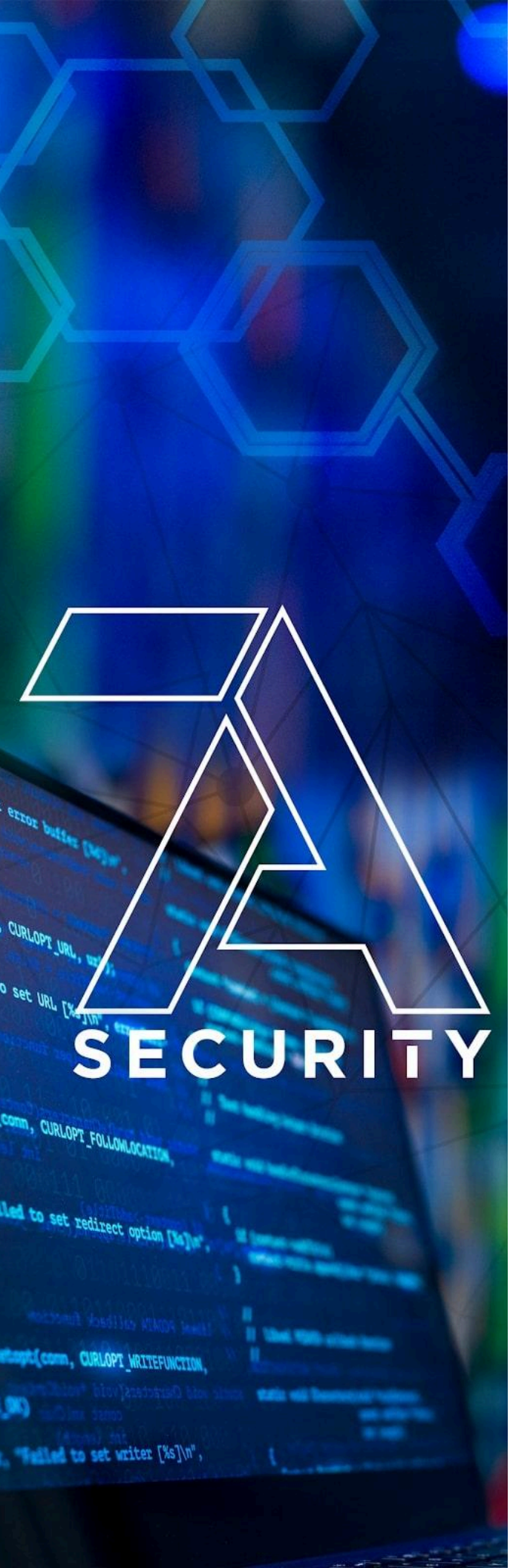

Test Targets:

AmneziaVPN Android & iOS
AmneziaVPN Desktop clients
AmneziaWG
AmneziaVPN XRay

Pentest Report

Client:
AmneziaVPN

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Daniel Ortiz, MSc.
- Jesus Arturo Espinoza Soto
- Miroslav Štampar, PhD.

7ASecurity
Protect Your Site & Apps
From Attackers
sales@7asecurity.com
7asecurity.com

INDEX

Introduction

"Create your personal VPN
Amnezia VPN — simple and free app to run a self-hosted VPN with high privacy requirements"

From https://amnezia.org/en

This document outlines the results of a penetration test and *whitebox* security review conducted against the AmneziaVPN platform. The project was solicited by AmneziaVPN and executed by 7ASecurity in July 2024. The audit team dedicated 16 working days to complete this assignment. Please note that this is the second penetration test for this project. Consequently, the identification of security weaknesses was expected to be more difficult during this engagement, as more vulnerabilities are identified and resolved after each testing cycle.

During this iteration the goal was to review the solution as thoroughly as possible, to ensure AmneziaVPN users can be provided with the best possible security. The methodology implemented was *whitebox*: 7ASecurity was provided with access to a staging environment, documentation, test users, and source code. A team of 4 senior auditors carried out all tasks required for this engagement, including preparation, delivery, documentation of findings and communication.

A number of necessary arrangements were in place by June 2024, 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 Signal chat group. The AmneziaVPN team was helpful and responsive throughout the audit, which ensured that 7ASecurity was provided with the necessary access and information at all times, thus avoiding unnecessary delays. 7ASecurity provided regular updates regarding the audit status and its interim findings during the engagement.

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

Identified Vulnerabilities	*Hardening Recommendations*	*Total Issues*
4	*6*	*10*

Moving forward, the scope section elaborates on the items under review, while 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 AmneziaVPN applications.

Scope

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

- **WP1: Whitebox Tests against AmneziaVPN Android & iOS apps**
  - Android:
    - https://github.com/amnezia-vpn/amneziawg-android
    - https://play.google.com/store/apps/details?id=org.amnezia.vpn
  - iOS:
    - https://github.com/amnezia-vpn/amneziawg-apple
    - https://apps.apple.com/us/app/amneziavpn/id1600529900
- **WP2: Whitebox Tests against AmneziaVPN Desktop clients**
  - https://github.com/amnezia-vpn/amnezia-client
- **WP3: Whitebox Tests against AmneziaWG and XRay**
  - https://github.com/amnezia-vpn/amneziawg-go
  - https://github.com/amnezia-vpn/amneziawg-windows
  - https://github.com/amnezia-vpn/amneziawg-exporter

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-02-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

AVP-02-001 WP1: Config Access via incorrect Android Keystore Usage *(Medium)*

Affected version: AmneziaVPN for Android v4.6.0.1.

