Back to School: On the (In)Security of Academic VPNs

Ka Lok Wu[†] Man Hong Hue[†,‡,1] Ngai Man Poon[†] Kin Man Leung[§]
Wai Yin Po[†] Kin Ting Wong[†] Sze Ho Hui[†] Sze Yiu Chau[†,2]

[†] *The Chinese University of Hong Kong*
[‡] *Georgia Institute of Technology*
[§] *The University of British Columbia*

Abstract

In this paper, we investigate the security of academic VPNs around the globe, covering various protocols that are used to realize VPN services. Our study considers 3 aspects that can go wrong in a VPN setup, which include (i) the design and implementation of VPN front-ends, (ii) the client-side configurations, and (iii) the back-end configurations. For (i), we tested more than 140 front-ends, and discovered numerous design and implementation issues that enable stealthy but severe attacks, including credential theft and remote code execution. For (ii), we collected and evaluated 2097 VPN setup guides from universities, and discovered many instances of secret key leakage and lack of consideration to potential attacks, leaving many client-side setups vulnerable. Finally, for (iii), we probed more than 2000 VPN back-ends to evaluate their overall health, and uncovered some concerning configuration and maintenance issues on many of them. Our findings suggest that severe cracks exist in the VPN setups of many organizations, making them profitable targets for criminals.

1 Introduction

Institutions and companies often use Virtual Private Networks (VPNs) to facilitate remote access to internal services and resources. In recent years, reliance on VPNs has soared worldwide in part due to social distancing policies during the COVID pandemic [6]. Unfortunately, despite a connotation of security, not all VPN implementations and deployments are adequately secure. In fact, industry-led research suggests that leaked VPN credentials are often sold by Initial Access Brokers (IABs) to criminals for launching further attacks [7,9], for example, the infamous Colonial Pipeline ransomware attack that led to a temporary fuel shortage in several U.S. states [8].

The aforementioned attacks motivate a systematic review of the security of *organizational VPNs*. Since the practices of most companies are opaque to researchers, we focus instead on VPNs deployed at academic institutions (hereafter *academic VPNs*) which typically disclose more information on their deployments. We believe many of the findings and lessons learned from academic VPNs can also transfer to organizational VPNs. Moreover, report suggests academic VPNs are also being targeted by IABs [10], thus they warrant a thorough investigation in their own merit. Despite an expectation of enabling secure remote access to internal resources, unfortunately, as will be explained later, many academic VPNs actually enable trivial compromise of user credentials as well as the client machines.

Although previous studies covered different types and aspects of VPNs [19,28,36,50,57,58], they do not paint a comprehensive picture of the current landscape of academic VPNs. Under the typical "bring your own device" (BYOD) model, VPN users configure *front-ends* following setup guides prescribed by their IT admins, who in turn maintain and configure the VPN *back-ends*, also known as *gateways*, purchased from and implemented by *vendors* of networking equipment. Additionally, front-ends are either part of the operating systems (OSes), or implemented and released as standalone apps by the same vendor of the VPN back-end, in which case the apps might incorporate proprietary designs. More importantly, organizational VPNs often reuse existing Single Sign-On (SSO) credentials, making them ideal targets for IABs.

As such, in this study, we consider security issues in academic VPNs that can arise from 3 different *perspectives*: ⟨P1⟩ poor designs and implementations of the front-end apps by network equipment and OS vendors; ⟨P2⟩ poor front-end settings prescribed to users by IT admins; and ⟨P3⟩ poor back-end configurations adopted by IT admins. Partly motivated by the proliferation of IABs, we are particularly interested in understanding *server authentication* issues that can lead to credential theft and other forms of access compromise. Our study of the academic VPN ecosystem is thus structured as follows. For studying ⟨P1⟩, we test the behaviors of front-end implementations under a back-end impersonation attack. This helps us to determine whether a front-end is inherently inse-

[1] Work done while at The Chinese University of Hong Kong.
[2] Corresponding author.

cure or requires *user precautions* to achieve secure outcomes. The discovery of test subjects for ⟨P1⟩ is informed by our efforts on studying ⟨P2⟩, in which we inspect and evaluate the setup guides released by academic institutions worldwide. Setup guides are evaluated based on the results of studying ⟨P1⟩, which allow us to determine the likely security outcomes of VPN setup guides (*e.g.*, whether user precautions are explicitly mentioned). Finally, with the gateway addresses discovered from studying ⟨P2⟩, we also measure and evaluate configuration issues on VPN back-ends, thus covering ⟨P3⟩.

