Abstract—Corporate Virtual Private Networks (VPNs) enable users to work from home or while traveling. At the same time, VPNs are tied to a company’s network infrastructure, forcing users to install proprietary clients for network compatibility reasons. VPN clients run with high privileges to encrypt and reroute network traffic. Thus, bugs in VPN clients pose a substantial risk to their users and in turn the corporate network. Cisco, the dominating vendor of enterprise network hardware, offers VPN connectivity with their AnyConnect client for desktop and mobile devices. While past security research primarily focused on the AnyConnect Windows client, we show that Linux and iOS are based on different architectures and have distinct security issues. Our reverse engineering as well as the follow-up design analysis and fuzzing reveal 13 new vulnerabilities. Seven of these are located in the Linux client. The root cause for privilege escalations on Linux is anchored so deep in the client’s architecture that it only got patched with a partial workaround. A similar analysis on iOS uncovers three AnyConnect-specific bugs as well as three general issues in iOS network extensions, which apply to all kinds of VPNs and are not restricted to AnyConnect.

Index Terms—Virtual Private Network, Fuzzing, iOS, Linux

I. INTRODUCTION

When corporations build an internal network, they often stick to the same vendor for all components due to compatibility reasons. A vendor should offer a variety of solutions meeting all the customer’s needs. Creating and maintaining such a product range is a huge effort, and, thus, the corporate network landscape is dominated by very few vendors. Cisco’s market share including VPNs and other enterprise network equipment is around 50% [1]. Thus, users connecting to corporate VPNs will likely face a setup that requires them to install the Cisco AnyConnect client. This client supports the most popular desktop and mobile operating systems Windows, Linux, macOS, iOS, and Android. While they have platform-dependent feature sets, they are all compatible with Cisco’s Adaptive Security Appliance (ASA), which, amongst others, also provides VPN server functionality. As a product that is meant to provide secure network access and protect corporate networks, VPN clients should have a high security standard.

In this paper, we analyze the AnyConnect client for iOS and Linux. These operating systems have very different security mechanisms and network stacks, enforcing a fundamentally different implementation on both platforms. We reverse-engineer the proprietary clients and their operating system integration, analyze design issues, and test interesting interfaces with automated fuzzing. On Linux, issues anchored deep in the client’s architecture lead to privilege escalations that can only be prevented with workarounds. Even after our report and an official advisory by Cisco, the default configuration remains insecure. On iOS, our findings are not limited to the AnyConnect client. Third-party VPN applications can be integrated into the iOS network stack using the network extension framework, in which we uncover three issues. Despite these findings, iOS network extensions conceptually prevent a multitude of attack vectors that have been previously reported for desktop clients. Our main contributions are as follows.

- Reverse-engineering of the AnyConnect iOS and Linux client functionality to understand the underlying design and security assumptions.
- Protocol and design analysis of the Linux client, revealing one version downgrade and two privilege escalation bugs.
- Analysis of the iOS client, uncovering multiple issues, including plaintext data transmission.
- Fuzzing of interfaces identified during the initial analysis, discovering multiple memory corruption bugs on Linux, as well as a permanent Wi-Fi Denial of Service (DoS) that persists through iOS reboots.
- Analysis under unstable networking conditions, revealing a double-free memory corruption bug in iOS network extensions that can be triggered over-the-air.

We responsibly disclosed all identified issues. The remainder of this paper is structured as follows. Previous work is categorized to get a better understanding about which types of bugs affected which components in Section II. The security analysis in Section III focuses on the Linux client while Section IV focuses on the iOS client. Both sections follow the same structure that explains the client’s design, the resulting security assumptions as well as the individual findings. Section V concludes this paper.

II. PREVIOUS WORK

An overview of security issues that existed prior to our work is shown in Figure 1. Since the first public release of an AnyConnect vulnerability in 2011, twelve vulnerabilities per year were published on average [2]. Cisco rates issues as low, medium, high, or critical. However, no issue with the rating critical was published since a remote code execution vulnerability in 2012. We categorize all issues from Cisco’s security bulletins to get a better understanding before starting our own security analysis.
A. Cryptography

Not all published issues are directly located within the AnyConnect client—the majority is attributable to the third-party cryptographic library OpenSSL. This library had vulnerabilities like Logjam and SWEET32 [3, 4]. Thus, 63.5% of the issues fall into the cryptography category. When analyzing Cisco-specific issues, this category is out of scope.

B. Privilege Escalation

A VPN client configures network routes and encrypts all traffic. Hence, AnyConnect has components like vpndownloader requiring system or administrator permissions. These pose an interesting attack surface for privilege escalations. When excluding third-party components, privilege escalations are the most frequently reported attack vector. Since privilege escalations are systemic to VPN clients and stem from architectural issues, we provide further details about these. However, the official advisories are missing this information, and we need to rely on externally published write-ups.

