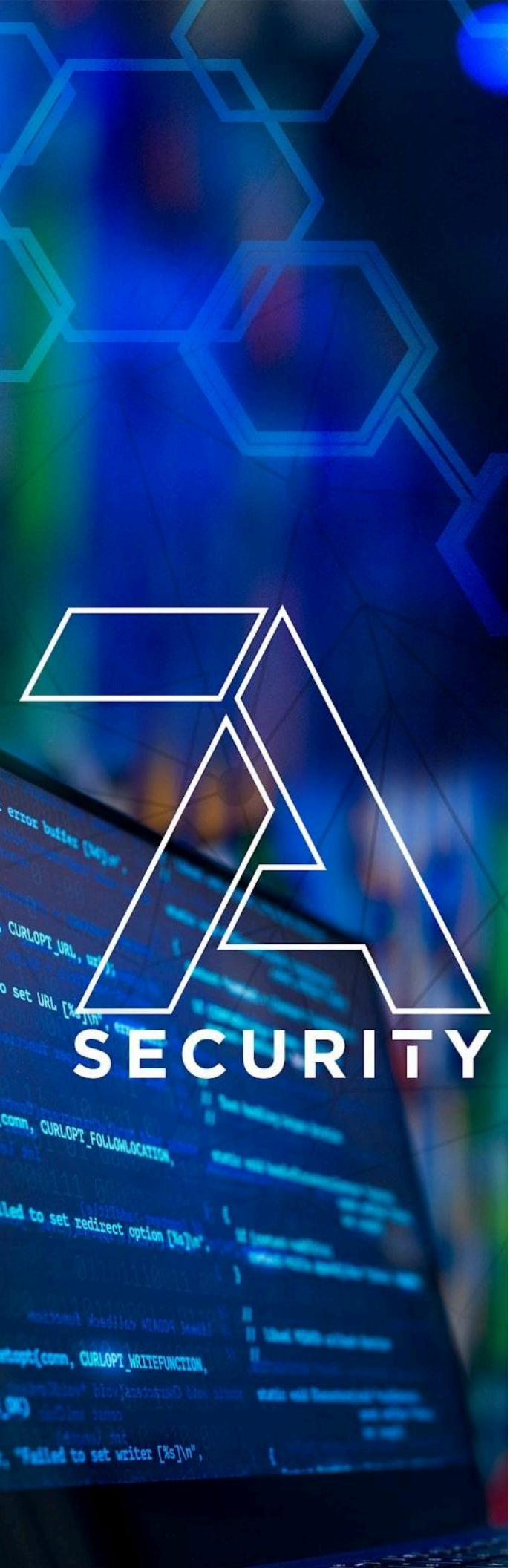

Test Targets:

AmneziaVPN Security
AmneziaVPN Privacy
AmneziaVPN Code Changes
AmneziaVPN Supply Chain
AmneziaVPN Threat Model

Pentest Report

Client:
AmneziaVPN

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Daniel Ortiz, MSc.
- Miroslav Štampar, PhD.
- Szymon Grzybowski, MSc.

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

INDEX

Introduction

“More than a VPN
Access resources blocked in your region or create your own private VPN with ease”
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, funded by the Open Technology Fund (OTF), and executed by 7ASecurity in December 2024 and January 2025. The audit team dedicated 18 working days to complete this assignment. Please note that this is the third 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.

Due to a recent audit in July 2024[1], during this iteration the goal was to review a number of security and privacy features, recent code changes, review the supply chain security and create a lightweight threat model. The methodology implemented was *whitebox*: 7ASecurity was provided with access to a staging environment, documentation, 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 December 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.

This audit split the scope items into the following work packages, which are referenced in the ticket headlines as applicable:

- WP1: Whitebox tests for AmneziaVPN Security, Privacy & Recent Code Updates
- WP2: Whitebox Tests against AmneziaVPN Supply Chain Implementation
- WP3: AmneziaVPN Lightweight Threat Model documentation

[1] https://7asecurity.com/reports/pentest-report-amneziavpn2.pdf

The findings of the security audit (WP1) can be summarized as follows:

Identified Vulnerabilities	*Hardening Recommendations*	*Total Issues*
3	*3*	*6*

Please note that the analysis of the remaining work packages (WP2, WP3) is provided separately, in the following section of this report:

- WP2: AmneziaVPN Supply Chain Implementation
- WP3: AmneziaVPN Lightweight Threat Model

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 for AmneziaVPN Security, Privacy & Recent Updates**
  - https://github.com/amnezia-vpn/amnezia-client
  - https://github.com/amnezia-vpn/amneziawg-go
  - https://github.com/amnezia-vpn/amneziawg-windows
  - https://github.com/amnezia-vpn/amneziawg-windows-client
  - https://github.com/amnezia-vpn/amneziawg-exporter
  - https://github.com/amnezia-vpn/amneziawg-android
  - https://github.com/amnezia-vpn/amneziawg-apple
- **WP2: Whitebox Tests against AmneziaVPN Supply Chain Implementation**
  - As above
- **WP3: AmneziaVPN Lightweight Threat Model documentation**
  - As above

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-03-001*) for ease of reference, and offers an estimated severity in brackets alongside the title.

AVP-03-004 WP1: DoS via Insecure Communication in AmneziaVPN Client *(High)*

Retest Notes: Resolved[2] by AmneziaVPN and confirmed by 7ASecurity.

The AmneziaVPN client has a Denial-of-Service (DoS) vulnerability affecting both free and paid VPN services. This issue arises from insecure communication within the function chain *getServicesList()* → *shouldBypassProxy()* → *getProxyUrls()*. The *getServicesList()* function sends an unencrypted HTTP request to *http://gw.amnezia.org/v1/services*, making it susceptible to manipulation. If *shouldBypassProxy()* returns *true* due to redirections or timeouts, *getProxyUrls()* retrieves proxy endpoints. Although the response is encrypted, it can be easily decrypted using a known key and IV, leaving the proxy list vulnerable to blocking.

This allows attackers to disrupt gateway and proxy resolution, leading to connectivity failures. No evidence suggests data exposure or unauthorized access, but affected users may be unable to connect to AmneziaVPN services. Even though this covers both the gateway and failback proxy communication, During proof-of-concept testing, it was confirmed that redirecting the DNS resolution of *gw.amnezia.org* to 127.0.0.1 is sufficient to disrupt both free and paid VPN services.

Issue 1: DoS via Gateway DNS spoofing

To demonstrate this issue, the *dnschef*[3] tool was used to spoof the DNS response for the gateway domain, redirecting it to *127.0.0.1*, effectively disrupting the connectivity.

Command:

```
dnschef.py -i 192.168.0.107 --fakeip 127.0.0.1 --fakedomains gw.amnezia.org
```

Output:

```
[...]
(00:47:04) [*] Cooking A replies to point to 127.0.0.1 matching: gw.amnezia.org
[...]
```

[2] https://github.com/amnezia-vpn/amnezia-client/releases/tag/4.8.7.2
[3] https://github.com/iphelix/dnschef

Result:
Attempting to initiate a "VPN by Amnezia" connection within the AmneziaVPN client results in an error message, preventing the user from connecting to the VPN service.