Concerning server authentication, VPN protocols can generally be classified into three types: ⟨SA0⟩ no server authentication; ⟨SA1⟩ server authentication based on pre-shared key (PSK); and ⟨SA2⟩ server authentication based on public key infrastructure (PKI). However, as we will explain later, due to flaws in design, implementation and configuration, many ⟨SA1⟩ and ⟨SA2⟩ VPNs in practice are *effectively* as weak as their ⟨SA0⟩ counterparts under impersonation attacks.

Finding summary. We found that a non-negligible number of schools are vulnerable to VPN credential theft due to server authentication issues. First, we discovered 44 ⟨SA0⟩ VPNs and 149 ⟨SA1⟩ VPNs, 119 of which have their PSKs exposed. We then uncovered numerous implementation issues in various ⟨SA0⟩ and ⟨SA2⟩ VPN front-ends that make them vulnerable to credential theft and in some cases remote code execution (RCE). Additionally, we found that many setup guides do not achieve the security promises of ⟨SA2⟩ VPNs, effectively downgrading them to ⟨SA0⟩. Finally, we uncovered several maintenance and configuration issues on VPN gateways, including unpatched cryptographic vulnerabilities and poor certificate management practices.

Contributions. To the best of our knowledge, we are the first to present a comprehensive review of academic VPNs. The fact that variants of known attacks worked well in this study echoes with our belief that the academic VPN ecosystem has not been thoroughly scrutinized before. To help practitioners and future research, we demonstrate and document weakness patterns in academic VPNs through the following exercises:

1. We test more than 142 VPN front-ends for various protocol stacks on 4 major OSes, and discovered numerous design and implementation flaws that can lead to stealthy but severe attacks, including credential theft and RCEs.

2. We present the first large scale evaluation of academic VPN setup guides. 2097 setup guides were collected and inspected, and we found many to ignore important precautions and exception handling, leading to insecure client-side setups.

3. We probe more than 2000 VPN gateways to better understand their configurations and security hygiene. We also discuss the implications of cryptographic weaknesses and other management issues discovered by the probe.

4. We engage in responsible disclosure to help vendors and schools fix their VPN setups. We also provide recommendations to better secure organizational VPNs in general.

2 Background and Scope of the Study

In this study we consider VPNs on major OSes, *i.e.*, macOS, Windows, Android, and iOS, as organizations typically only support those four. Here we give a summary of the adversary and protocols being considered.

Adversary Model. We primarily consider an active on-path attacker, otherwise known as a man-in-the-middle (MITM), which can observe, modify and redirect network traffic, but is not a member of the target organization. We note that this is a very realistic model, for instance, when a user connect to VPN through a hotel Wi-Fi, the network admins and middleboxes at the hotel naturally have such adversarial capabilities. Additionally, we also consider the threat of an insider (*e.g.*, a disgruntled student) colluding with the MITM.

Protocols Considered. For this study, we consider VPN protocols that (i) have native support on the 4 major OSes, or (ii) are supported by front-end apps used by actual organizations. As discussed before, they can be classified into ⟨SA0⟩, ⟨SA1⟩ and ⟨SA2⟩, based on how server authentication is achieved:

⟨SA0⟩ **no server authentication.** A prominent example that we consider is **PPTP** [31], which is one of the oldest VPN protocols still in use. PPTP is natively supported on Windows and Android. The typical authentication methods used with PPTP include the Password Authentication Protocol (PAP) and the Microsoft Challenge-Handshake Authentication Protocol version 2 (MSCHAPv2). In the most common case of PAP, the username and password are sent to the server in cleartext. In MSCHAPv2, the client computes and sends a response based on its password and a challenge from server. Although in theory MSCHAPv2 achieves *mutual* authentication, by design the client sends its response *before* the server does. Thus a server impersonator can choose arbitrary challenge and get the client response, and then attempt an *offline* dictionary attack to recover the password. Moreover, after at most 2^{56} brute-force attempts, one can recover the MD4 hash of client password and use that to login as the victim via MSCHAPv2 [39]. Since PPTP does not offer additional authentication and confidentiality, we consider *direct* usage of PPTP with these methods vulnerable to credential theft.

PPTP is sometimes used with the Microsoft Point-To-Point Encryption (MPPE) protocol [44], which adds encryption to the VPN traffic. However, this does not protect the user authentication methods. In fact, the key derivation in MPPE depends on the completion of a challenge-response protocol (*e.g.*, MSCHAPv2) and thus does not work with PAP [59]. PPTP can also use the Extensible Authentication Protocol (EAP) framework for user or mutual authentication. In theory it is possible to first set up a TLS tunnel (*e.g.*, EAP-TTLS) to protect the likes of PAP and MSCHAPv2, which upgrades PPTP from ⟨SA0⟩ to ⟨SA2⟩. Although MSCHAPv2 inside a TLS tunnel is quite common in enterprise Wi-Fi [14, 33], as we will see later, it is rarely used with PPTP.