One of the first privilege escalation vulnerabilities was published by Kostya Kortchinsky in 2015 [5]. This vulnerability marks a turning point in identifying further, similar privilege escalations. On Windows, vpndownloader runs with system privileges. Moreover, vpndownloader.exe parses a special Inter-Process Communication (IPC) message that defines a binary and arguments to execute it. This is not directly exploitable because the binary needs to be signed by Cisco. Kortchinsky identified a Cisco-signed executable VACon64.exe, which allows installing additional services, leading to arbitrary code execution. Cisco fixed this vulnerability by restricting vpndownloader.exe to only execute vpndownloader.exe.

Shortly after, James Forshaw and Yorick Koster independently discovered that this restriction did not include the full path. An unprivileged user could copy vpndownloader.exe to another directory and plant a malicious .dll into the same directory, again leading to code execution [6].

In 2016, Duarte Silva found a flag in the same IPC message [7]. This flag sets if vpndownloader.exe is launched from a temporary or secured application directory. Unprivileged users can modify the temporary directory.

Koster’s first report was a duplicate, but he identified a path traversal issue several years later in 2020. The directory check was flawed because Windows considers both \ and / directory separator [8].

All these issues are specific to Windows. The history of these bugs shows that they were fixed individually but the underlying design issues were not solved. Even the latest advisory of January 2021 once again includes a malicious .dll injection [9]. To the best of our knowledge, there are no write-ups for Linux or iOS privilege escalations.

C. Remote Code Execution

All public vulnerabilities that lead to code execution require user interaction. The only vulnerability with the rating critical, fixed in 2012, still demands the user to visit a malicious website that loads a Java applet [10].

D. Denial of Service

The VPN client might be interrupted or stopped. Depending on the remaining system configuration, this can either lead to disrupted network connectivity or traffic being exchanged without the additional layer of VPN encryption. For example, one vulnerability classified as DoS allows a local attacker to stop the vpndownloader service [11].

E. Sensitive Information

A VPN client has access to a lot of sensitive information that it could leak. In a VPN setting, remote information leakage is much more severe than local leakage. One vulnerability of this category allows a remote attacker to exploit insufficient boundary checks to read confidential system information [12]. This vulnerability is still only rated as medium.

F. Version Downgrade

A version downgrade of the installed client is a first step to exploit previously fixed and known vulnerabilities. The most severe issue in this category can be exploited remotely. The web launch feature allows websites to start AnyConnect using ActiveX or Java applets, but it also allowed downgrading the AnyConnect client [13]. Despite being exploitable remotely and an interesting component for exploit chains that could lead to code execution, this bug was only rated as medium.

G. Overflow

Missing input checks can cause buffer, heap, and integer overflows. These in turn lead to crashes or might change the program flow. Overflows could be assigned to the previous categories. However, they can be identified automatically using fuzzing. From a security research perspective it is hence interesting to list them separately to see how common simple programming mistakes are within the code base. One of the overflow issues allows an attacker to execute arbitrary code with system permissions [14]. It is only rated as medium, similar to most other privilege escalation bugs.

![Fig. 1: Public security bulletin vulnerability categories.](image)
H. Other

The remaining vulnerabilities stem from a variety of root causes. Issues in this category include modifying configuration files as unprivileged user or damaging existing files owned by the system user.

III. Linux Client

In the following, we analyze the AnyConnect client version 4.9.00086 on Linux. The overall architecture is similar to Windows, and despite not having source-code access, we assume that major parts of the code base are shared. However, both operating system have fundamentally different network stacks and different process interaction, resulting in various implementation-specific details. Especially platform-dependent bugs can therefore have similar root causes in the architecture, but require different fixes for each platform as they are not part of a shared code base.

We start with a component overview in Section III-A and explain the basic connection setup in Section III-B. Based on this, we can make security assumptions in Section III-C—for example, the VPN server must be ultimately trusted by a Linux client to not execute malicious code. Even with this assumption, the client’s design allows for unpatchable privilege escalations, as explained in Section III-D. Moreover, we identify further bugs with fuzzing in Section III-E.

A. Component Overview

The Linux client consists of three main binaries.
1) vpndownloader establishes VPN tunnels and applies network settings.
2) vpnui is responsible for user interaction.
3) vpndownloader downloads profile files and updates provided by the VPN server.

The core VPN functionality requires all these binaries and their interaction via IPC. IPC messages are implemented as network messages sent to local TCP sockets. The precise message format is Linux-specific and shown in Table II. While these messages are meant to enable communication between the three main binaries, they also pose an attack surface to privilege escalations.

Moreover, the binaries rely on various libraries, including ports of open-source libraries, as well as resources, which are processed frequently and control its actions. The most noteworthy resources are profiles and local policies. Profile files contain special features and rules to be used when connecting to a specific VPN server. The local policy file contains various settings also affecting the security. This overview is still very brief and simplified. The /bin directory contains 8 binaries and 2 shell scripts, and 14 libraries reside in the /lib directory.

B. Connection Setup

The AnyConnect client encrypts traffic based on Transport Layer Security (TLS). Specific actions like authentication, file download, or tunnel setup rely on HTTPS. The content-type of these messages is XML, which makes interpretation rather simple. Local IPC uses plaintext communication in a binary format over TCP sockets. On the server side, Cisco VPNs run ASA, which provides remote access VPNs, site-to-site VPNs, and firewall functionality.