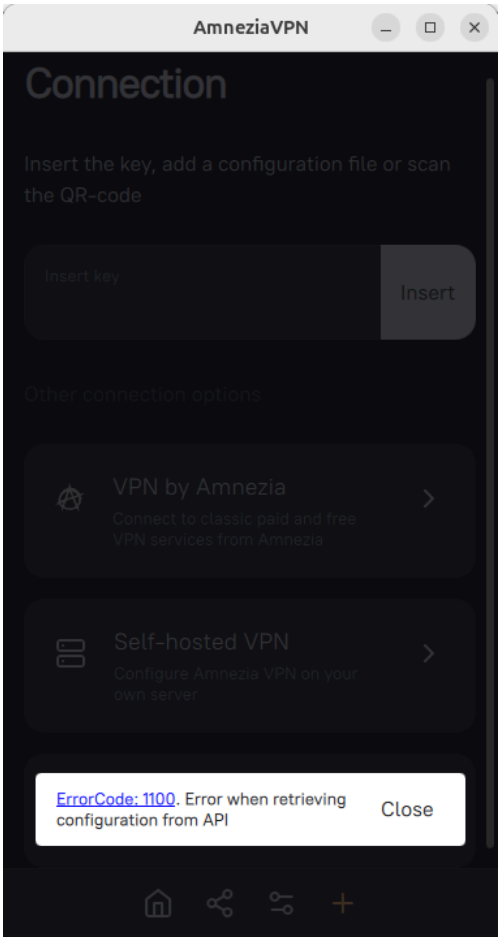

Fig.: Service Disruption in AmneziaVPN Client Due to DoS Attack

Issue 2: DoS via DNS spoofing of Failback Proxies

An attacker could escalate the attack by enumerating the failback proxy list used by the AmneziaVPN client and blocking their DNS resolution to disrupt connectivity. The following script demonstrates how to retrieve these failback proxies, which could then be rendered inaccessible through targeted DNS manipulation:

PoC Script:
https://7as.es/AmneziaVPN_uEuw9k6N/fetch_proxies.py

Command:
```
python3 fetch_proxies.py
```

Output:

```
[
  "https://sbeo7y6ouqvhhxviizzbfvxale0lyhsz.lambda-url.eu-north-1.on.aws:443/"
]
```

The root cause for this issue can be observed in the following code paths:

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

Affected Code:

```
namespace
{
   const char cloudFlareNs1[] = "1.1.1.1";
   const char cloudFlareNs2[] = "1.0.0.1";

   constexpr char gatewayEndpoint[] = "http://gw.amnezia.org:80/";
}
```

Affected File:
https://github.com/amnezia-vpn/amnezia-client/[...]/core/controllers/apiController.cpp

Affected Code:

```
ErrorCode ApiController::getServicesList(QByteArray &responseBody)
{
    [...]
    request.setUrl(QString("%1v1/services").arg(m_gatewayEndpoint));
    [...]
    if (sslErrors.isEmpty() && shouldBypassProxy(reply, responseBody, false)) {
        m_proxyUrls = getProxyUrls();
        [...]
        for (const QString &proxyUrl : m_proxyUrls) {
              request.setUrl(QString("%1v1/services").arg(proxyUrl));
        [...]
}

bool shouldBypassProxy(QNetworkReply *reply, const QByteArray &responseBody, bool
    checkEncryption, const QByteArray &key = "",
    const QByteArray &iv = "", const QByteArray &salt = "")
{
    if (reply->error() == QNetworkReply::NetworkError::OperationCanceledError
    || reply->error() == QNetworkReply::NetworkError::TimeoutError) {
        qDebug() << "Timeout occurred";
        return true;
    } else if (responseBody.contains("html")) {
        qDebug() << "The response contains an html tag";
        return true;
    }
```

```
    [...]
}

QStringList ApiController::getProxyUrls()
{
    [...]
    QStringList proxyStorageUrl;
    if (m_isDevEnvironment) {
        proxyStorageUrl = QStringList { DEV_S3_ENDPOINT };
    } else {
        proxyStorageUrl = QStringList { PROD_S3_ENDPOINT };
    }

    QByteArray key = m_isDevEnvironment ? DEV_AGW_PUBLIC_KEY : PROD_AGW_PUBLIC_KEY;

    for (const auto &proxyStorageUrl : proxyStorageUrl) {
        request.setUrl(proxyStorageUrl);
        reply = amnApp->manager()->get(request);

        auto encryptedResponseBody = reply->readAll();

        if (!m_isDevEnvironment) {
            QCryptographicHash hash(QCryptographicHash::Sha512);
            hash.addData(key);
            QByteArray hashResult = hash.result().toHex();

            QByteArray key = QByteArray::fromHex(hashResult.left(64));
            QByteArray iv = QByteArray::fromHex(hashResult.mid(64, 32));
            QByteArray ba = QByteArray::fromBase64(encryptedResponseBody);

            QSimpleCrypto::QBlockCipher blockCipher;
            responseBody = blockCipher.decryptAesBlockCipher(ba, key, iv);
            [...]
}
```

To resolve this issue, all gateway communications must be enforced over HTTPS to prevent interception and manipulation. Error handling in *shouldBypassProxy()* should be strengthened to resist exploitation through forced timeouts or crafted responses, ensuring secure proxy resolution.

Since the proxy list must remain accessible without authentication, encryption with a known key and IV provides no meaningful protection and should not be relied upon. Instead, resilience should be prioritized through the following measures:

- Regularly rotate proxy endpoints to reduce the impact of DNS-based blocking.
- Distribute proxy lists via multiple channels, including hardcoded fallback endpoints and dynamically updated lists.

- Protect DNS resolution using *DNS-over-HTTPS (DoH)*[4] or *DNS-over-TLS (DoT)*[5] to prevent interception and manipulation.
- Explore decentralized, peer-assisted and deliberately slow proxy discovery mechanisms to reduce reliance on a single retrieval method, and make proxy identification as difficult as possible, without compromising the user experience.

As attackers can inevitably retrieve the proxy list, the focus should be on resilience rather than obfuscation. Nevertheless, *Psiphon*[6] employs a strategy to prevent censors from easily obtaining and blocking its entire proxy or node list, which could be explored by AmneziaVPN for further resilience against censorship.

AVP-03-005 WP1/3: VPN Config Tampering via Exposed Admin API *(**Critical**)*

Retest Notes: Resolved by AmneziaVPN and confirmed by 7ASecurity.

The AmneziaVPN Premium service was found to expose the *amnezia-backend* service to the Internet with only basic authorization. Any user purchasing the premium service via platforms such as *vpnpay.io* can extract a valid API key and the API backend IP address from the *vpn://textkey* used to import VPN configurations into the AmneziaVPN client, allowing manual queries to administrative backend endpoints hosted in AWS.

A source code review revealed that predictable S3 bucket names storing configuration files could enable unauthorized creation or modification of VPN configurations for premium users. This could result in a full compromise of all service users, allowing an attacker to redirect traffic to attacker-controlled machines or deliver malicious VPN configurations to end-user devices.

Issue 1: Creation of arbitrary VPN configuration in existing bucket

The following cURL command, using a valid regular-user *api-key* extracted from the Amnezia Premium VPN configuration, creates an empty VPN host configuration pointing to a bogus localhost, confirming backend resource creation.

Command:

```
curl -i -H "content-type: application/json" -H "Authorization: Api-Key Def[...]"
https://52.223.54.40/api/v1/worker/create-or-update -d
'{"host":"127.0.0.1","bucket_name":"pl","config":""}'
```

Output:

```
HTTP/2 201
```

[4] https://developers.cloudflare.com/1.1.1.1/encryption/dns-over-https/
[5] https://www.cloudflare.com/learning/dns/dns-over-tls/
[6] https://psiphon.ca/

```
date: Thu, 30 Jan 2025 20:09:42 GMT
content-type: application/json
[...]

"A new Worker [76][127.0.0.1] was created."
```

Code inspection indicated that existing configurations could be modified, but this was not tested to avoid service disruption.

Affected File:
amnezia-backend/app/amnezia/views/upload.py

Affected Code:

```
class WorkerCreateUpdateView(CreateAPIView):
    """Updates the config in the provided bucket-worker pair..."""
[...]
        worker_query = {'host': worker_host, 'bucket_id': bucket.id}
        if country_code:
            worker_query['country_code'] = country_code
        try:
            worker = Worker.objects.filter(**worker_query).get()

            if worker.config == config:
                response_messages.append(f'The `config` field is the same as in the
Worker [{worker.host}].')
                return Response(
                    data='\n'.join(response_messages),
                    status=status.HTTP_304_NOT_MODIFIED,
                )
            worker.config = config
            worker.save()

            response_messages.append(f'Worker [{worker.id}][{worker.host}] was
updated.')
            return Response(
                data='\n'.join(response_messages),
                status=status.HTTP_204_NO_CONTENT,
            )

        # Create a new Worker...
        except Worker.DoesNotExist:
            worker_query['config'] = config
            worker = Worker.objects.create(**worker_query)
            worker.save()
[...]
```

Issue 2: Extraction of Worker IP addresses via admin API endpoint

The following command confirms broken authorization, enabling the listing of worker IP addresses for a guessable *bucket_id* via an administrative endpoint:

Command:

```
curl -i -H "content-type: application/json" -H "Authorization: Api-Key Def[...]"
https://52.223.54.40/api/v1/admin/get_workers/?bucket_id=1
```

Output:

```
HTTP/2 200
date: Fri, 31 Jan 2025 18:18:28 GMT
content-type: application/json
content-length: 303
server: nginx
vary: Accept, Cookie
allow: GET, HEAD, OPTIONS
x-frame-options: DENY
x-content-type-options: nosniff
referrer-policy: same-origin
cross-origin-opener-policy: same-origin

{"1": "185.88.141.48", "7": "185.200.106.68", "21": "51.158.237.213", "26":
"185.200.105.13", "33": "51.159.224.137", "38": "51.159.224.212", "42":
"51.159.224.209", "49": "51.159.225.1", "47": "51.159.224.208", "52": "176.56.182.155",
"58": "176.56.182.121", "67": "62.197.45.18", "70": "89.41.181.69"}
```

To avoid disrupting production, other administrative API endpoints were not tested. However, proof-of-concept analysis strongly indicates incorrect authorization implementation, exposing administrative features to any Amnezia Premium user for $21.