⟨SA1⟩ **PSK-based server authentication.** For this, we con-

sider the various protocol stacks that involve **IPSec**. In such setups, IPSec provides a secure tunnel that protects the user authentication exchange and the subsequent traffic. Although TLS also supports the use of PSK [26], such ciphersuites are seldom used in practice, so we do not consider TLS with PSK in this paper. Different combinations of protocol stacks can realize the PSK-based IPSec VPN, and they are confusingly named by different vendors. A very common stack is known as **L2TP/IPSec-PSK VPN**, which uses IPSec to protect L2TP traffic [45], where L2TP can support the likes of PAP and MSCHAPv2 for user authentication. Another common stack uses IPSec-PSK with XAUTH, where the user password is typically sent to the server directly inside the IPSec tunnel, especially when the setup uses RADIUS to implement SSO. The so-called "Cisco IPSec" is also a variant of **IPSec-PSK XAUTH**. Currently two versions (v1 and v2) of the Internet Key Exchange (IKE) protocol exist in IPSec. IKEv2 can also use EAP or PKI-based methods for user or mutual authentication, though PSK continues to be supported. In general, when (i) password-based methods are used for user authentication, and (ii) server authentication hinges solely on the PSK, knowledge of the PSK allows an attacker to set up an impersonator and perform credential theft.

⟨SA2⟩ **PKI-based server authentication.** Another approach to the authentication problem is to use public-key cryptography, which often involves the use of X.509 certificates. Both **TLS** and **IPSec** support this and will be considered. In IPSec, both IKEv1 and v2 support some server authentication methods based on certificates and signatures, and user authentication can happen either through password-based methods or also based on certificates and signatures. For TLS-based VPNs, vendors might have their own proprietary designs for authenticating users inside the TLS tunnel. Additionally, we also consider OpenVPN, which uses TLS as its underlying authentication and key negotiation protocol.

3 ⟨SA0⟩ and ⟨SA1⟩ VPNs in action

3.1 Attack setups and testing front-ends ⟨P1⟩

Although the weaknesses of ⟨SA0⟩ and ⟨SA1⟩ protocols were discussed in previous work [19], we are not aware of any testbeds that perform impersonation attacks against them. We thus built our own setups for testing the front-ends. For both PPTP and IPSec-based VPNs, we use a Linux laptop as the Wi-Fi access point, and set up dummy interfaces (virtual network) with addresses of the target back-end servers. Then for PPTP VPNs, we use the open-source `pptpd` as the impersonating back-end. For IPSec-based VPNs, we used open-source implementations including `strongSwan` and `Libreswan`, depending on the authentication methods and protocol versions needed. The main challenge was to come up with back-end configurations compatible with the front-ends, as there are many options at each layer of the stack (*e.g.*, IKE version and mode, ciphers, authentication methods, *etc.*). In some cases we had to modify the code of IPSec implementations to accommodate non-standard behaviors of certain front-ends. In the end, we built working setups of PPTP impersonator that works with the native Windows and Android front-ends, and various IPSec-PSK impersonators that can work with the native front-ends on macOS, Windows, Android, iOS, as well as front-ends apps from network equipment vendors like Fortinet and Aruba. These setups helped us demonstrate the PPTP front-ends tested are indeed vulnerable to credential theft, and similarly so for IPSec-PSK front-ends if attacker knows the PSK. Besides, we also discovered some product-specific weaknesses, as discussed below.

Lack of choice on Android. We found that on Android, the user cannot choose the preferred user authentication methods when configuring PPTP and IPSec-PSK VPNs. As such, the back-end server will decide the method. A rational impersonator would thus choose PAP over methods like MSCHAPv2 to get the password directly. In fact, as discussed before, MPPE cannot be used with PAP due to its key derivation requirements. However, we found that on Android, when the user enables MPPE for PPTP, it still attempts to connect even if the server chooses PAP. This could lead to a false sense of security as the user cannot cherry-pick authentication methods but direct password theft continues to be possible.

Encrypted PSK on Aruba VIA. While testing, we found that the Aruba VIA apps fetch a configuration profile via TLS at first login before subsequent IPSec-PSK VPN connections can be made, but the PSK is nowhere to be found on the UI. Interestingly, the fetched profile can be found on local storage of Windows, which contains an *encrypted PSK*. Alternatively, under the insider threat model, we found that one can also perform a self-MITM to obtain the encrypted PSK from the profile-fetching TLS with Aruba app on other OSes. Although this can be seen as an attempt to protect the PSK from insiders, we found that it can be easily worked around. We decompiled the Aruba VIA Android app and quickly found that the PSK is encrypted using 3DES-CBC with a secret key hardcoded inside the app. Using the hardcoded secret key and some simple Java code, any insiders can decrypt the PSK in an offline manner, rendering the protection moot.