With all those binaries on the client side and configuration by the ASA server, the connection setup works as follows:
1) vpnui establishes a TLS connection to the ASA server to perform user authentication. The ASA server requires a valid certificate and the user has to provide valid credentials, meaning that both parties are mutually authenticated after this step.
2) ASA replies with an XML file, containing session tokens and a list of downloadable files.
3) vpnui launches vpndownloader, which continues running in the background.
4) vpnui transfers the essential XML contents to vpndownloader via IPC.
5) vpndownloader parses the XML and downloads available profile files and other resources.
6) vpndownloader notifies vpnui about the successful download.
7) vpnui advises vpndagentd to establish a VPN tunnel.
8) vpndagentd sends a HTTPS CONNECT request to ASA to initiate a tunnel.
9) ASA replies with tunnel parameters such as a DNS server and routes.
10) From now on, VPN traffic is exchanged between vpndagentd and ASA.

Even this simplified connection process shows the AnyConnect complexity. The binary separation and local communication via IPC is meant to decouple processes running with user permissions from vpndagentd with root privileges.

| Name            | Cause                        | Impact                          | Report   | Fix     |
|-----------------|------------------------------|---------------------------------|----------|---------|
| vpndagentd-vd   | Missing version validation   | Version downgrade               | Jul 5 2020| Sep 24 2020|
| vpndagentd-pe1  | Scripts can be overwritten   | Vertical privilege escalation   | Aug 6 2020| Nov 4 2020*|
| vpndagentd-pe2  | Profiles can be overwritten  | Vertical privilege escalation   | Aug 6 2020| Nov 4 2020*|
| vpndagentd-c1   | Invalid memory address       | Crash                           | Oct 28 2020| Feb 24 2021|
| vpndagentd-c2   | Double free                  | Crash                           | Oct 28 2020| Feb 24 2021|
| vpndagentd-c3   | Heap corruption              | Crash                           | Oct 28 2020| Feb 24 2021|
| vpndagentd-c4   | Heap corruption              | Crash                           | Oct 28 2020| Feb 24 2021|

* Only a workaround configuration, still insecure by default.
C. Security Assumptions

We assume the ASA server has no malicious intent. It could spy on the client’s network traffic or modify it. Moreover, the vpndownloader can be advised to download scripts, executed by vpnui. Thus, the server needs to be ultimately trusted. This is already quite exceptional, considering that the user does not get any warnings if ASA pushes scripts. We set vpndownloader and vpnui out of scope since they do not run as privileged processes.

Furthermore, we assume TLS is secure. Breaking authenticated, end-to-end encrypted communication has the same severe impact as a malicious server. Moreover, the server and client are mutually authenticated. The server authenticates with a certificate signed by a trusted authority and the client provides a certificate or credentials.

The client is using Linux in default configuration without special security mechanisms activated. However, the operating system is compliant with a secure user role model. The AnyConnect client is also installed in default configuration, meaning that the application directory is readable by all users but only writable with root privileges.

A local attacker aims at compromising confidentiality, integrity, and availability. Moreover, they want to escalate their privileges. They have the permission to run unprivileged code in a shell and modify files in the /tmp directory. They can also open TCP ports above 1023 and connect to TCP ports on the loopback interface.

D. Architectural Issues and Logic Bugs

Overall, we discovered three bugs by manually analyzing the protocol steps: a version downgrade, overwriting scripts, as well as overwriting profiles (see Table I).

1) Version Downgrade: The vpndownloader is responsible for downloading client updates from the server. After a successful download, it sends an IPC message specifying an installer executable. Yet, the actual binary can have an older version, including downgrades.

A downgrade enables exploitation of previously disclosed bugs. However, Cisco rated this version downgrade vulnerability so low that they did not publish any advisory. This is surprising given that other downgrade vulnerabilities were included in advisories. Since version downgrades have also been reported for other desktop clients, this indicates that Cisco does not validate and fix the root cause of each vulnerability in all clients.

2) Privilege Escalation: An attacker can overwrite the OnDisconnect script and manually trigger it by disconnecting from the VPN. If scripting is not enabled in a profile, this can be bypassed by also overwriting the profile. Both can be combined for a reliable privilege escalation.

| Offset | Purpose | Default |
|--------|---------|---------|
| 00-03  | Magic byte | OCSC |
| 04-05  | Head length | 26 |
| 06-07  | Body length | |
| 08-0f  | IPC response pointer | |
| 10-17  | Unknown | |
| 18-1b  | Unknown | |
| 1c-23  | Return IPC object | |
| 24     | Message type | |
| 25     | Message identifier | |
| 26-nn  | Body | |

Listing 1: Script deployment IPC message.
a) Overwriting Scripts: Attackers can run scripts with permissions of active VPN users, since scripts are executed by vpnui. Vulnerable parts within the full script deployment process are depicted in Figure 2.