It was found that the Android app fails to correctly leverage the Android Keystore[1], 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[2] and the 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*[3].

Affected File:
files/settings/AmneziaVPN.ORG/AmneziaVPN.conf

Affected Contents:

```
\"container\": \"amnezia-awg\"\n            }\n        ],\n
\"defaultContainer\": \"amnezia-awg\",\n        \"description\": \"Server 1\",\n
\"hostName\": \"[...].223.178.[...]\",\n        \"password\": \"r1F[...]\",\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*[4] or the *Android Keystore*[5] 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

[1] https://developer.android.com/training/articles/keystore
[2] https://www.cvedetails.com/vulnerability-list.php?vendor_id=1224&product_id=19997...
[3] https://developer.android.com/topic/security/data
[4] https://developer.android.com/topic/security/data
[5] https://developer.android.com/training/articles/keystore

protection features can be found in the official Android documentation[6].

AVP-02-006 WP2: Possible VPN Traffic Access via TunnelVision *(Medium)*

Retest Notes: Due to the killswitch AmneziaVPN features, which blocks all traffic routed outside the VPN, this issue may only be leveraged for selective Denial of Service purposes.

It was found that AmneziaVPN Windows, macOS and iOS clients are susceptible to the *TunnelVision*[7] attacking technique (CVE-2024-3661[8]). An attacker can bypass VPN encapsulation and route traffic outside the VPN tunnel to obtain the unencrypted VPN traffic from victim users, de-anonymize the traffic destination and leak the real IP addresses. This technique uses the *Dynamic Host Configuration Protocol* (DHCP), specifically exploiting DHCP option 121[9], which manages classless static routes in a client routing table. Please note Android VPN clients are unaffected by this issue, as they do not support DHCP option 121[10].

The TunnelVision technique can be used in various scenarios where an attacker can set up a rogue DHCP server and inject malicious routes into a target routing table. Here are some potential scenarios:

- ***Public Wi-Fi networks***: Attackers can set up rogue access points in public areas like cafes, airports, or libraries. Unsuspecting users who connect to these networks may receive DHCP configurations from the attacker server, including the malicious DHCP option 121. This would redirect the user traffic outside their VPN tunnel, allowing the attacker to intercept sensitive data.
- ***Compromised Local Networks***: In scenarios where an attacker has gained access to a local network, such as a corporate or home network, they can set up a rogue DHCP server. This server can issue DHCP responses with option 121, creating routes that bypass the VPN tunnel. This allows the attacker to monitor and manipulate the internet traffic from the victim.
- ***Man-in-the-Middle (MitM) Attacks***: In MitM attack scenarios, attackers position themselves between the victim and the VPN server. By controlling the DHCP responses, the attacker can inject malicious routes, forcing traffic to be routed outside the encrypted VPN tunnel. This would effectively nullify the privacy and security benefits of the VPN.

[6] https://developer.android.com/training/articles/keystore
[7] https://www.leviathansecurity.com/blog/tunnelvision
[8] https://nvd.nist.gov/vuln/detail/CVE-2024-3661
[9] https://datatracker.ietf.org/doc/html/rfc3442
[10] https://www.wired.com/story/tunnelvision-vpn-attack/

Proof of Concept (PoC):
This issue was confirmed using a rogue DHCP server in a lab environment. The *TunnelVision* GitHub page[11] contains all setup details. Here is a summary of the steps involved during the attack. Please note that the steps below belong to the Windows operating system but were also confirmed on macOS and iOS.

Steps to reproduce:

1. The attacker sets up a rogue DHCP Server in a Public Wi-Fi network or compromised local network.
2. The attacker configures DHCP option 121, a classless static route option to inject route entries in the routing table of victims.
3. A victim connects to the compromised Public Wi-Fi network or local network and receives an IP address from the DHCP server with the desired 121 option. Here is an example of a packet received by DHCP clients:

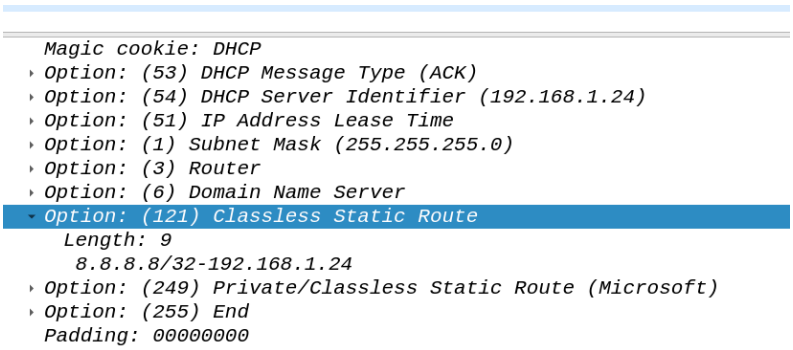

Fig.: Example DHCP packet with option 121, including a Classless Static Route

The route entry injected by the attacker can be confirmed on the victim routing table by executing the following commands.

Command:
```
route print | findstr /i "8.8.8.8"
```

Output:
```
8.8.8.8  255.255.255.255     192.168.1.24     192.168.1.10     26
```

4. The victim connects to the VPN using the desired client.
5. Using the route entry injected to the victim via a rogue DHCP, the attacker can leak VPN traffic or cause a selective *Denial of Service* (DoS):

Command (on victim machine):
```
ping 8.8.8.8
```

[11] https://github.com/leviathansecurity/TunnelVision?tab=readme-ov-file

Output (on victim machine):