3.2 Setup guides prescribed to users ⟨P2⟩

Motivated by the aforementioned attacks against ⟨SA0⟩ and ⟨SA1⟩ VPNs, we are curious to see if *academic units*, which include universities and their departments, actually use such VPNs in practice. In this part of the study, we locate and inspect the VPN setup guides that are publicly accessible on school websites. This exercise serves three purposes: (1) provide an overview of the distribution of various VPN protocols, including ⟨SA2⟩ ones; (2) identify ⟨SA2⟩ front-ends to be tested in Section 4; and (3) to better understand ⟨P2⟩ of ⟨SA0⟩ and ⟨SA1⟩ VPNs. Specifically, we focus on whether

academic units uses PPTP or IPSec-PSK, and in the latter case, whether the PSK itself is also publicly accessible.

We first consolidate a list of universities by consulting public resources (*e.g.*, Wikipedia and the list from [33]), which includes 6600 schools in 45 regions. For each university, we programatically collect Google's top 8 search results with the help of Selenium, using the following search keywords: `VPN site:<domain>`, `<domain> AND (VPN OR Virtual Private Network OR Off-Campus Network Access OR (VPN AND (IPSEC OR L2TP OR SSL OR PPTP)))`, `IPSEC or "L2TP" or "SSL" or "PPTP" site: <domain>`. We then inspect the crawled search results to locate applicable VPN setup guides manually. A setup guide is applicable if it prescribes VPN protocols considered by the study. If we find a setup guide prescribing the use of ⟨SA0⟩ and ⟨SA1⟩ VPNs, we archive the corresponding setup guide as a snapshot and log the contact information for responsible disclosure. For TLS-based VPN setup guides, we curate a list of vendors and their corresponding front-end apps that are used by some schools, which are tested in Section 4.1.2. This data collection and evaluation exercise spanned from September 2021 to March 2022, involving six authors. Three of them were in charge of the major portion of the data entries, with each covering thousands of the schools considered. Setup guides from non-Anglosphere nations tend to be written in their local languages (*e.g.*, CJK, German, French, and Thai). If a school has setup guides in multiple different languages, we favor the one in local language of the region, as that tends to be more informative than the translated English version. Depending on our language proficiency, we use Google Translate (GT) and other online dictionaries to help us understand unfamiliar words. Specifically, setup guides in languages of high proficiency (e.g., Japanese and Chinese) were evaluated without using GT. For other languages of moderate or basic proficiency (e.g., German and French), GT was used as an assistive tool. Only for languages of no proficiency (e.g., Thai and Arabic), we relied on GT.

Findings. From the 6600 universities considered, we found 2097 applicable setup guides. One school can have multiple setup guides, as it can run several VPNs for different staff and students, and some departments might also have their own VPNs. Out of the 2097 applicable setup guides, 149 (7.1%) of them rely on IPSec-PSK VPNs (106 are L2TP/IPSec-PSK, 29 are "Cisco IPSec", and the rest are from other vendors). Among them, 118 (79.2%) PSKs are *publicly exposed* on the setup guides. The others either require user login or contact IT admins to obtain the PSK. Although this is arguably better than publicly exposing their PSKs, it is still susceptible to credential theft by a MITM colluding with an insider. Moreover, we found 44 (2.1%) setup guides prescribe the use of PPTP VPNs. To make sure that they are indeed using PPTP as ⟨SA0⟩ and not ⟨SA2⟩, we further inspected the 19 setup guides that apply to Windows, and noticed that none of them are using EAP. As such, members of these schools that rely on insecure usage of ⟨SA0⟩ and ⟨SA1⟩ VPNs are likely susceptible to credential theft. Finally, we also found that 4 (0.2%) setup guides prescribe PKI-based IPSec for Android devices, and 1981 (94.4%) setup guides prescribe the use of TLS-based VPNs, both of which will be considered in Section 4.

3.3 Configuration issues on back-ends ⟨P3⟩

Here we focus on a known weakness in IPSec-PSK. IKEv1 has an aggressive mode, which exchanges fewer messages and delivers a faster establishment of the IKE security association than the main mode. However, it is known that the aggressive mode enables an offline dictionary attack on the PSK [1, 28], as the hash of the PSK is transmitted in cleartext. This hash can be easily obtained by a *passive on-path* attacker when a user connects using aggressive mode, or by someone who *actively probes* the back-end. VPN gateways are thus recommended to disable the IKEv1 aggressive mode [1].