Role-based authorization should be implemented to distinguish administrative users from regular users. The administrative API should not be exposed to the Internet, should be hosted on a separate port, and should only be accessible via an internal IP after establishing a VPN connection. A full security assessment of the backend infrastructure and services is strongly recommended.

Given the severity of this issue, all logs should be collected, and a forensic investigation should determine if exploitation has occurred. Findings should guide further actions to assess the impact and potential infrastructure compromise.

AVP-03-006 WP1: Arbitrary RCE via OpenVPN Config Import *(**Critical**)*

Retest Notes: Resolved[7] by AmneziaVPN and confirmed by 7ASecurity.

The configuration import functionality of the AmneziaVPN client is vulnerable to arbitrary code execution due to insufficient validation of dangerous OpenVPN directives. The *checkForMaliciousStrings()* function fails to block configurations containing script-triggering tags, allowing attackers to execute arbitrary commands with root privileges.

Two core issues exist:

1. *dangerousTagsMaxCount = 3* permits configurations with up to two dangerous tags (e.g., *up*, *tls-verify*) without warnings, allowing malicious tags to bypass detection.
2. The *\w+-\w+|\w+* regex fails to capture multi-hyphenated tags (e.g., *route-pre-down*, *auth-user-pass-verify*), enabling attackers to evade detection using tags with multiple hyphens.

The following AmneziaVPN configuration contains a route-up directive executing */usr/bin/touch /tmp/7a.poc*. Upon VPN connection, this command runs with root privileges, creating */tmp/7a.poc*. This demonstrates arbitrary command execution on the client system through a malicious configuration import.

PoC Malicious Configuration:
https://7as.es/AmneziaVPN_uEuw9k6N/malicious_vpn_config.txt

Result:

```
$ ls -la /tmp/7a.poc
-rw-r--r-- 1 root root 0 velj  1 01:32 /tmp/7a.poc
```

The root cause for this issue can be observed in the following code path:

Affected File:
https://github.com/amnezia-vpn/amnezia-client/[…]/ui/controllers/importController.cpp

Affected Code:

```
bool ImportController::extractConfigFromData(QString data)
{
    [...]
    m_configType = checkConfigFormat(config);
    if (m_configType == ConfigTypes::Invalid) {
        data.replace("vpn://", "");
```

[7] https://github.com/amnezia-vpn/amnezia-client/pull/1571

```
        QByteArray ba = QByteArray::fromBase64(data.toUtf8(),
QByteArray::Base64UrlEncoding | QByteArray::OmitTrailingEquals);
        QByteArray ba_uncompressed = qUncompress(ba);
        if (!ba_uncompressed.isEmpty()) {
            ba = ba_uncompressed;
        }

        config = ba;
        m_configType = checkConfigFormat(config);
    }

    switch (m_configType) {
        case ConfigTypes::OpenVpn: {
        m_config = extractOpenVpnConfig(config);
        if (!m_config.empty()) {
            checkForMaliciousStrings(m_config);
            return true;
        }
        return false;
    }
    [...]
}

void ImportController::checkForMaliciousStrings(const QJsonObject &serverConfig)
{
    [...]
    QString protocolConfig =
containerConfig[ProtocolProps::protoToString(Proto::OpenVpn)].toObject()[config_key::la
st_config].toString();
    QString protocolConfigJson =
QJsonDocument::fromJson(protocolConfig.toUtf8()).object()[config_key::config].toString(
);

    const QRegularExpression regExp { "(\\w+-\\w+|\\w+)" };
    const size_t dangerousTagsMaxCount = 3;

    QStringList dangerousTags {
        "up", "tls-verify", "ipchange", "client-connect", "route-up", "route-pre-down",
"client-disconnect", "down", "learn-address", "auth-user-pass-verify"
    };

    QStringList maliciousStrings;
    QStringList lines = protocolConfigJson.replace("\r", "").split("\n");
    for (const QString &l : lines) {
        QRegularExpressionMatch match = regExp.match(l);
        if (dangerousTags.contains(match.captured(0))) {
            maliciousStrings << l;
        }
    }

    if (maliciousStrings.size() >= dangerousTagsMaxCount) {
```

```
        m_maliciousWarningText = tr("In the imported configuration, potentially
dangerous lines were found:");
        for (const auto &string : maliciousStrings) {
              m_maliciousWarningText.push_back(QString("<br><i>%1</i>").arg(string));
        }
    }
[...]
```

To address this issue, the *checkForMaliciousStrings()* function must reduce *dangerousTagsMaxCount* to *1*, flagging any non-whitelisted dangerous tag to ensure a single malicious directive triggers a warning. The *regExp* must be replaced with logic that splits configuration lines by whitespace and validates the first token against the dangerous tags list, preventing evasion via multi-hyphenated or malformed tags.

Trusted directives like *up* and *down* with predefined scripts should be explicitly whitelisted, while all others must be blocked. Input validation must be hardened to strip or sanitize inline comments and unexpected characters that could bypass detection. For additional guidance, please see the *OWASP Input Validation Cheat Sheet*[8].

[8] https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

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-03-001 WP1: Lack of Perfect Forward Security *(Medium)*

Retest Notes: Resolved[9] by AmneziaVPN and confirmed by 7ASecurity.

Perfect Forward Security (PFS)[10] ensures that session keys in VPN connections remain secure even if the private key of the server is compromised. Without PFS, an attacker with access to the private key can decrypt previously captured VPN traffic, compromising data confidentiality. During the code audit, instances were identified where PFS was explicitly disabled in the implementation of AmneziaVPN. This was confirmed as follows:

Example 1: Windows Client C++ Implementation

Affected File:
https://github.com/amnezia-vpn/amnezia-client/[...]/ikev2_vpn_protocol_windows.cpp

Affected Code:

```
QString arguments = QString("-command \"Set-VpnConnectionIPsecConfiguration\" "
                            "-ConnectionName '%1' "
                            "-AuthenticationTransformConstants GCMAES128 "
                            "-CipherTransformConstants GCMAES128 "
                            "-EncryptionMethod AES256 "
                            "-IntegrityCheckMethod SHA256 "
                            "-PfsGroup None "
                            "-DHGroup Group14 "
                            "-PassThru -Force\"")
                    .arg(tunnelName());
```

Example 2: Server Configuration Scripts

Affected File:
https://github.com/amnezia-vpn/amnezia-client/[...]/ipsec/configure_container.sh

[9] https://github.com/amnezia-vpn/amnezia-client/pull/1382
[10] https://en.wikipedia.org/wiki/Forward_secrecy

Affected Code:

```
cat > /etc/ipsec.conf <<EOF
[...]
conn shared
  left=%defaultroute
  leftid=$SERVER_IP_ADDRESS
  right=%any
  encapsulation=yes
  authby=secret
  pfs=no

[...]

cat > /etc/ipsec.d/ikev2.conf <<EOF
conn ikev2-cp
  left=%defaultroute
  leftcert=$SERVER_IP_ADDRESS
  leftid=$SERVER_IP_ADDRESS
  leftsendcert=always
  leftsubnet=0.0.0.0/0
  leftrsasigkey=%cert
  right=%any
  rightid=%fromcert
  rightaddresspool=192.168.43.10-192.168.43.250
  rightca=%same
  rightrsasigkey=%cert
  narrowing=yes
  dpddelay=30
  dpdtimeout=120
  dpdaction=clear
  auto=add
  ikev2=insist
  rekey=no
  pfs=no
```

It is advised to enable PFS in all configurations to protect the confidentiality of past VPN sessions, even if the private key of the server is compromised. In the C++ implementation, update the PowerShell command by replacing *-PfsGroup None* with a secure PFS group, such as *PFS2048*[11], to enable PFS for client-side VPN configurations.