```
Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=9ms TTL=127
Reply from 8.8.8.8: bytes=32 time=9ms TTL=127
Reply from 8.8.8.8: bytes=32 time=8ms TTL=127
Reply from 8.8.8.8: bytes=32 time=9ms TTL=127

Ping statistics for 8.8.8.8:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss)
[...]
```

Please note that the traffic goes outside the VPN interface and is sent to the rogue DHCP server via injected route entry.

Command (on rogue DHCP server):

```
sudo tcpdump -i ens33 icmp
```

Output (on rogue DHCP server):

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens33, link-type EN10MB (Ethernet), capture size 262144 bytes
09:49:43.997912 IP 192.168.1.10 > dns.google: ICMP echo request, id 1, seq 92,
length 40
09:49:44.006659 IP dns.google > 192.168.1.10: ICMP echo reply, id 1, seq 92,
length 40
09:49:45.020195 IP 192.168.1.10 > dns.google: ICMP echo request, id 1, seq 93,
length 40
09:49:45.028975 IP dns.google > 192.168.1.10: ICMP echo reply, id 1, seq 93,
length 40
09:49:46.026615 IP 192.168.1.10 > dns.google: ICMP echo request, id 1, seq 94,
```

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

- To ensure secure VPN usage on hostile networks, educate users to:
  - Follow the *WireGuard* documentation[12] to implement network namespaces on Linux to completely fix this behavior.
  - Use a personal hotspot with their VPN
  - Run their VPN inside a virtual machine without a bridged network adapter
  - Where possible, avoid untrusted networks like public Wi-Fi
  - Use ad-blockers and privacy-focused browsers that reject tracking cookies to enhance online privacy and protect against tracking and surveillance.
- In situations where VPN clients control their local network, the following additional mitigation measures could be implemented:

[12] https://www.wireguard.com/netns/#the-new-namespace-solution

- Enabling DHCP Snooping[13] and ARP Protection[14] to prevent rogue DHCP servers from operating on the network.
- Using Port Security[15] on Switches to limit the number of MAC addresses that can connect through a single port, preventing unauthorized devices from injecting malicious DHCP responses.
- Ignoring DHCP Option 121 on VPN clients. However, this may lead to connectivity issues and should be tested thoroughly.

AVP-02-009 WP1: VPN Config Access via Logcat Messages *(Medium)*

Affected version: AmneziaVPN for Android v4.6.0.1.

It was found that the AmneziaVPN Android app leaks the entire VPN configuration via *logcat* messages. A malicious attacker with access to an unlocked device could enable USB debugging and retrieve the VPN configuration from the *logcat buffer*[16], revealing both recent and previous messages that may contain sensitive information.

Proof of Concept:
This issue was identified while looking for *logcat* leaks, when an intent is sent to the *ImportConfigActivity* this leaks VPN configuration like the following:

Step 1: Send an intent to AmneziaVPN Android app with the VPN configuration

Command:
```
adb shell am start -W -a android.intent.action.VIEW -d
"vpn://eyJjb25maWdfdmVyc2lvbiI6IDEuMCwgImFwaV9lbmRwb2ludCI6ICJodHRwczovLzEzLjI0OC4xMzkuNDQvYXBpL3YxL3JlcXVlc[....]" org.amnezia.vpn
```

Step 2: Execute the following command on a USB debugging device

Command:
```
adb logcat -d  |  grep -i "ImportConfigActivity"
```

Output:
```
07-23 10:20:19.322  4908  7039 I ActivityTaskManager: START u0
{act=android.intent.action.VIEW
dat=vpn://eyJjb25maWdfdmVyc2lvbiI6IDEuMCwgImFwaV9lbmRwb2ludCI6ICJodHRwczovLzEzLjI0OC4xMzkuNDQvYXBpL3YxL3JlcXVlc[....] flg=0x10000000 pkg=org.amnezia.vpn
cmp=org.amnezia.vpn/.ImportConfigActivity} from uid 2000
```

[13] https://en.wikipedia.org/wiki/DHCP_snooping
[14] https://en.wikipedia.org/wiki/ARP_spoofing
[15] https://kb.netgear.com/21786/What-is-port-security-and-how-does-it-work-with-my-managed-switch
[16] https://developer.android.com/studio/command-line/logcat

As seen above, *ActivityTaskManager*[17] logs the VPN configuration in *logcat*. This Android class provides APIs for managing activities and tasks, and is controlled by the operating system. The intent filter involved is defined in the following file:

Affected File:
https://github.com/amnezia-vpn/[...]client/android/AndroidManifest.xml

Affected Code:

```
<activity
   android:name=".ImportConfigActivity"
   android:excludeFromRecents="true"
   android:launchMode="singleTask"
   android:taskAffinity=""
   android:exported="true"
   android:theme="@style/Translucent">

   [...]

   <intent-filter>
       <action android:name="android.intent.action.VIEW" />
       <category android:name="android.intent.category.DEFAULT" />

       <data android:scheme="file" />
       <data android:scheme="content" />
       <data android:mimeType="*/*" />
       <data android:host="*" />

       <data android:pathPattern=".*\\.vpn" />
       <data android:pathPattern=".*\\..*\\.vpn" />
       <data android:pathPattern=".*\\..*\\..*\\.vpn" />
       <data android:pathPattern=".*\\..*\\..*\\..*\\.vpn" />
       <data android:pathPattern=".*\\..*\\..*\\..*\\..*\\.vpn" />
       [...]
   </intent-filter>
</activity>
```

Limiting deep link leakage in *logcat* due to *ActivityTaskManager* is challenging because it is controlled by Android. However, the risk can still be mitigated using some of the following options:

1. Eliminate the option for sending VPN information via DeepLinks, or replace it with an alternative.
2. Limit Sensitive Data in URLs: Avoid passing sensitive information via deep links. Use tokens or session IDs verified server-side.
3. Encrypt Deep Link Parameters: Encrypt parameters to protect sensitive data, even if URLs are logged.

[17] https://android.googlesource.com/platform/.../core/java/android/app/ActivityTaskManager.java

More broadly, it is further advised to avoid logging sensitive information, especially when the *debug* flag is not set in the APK, as was the case in this assessment. Common approaches to implement this are:

- To create a *log wrapper*, check if the build is a *debug* build there, only log debug and verbose messages for a *debug* build[18].
- To create ProGuard rules so that *Log.d* and *Log.v* are removed when the build is for production[19].
- Avoid the usage of intent filters that may leak sensitive information to *logcat*.

AVP-02-010 WP1: API Key Access via Memory Leak *(Medium)*

Affected version: AmneziaVPN for Android v4.6.0.1.

It was found that the AmneziaVPN Android app keeps the configuration in memory. This approach is insecure because that information could be accessed by a malicious attacker with physical access or memory access. Furthermore, given the large volume of publicly known Android kernel vulnerabilities[20] and the 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.

To confirm this issue, filesystem usage was reviewed but no sensitive information was found. Hence, subsequently, the app process memory was dumped and the contents were reviewed for possible leaks. In particular, a search for the API key discovered an occurrence in memory.

Command 1: Search api_key in memdump file.

```
grep -ir "api_key" ./0x6fcd264000_dump.data –text
```

Output:

```
./0x6fcd264000_dump.data: "api_key": "Ez39deUm.e[...]ZIqlW2kYYb3Wi2P1A2JKd",
```

Command 2: Usage of api_key via cURL.

```
curl -X OPTIONS -H "Authorization: Api-Key Ez39deUm.e[...]ZIqlW2kYYb3Wi2P1A2JKd"
https://13.248.139.44/api/v1/request/awg/
```

Output:

```
{"name":"Awg Config Request","description":"Generates an AWG
config...","renders":["application/json","text/html"],"parses":["application/json","app
lication/x-www-form-urlencoded","multipart/form-data"],"actions":{"POST":{"public_key":
```

[18] https://stackoverflow.com/a/4592958
[19] https://stackoverflow.com/a/2466662
[20] https://www.cvedetails.com/vulnerability-list.php?vendor_id=1224&product_id=19997...

{"type":"string","required":true,"read_only":false,"label":"Public key"},"installation_uuid":{"type":"string","required":false,"read_only":false,"label":"Installation uuid"},"config":{"type":"string","required":false,"read_only":true,"label":"Config"},"os_version":{"type":"string","required":false,"read_only":false,"label":"Os version"},"app_version":{"type":"string","required":false,"read_only":false,"label":"App version"}}}}

The root cause for this issue may appear to be in the following code path, which uses a *QString* object to store the configuration data in memory without clearing it:

Affected File:
https://github.com/amnezia-vpn/amnezia-client/[...]/client/amnezia_application.cpp

Affected Code:

```
connect(AndroidController::instance(), &AndroidController::importConfigFromOutside,
[this](QString data) {
   m_pageController->replaceStartPage();
   m_importController->extractConfigFromData(data);
   m_pageController->goToPageViewConfig();
});
```

To resolve this issue, at a minimum, sensitive configuration information should be regularly wiped from memory to avoid potential leaks. Additionally, sensitive data like encryption keys, should only be retained in RAM briefly. Variables storing keys ought to be nullified after use. Also, when using Qt containers (like *QByteArray*[21], *QString*[22], etc.), ensure they are cleared explicitly using the *clear()*[23] function. Even after removing or nullifying references to immutable objects, they might persist in memory until garbage collection, which apps are unable to enforce. For additional mitigation guidance, please see the *Testing Memory for Sensitive Data* section of the *Mobile Application Security Testing Guide (MASTG)*[24].