As part of our evaluation, we probed the IKE back-ends discovered in Section 3.2 using the open source tool `ike-scan`, to see if they enabled support for IKEv1 aggressive mode. Notice that one setup guide could mention multiple backup gateways. Among the 173 back-ends that we probed, 25 (14.5%) responded with an aggressive mode handshake message. Subsequently, we used `psk-crack` and its default dictionary file to attempt an offline dictionary attack, and were able to uncover *one additional PSK* that was not exposed in the setup guides. Furthermore, by incorporating the exposed PSKs from Section 3.2 into the dictionary file, we were able to confirm that 13 out of the 25 aggressive mode handshake responses are indeed using the publicly exposed PSKs previously found. Given that the purpose of this exercise is to measure the support of aggressive mode instead of the predictability of PSKs, we refrain from attempting massive scale attacks with large dictionaries. Our results nonetheless suggest that IKEv1 aggressive mode continues to be enabled on a considerable number of IPSec gateways, despite its known weaknesses.

4 ⟨SA2⟩ VPNs in action

4.1 Attack setups and testing front-ends ⟨P1⟩

4.1.1 IPSec with certificates and signatures

For testing IPSec-PKI front-ends, we modified the setup described in 3.1, and managed to make it work with the Android native front-end under the "IPSec Xauth RSA" and "IPSec Hybrid RSA" modes. In both modes, instead of relying on a PSK, the client is supposed to validate the server certificate (and a fresh signature) for authenticating the server. Interestingly, we found that the Android 11 UI *by default* does not verify the server certificate, which enables our impersonating setup with self-issued certificates to perform credential theft.

This is highly reminiscent of the Wi-Fi UI problem found on earlier versions of Android [33].

4.1.2 TLS-based (and OpenVPN-based) VPNs

Given that 94.4% of the setup guides prescribe the use of TLS-based VPNs (Section 3.2), we also test the various VPN front-ends used by those schools. In practice, except for SSTP on Windows, all the other TLS-based VPNs we discovered require the installation of an additional app, which are typically also implemented by the vendor of the corresponding VPN back-end. From Section 3.2, we discovered and tested 132 TLS-based VPN apps (160 including different modes) on 4 major OSes from 30 vendors. Some vendors do not have apps for all 4 OSes, but some might have multiple front-ends apps even on the same OS. For instance, we found that Fortinet has both v6 and v7 in circulation and used by various schools. OpenVPN, on the other hand, has two apps for Windows, one is open-source (named OpenVPN GUI) and the other is not (named OpenVPN Connect). Similarly, on top of the traditional desktop app, Cisco AnyConnect also has a UWP version from Microsoft Store.

For this round of testing, we consider 2 test cases: (i) *untrusted* server certificate with correct name; and (ii) trusted server certificate with *incorrect* name. Both test cases represent realistic threats within reach of the adversary we consider. Test case (i) captures the case where an on-path attacker uses an impersonating certificate issued by an untrusted issuer (*e.g.*, self-signing), while test case (ii) captures the case where the attacker purchases a valid certificate from a trusted commercial CA for an arbitrary domain under control, and then uses that as the impersonating certificate. To realize an active on-path attacker, we use a Linux laptop as the Wi-Fi access point, and run `mitmproxy` [22] on it with certificates of our choosing to attempt TLS interception. We noticed that `mitmproxy` can run into compatibility issues with certain front-ends, and thus we augment the adversarial setup with `SSLproxy` [56] if necessary. For testing OpenVPN and other apps based on it, we set up an OpenVPN back-end that uses our certificates and listens on a dummy interface (see Section 3.1). Due to space constraints, we discuss some of the compatibility issues and other interesting findings in Appendix A.

(i) Untrusted certificate with correct name. For ease of discussion, the results of this test are shown in 3 tables. Table 1 shows the list of apps found to be vulnerable under this test case. Table 2 presents the behavior of apps that rely on user precautions to achieve secure outcomes, which serves as the basis of the setup guide evaluation in Section 4.2. Finally, Table A1 in Appendix shows the apps that programatically reject untrusted certificates without relying on user precautions, and are thus not found to be vulnerable. Due to space constraints, we use shorthands instead of full app names in the tables, and show the mapping as well as the tested versions and origins of the apps in Table **??** in Appendix. Also notice that some apps support multiple *modes* of operation, for example, whether to use SSO protocols such as SAML for user authentication. However, not all apps from the same vendor support such modes. For instance, the SAML mode is not available on the UWP version of the Cisco AnyConnect app for Windows 10. Additionally, some apps have a distinct *bootstrapping* phase for fetching configuration profiles at the first successful login. For example, the OpenVPN and Aruba VIA apps first fetch configuration profiles via TLS, and then later use a different protocol (*i.e.*, OpenVPN and IKE-PSK) for the actual VPN connection. GlobalProtect also has 2 distinct phases, which behave differently as shown in the tables. For such apps, we can only test the *subsequent* VPN login phase with gateways of our affiliated institutes using our own accounts.