On the server side, update *ipsec.conf* and *ikev2.conf* by setting *pfs=yes* instead of *pfs=no*. Additionally, specify secure *Diffie-Hellman* groups, such as *modp2048*, under the *ike* and *phase2alg* parameters. These changes will align the IPsec server configuration with modern security best practices, significantly enhancing VPN cryptographic strength.

[11] https://docs.azure.cn/en-us/vpn-gateway/vpn-gateway-ipsecikepolicy-rm-powershell

AVP-03-002 WP1: Multiple Vulnerabilities in Go Versions Used *(Low)*

Retest Notes: Resolved[12][13][14][15] by AmneziaVPN and confirmed by 7ASecurity.

Outdated or vulnerable Go versions[16] are used in several components of the AmneziaVPN software, introducing potential security risks. Although these vulnerabilities are limited to the application runtime and do not directly impact host systems, updating Go versions is strongly recommended to mitigate these risks.

Affected File	Go Version	Vulnerabilities
amnezia-client/.github/workflows/deploy.yml	1.22.1	9[17]
amneziawg-go/Dockerfile	1.20	11[18] / 98[19]
amneziawg-windows/build.cmd	1.20.8	18[20]
amneziawg-android/tunnel/tools/libwg-go/Makefile	1.22.3	6[21]

All instances of Go used in the project should be updated to the latest stable version[22] to address known vulnerabilities. Hardcoding specific minor versions in files such as Dockerfiles and build scripts should be avoided, as it can result in reliance on outdated versions. Semantic versioning ranges or referencing the latest minor revisions with appropriate tags (e.g., *:latest*) should be used to facilitate updates while ensuring compatibility.

Automated tools such as *Dependabot*[23] or *Trivy*[24] should be integrated into the development workflow to identify outdated or vulnerable components proactively, enhancing security and simplifying dependency management. Although the current vulnerabilities are of low severity due to the isolation of the Go runtime, prompt resolution is essential to minimize the attack surface and ensure robust software

[12] https://github.com/amnezia-vpn/amneziawg-go/commit/71b…52dc27
[13] https://github.com/amnezia-vpn/amnezia-client/pull/1517
[14] https://github.com/amnezia-vpn/amneziawg-windows/pull/13
[15] https://github.com/amnezia-vpn/amneziawg-android/pull/30
[16] https://www.cvedetails.com/version-list/14185/29205/1/Golang-GO.html
[17] https://www.cvedetails.com/vulnerability-list/vendor_id-14185/…/Golang-GO-1.22.1.html
[18] https://www.cvedetails.com/vulnerability-list/vendor_id-14185/…/Golang-GO-1.20.12.html
[19] https://hub.docker.com/layers/library/golang/1.20/images/sha256-c1a…b5
[20] https://www.cvedetails.com/vulnerability-list/…/Golang-GO-1.20.8.html
[21] https://www.cvedetails.com/vulnerability-list/vendor_id-14185/…/Golang-GO-1.22.3.html
[22] https://go.dev/doc/devel/release
[23] https://docs.github.com/en/code-security/getting-started/dependabot-quickstart-guide
[24] https://github.com/aquasecurity/trivy

security.

AVP-03-003 WP1: Traffic Masking Weakness via Insecure PRNG *(Low)*

Retest Notes: Resolved[25][26] by AmneziaVPN and confirmed by 7ASecurity.

An instance of insecure *Pseudo-Random Number Generator* (PRNG)[27] usage was identified, potentially undermining the traffic masking mechanism. The junk packet creation mechanism uses the non-cryptographically secure *math/rand* package, resulting in predictable packet sizes. This predictability could enable an attacker to distinguish junk packets from legitimate traffic through traffic analysis, compromising the *traffic masking* mechanism[28]. This can be confirmed observing the following code path:

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 sizes is insecure, as it generates predictable values exploitable through traffic analysis. While the *randomJunkWithSize* function correctly uses *crypto/rand* for secure packet content generation, the *createJunkPackets* function produces predictable packet sizes, reducing obfuscation effectiveness and weakening resistance to traffic analysis.

The replacement of *math/rand* with *crypto/rand* is recommended to ensure

[25] https://github.com/amnezia-vpn/amneziawg-go/pull/61
[26] https://github.com/amnezia-vpn/amneziawg-go/pull/70
[27] https://en.wikipedia.org/wiki/Pseudorandom_number_generator
[28] https://docs.amnezia.org/documentation/how-amnezia-works/

cryptographically secure random number generation and enhance the unpredictability of junk packet sizes. This change will significantly improve resilience to traffic analysis attacks and strengthen the security of junk packet generation.