[21] https://doc.qt.io/qt-6/qbytearray.html
[22] https://doc.qt.io/qt-6/qstring.html
[23] https://doc.qt.io/qt-6/qbytearray.html#clear
[24] https://mas.owasp.org/MASTG/tests/android/MASVS-STORAGE/MASTG-TEST-0011/

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-02-002 WP1: Android Config Hardening Recommendations *(Info)*

Affected version: AmneziaVPN for Android v4.6.0.1.

It was found that the AmneziaVPN Android app fails to leverage optimal values for a number of security-related settings. This unnecessarily weakens the overall security posture of the application. For example, the application fails to mitigate potential Tapjacking attacks. These weaknesses are documented in more detail next.

Issue 1: Missing Tapjacking Protection

The Android app accepts user taps while other apps render anything on top of it. Malicious attackers might leverage this weakness to impersonate users using a crafted app, which launches the victim app in the background while something else is rendered on top. Please note that this attack vector is mitigated in Android 12[25]. Since the app supports Android 7.0, this leaves users on Android 7.0-11 vulnerable to this attack. The following command confirms that Tapjacking protections are missing on the source code provided and the decompiled app:

Command:

```
egrep -r
'(filterTouchesWhenObscured|FLAG_WINDOW_IS_OBSCURED|FLAG_WINDOW_IS_PARTIALLY_OBSCURED)'
* | wc -l
```

Output:

```
0
```

Since Android API level 9 (Android 2.3), it is possible to mitigate Tapjacking attacks utilizing at least one of the following approaches:

[25] https://developer.android.com/topic/security/risks/tapjacking#mitigations

Approach 1: The *filterTouchesWhenObscured*[2627] attribute could be set at the Android root view level[28]. This will ensure that taps will be ignored when the Android app is not displayed on top.
Approach 2: Alternatively, *MotionEvents* could be checked against the following flags to present a protection screen on top:

1. *FLAG_WINDOW_IS_OBSCURED*[29] (since Android 2.3)
2. *FLAG_WINDOW_IS_PARTIALLY_OBSCURED*[30] (since Android 10)

Issue 2: Undefined *android:hasFragileUserData*

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

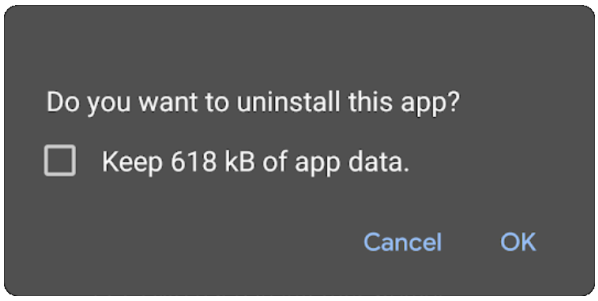

Fig.: Uninstall prompt with check box for keeping the app data

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 prompts asking whether data should be preserved or not.

[26] http://developer.android.com/reference/[...]/View.html#setFilterTouchesWhenObscured(boolean)
[27] http://developer.android.com/reference/[...]/View.html#attr_android:filterTouchesWhenObscured
[28] https://developer.android.com/reference/android/view/View#security
[29] https://developer.android.com/reference/android/view/MotionEvent#FLAG_WINDOW_IS_OBSCURED
[30] https://developer.android.com/reference/android/view/MotionEvent#FLAG_WINDOW_IS_PARTIALLY...

AVP-02-003 WP1: Missing Android root & iOS Jailbreak Detection *(Info)*

Affected versions: AmneziaVPN for Android v4.6.0.1, AmneziaVPN for iOS v4.5.4.

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 applications fail 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/jailbroken device and validating the complete lack of application warnings.

It is recommended to implement a root/jailbreak 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*[31] and *DTTJailbreakDetection*[32], although custom checks are also possible in Swift applications[33]. 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[34][35] solutions against well-known *Cydia tweaks* like *LibertyLite*[36], *Shadow*[37], *tsProtector 8+*[38] or *A-Bypass*[39]. Based on this, the development team could determine the most solid approach.

The freely available *rootbeer* library[40] 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.

[31] https://cocoapods.org/pods/IOSSecuritySuite
[32] https://github.com/thii/DTTJailbreakDetection
[33] https://sabatsachin.medium.com/detect-jailbreak-device-in-swift-5-ios-programatically-da467028242d
[34] https://github.com/thii/DTTJailbreakDetection
[35] https://github.com/securing/IOSSecuritySuite
[36] http://ryleyangus.com/repo/
[37] https://ios.jjolano.me/
[38] http://apt.thebigboss.org/repofiles/cydia/
[39] https://repo.rpgfarm.com/
[40] https://github.com/scottyab/rootbeer

AVP-02-004 WP2/3: Possible Weaknesses via Insecure Function Usage *(Medium)*

During the code review, multiple instances of insecure coding practices were identified across several application components. These involve using functions known to be potentially unsafe or banned due to their susceptibility to memory corruption vulnerabilities, buffer overflows, or undefined behavior. These functions should be avoided or replaced with safer alternatives. The following examples illustrate this issue:

Issue 1: Unsafe usage of strncpy

strncpy may not null-terminate the destination string if the source string length equals or exceeds the destination size, potentially causing buffer overflow issues.