Table 1: VPN front-ends vulnerable to untrusted certificates

app (mode)	OS	Out
DPTech	Mac	R
	W10	R
	And	R
	iOS	R
Huawei Anyoffice	Mac	P
	W10	P
	iOS	P
Huawei Seco	And	P
	iOS	P
Hillstone	Mac	P
	W10	P
	And	P
	iOS	P
Hillstone (gmssl)	And	P
H3C	Mac	P
	W10	P
	And	P
	iOS	P
V5	W10	R
	Mac	R
	And	R
	iOS	R
V5 (gmssl)	W10	R
Qianxin	W10	P
	Mac	P
	And	P
	iOS	P
AhnLab	W10	P
Aruba (bootstrapping)	W10	P
Sangfor	Mac	R
	W10	R
	And	R
	iOS	R
TOPSec	Mac	P
	W10	P
	And	P
	iOS	P
Array MotionPro	Mac	P
	W10	P$^{\ominus}$
Ruijie	Mac	P
	W10	P
	And	P
	iOS	P
Enlink	iOS	R
EnAgent	Mac	R
	W10	R
	And	R
	iOS	R
Forti new (SSO)	Mac$^{\boxminus}$	P$^{\alpha}$ or H
	W10$^{\boxminus}$	P$^{\alpha}$ or H
	And	P$^{\alpha}$ or H
	iOS	P or H
Forti Fabric (SSO)	And	P$^{\alpha}$ or H
	W10$^{\boxminus}$	P$^{\alpha}$ or H
	iOS$^{\boxminus}$	P or H
	Mac$^{\boxminus}$	P$^{\alpha}$ or H

$\boxminus$ has UI option on certificate validation, but does not affect the browser invoked for SSO
α additional prompts about invalid cert will be shown and need to be accepted by user

For both Table 1 and Table 2, the Out column captures what kind of credential or access compromise the attacker can perform (Plaintext/Reverse engineered/Dictionary attack/pHishing/Mitm login/Unknown), if the server certificate is accepted. In other words, it represents the *outcome* of successful attacks. Such a fine-grained classification is needed because not all apps directly transmit the username and password inside TLS. Quite a few apps have additional encoding or scrambling of the user credentials, many of which we man-

aged to reverse engineer, though for 2 apps the mechanism remains unknown to us.

Insecure VPN front-ends. To our surprise, 56 distinct apps from 15 vendors automatically accept untrusted server certificate *without* any user interventions (Table 1). We consider these apps vulnerable because there are no ways for users to enable or manually enforce certificate validation. In other words, using these VPN front-ends directly exposes users to a realistic threat of stealthy credential theft. It is also interesting to see that despite coming from the same vendor, apps for different OSes might not deliver the same behavior in response to potential impersonators. For example, while the Aruba VIA apps on iOS, macOS and Android all prompt the user to decide whether to accept the untrusted certificate (thus in Table 2), the Windows app blindly accepts it.

Misguided and vulnerable designs. We found that instead of performing proper certificate validation to secure the TLS tunnel, several vendors attempt to introduce additional scrambling to protect the password being transmitted, which we reverse engineered (Out = R) through observing and manipulating the TLS traffic, and in some cases by inspecting the front-end code. For example, the DPTech apps base64 encode the password before sending it to the back-end, which is trivial to decode. Moreover, the Sangfor EasyConnect front-end apps use RSA to encrypt the password, with an RSA public key that was received earlier in the TLS tunnel. To obtain the actual user password, an active MITM can simply replace the RSA exponent with a value of 1 after compromising the TLS session. This way the RSA modular exponentiation does not hide the message. Similarly, the Enlink EnApp and EnAgent apps use AES-CBC to encrypt the password with a secret key that was received from the back-end earlier in the TLS tunnel, and the IV is simply the reverse of the secret key. As such, the MITM can observe both the key and ciphertext after compromising the TLS, and easily decrypt the user password. The V5 apps use a similarly vulnerable design, but with a hardcoded secret key and IV instead.

Additionally, we found the Array MotionPro app on Windows 10 decides whether to perform certificate validation based on a policy file it fetched from the gateway prior to user login. However, since the TLS used for this fetching also accepts untrusted certificates, the MITM can simply intercept and rewrite the policy file into disabling future certificate validation, and later intercept the user password (Out = $\mathsf{P}^{\ominus}$). Array front-end apps for other OSes do not exhibit this behavior, and are directly susceptible to MITM credential theft.