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
    [...]
}
```

WP2: AmneziaVPN Supply Chain Implementation

Introduction and General Analysis

The *8th Annual State of the Software Supply Chain Report*, released in October 2022[29], revealed a 742% average yearly increase in software supply chain attacks since 2019. Some notable compromise examples include *Okta*[30], *Github*[31], *Magento*[32], *SolarWinds*[33], and *Codecov*[34], among many others. To mitigate this concerning trend, Google released an End-to-End Framework for *Supply Chain Integrity* in June 2021[35], named *Supply-Chain Levels for Software Artifacts* (*SLSA*)[36].

This section of the report elaborates on the current state of the supply chain integrity implementation of the AmneziaVPN project, as audited against versions 0.1 and 1.0 of the SLSA framework. SLSA assesses the security of software supply chains and aims to provide a consistent way to evaluate the security of software products and their dependencies.

Current SLSA practices of AmneziaVPN

The AmneziaVPN project uses a public GitHub repository[37] for source code management and releases. Artifact distribution is fully automated via GitHub Actions[38] for Linux, Android, and macOS, while Windows releases are handled using a dedicated virtual machine. These practices align with SLSA framework security guidelines. The following sections focus on specific practices and requirements unique to the project:

Source

AmneziaVPN uses Git and GitHub for version control and enforces strict rules to maintain codebase integrity. All contributions are reviewed by trusted developers, ensuring controlled and accountable repository access. Changes to the source code are made transparently, with pull requests reviewed and approved only by trusted developers.

[29] https://www.sonatype.com/press-releases/2022-software-supply-chain-report
[30] https://www.okta.com/blog/2022/03/updated-okta-statement-on-lapsus/
[31] https://github.blog/2022-04-15-security-alert-stolen-oauth-user-tokens/
[32] https://sansec.io/research/rekoobe-fishpig-magento
[33] https://www.techtarget.com/searchsecurity/ehandbook/SolarWinds-supply-chain-attack...
[34] https://blog.gitguardian.com/codecov-supply-chain-breach/
[35] https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html
[36] https://slsa.dev/spec/
[37] https://github.com/amnezia-vpn
[38] https://github.com/amnezia-vpn/amnezia-client/actions

Build

The AmneziaVPN project uses GitHub, a hardened hosted platform, for its scripted build process, defined as code within the repository[39]. Changes to the build script are made through pull requests, reviewed and approved by maintainers, and take effect only after merging. This strict review process enhances build security and reliability. Builds are timestamped and reproducible on GitHub Actions.

Provenance

No evidence of properly formatted provenance compliant with the SLSA Framework[40] was found in the AmneziaVPN repository. This result is expected, as SLSA standard adoption remains an ongoing process in the industry. Tools like *GitHub Artifacts Attestations*[41] are gradually enabling provenance generation, but widespread implementation is still lacking. However, unsigned and unformatted provenance was identified in the GitHub Workflow build configuration file.

SLSA v1.0 Framework Analysis

SLSA v1.0 defines a set of four levels that describe the maturity of the software supply chain security practices implemented by a software project as follows:

- **Build L0: No guarantees** represent the lack of SLSA[42].
- **Build L1: Provenance exists**. The package has provenance showing how it was built. This can be used to prevent mistakes but is trivial to bypass or forge[43].
- **Build L2: Hosted build platform**. Builds run on a hosted platform that generates and signs the provenance[44].
- **Build L3: Hardened builds**. Builds run on a hardened build platform that offers strong tamper protection[45].

Based on the documentation provided by the AmneziaVPN team, 7ASecurity conducted a SLSA v1.0 analysis, with the following results.

[39] https://github.com/amnezia-vpn/amnezia-client/tree/dev/deploy
[40] https://slsa.dev/spec/v1.0/provenance
[41] https://github.blog/changelog/2024-06-25-artifact-attestations-is-generally-available/
[42] https://slsa.dev/spec/v1.0/levels#build-l0
[43] https://slsa.dev/spec/v1.0/levels#build-l1
[44] https://slsa.dev/spec/v1.0/levels#build-l2
[45] https://slsa.dev/spec/v1.0/levels#build-l3

SLSA v1.0 Assessment Results

The table below presents the results of AmneziaVPN according to the Producer and Build platform requirements in the SLSA v1.0 Framework. The categories (source, build, provenance, and contents of provenance) are logically separated. Each row shows the SLSA level for each control, with green check marks indicating compliance and red boxes indicating the lack of evidence for compliance.

Implementer	Requirement		L1	L2	L3
Producer	Choose an appropriate build platform		✅	✅	✅
	Follow a consistent build process		✅	⛔	⛔
	Distribute provenance		✅	⛔	⛔
Build platform	Provenance generation	Exists	✅	⛔	⛔
		Authentic		⛔	⛔
		Unforgeable			⛔
	Isolation strength	Hosted		✅	✅
		Isolated			✅

SLSA v1.0 Assessment Justification

Producer requirements

Choose an Appropriate Build Platform

AmneziaVPN artifacts are built on GitHub, a hosted build platform. GitHub Actions offer a robust environment for automating builds and support SLSA-compliant provenance generation using tools like *GitHub Artifact Attestation*[46], enabling cryptographic verification of artifact origin and integrity. Sandboxed build environments provided by GitHub reduce interference risks. With proper configuration, GitHub Actions can support SLSA Level 3 compliance, allowing AmneziaVPN to target SLSA L3 using platform-provisioned features.

[46] https://github.blog/news-insights/product-news/introducing-artifact-attestations-now-in-public-beta/

Follow a Consistent Build Process

This requirement mandates producers to generate artifacts through a consistent build process, enabling consumers to set clear expectations[47]. AmneziaVPN artifacts are created using defined build scripts[48] with step-by-step instructions, triggered by GitHub Actions[49] configured through YAML files.

Build requirements

Distribute provenance

AmneziaVPN distributes configuration files in the source code via GitHub. According to the SLSA Framework, the configuration build file is classified as unsigned, unformatted provenance.

Provenance Exists

AmneziaVPN does not use GitHub tools like *GitHub Artifact Attestations*, which generate formatted provenance and leverage *Sigstore*[50] for signing and verifying artifacts, essential for meeting SLSA L2 and L3 requirements. However, AmneziaVPN uses a build configuration file classified as unsigned, unformatted provenance under the SLSA Framework, meeting SLSA L1 requirements.

Provenance is Authentic

Provenance must be signed with a private key accessible only to the hosted build platform, ensuring trustworthiness and preventing tampering. This requirement can be met by enabling tools such as *GitHub Artifact Attestation* or generating verifiable artifact attestations.

Provenance is Unforgeable

The hosting platform must generate Provenance L3 to ensure resistance to tenant forgery. This requirement can be met by enabling tools such as *GitHub Artifact Attestation*.

[47] https://slsa.dev/spec/v1.0/requirements#follow-a-consistent-build-process
[48] https://github.com/amnezia-vpn/amnezia-client/tree/dev/deploy
[49] https://github.com/amnezia-vpn/amnezia-client/actions
[50] https://www.sigstore.dev/

Hosted

The requirement mandates that all build steps be executed on a hosted build platform, either on shared or dedicated infrastructure, not on individual workstations. AmneziaVPN meets this requirement using GitHub.

Isolated

The requirement mandates execution of build steps in an isolated environment, with external influence only initiated by the build process. AmneziaVPN meets this requirement by using GitHub Actions agents provisioned for each build.

SLSA v0.1 Results

The following sections summarize the results of the software supply chain security implementation audit based on the SLSA v0.1 framework. Green check marks indicate that evidence of the noted requirement was found.

Requirement	L1	L2	L3	L4
Source - Version controlled		✅	✅	✅
Source - Verified history			✅	✅
Source - Retained indefinitely			✅18mo.	✅
Source - Two-person reviewed				✅
Build - Scripted build	✅	✅	✅	✅
Build - Build service		✅	✅	✅
Build - Build as code			✅	✅
Build - Ephemeral environment			✅	✅
Build - Isolated			✅	✅
Build - Parameterless				✅
Build - Hermetic				✅

Build - Reproducible				✅
Provenance - Available	✅	⛔	⛔	⛔
Provenance - Authenticated		⛔	⛔	⛔
Provenance - Service generated		⛔	⛔	⛔
Provenance - Non-falsifiable			⛔	⛔
Provenance - Dependencies complete				⛔
Common - Security				⛔
Common - Access				⛔
Common - Superusers				⛔

SLSA v0.1 & v1.0 Conclusion

The evaluation of AmneziaVPN software supply chain security practices determined that the project achieves SLSA Level 1, reflecting basic measures such as source code version control and automated builds via GitHub Actions. Gaps remain in meeting higher SLSA levels, primarily due to the absence of formatted and signed provenance data.

It is recommended to deploy Signed Provenance Generation to reach SLSA Level 3. Specifically, *GitHub Actions* can use *GitHub Artifact Attestations* to establish an unforgeable link between artifacts and the build process, ensuring software supply chain integrity by creating and verifying signed provenance.