Affected Files:
https://github.com/amnezia-vpn/amnezia-client/[...]/linux/daemon/iputilslinux.cpp
https://github.com/amnezia-vpn/amnezia-client/[...]/macos/daemon/iputilsmacos.cpp

Affected Code:
```
strncpy(ifr.ifr_name, WG_INTERFACE, IFNAMSIZ);
[...]
strncpy(ifr.ifra_name, qPrintable(ifname), IFNAMSIZ);
```

It is advised to replace *strncpy* with *strncpy_s*[41] or *strlcpy*[42]: These functions ensure proper null-termination and prevent buffer overflows by limiting the number of characters copied.

Issue 2: Unsafe usage of strcpy

strcpy does not check the buffer size and can lead to buffer overflows, if the destination buffer is not large enough to hold the source string.

Affected File:
https://github.com/amnezia-vpn/amnezia-client/[...]/service/server/router_mac.cpp

Affected Code:
```
strcpy(argv[i], parts.at(i).toStdString().c_str());
```

It is recommended to replace *strcpy* with *strcpy_s*[43] or *strlcpy*[44]: These functions perform bounds checking and provide a safer alternative to *strcpy*.

[41] https://what.thedailywtf.com/topic/4012/more-secure-than-strncpy-strncpy_s
[42] https://what.thedailywtf.com/post/77038
[43] https://stackoverflow.com/questions/59239734/why-strcpy-s-is-safer-than-strcpy
[44] https://stackoverflow.com/questions/6987217/strncpy-or-strlcpy-in-my-case

Issue 3: Insecure use of memset

memset can be optimized out by the compiler in certain scenarios, especially when used to clear sensitive data, leading to potential security risks.

Affected Files:
https://github.com/amnezia-vpn/amnezia-client/[...]/service/server/helper_route_mac.c
https://github.com/amnezia-vpn/amneziawg-apple/[...]/Shared/Logging/ringlogger.c
https://github.com/amnezia-vpn/amneziawg-apple/[...]/macOS/View/highlighter.c

Affected Code:
```
memset((void *)&so_mask, 0, sizeof(so_mask));
[...]
memset(&in6, 0, sizeof(in6));
[...]
memset(&hints, 0, sizeof(hints));
```

It is encouraged to replace *memset* with *memset_s*[45]: This ensures the memory set operation is not optimized out, which is particularly useful for clearing sensitive data.

Issue 4: Unsafe usage of sprintf

sprintf does not perform bounds checking, making it susceptible to buffer overflows, if the destination buffer is too small.

Affected File:
https://github.com/amnezia-vpn/amnezia-client/[...]/service/server/helper_route_mac.c

Affected Code:
```
#define C(x)   (unsigned)((x) & 0xff)
    if (cp != NULL)
        strlcpy(line, cp, sizeof(line));
    else if ((in.s_addr & 0xffffff) == 0)
        (void) sprintf(line, "%u", C(in.s_addr >> 24));
    else if ((in.s_addr & 0xffff) == 0)
        (void) sprintf(line, "%u.%u", C(in.s_addr >> 24),
            C(in.s_addr >> 16));
    else if ((in.s_addr & 0xff) == 0)
        (void) sprintf(line, "%u.%u.%u", C(in.s_addr >> 24),
            C(in.s_addr >> 16), C(in.s_addr >> 8));
    else
        (void) sprintf(line, "%u.%u.%u.%u", C(in.s_addr >> 24),
            C(in.s_addr >> 16), C(in.s_addr >> 8),
```

[45] https://stackoverflow.com/questions/…/memset-s-what-does-the-standard-mean-with-this-piece-of-text

```
            C(in.s_addr));
#undef C
```

It is suggested to replace *sprintf* with *sprintf_s*[46] or *snprintf*[47]: These functions include bounds checking to prevent buffer overflows.

Issue 5: Incorrect use of atoi

atoi does not handle errors gracefully and can lead to undefined behavior, if the input is not a valid integer.

Affected File:
https://github.com/amnezia-vpn/amnezia-client/[...]/service/server/helper_route_mac.c

Affected Code:
```
*valp = atoi(value);
[...]
int len = atoi(s), q, r;
```

It is advised to replace *atoi* with *strtol*[48]: This offers better error handling and avoids undefined behavior by checking the input string validity.

By addressing these issues, the resilience against potential memory-based attacks and undefined behavior can be significantly improved.

AVP-02-005 WP1/3: Possible Vulnerabilities via Outdated Go *(Low)*

The *net/netip* library in *Go 1.22.3* is affected by *GO-2024-2887*[49] (*CVE-2024-24790*[50]), impacting the behavior of Is-prefixed methods for IPv4-mapped IPv6 addresses. Methods like *IsPrivate*, *IsLoopback*, etc. return false for IPv4-mapped IPv6 addresses, which would return true in their IPv4 forms. This may lead to incorrect network address classification, potentially affecting security decisions and improper handling of IPv4-mapped IPv6 addresses in network operations.