During normal operation, vpndownloader stores scripts in a temporary directory and then advises the vpnagentd to move them to an AnyConnect directory using an IPC message. This IPC message contains a temporary script path, final script path, and script hash as shown in Listing 1. The CAC-move command takes both script paths as argument. The hash value is used for a file integrity check and prevents vpndownloader to move files without read access. The value after the hash is set to 1, meaning that the script will be saved with -rwxr-xr-x permissions. Since this is an IPC message from vpndownloader to vpnagentd, it also indicates its listening port 37319 for replies. vpnagentd does not move the script directly but launches a second, privileged instance of vpndownloader moving the scripts.

The IPC messages lack authentication. Thus, every user on the system can send them to the vpnagentd port 29754. Moreover, all users can create scripts in the /tmp directory and they will be accepted by the CAC-move command that moves them to the final directory. An attacker can trigger OnDisconnect scripts immediately by sending an additional IPC disconnect message.

AnyConnect version 4.9.04053 adds a new configuration option as a workaround. Using the RestrictScriptWebDeploy element in a local policy file, it is now possible to skip the distribution of scripts. However, this is set to false by default. Users need to know this specific setting and manually disable it after the AnyConnect client installation to prevent script deployment by the server.

b) Overwriting Profiles: In case a VPN connection profile has scripting disabled, it is possible to activate scripting by overwriting the profile. The new profile needs to set the EnableScripting element to true.

The overall approach for overwriting profiles is similar to scripts. Profiles are stored in XML format and non-executable. Instead of setting the IPC message’s last value to 1, it is set to 0, which corresponds to -rw-r---r-- permissions.

In contrast to scripts, profiles are usually only applied once, even when overwriting an existing profile. However, during a reconnect, vpnui reads and processes the profile again. Under normal circumstances this would only be a simple bug that results in unnecessary parsing overhead. In this scenario, a reconnect enables attackers apply a new profile.

Cisco treated both file override bugs as one, since the resulting code execution can optionally be prevented by the new RestrictScriptWebDeploy flag. The underlying bug that enables attackers to inject arbitrary profiles and scripts remains unpatched and the default configuration is insecure. Most likely Cisco decided to leave the script deployment intact by default due to the risk of breaking existing setups. Leaving the RestrictScriptWebDeploy flag disabled by default means that users need to manually enable this flag—also after every AnyConnect client update.

e) Inter-Process Communication Parsing Mistakes

We further automate identifying bugs in the Linux IPC implementation with fuzzing. Based on the reverse-engineered IPC message format, we can inject messages while the AnyConnect binaries are running and have Internet connectivity. As listed in Table I, this reveals four individual bugs.

1) Inter-Process Communication Message Format: AnyConnect implements IPC on Linux with TCP sockets. The reverse-engineered message format is shown in Table II. Similar to the message in Listing 1, all messages start with the string OCSC. The messages can even have pointers to objects or functions. The body contains multiple Type Length Value (TLV) fields, carrying the actual payload.

2) Fuzzing Setup: We fuzz vpnagentd, because it runs with root privileges and parses IPC messages. The vpnagentd IPC interface is a non-trivial target. Without source code, we can only perform blackbox fuzzing. Additionally, vpnagentd requires a fully-functional network stack. Moreover, some bugs might only occur when injecting a message sequence or while vpnagentd is in a certain state. Thus, we fuzz on a fully functional Linux system. In a first version of the fuzzer, we tried collecting coverage with FFIDA [15]. Coverage collection significantly slows down the target and the resulting coverage is inconsistent due to the target’s statefulness. Instead, we create a dumb fuzzer that injects packets via a TCP socket. Once the socket is closed, vpnagentd likely crashed due to fuzzing. New inputs are generated with radamsa [16].

Our fuzzer discovers multiple memory corruption bugs that require a sequence of packets. Messages the fuzzer injects into vpnagentd can reach the whole network stack, and despite not being state-aware, it is very stateful and logs packet sequences. If the vpnagentd IPC interface has been fuzzed before, this was likely only with a more common single-packet, coverage-based fuzzer.

3) Fuzzing Results: In the following, we briefly describe the bugs found during fuzzing.

a) Invalid Memory Address: This bug requires sending multiple IPC messages simultaneously to vpnagentd. If an IPC message corresponds to a certain type (type=0) and ID (id!=0xd||id!=0x00), it is processed by the IPCDepot, which notifies all registered handlers. In one of the handlers, the IPC message is then decomposed into its TLV tuples. CsingleTLV::SetBuffer is called to extract the value from a TLV tuple with a memcpy operation. The application crashes when accessing the address pointer with a SIGSEGV.

b) Double Free: The following bug can be triggered when replacing the message length in the header with zero. This can lead to the message being rejected, which includes replacing the message’s memory area with zeros. Then, operator.delete(this); clears the address area belonging to the message. This causes free being called twice for the same address range, resulting in a double free. Calling free more than once to an address pointer damages the memory management data structure, which can allow arbitrary memory writes.
c) Heap Corruption #1: Another bug can be reproduced by extracting an existing IPC status message, which contains a notification about the current download progress of vpndownloader, and slightly modifying it. This status message is sent by vpndownloader to vpnagentd, which forwards it to vpnui. The TLV tuple at offset 2e defines the string to be displayed by vpnui. Replacing the message’s length field to 0006 before sending it to vpnagentd causes a crash. However, the crash does not occur during message processing, since it only corrupts parts of the vpnagentd heap. This results in a SIGABRT during later heap usage. The crash is difficult to reproduce because it only occurs when a timer expires and a new network manager client object is created. Several minutes can pass between sending the message and the SIGABRT signal.