Although AmneziaVPN meets SLSA Level 1, advancing to higher levels requires a defined strategy for generating signed, formatted provenance. These measures will enhance resilience against supply chain threats, ensuring artifact integrity, authenticity, and traceability.

WP3: AmneziaVPN Lightweight Threat Model

Introduction

AmneziaVPN is an all-in-one solution enabling users to connect to a managed VPN service or configure a self-hosted VPN. It is designed to provide censorship-resistant access in countries with extreme internet restrictions. To enhance anonymity and bypass government censorship, it includes AmneziaWG, a custom-modified WireGuard version with obfuscation techniques against Deep Packet Inspection (DPI) and surveillance. The software supports multiple protocols for adversarial environments, including WireGuard, OpenVPN with Cloak, Shadowsocks, and IKEv2/IPsec[51].

Threat model analysis identifies security threats and vulnerabilities, enabling effective mitigation before exploitation. Lightweight threat modeling follows a simplified *STRIDE*[52] approach, focusing on system analysis through documentation, specifications, source code, and existing threat models, with client representative input.

This section categorizes potential attack scenarios, identifies vulnerabilities, and suggests mitigations. The analysis covers client applications, infrastructure, design, and processes based on all available resources during the engagement.

Relevant assets and threat actors

The following key assets were identified as significant from a security perspective:

- Source code repository
- Build artifacts
- Build artifacts signing keys
- Accounts in Google Play Store/Apple Store authorized to release the software
- GitHub accounts of the team
- Domains used to publish documentation and release artifacts
- Infrastructure hosting publicly available services (AWS/CloudFlare/GCP)
- Private key used by servers to encrypt data provided to clients on managed AmneziaVPN servers, eliminating reliance on HTTPS certificate infrastructure.
- Symmetric session encryption key to encrypt parameters exchanged between the AmneziaVPN client and backend services.
- QR codes and *textkey* used to share access to VPN instances.

The following threat actors are considered relevant for the analysis:

- Advanced Persistent Attacker (Nation State Attacker)
- External Attacker

[51] https://docs.amnezia.org/documentation/protocols-info/
[52] https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats#stride-model

- LAN Attacker
- Compromised Developer

Attack surface

In threat modeling, the attack surface includes all potential entry points an attacker could exploit to compromise a system, access or manipulate sensitive data, or disrupt application availability. Identifying the attack surface helps pinpoint vulnerabilities and implement defenses to mitigate risks.

By analyzing threats and attack scenarios, organizations gain insight into techniques that could compromise system security.

The following data flow diagram outlines the system, highlighting key countermeasures and components handling various assets as envisioned by 7ASecurity:

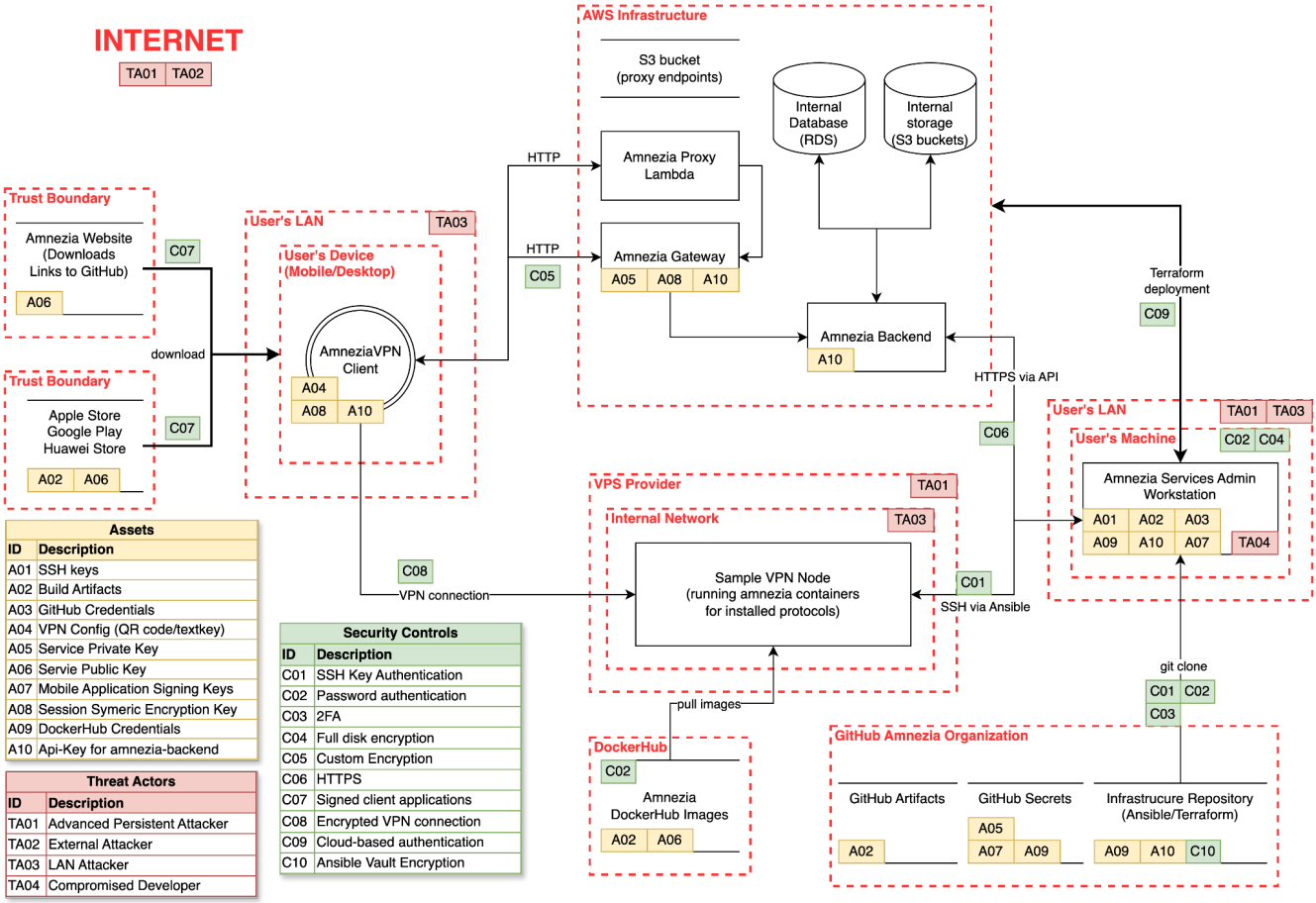

Fig.: Data flow diagram of a simplified Amnezia Free involving an end-user, infrastructure engineer (admin) and backend components hosted in AWS

Threat 01: Attacks Against Custom Cryptography Implementation

Overview

The risk of HTTPS certificate infrastructure compromise, particularly in adversary states, is considered. A custom encryption mechanism using public-key cryptography and a session key for symmetric encryption is implemented.

Cryptographic flaws are difficult to detect due to their complexity. If encryption is compromised, the system and users may face threats to privacy, anonymity, or more advanced attacks targeting the infrastructure.

Countermeasures

The Amnezia Free gateway component uses a custom cryptography protocol with public-key cryptography and AES-based symmetric encryption using a session key. Amnezia Premium does not implement this scheme, but implementation is planned.

Attack Scenarios

The following attack scenarios should be considered in security assessments and cryptanalysis:

- Ciphertext attacks due to insufficient randomness in algorithm parameters, enabling decryption.
- Known or chosen plaintext attacks to break the encryption key.
- Padding oracle attacks using chosen ciphertext.
- Private key disclosure, allowing decryption and tampering of user-backend communication.
- Attacks on cryptographic libraries (e.g., OpenSSL), insecure wrappers, or insecure defaults lacking mitigations for known vulnerabilities.

Recommendation

To mitigate risks from custom cryptography and strengthen defenses against the listed attacks, the following measures may be considered:

- Conduct a cryptanalysis alongside a security assessment to ensure compliance with security best practices.
- Review algorithms and modes to identify potential attack techniques.
- Use reputable libraries with securely configured parameters to mitigate known attacks.
- Ensure secure parameter generation using a reliable randomness source.
- Review private key handling and potential backend key material attacks.

- Implement strong encryption for key management (e.g., AWS CMK KMS or HSM solutions).

Threat 02: Network-based Amnezia Backend Service Impersonation

Overview

The VPN solution, designed for highly adversarial areas, faces threats from nation-state actors using advanced techniques such as issuing legitimate certificates, large-scale Man-in-the-Middle (MiTM) attacks, and DNS spoofing. Countermeasures must prevent and detect these attacks.

Countermeasures

The application considers common certificate infrastructure vulnerable to Man-in-the-Middle (MiTM) attacks in adversarial environments. Amnezia Free does not rely on issued certificates but uses custom cryptography with public-key and symmetric encryption, as detailed separately.

Attack Scenarios

The following network-level attack scenarios, previously used by sophisticated attackers, should be included in risk assessments:

- DNS spoofing to redirect AmneziaVPN users to an attacker-controlled server.
- Large-scale Man-in-the-Middle (MiTM) attacks via BGP hijacking or ISP-level traffic manipulation.
- Certificate issuance by nation-state attackers using state-controlled root CAs installed on devices.
- Coercion of a trusted CA to issue a malicious but valid certificate for a targeted domain.
- Private key extraction from misconfigured cloud infrastructure, servers, or VPS providers, compromising data integrity in Amnezia backend services.

Recommendation

The common certificate infrastructure was correctly identified as vulnerable in censored countries, leading to a custom encryption protocol implementation. To further protect against sophisticated attackers, the following measures should be considered:

- Security review of backend services, focusing on sensitive parameter storage and secrets handling.
- Monitoring of DNS and BGP[53] routes to detect protocol-based attacks.

[53] https://radar.cloudflare.com/routing

- Secure key storage using Hardware Security Modules (HSM) for cryptographic operations.
- Key revocation and rotation processes for compromised key material.
- *DNS over HTTPS (DoH)*[54] in all client applications (i.e. mobile, desktop) to prevent tampering, regardless of OS or network configuration.

Threat 03: Release Binary Tampering

Overview

The application includes mobile and desktop versions for multiple platforms. Server-side components are deployed in Amnezia-managed infrastructure, with key parts also deployable for self-hosted VPN servers. Ensuring artifact integrity throughout development, release, and installation is critical to protecting end-user communication.

Countermeasures

Artifacts are built using GitHub Actions. Mobile binaries are signed before release to Google Play and Apple Store. The Windows binary is signed for verification during installation. Docker containers and Linux binaries lack integrity checks.

Attack Scenarios

The following supply chain attack scenarios could lead to severe compromise by injecting malicious code into released binaries:

- GitHub pipeline compromise or external action manipulation to inject malicious code without modifying source code.
- Domain hijacking to tamper with hosted data and serve malicious binaries, especially for unsigned artifacts.
- Workstation attacks on infrastructure engineers to backdoor deployed services.
- Docker Hub account takeover to release malicious images under the company name.
- Library supply chain attacks to introduce malicious code, similar to the 2024 *XZ Utils* case, where a threat actor contributed for two years before introducing vulnerable code[5556].
- Signing key compromise to release modified binaries in mobile stores or coercion by a nation-state to distribute malicious versions.

[54] https://en.wikipedia.org/wiki/DNS_over_HTTPS
[55] https://tukaani.org/xz-backdoor/
[56] https://www.akamai.com/blog/security-research/critical-linux-backdoor-xz-utils-discovered-what-to-know

Recommendation

To enhance defenses against the identified scenarios, consider the following measures:

- Achieve the highest possible SLSA compliance to mitigate supply chain attacks.
- Enforce strict control over signing keys and accounts used for artifact releases.
- Require MFA for critical systems, including source code, build pipelines, and releases, preferably using physical keys.
- Use isolated, monitored workstations for signing, releasing artifacts, and infrastructure deployment (e.g., Ansible Tower).
- Pin dependencies in the application and Dockerfiles to prevent fetching malicious images.
- Apply strong security settings to company-owned domains to prevent hijacking.