Affected Files:
https://github.com/amnezia-vpn/amneziawg-go/[...]/go.mod
https://github.com/amnezia-vpn/amneziawg-go/[...]/Dockerfile
https://github.com/amnezia-vpn/amneziawg-apple/[...]/Sources/WireGuardKitGo/go.mod

[46] https://www.reddit.com/r/learnprogramming/comments/t9s3ia/is_cs_sprintf_function_actually_unsafe/
[47] https://github.com/libarchive/libarchive/issues/1743
[48] https://blog.mozilla.org/nnethercote/2009/03/13/atol-considered-harmful/
[49] https://pkg.go.dev/vuln/GO-2024-2887
[50] https://nvd.nist.gov/vuln/detail/CVE-2024-24790

https://github.com/amnezia-vpn/amneziawg-android/[...]/tunnel/tools/libwg-go/Makefile
https://github.com/amnezia-vpn/amneziawg-android/[...]/tunnel/tools/libwg-go/go.mod

Example Code:

```
module github.com/amnezia-vpn/amneziawg-go

go 1.22.3

require (
    github.com/tevino/abool/v2 v2.1.0
    golang.org/x/crypto v0.21.0
    golang.org/x/net v0.21.0
    golang.org/x/sys v0.18.0
    golang.zx2c4.com/wintun v0.0.0-20230126152724-0fa3db229ce2
    gvisor.dev/gvisor v0.0.0-20230927004350-cbd86285d259
)

require (
    github.com/google/btree v1.0.1 // indirect
    golang.org/x/time v0.0.0-20220210224613-90d013bbcef8 // indirect
)
```

It is recommended to update the Go version to 1.22.4 or later in all affected components, which contains the fix for this vulnerability. Furthermore, additional vulnerabilities were found in other modules, but the code does not appear to invoke the code affected by these vulnerabilities directly. Thus, running the following command will ensure all dependencies are updated and the affected components are rebuilt. Please note everything ought to be thoroughly tested prior to deployment.

Proposed Fix Command:

```
go mod tidy
```

AVP-02-007 WP2/3: Masking/Config Weaknesses via Insecure PRNGs *(Medium)*

Two instances of insecure *Pseudo-Random Number Generator* (PRNG)[51] usage were identified, potentially compromising traffic masking effectiveness and configuration data security.

Issue 1: Usage of Insecure PRNG in Junk Packet Creation

The junk packet creation mechanism uses a non-cryptographically secure random number generator provided by the *math/rand* package, which could lead to predictable outcomes and compromise the *traffic masking* mechanism[52].

[51] https://en.wikipedia.org/wiki/Pseudorandom_number_generator
[52] https://docs.amnezia.org/documentation/how-amnezia-works/

Affected File:
https://github.com/amnezia-vpn/amneziawg-go/[...]/device/send.go

Affected Code:
```
import (
    "math/rand"
[...]
func (peer *Peer) createJunkPackets() ([][]byte, error) {
    [...]
    junks := make([][]byte, 0, peer.device.aSecCfg.junkPacketCount)
    for i := 0; i < peer.device.aSecCfg.junkPacketCount; i++ {
        packetSize := rand.Intn(
peer.device.aSecCfg.junkPacketMaxSize-peer.device.aSecCfg.junkPacketMinSize,
        ) + peer.device.aSecCfg.junkPacketMinSize
        junk, err := randomJunkWithSize(packetSize)
        [...]
        junks = append(junks, junk)
    }
    return junks, nil
}
```

The use of *math/rand.Intn()* for determining junk packet size is not cryptographically secure. This can lead to predictable packet sizes, potentially allowing an attacker to distinguish junk packets from real ones through traffic analysis.

It is worth noting that the subsequently called function *randomJunkWithSize* is not affected by this same vulnerability, as it properly utilizes *crypto/rand* for generating the actual content of the junk packets. However, the predictability of packet sizes could still compromise the effectiveness of the obfuscation mechanism.

It is recommended to replace the use of *math/rand* with *crypto/rand*. This ensures cryptographically secure random number generation, significantly enhancing the unpredictability of junk packet sizes.

Proposed Fix:
```
import (
    "crypto/rand"
    "encoding/binary"
[...]
func (peer *Peer) createJunkPackets() ([][]byte, error) {
    [...]
    randomBytes := make([]byte, 8)
    _, err := rand.Read(randomBytes)
    if err != nil {
        return nil, err
    }
```

```
    packetSize := int(binary.BigEndian.Uint64(randomBytes) % uint64(peer.device.
aSecCfg.junkPacketMaxSize-peer.device.aSecCfg.junkPacketMinSize)) + peer.device.
aSecCfg.junkPacketMinSize
    [...]
}
```

Issue 2: Usage of Insecure PRNG in Configuration Generation

The random string generation mechanism used for filling configuration values uses *QRandomGenerator::global()*, which is not suitable for security-sensitive purposes.

Affected File:
https://github.com/amnezia-vpn/amnezia-client/[...]/client/utilities.cpp

Affected Code:

```
QString Utils::getRandomString(int len)
{
    const QString possibleCharacters("
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789");

    QString randomString;
    for (int i = 0; i < len; ++i) {
       quint32 index = QRandomGenerator::global()->generate() % possibleCharacters.
length();
        QChar nextChar = possibleCharacters.at(index);
        randomString.append(nextChar);
    }
    return randomString;
}
```

The use of *QRandomGenerator::global()* for generating random values could lead to predictable outcomes or non-random values[53], potentially compromising the security of secret connection data configuration values.