Phishing FortiClient SSO mode. Another class of subtle vulnerabilities concerns the SSO mode of the various FortiClient apps. In the SSO mode, a built-in browser will be invoked to first fetch policy files `<gateway>/remote/info` and `<gateway>/remote/saml/start`, and then perform the actual SAML for user authentication. Interestingly, we found that the browsers launched accept untrusted certificates for the 2 initial requests of policy files, which opens up possibilities for attacks. Most importantly, one can intercept the *request* for `<gateway>/remote/saml/start` and exploits it to stealthily redirect the client to a phishing website (Out = H). The main idea is correct the browser's expectation of the SSO server identity by using HTTP 301. Since none of the browsers launched by the FortiClient apps under SSO mode show the address bar, users will have no ways to tell that the SAML request has been redirected to a malicious phishing site. The exact behavior differs for the apps on different OSes, but we assume the attacker has set up `http://www.attacker.com/login` that HTTP 301 redirects to `https://attacker.com/login`, which is the actual phishing site but has a valid certificate.

For macOS, the MITM can simply rewrite the request for `<gateway>/remote/saml/start` so that it becomes a request for `http://www.attacker.com/login`, and upon receiving the HTTP 301, the browser launched for SAML will continue with `https://attacker.com/login`. No alerts will be shown throughout this process if `attacker.com` has a valid certificate. For Windows, the same attack also achieves stealthy redirection to a phishing site. Moreover, we found that if the final `attacker.com` has an invalid certificate, the browser will attempt to import it into the trusted CA store on Windows, and a prompt will be shown to the user to decide whether to import or not. For a screenshot, see figure B2 in Appendix B. Depending on how alert a user is, this gives a pathway for an attacker to inject a root CA on the machine, potentially enabling MITM attacks against other TLS traffic.

For Android, the app expects a formatted string that describes the scheme and URL of the SAML login page from `<gateway>/remote/saml/start`. We reverse engineered the format by decompiling the app, and can thus feed a target URL of our choosing (*i.e.*, `www.attacker.com/login`) as the MITM. In fact, if the attacker explicitly specify the scheme of `http://` in the target URL, the browser is even willing to continue with the phishing site without TLS. The same attack, including the specification of `http://` also works against the iOS app under SSO mode. In fact, based on our testing, the built-in browser launched by the iOS app even accepts invalid certificates for `https://attacker.com`. Thus instead of the phishing attack described above, the MITM can also simply intercept the TLS with the SSO login page and get the user password (Out = P). For non-iOS apps, intercepting the SSO login page can also work if the user accepts invalid server certificates when prompted (Out = P^{α}), as certificate validation of the built-in browser kicks in after the redirect.

Reliance on user precautions. As discussed before, many VPN front-ends depend on user precautions to achieve secure outcomes. The precise behaviors of such apps are presented in Table 2 with the help of additional columns: CertChkOpt captures whether certificate validation is a configurable option in the user interface (UI); SuffInf captures whether sufficient information (*e.g.*, certificate fingerprint) is given to the user for deciding if the certificate is trustworthy; CertDisBehav

only applies to apps with CertChkOpt = T, and captures what the apps do (Prompt or Accept) with an untrusted certificate when validation is configured as disabled.

As shown in Table 2, apps vary quite significantly in terms of their UI and certificate validation enforcement, and some design choices are alarming. In particular, several apps by default have certificate validation disabled (CertChkOpt = T!). For the ones that also have CertDisBehav = A, they are vulnerable to MITM attacks right out of the box, which defies the *"secure by default"* principle. Similarly, apps on some OSes do not show informative alerts to facilitate user decision-making (SuffInf = F), which could provide incentives for users to just accept arbitrary certificates when prompted. In Section 4.2 we will investigate whether setup guides carefully consider and articulate the necessary user precautions.

RCE and drive-by attacks. On top of credential theft, we also investigate the possibility of injecting code onto the user machine running vulnerable VPN apps, by exploiting the lack of certificate validation in the TLS used for software updates. For this, we run the aforementioned TLS intercepting setup and monitor for intercepted traffic that appear related to software updates (*e.g.*, app querying the gateway or vendor for the latest version number), while we interact with the VPN apps (*e.g.*, attempt user login) and accept prompts of untrusted certificates, if any. In the end, we identified 22 apps that have RCE potentials, as presented in Table 3.