d) Heap Corruption #2: Another discovered bug is based on the previous heap corruption. The previous bug relied on a single IPC message and waiting for timers to expire. By sending a specially crafted message sequence, the bug can be triggered faster. During the crash, a basic validation of the message takes place within CIpcTransport::OnSocketReadComplete. This involves creating an empty response message stub, which in turn calls malloc, and since the heap is already corrupted, this directly causes a SIGABRT.

4) Bug Impact: All AnyConnect binaries are compiled with the most recent binary security features, as tested with checksec [17]. The checksec output looks as follows for all binaries:

```
gef> checksec
[*] checksec for '/opt/cisco/anyconnect/bin/vpnagentd'
Canary : ✓ (value: 0x19c289dbffe6c00)
NX, PIE, Fortify : ✓
RelRO : ✓
```

Thus, we were only able to crash vpnagentd but could not alter the control flow. Note that advanced exploitation techniques might still allow to exploit such bugs under certain conditions, and as such, they should be patched.

The vpnagentd service is managed by systemd and immediately restarted upon a crash. Currently active VPN connections are deactivated, and connections residing on top of it might be dropped. However, AnyConnect cannot be used if vpnagentd keeps crashing continuously. Thus, even simple crashes can be used for a permanent DoS, which might motivate the user to manually disconnect from the VPN and use a plaintext Internet connection.

IV. iOS Client

The iOS client implementation has a very different architecture and feature set. As a result, the app components (see Section IV-A), connection setup (see Section IV-B), and security assumptions (see Section IV-C) differ a lot. Nonetheless, we identify multiple bugs listed in Table III, which are explained in Section IV-D and Section IV-E. Based on these bugs, we start further manual analysis and find that an attacker that can drop or modify network packets, such as disabling a Wi-Fi access point, can trivially cause VPN crashes without user interaction. As explained in Section IV-F, these originate from a double-free memory access while parsing VPN configurations.

A. Component Overview

iOS sandboxes all applications and limits system functions apps can access. System functionality is provided via public frameworks, which abstract system functions and add various checks. Creating VPN connections is part of the network extension framework [18]. It offers multiple variants to integrate and implement VPNs. The VPN server component, ASA, only supports TLS. AnyConnect must therefore use the Packet Tunnel Provider feature of the network extension framework. This is implemented in a custom network extension called ACExtension. The extension encrypts traffic with the OpenSSL library, similar to the Linux implementation.

The AnyConnect application can be extracted and decrypted from a jailbroken iPhone using FRIIDA [15] for further analysis. The main app binary is called AnyConnect and provides the user interface. VPN functionality is contained in the ACExtension plugin. There are two more plugin components named ACShareExtension and ACSiriExtensionUI. The main binary and plugins are non-stripped, they still contain symbol information and debug strings despite being a compiled binary.

The iOS client only needs to implement a custom packet format on top of an existing VPN interface. Yet, the code base is gigantic. Table IV lists the number of functions per binary for the app version 4.9.00518, which are 24 245 in total. Major parts of the VPN logic are shared with other platforms, and only the necessary parts like the user interface and network extension are implemented in Objective-C. Despite the iOS framework concept that should unify network extensions and encourage light implementations, AnyConnect on iOS is very complex.

| Name               | Cause               | Impact                                      | Report           | Fix         |
|--------------------|---------------------|---------------------------------------------|------------------|-------------|
| ios-plaintext      | Missing crash handler | Data sent silently without VPN after a network extension crash | Dec 22 2019      | — (won’t fix) |
| ios-dos            | Missing interface name validation | Permanent Wi-Fi DoS | Jan 28 2021 | iOS 14.6 |
| ios-0click         | Double-free when parsing configs | Network zero-click VPN crash with invalid memory access | Feb 3 2021 | iOS 14.6 |
| anyconnect-crash1  | Memory corruption    | Configuration string controlled memory access within ACExtension | Dec 13 2019 | Dec 18 2020* |
| anyconnect-crash2  | Memory corruption    | Likely memory access in ACExtension         | Dec 13 2019 | Dec 18 2020* |
| anyconnect-crash3  | Fixed dereference    | Crash only                                  | Dec 13 2019      | — (non-reproducible) |

* Claimed to be patched by Cisco, reproducibility of these bugs is limited.
B. Connection Setup

Setting up connections is based on the iOS network extension, which creates a tunnel interface to route traffic. Outbound packets arriving on the tunnel interface are read by the network extension, encapsulated, and sent to the VPN server. The server unpacks the packets and routes them to the final destination. Similar, inbound packets from the server are encapsulated by the server, sent to the client, unpacked by the network extension, and written to the tunnel interface.