- Monitor published binaries across all locations to detect tampering by nation-state actors.

Threat 04: Connecting to a Malicious VPN Server

Overview

AmneziaVPN client supports multiple VPN protocols[57], including AmneziaWG, OpenVPN, and Shadowsocks. Configurations can be imported via file (e.g., .ovpn), QR code, or text key. The latter methods allow direct imports or loading an *api_key* from a custom JSON format to query backend services for VPN configurations.

Users should be aware of threats when importing configurations from untrusted sources. If connected to a rogue VPN server, no application-level protections prevent attacker interference.

Countermeasures

AmneziaVPN Free uses public key cryptography and symmetric session key encryption to exchange parameters between the application and backend, ensuring confidentiality and authentication without relying on HTTPS certificates.

AmneziaVPN Premium delivers configurations, including *api-key* and backend IP, via email or Telegram in *vpn://<textkey>* format. The *textkey* is a compressed, base64-encoded JSON relying on the security of delivery channels. After importing the configuration, the client connects to the *api_endpoint* over HTTPS to retrieve the VPN configuration. Public key cryptography is planned for the premium service but has not yet been implemented.

[57] https://docs.amnezia.org/documentation/protocols-info/

Attack Scenarios

The following attack scenarios demonstrate how configuration import flexibility may be exploited against Amnezia users:

- Spear phishing using similar domains[58] to deliver forged *vpn://<textkey>* links, tricking victims into connecting to attacker-controlled VPN servers by claiming improved Amnezia Premium nodes.
- Compromise of communication channels used to deliver *vpn://<textkey>* links, such as Telegram, email, or payment websites, to modify configurations.
- Public campaigns distributing QR codes leading to attacker-controlled servers, easily imported into AmneziaVPN clients.
- Automatic attacks against users connected to malicious VPN servers due to insufficient client-side firewall rules, enabling persistence.

Recommendation

Enhancements to improve security and complement existing mechanisms include:

- Integrity protection for *vpn://<textkey>* and QR codes to ensure configurations are legitimate and generated by the Amnezia backend.
- Clear warnings in the application for untrusted VPN configurations, allowing users to distinguish between Amnezia-generated and third-party configurations. A self-hosted server can share a public key for verification.
- Default deny-all firewall rules to prevent exposure after connecting to rogue VPN servers. Users should configure this firewall before connecting if no ports need to be exposed.
- Notifications if multiple devices connect to the same VPN endpoint using the same *api-key*, detecting potential key leakage.
- Passphrase protection when sharing QR codes or *textkey* strings for VPN access.

Threat 05: Disrupted Continuity of the Service (Denial of Service)

Overview

Denial of service occurs when the system fails to function as intended, temporarily or permanently. Ensuring availability is critical for VPN providers, especially against nation-state attacks. Additional attack scenarios and countermeasures must be considered to maintain service continuity.

[58] https://www.cisa.gov/resources-tools/services/domain-doppelganger

Countermeasures

Publishing limitations in mobile stores and domain blocks in some countries have been encountered. As a countermeasure, a static AWS S3 file with a custom proxy list, implemented via AWS Lambda functions, is used when *http://gw.amnezia.org* is blocked. However, this solution is suboptimal, as the S3 resource is static, and the Lambda function can be easily blocked at the DNS level in adversary-controlled networks.

Attack Scenarios

Reviewing and testing the following attack scenarios will improve detection and mitigation of threats targeting Amnezia users:

- DNS tampering to block Amnezia domains and IPs, disrupting Free and Premium services.
- Automated bots fetching proxy lists to add IPs to country-wide blocklists.
- Mass scanning of Amnezia infrastructure to fingerprint services via HTTP headers or error messages, enabling dynamic blocking or large-scale (D)DoS attacks.
- Nation-state actors purchasing VPN access to query and block VPN servers or identify connected users.
- Network filtering exploiting weaknesses in the anonymization protocol, using deep packet inspection, BGP hijacking[59], or IP filtering to block or capture AmneziaWG traffic.
- Bot farms mass-reporting the application in mobile stores, hindering installation, forcing users to enable untrusted sources, and preventing Amnezia client users from applying security patches.

Recommendation

To counter these threats, the following measures should be considered:

- Security assessments to prevent easy fingerprinting of Amnezia servers and backend components.
- Frequent rotation of IP addresses and DNS names to evade censorship.
- Utilization of unblockable services, such as CDNs or Tor Hidden Services.
- Short-lived, dynamically rotated proxy addresses to maintain access when backend components are unavailable.
- Alternative communication channels, such as Telegram, Signal, or approved messaging apps, to distribute backend service IPs in high-censorship areas.
- Extensive monitoring across multiple networks to detect blocked IPs and domains, enabling active resource rotation and configuration updates.

[59] https://www.cloudflare.com/en-gb/learning/security/glossary/bgp-hijacking/

Threat 06: Increased Local Attack Surface

Overview

VPN applications are a high-value target for attackers seeking initial compromise or privilege escalation on end-user devices. Support for multiple protocols, configuration parsers, and automated VPN setup via QR codes or JSON payloads increases the attack surface. Vulnerabilities in these components, when combined with other attacks, may be exploited by both regular and nation-state attackers. Comprehensive security testing is essential to mitigate risks and address the expanded attack surface on local devices.

Countermeasures

AmneziaVPN uses keyword-based filters to detect potentially malicious configuration options, such as in OpenVPN configurations. AmneziaVPN binaries have correct binary flags set to limit memory corruption issues, but integrated components for third-party protocols require better protection.

Due to the variety of configuration formats and loading methods, including custom Backup and Amnezia JSON-based types, current protections should undergo extensive testing, covering at least the listed scenarios.

Attack Scenarios

The following attacks should be considered during the design, development, and analysis of client binaries:

- Insecure local binaries, including third-party components, allowing library hijacking.
- Insecure configuration parsers, enabling deserialization, command injection, or memory corruption, especially in C/C++ parsers.
- Social engineering attacks using crafted malicious configurations to bypass regex-based filtering via QR codes or encoded text keys.
- Memory corruption attacks on QT components or integrated libraries due to insufficient protection or missing binary flags.
- Configuration type confusion, merging properties from multiple formats using incorrect logic.
- Attacks on templating engines, processing JSON-based data, leading to potential command injection.

Recommendation

To mitigate these threats, the following measures should be considered:

- Enforce security flags across all platforms using tools like *checksec*[60] and *PESecurity*[61], applying these settings to all integrated components.
- Conduct extensive fuzz testing on configuration file loading, focusing on custom parsing logic.
- Perform client-side API fuzzing to identify memory corruption or deserialization bugs in AmneziaVPN logic when fetching backend data, preventing local exploitation, especially with new features.

Threat 07: Attacks Against the Backend Infrastructure

Overview

Amnezia Free and Premium services use AWS and multiple VPS providers. Each Internet-exposed infrastructure is a potential target. API exploitation could grant attackers access to cloud infrastructure. Proper configuration and compromise detection are crucial to prevent breaches.

Countermeasures

The VPS infrastructure is accessible from infrastructure engineer workstations with access to Ansible playbooks (private GitHub repository), SSH keys, and Ansible Vault passwords used for encrypting secrets. Employees reported that self-managed laptops follow high security standards, including full disk encryption, multi-factor authentication, and up-to-date operating systems.

Cloud infrastructure was outside the assessment scope, and no assumptions about its security controls can be made.

Attack Scenarios

Future security assessments should consider the following attack scenarios due to the lack of backend cloud analysis:

- AWS account compromise via leaked access keys, weak credentials, or misconfigurations.
- Employee workstation compromise to backdoor VPN node servers via Ansible playbooks.

[60] https://github.com/slimm609/checksec
[61] https://github.com/NetSPI/PESecurity

- Backend vulnerabilities (e.g., SSRF, deserialization) targeting cloud resources to extract IAM tokens.
- Unauthorized S3 bucket access to redirect users to attacker-controlled servers.
- Backend database compromise due to misconfigured access controls, public exposure, or unauthorized access.
- Exploitation of administrative API features to manipulate licensing, backend applications, or VPN configurations.