It is recommended to replace the *QRandomGenerator::global()* with a cryptographically secure random number generator, such as *QRandomGenerator::system()*[54].

Addressing these issues will significantly improve the resilience against potential traffic analysis attacks and enhance the overall security of the junk packet generation and configuration data in the system.

[53] https://forums.gentoo.org/viewtopic-p-8647783.html?sid=29d2dcdf93977b3a96d1b7e2118359ba
[54] https://doc.qt.io/qt-6/qrandomgenerator.html#system

AVP-02-008 WP1: Insecure Local Networking via ATS Exception *(Low)*

Note: While TLS over port 443 is used by default, the ATS exception is needed for a fallback mechanism that does not rely on public certificate infrastructure, but still uses encryption.

Affected version: AmneziaVPN for iOS v4.5.4.

Since iOS 9, iOS disabled clear-text HTTP communications by default. However, developers can define exceptions to these secure defaults. It was found that the iOS app implements a local networking exception[55], which allows insecure communications on local networks. This may result in malicious Man-In-The-Middle (MitM) scenarios in situations where the application makes an insecure local network request while an attacker is able to modify clear-text HTTP communications.

This issue can be confirmed by reviewing the *Info.plist* file of the application:

Affected File:
Info.plist

Affected Setting:

```
<key>NSAppTransportSecurity</key>
        <dict>
                <key>NSAllowsArbitraryLoads</key>
                <false/>
                <key>NSAllowsLocalNetworking</key>
                <true/>
        </dict>
```

Please note that *NSAllowsLocalNetworking* is set to NO by default. Setting it to YES will disable ATS for connections over a local network[56].

It is recommended to remove all ATS exceptions in order to provide the best protection to application end users.

[55] https://developer.apple.com/.../nsapptransportsecurity/nsallowslocalnetworking
[56] https://www.nowsecure.com/...-nsallowsarbitraryloads-app-transport-security-ats-exceptions/

Conclusion

Despite the number of findings encountered in this exercise, the AmneziaVPN solution defended itself well against a broad range of attack vectors. The solution will become increasingly difficult to attack as additional cycles of security testing and subsequent hardening continue.

The AmneziaVPN solution provided a number of positive impressions during this assignment that must be mentioned here:

- The source code is very well-written, easy to read, and generally adheres to a number of security best practices.
- The Android and iOS mobile applications offer a minimal attack surface, reducing the potential for security vulnerabilities.
- The apps prevent screenshot leaks. This enhances security by safeguarding confidential information from being captured and shared without authorization.
- The Android app prevents backups, debugging, and task hijacking, which protects user privacy and eliminates entire attack vectors.
- The Android app exports only secure components with appropriate permissions, implements adequate exception handling and was found to be resilient against a number of intent-based attacks. It also avoids hardcoded API keys and enforces encryption for all traffic.
- Access control was found to be well-implemented, whereby users cannot read, modify, or delete data from other users.
- It was observed that the solution does not reveal any sensitive data or cookies to third-party websites.

The security of the AmneziaVPN solution will improve substantially with a focus on the following areas:

- **Hardware-backed Security Enclave Usage:** The Android application should leverage the hardware-backed security enclave available in the Android Keystore for the best protection of sensitive data at rest, such as PII, credentials, tokens, and alternative information (AVP-02-001).
- **User Education:** While mitigation of the *TunnelVision* attack is difficult, some countermeasures are possible via user education (AVP-02-006). This will protect users against de-anonymization and VPN traffic interception attacks from high-profile adversaries.
- **Information leakage:** The Android application should implement measures to mitigate risks associated with sensitive information leakage through *logcat* messages (AVP-02-009), and to prevent exposure of sensitive data in memory (AVP-02-010).
- **Software Patching:** The AmneziaVPN solution should implement appropriate software patching procedures that regularly apply security patches in a timely

manner (AVP-02-005). In a day and age when most lines of code come from underlying software dependencies, regularly patching these becomes increasingly important to avoid unwanted security vulnerabilities. Possible automation for this could include tools like *Snyk.io*[57] or *Renovate Bot*[58].

- **Removal of Unsafe Functions:** The AmneziaVPN 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, improve resilience against potential traffic analysis attacks, and enhance overall security (AVP-02-004, AVP-02-007).
- **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-02-003), removing all ATS exceptions to protect users from Man-In-The-Middle (MitM) attacks (AVP-02-008), and hardening several configuration options (AVP-02-002) to protect AmneziaVPN users.

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 application significantly, but also reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, ideally including another source code audit, is highly recommended to ensure adequate security coverage of the platform. This provides auditors with an edge over possible malicious adversaries that do not have significant time or budget constraints.

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 AmneziaVPN resources.

It is suggested to test the application 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 application highly resilient against online attacks over time.

[57] https://snyk.io/
[58] https://github.com/renovatebot/renovate

7ASecurity would like to take this opportunity to sincerely thank Mazay Banzaev and the rest of the AmneziaVPN team, for their exemplary assistance and support throughout this audit.