Similar to the Linux client, iOS can apply profiles for a connection. iOS only implements a subset of the functions [19]. Due to iOS-specific security restrictions, many features are impossible to implement and will not have any effect when configured by the server. However, there are also features specific to mobile clients, such as the roaming behavior when switching between Wi-Fi and Cellular. Moreover, rules for connect on demand can be configured, which offers automatic switching between Wi-Fi and Cellular. Moreover, rules for connect on demand can be configured, which offers automatic switching between Wi-Fi and Cellular. Moreover, rules for connect on demand can be configured, which offers automatic switching between Wi-Fi and Cellular. Moreover, rules for connect on demand can be configured, which offers automatic switching between Wi-Fi and Cellular. Moreover, rules for connect on demand can be configured, which offers automatic switching between Wi-Fi and Cellular. Moreover, rules for connect on demand can be configured, which offers automatic switching between Wi-Fi and Cellular.

C. Security Assumptions

The iOS framework concept and application sandboxing protect users. The most dangerous features like the connect and disconnect scripts on desktop clients cannot be implemented by iOS apps. This narrows down attack vectors for privilege escalations. Moreover, it means that an app user only needs to trust the VPN server with their network traffic. Replacing traffic to exploit overflows within the client would still be possible for someone controlling the server. Compared to implementing scripting out-of-the-box, this is a limited attack context of the network extension.

Since iOS apps can only be updated via the official App Store and updates are installed automatically, downgrade attacks via the app are prevented. Users can still disable app updates manually and run an outdated version. However, there is no VPN client interface that would allow downgrades or, worst case, installing arbitrary executables.

One feature provided by iOS is the so-called Always On VPN [20]. This feature ensures a VPN stays always activated, including across reboots. The only possibility to deactivate an Always On VPN is to uninstall the according VPN profile in the settings menu. AnyConnect does not support this feature. Thus, once a VPN connection is terminated, traffic is no longer tunneled through the VPN and sent directly via a potentially untrusted Wi-Fi without the additional TLS encryption layer. Thus, if the network extension crashes, traffic is sent without VPN encryption and rerouting. While most apps should use TLS on top, some services and websites might be plaintext, including DNS. This is different from the Linux configuration where vpagentd is automatically restarted by systemd. When the app crashes, it cannot warn the user as it is already terminated, and iOS does not warn the user either. Upon our request Apple confirmed that this is the expected behavior. Since we believe that this is a dangerous and unexpected default behavior, we list it as vulnerability ios-plaintext.

D. Fuzzing the Configuration Interface

The app can be almost completely controlled through a custom URL scheme starting with anyconnect:// followed by further action parameters [21]. This includes creating connection entries, importing VPN profiles, configuring localization, connecting with pre-filled credentials, importing certificates, disconnecting from a VPN, and closing the app. Only non-destructive operations are possible via this custom URL scheme, which follows Apple’s recommendation for developers [22]. It is not possible to delete connection entries, profiles, or localizations. However, profiles and localizations can be overwritten. The supported actions are as follows.

- create: Create connection entries.
- connect: Connect with a specific connection entry identified by its host.
- disconnect: Terminate the current connection.
- close: Dismiss the AnyConnect user interface.
- import: Import certificates, profiles, and localizations.

These actions can be customized with multiple parameters. For example, to connect to a specific existing profile with pre-filled credentials and opening a website in the AnyConnect user interface after a successful connection attempt, the following parameters can be passed:

```
anyconnect://connect?host=vpn.example.com&prefill_username=users&prefill_password=password&onsuccess=http%3A%2F%2Fwww.example.com
```

Users need to manually enable this URL scheme via the setting External Control – Enabled. Thus, this interface is neither controllable remotely nor a typical one-click attack. Nonetheless, we can use it to automatically fuzz test the app’s functions. For this, we build a FFIIDA-based fuzzer that opens anyconnect:// URLs. Additionally, the user interface needs to be hooked with FFIIDA to automate the manual connection confirmation. The AnyConnect core module that is responsible for the user interface implements user prompts, which we hook, is implemented in CredentialPromptsViewController. We also automate further steps like deleting all created VPN connections later on.

Identified crashes are simple to verify by providing the according URL via a browser. Using this method, we found one bug in the iOS network stack. If the connection description string is too large, the iOS-internal settings app becomes very slow and unresponsive. Moreover, it is no longer possible to connect to Wi-Fi networks, even after a reboot. Thus,

| Module                  | Number of Functions |
|-------------------------|---------------------|
| AnyConnect              | 6653                |
| ACExtension             | 13 684              |
| ACSShareExtension       | 3557                |
| ACSiriExtension         | 351                 |
this is a DoS that affects the whole iOS network stack. AnyConnect needs to be uninstalled to get Wi-Fi working again. Sometimes, the AnyConnect VPN profile is still cached, and reinstalling the app again leads to the same Wi-Fi DoS even without installing a profile.