- Cloud privilege escalation via compromised service instances (e.g., gateway, Lambda, backend) leveraging privileged IAM roles or access to resources.

Recommendation

To mitigate and prepare for the aforementioned attacks, it is recommended to:

- Conduct a full security review of the cloud infrastructure.
- Isolate free and premium backend services and separate test environments to limit compromise impact.
- Perform a security review of the *amnezia-backend* component.
- Implement robust logging and monitoring across cloud and VPS-hosted servers, focusing on anomaly detection.
- Secure key material handling to prevent extraction (e.g., using HSM modules).
- Establish key revocation and rotation procedures in case of key disclosure.

Threat 08: Bug Omission Risk from Missing Automated Security in CI/CD

Overview

No software is bug-free, and thorough code reviews can still miss security vulnerabilities. All components, including source code, Terraform, Docker images, and CI/CD configurations (e.g., GitHub Actions), should be continuously scanned using SAST tools in CI/CD pipelines. This enhances software resilience by complementing internal development practices.

Countermeasures

The team reported minimal use of security tools for static code, image, or build configuration scanning. Codacy, configured with GitHub Actions, was mentioned but not found in the analyzed repositories. Despite the lack of automated scanning, overall code quality is decent.

Attack Scenarios

The absence of security tooling in pipelines introduces the following risks:

- Accidental commits of sensitive data (e.g., private keys, unencrypted credentials), especially in private repositories.
- Insecure Docker configurations.
- Introduction of outdated libraries or detectable bugs by new developers bypassing code review.

Recommendation

To enhance component security, the following should be integrated into CI/CD pipelines:

- Dependency scanning to detect outdated or vulnerable dependencies.
- Periodic secret scanning in repositories using tools like *TruffleHog*[62] or *GitLeaks*[63].
- Commit scanning with hooks to prevent secret exposure.
- Docker and infrastructure code scanning using tools like *Checkov*[64] .
- Source code scanning with SAST tools (e.g., *CodeQL*[65], *semgrep*[66]) for all languages, including *Go*, *Python*, and *C++*.

Threat 09: Public Relation Failures and Loss of Public Trust

Overview

Software that bypasses restrictions or censorship, preventing government tracking and monitoring, can be used for both positive and negative purposes. Such tools are often criticized as enablers of illegal activities, sometimes justifiably. Organizations in this sector must prioritize transparent communication to ensure project success amid potential attacks and accusations. Poorly handled security incidents or responses may lead to the downfall of the company.

Attack Scenarios

The following potential scenarios may cause PR damage and should be considered:

- Misconfigured integration enabling mass exploitation of self-hosted AmneziaVPN servers.
- Insecure defaults compromising anonymity or confidentiality for self-hosted or managed users.
- Cryptographic attacks on the custom encryption protocol replacing HTTPS certificate infrastructure.

[62] https://github.com/trufflesecurity/trufflehog
[63] https://github.com/gitleaks/gitleaks
[64] https://github.com/bridgecrewio/checkov
[65] https://github.com/github/codeql-action
[66] https://github.com/semgrep/semgrep

- Undetected data breaches exposed by the media, eroding customer trust.
- Association with illegal activities similar to accusations against Tor or cryptocurrencies.
- Lack of responsible disclosure and security bounty policies, and poor handling of bug reports, damaging reputation.

Recommendation

The following measures should be implemented to limit PR damage from listed or similar cases:

- Monitor all supported protocols and ensure VPN server configurations follow best security practices (e.g., hardened OpenVPN setups).
- Conduct periodic penetration tests to identify misconfigured services or exposed ports early.
- Regularly review VPS provider accounts and apply the most secure settings to prevent administrative account takeovers.
- Test incident response processes, including PR communication, to ensure effective handling.
- Develop strategies to manage media accusations and incidents involving illegal activities.
- Establish a clear responsible disclosure policy, using formats like *SECURITY.md*[67] or *security.txt*[68].

[67] https://docs.github.com/en/code-security/getting-started/adding-a-security-policy-to-your-repository
[68] https://securitytxt.org/

Conclusion

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

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

- AmneziaVPN demonstrates a well-structured codebase, effective vulnerability management, and prior experience with penetration testing and code reviews.
- The Android components effectively mitigate common vulnerabilities, reflecting a strong security posture.
- Well-documented and structured source code facilitates efficient review and assessment.
- Alignment with SLSA Level 1 indicates the adoption of fundamental supply chain security practices, establishing a foundation for higher compliance levels.
- No obfuscation weaknesses were identified in the static and dynamic review, demonstrating strong privacy and security practices.
- Despite the absence of SAST tools, code quality is solid. The focus on supporting multiple protocols increases the potential attack surface, particularly with QR code and encoded string-based configuration imports, which could be leveraged for phishing attacks.
- Automation frameworks like Terraform and Ansible streamline development and deployment. However, Ansible playbook access lacks auditability and is executed from a local infrastructure engineer machine, which is suboptimal.

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

- **Administrative API Security:** Administrative APIs should be secured to prevent unauthorized VPN configuration tampering. IP-whitelisting, Access through VPN, role-based access controls, strict authentication, and input validation should be enforced to mitigate this and similar issues (AVP-03-005).
- **Configuration Import Hardening:** The VPN configuration import process should be secured against arbitrary remote code execution. Strict input validation, sanitization, and execution restrictions should be implemented to prevent exploitation through malicious configuration files (AVP-03-006).
- **Secure Gateway Communication:** HTTPS enforcement should be applied for gateway communication, error handling in proxy failback logic should be improved, and resilient proxy distribution methods should be implemented to mitigate DNS-based blocking attacks (AVP-03-004).

- **Supply Chain Security:** Signed provenance should be implemented to achieve higher SLSA Framework levels, and artifact signatures should be applied universally, including Linux binaries and Docker containers, which remain unsigned (WP2).
- **Static Application Security Testing (SAST):** A comprehensive SAST toolchain should be implemented for automated code scanning for vulnerabilities, this will complement security audits and facilitate the identification of potential weaknesses.
- **Security Model Gaps:** Adequate infrastructure mechanisms ought to be in place to ensure appropriate logging, monitoring, and anomaly detection. Without detection mechanisms, node compromise would remain unnoticed.
- **Infrastructure Access Risks:** VPS node access via SSH and Ansible playbooks should be secured by enforcing strong authentication, restricting SSH access, implementing credential rotation, and monitoring for unauthorized access to prevent attackers from exploiting leaked credentials.
- **Cryptographic Security:** The custom cryptography implementation, based on public-key and symmetric AES encryption, should undergo a dedicated cryptanalysis. The AES wrapper should be reviewed for vulnerabilities, particularly against ciphertext attacks. Similarly, secure *Pseudorandom-Number-Generators (PRNGs)* ought to be employed for best security (AVP-03-003).
- **Software Patching:** The AmneziaVPN solution should implement appropriate software patching procedures which regularly apply security patches in a timely manner (AVP-03-002). 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*[69] or *Renovate Bot*[70].

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

[69] https://snyk.io/
[70] https://github.com/renovatebot/renovate

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.

It is advised to consider the following areas in future engagements:

- AWS Cloud Security Review: A comprehensive assessment of the AWS cloud infrastructure should be performed to ensure security best practices are applied, as it serves as the primary backend environment.
- Host Hardening and Ansible Configuration Review: A review of host hardening guidelines enforced via Ansible should be conducted. The assessment should include CIS benchmarks as a baseline, along with additional security checks for deployed services such as Docker.
- Fuzz Testing: Automated fuzzing should be incorporated to identify vulnerabilities in input processing, configuration handling, and exposed interfaces, improving overall security resilience.
- Ongoing Source Code Reviews: Continuous code reviews should be conducted, as vulnerabilities often reside in small details. Some of the issues in this report were identified through a focused review of a single file, highlighting the need for regular, in-depth assessments.

7ASecurity would like to take this opportunity to sincerely thank 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.