E. Regular Connectivity Issues and Crashes

As previously stated in Section IV-C and claimed as separate vulnerability ios-plaintext, a network extension crash on iOS leads to plaintext network traffic being sent without warning the user. These crashes occur frequently during regular usage. Everything required to trigger these crashes is an unstable network connectivity that switches between Wi-Fi, Cellular, and no connectivity.

Nonetheless, bad network connectivity during regular usage yielded in three unique crashes on the AnyConnect versions 4.8.00825 and 4.8.01097. Cisco claims to have fixed two of these crashes. In fact, we were not able to reproduce crashes with an up-to-date AnyConnect client. However, reproducing a crash requires physically moving to places with bad network connectivity. Moreover, the crashes only occurred every few days to weeks prior to the claimed bugfix. We assume that Cisco was able to find the crash sources based on our reports and fixed them.

1) Memory Corruption #1: Upon a reconnect, the configuration is applied again and the tunnel is reinitialized. This happens via the functions PacketTunnelProvider_applyVpnConfig_cb and -[PacketTunnelProviderCallback tTunnelBuffers:]. When calling objc_release in the initialization function after calling initWithCapacity, a memory corruption can occur. The accessed memory address is invalid because it contains strings like IPv4 and 86k\n, which likely stem from the VPN configuration. Thus, this memory access might be controllable by an attacker that can modify network traffic or runs the server. This crash happened multiple times with different configuration strings.

2) Memory Corruption #2: The second memory corruption causes a SIGABRT in Libsystem_malloc after calling the function -[PacketTunnelProviderwrite Packet: dataLen:isIPv4:]. The full stack trace originates from the event handler when a TLS packet is received, which in turn calls CTlsProtocol::OnSocketReadComplete. After further intermediate function calls, this results in calling CTunTapMgr::postHostBoundPacket. There is one more call to the AxtSNAKTuntap::Write handler before finally crashing in -[PacketTunnelProviderwritePacket:DataLen:isIPv4:]. Most parts in the stack trace sound rather generic, except from the Cisco-specific System Network Abstraction Kit (SNAK). Even though the crash only resulted in an abort instead of an invalid memory access, crashing via malloc indicates a memory corruption. This crash only occurred once.

3) Fixed Dereference: The third crash does not look controllable and is a simple fixed dereference at a pointer to 0x04. While it is not worth to reverse-engineer its origin to determine if it could be controllable by an attacker, it is still a crash that disconnects from the VPN server.

4) Crash Debugging: The AnyConnect app has a configuration option to enable debug logs. These logs contain messages detailing how connections are established and which settings are applied. After enabling debug logs on one of our test devices, AnyConnect kept crashing—but without producing iOS crash logs and without saving debug logs that lead to the crash. This crash behavior might explain why Cisco was not able to identify such issues during internal testing.

F. Attacker-Controlled Connectivity Issues and Crashes

We were not able to program a Packet Tunnel Provider fuzzer that causes exactly same behavior as described in the previous section. Switching network interfaces and reconnecting to VPNs are implemented within the kernel. Hence, this cannot be reproduced by injecting packets into the Packet Tunnel Provider, which is only responsible for encrypting and decrypting packets on an upper network layer before forwarding them to the iOS VPN tunnel interface. As of now, Frida only works in the user space. Fuzzing techniques that also apply to the kernel were published recently [23], but still come with a lot of limitations like restriction to selected modules and a lot of customized harnessing.

Even without specialized tooling, multiple actions can shut down the network interface or drop packets during VPN connection establishment. For testing purposes, the ifconfig command can be used on jailbroken iPhones, but connections can also be interrupted as a regular user on a standard device by switching off Wi-Fi via menus. The same behavior can be achieved without user interaction by manually switching of the Wi-Fi access point, which is something any attacker within wireless range could do by jamming packets.

Interestingly, the network interface state change caused by all of the above options triggers a completely new bug. While originally trying to reproduce the AnyConnect bugs after Cisco claimed to have fixed them, this regularly causes another crash due to an invalid memory access in the iOS network extension agent process neagent. The crash happens in the com.apple.NSXPCConnection.user. endpoint thread while deallocating an immutable dictionary (NSDictionary) object. Memory corruptions during deallocation are also known as double-free, which can be abused for accessing arbitrary memory. Memory control depends a lot on the object causing the double-free. The initial crash log contains 19 entries just for the traceback of the crashed thread, plus various additional information on register states. To locate the actual root-cause and freed object, we use the frida-trace tool to print dictionary access in neagent observed in the initial crash log. The trace output when interrupting VPN connection establishment via Wi-Fi looks as follows:

```c
/* TID 0x26803 */
-[
-[
-[
-[
-[
```
**V. CONCLUSION**

Corporate VPN solutions cannot provide the security they promise, if they continue to be developed in a non-security conscious fashion. Ideally, they add encryption to facilitate secure access to corporate networks. At the same time, their ultimate control over a user’s network traffic and integrated scripting engines controllable via the server significantly endanger an end-user’s system security. Similar to previous findings in *Cisco AnyConnect* on *Windows*, the desktop client for *Linux* has various possibilities for privilege escalations and allows the user to push scripts to be executed on the client by default. Our findings show that even the restricted *iOS* network extension framework allows integrating bloated VPN clients and has severe bugs in itself. Despite being marketed as security product, users should be very skeptical about installing and using VPN clients.

**ACKNOWLEDGMENT**

We thank the *Cisco* incident response team for their timely answers and 90-day disclosure coordination. Moreover, we thank *Apple* for confirming the crash behavior of VPNs that do not implement the *Always On VPN* feature and fixing network extension related vulnerabilities.

This work has been funded by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

**REFERENCES**

[1] Datanyze. “Market Share Category Virtual Private Networks,” https://www.datanyze.com/market-share/vpn--326, Jan. 2021.

[2] “Cisco Security Advisories,” https://tools.cisco.com/security/center/search.x?publicationTypeIds=1&resourceIds=109810&prodTypeId=Cisco&prodRId=109810&limit=100, Jan. 2021.

[3] “Multiple Vulnerabilities in OpenSSL Affecting Cisco Products,” https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20160927-openssl, Sep. 2016.

[4] “Multiple Vulnerabilities in OpenSSL Affecting Cisco Products: September 2016,” https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20160927-openssl, Sep. 2016.

[5] K. Korthinsky, “Cisco AnyConnect Secure Mobility Client v3.1.06073 E0P,” https://expertmiami.blogspot.com/2015/06/cisco-anyconnect-secur-secure-mobility-client.html, Jun. 2015.

[6] J. Forshaw, “Issue 460: Cisco AnyConnect Secure Mobility Client v3.1.08089 Elevation of Privilege,” https://bugs.chromium.org/p/project -zer0/issues/detail?id=460, Jun. 2015.

[7] D. Silva, “AnyConnect Elevation of Privileges, Part 2,” https://www.serilizing.me/2016/12/20/anyconnect-elevation-of-privileges-part-2/, Dec. 2016.

[8] Y. Koster, “Cisco AnyConnect Privilege Elevation through Path Traversal,” https://ssd-disclosure.com/ssd-advisory-cisco-anyconnect -privilege-elevation-through-path-traversal, Feb. 2020.

[9] A. Thongthua, N. Intarasorn, and S. Sangrattanapisit, “Cisco AnyConnect Secure Mobility Client for Windows DLL Injection Vulnerability,” https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-anyconnect-dll-injec-pQnryXLf, Jan. 2021.

[10] “Multiple Vulnerabilities in Cisco AnyConnect Secure Mobility Client,” https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20120620-ac, Jun. 2012.

[11] “Cisco AnyConnect Secure Mobility Client for Windows Denial of Service Vulnerability,” https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-anyconnect-dos-foXp4AV, Aug. 2020.

[12] “Cisco AnyConnect Secure Mobility Client for Linux Out-of-Bounds Memory Read Vulnerability,” https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20190515-anyconnectclient -oob-read, May 2019.

[13] “Cisco AnyConnect Secure Mobility Client Software Downgrade Vulnerability,” https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/Cisco-SA-20120620-CVE-2012-2494, Jun. 2012.

[14] “Cisco AnyConnect Secure Mobility Client VPNAPI COM Buffer Over-Flow Vulnerability,” https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/Cisco-SA-20131104-CVE-2013-5559, Nov. 2013.

[15] O. A. V. Ravns, “Frida: A world-class dynamic instrumentation framework,” https://frida.re/, Jan. 2021.

[16] A. Helin, “radamsa,” https://github.com/akih/e-radamsa, Jan. 2021.

[17] C. Alladoum, “GFE - GDB Enhanced Features,” https://gfe.readthedocs.io/en/master/, Jan. 2021.

[18] Apple, “Network Extension Framework,” https://developer.apple.com/documentation/networkextension, Jan. 2021.

[19] “Cisco AnyConnect Secure Mobility Client Administrator Guide, Release 4.9, The AnyConnect Profile Editor,” https://www.cisco.com/c/en/us/td/docs/security/vpn_client/anyconne ct/anyconnect49/administration/guide/f.AnyConnect_Administrator_ Guide_4-9/anyconnect-profile-editor.html, Jan. 2021.

[20] Apple, “Always On VPN overview,” https://support.apple.com/guide /deployment-reference-ios/always-on-vpn-tore8663896/1/web1, Dec. 2020.

[21] “Cisco AnyConnect Secure Mobility Client Administrator Guide, Release 4.9, AnyConnect on Mobile Devices,” https://www.cisco.com/c/en/us/td/docs/security/vpn_client/anyconnect /anyconnect49/administration/guide/f.AnyConnect_Administrator_Gui de_4-9b.AnyConnect_Administrator_Guide_4-4.chapte r_0110.html, Jan. 2021.

[22] Apple, “Defining a Custom URL Scheme for Your App,” https://developer.apple.com/documentation/xcode/allowing_apps_and_websites_to o_link_to_your_content/defining_a_custom_url_scheme_for_your_app, Jan. 2021.

[23] N. Williamson, “Designing sockfuzzer, a network syscall fuzzer for XNU,” https://googleprojectzero.blogspot.com/2021/04/designing-sockfuzzer-network-syscall.html, Apr. 2021.