Table 2: Behaviors of VPN front-ends reliant on users

app (mode)	OS	CertChk Opt	Suff Inf	CertDis Behav	Out
Cisco	Mac	T	F	P	P
	W10	T	F	P	P
	W10 (UWP)	T	T	P	P
	And	T	T	P	P
	iOS	T	T	P	P
Pulse	Mac	F	T	–	P
	W10	F	T	–	P
	And	F	T	–	P
	iOS	F	F	–	P
Citrix SSO	And	T	F	P	P
Citrix Workspace	And	T!	T	P	P
	iOS	T!	T	P	P
GlobalProtect (bootstrapping)	Mac	F	T	–	P
	W10	F	T	–	P
	And	F	T	–	P
	iOS	F	T	–	P
Aruba (bootstrapping)	Mac	F	F	–	P
	And	F	F	–	P
	iOS	F	F	–	P
F5	And	F	T	–	P
F5 (SSO)	And	F	T	–	P
F5 BIG-IP Edge	W10	F	T	–	P
	Mac	F	F	–	P$^{\curlywedge}$ or H$^{\curlywedge}$
Forti new	W10	T	T	A	P$^{\perp}$
	And	F	F	–	P
	iOS	F	F	–	P
Forti old	Mac	T	T	A	P
	W10	T	T	A	P$^{\perp}$
	And	T!	F	A	P
	iOS	T	F	A	P
Forti old (SSO)	Mac	T	F	A	P
Forti MSFT Store	W10	T$^{£}$	–	A	P
Forti Fabric	And	T	F	A	P
	W10	T	T	A	P
	iOS	T	F	A	P
Array MotionPro	And	F	F	–	P
	iOS	F	F	–	P
Array MotionPro Plus	iOS	F	F	–	P
SonicWall MobConn	Mac	F	T	–	P
	And	F	T	–	P
	iOS	F	T	–	P
SonicWall ConnTunn	Mac	F	T	–	P
	W10	F	T	–	P
SonicWall NetExt	W10	F	T	–	P
CP EndSec	Mac	F	T	–	U
	W10	F	T	–	U
CP CapConn	And	F	T$^{\triangledown}$	–	R
	iOS	F	T$^{\triangledown}$	–	R
CP CapWork	And	F	T$^{\triangledown}$	–	R
	iOS	F	T$^{\triangledown}$	–	R
Huawei Seco	W10	T	F	A	P
	Mac	T	F	A	P
UniVPN	W10	T	F	A	P
	Mac	T	F	A	P
	And	T!	F	A	P
	iOS	T!	F	A	P
SoftEther (RADIUS)	W10	T!	T	A	P
(Default)	W10	T!	T	A	M or D
WatchGuard	W10	F	T	–	P
	Mac	F	T	–	P
ZyXEL	W10	F	T	–	P
	Mac	F	T	–	P
Barracuda CloudGen	W10	F	F	–	P
	MacOS	F	T	–	P
	And	F	F	–	P
	iOS	F	F	–	P
OpenVPN (bootstrapping)	Mac	F	T	–	R
	W10 (Connect)	F	T	–	R
	And	F	T	–	R
	iOS	F	T	–	R

CertChkOpt = certificate validation is a configurable option (can be enabled/disabled)
SuffInf = sufficient information available for manual certificate validation
CertDisBehav = behavior when the certificate validation option is disabled

! cert validation disabled by default ▽ common name + RFC1751 w/ truncated SHA1
⊥ might encrypt credentials depending on policy file fetched from gateway, which can be modified by the MITM to disable £ through a parameter of the server URL
ㅅ depends on whether the back-end performs authentications itself or relies on SSO

We then attempt to trigger software updates by changing the version numbers exchanged, and reverse engineer the ways of injecting malicious payloads. As can be seen from the table, mobile platforms are generally less susceptible to this, which we attribute to the fact that many vendors rely on official app stores for distributing and updating their apps. Also notice that not all the identified apps are indeed vulnerable to RCE. For some, we were not able to trigger software updates despite modifying the version numbers exchanged (*e.g.*, MotionPro macOS and Qianxin Android). Some might only be vulnerable to Drive-by Install or Download, and some might have additional verification to reject untrusted payloads,

even if users accept untrusted certificates when prompted during software updates (*e.g.*, F5 Big-IP Edge). To demonstrate RCE vulnerabilities, we wrote some simple programs as the attack payload, and fit them in the different package formats expected by the apps. Our payload writes files to certain directories so that we can determine if it gets executed with root (macOS) or administrator (Windows) privilege. We also purchased an individual code signing certificate from a commercial CA (Sectigo), in case the payloads need to be signed. We now discuss the findings of the RCE test.

For AhnLab, Huawei Seco, and V5 on Windows, after determining update is available, the apps fetch from their back-end a collection of files through TLS sessions that can be intercepted silently. As such, the MITM can easily replace the installer files with any preferred attack payloads. The payload does not need to be signed, and will be *automatically* run with administrator privilege without the Windows UAC prompt. The EnAgent app on Windows has a similar vulnerability, but it always shows the UAC prompt before running the fetched installer (even for genuine updates). Similar vulnerabilities exist in the Ruijie and MotionPro apps on Windows, albeit the attack payload is automatically executed without administrator privilege. For Qianxin on Windows, the app directly fetches app files (libraries and executables) and replace the ones currently installed on the system. It then prompts the user to restart the app. The MITM can intercept and replace those files, but since the app technically did not execute the code automatically, we consider it a case of drive-by install.

For WeGuardia on Windows, we found that it is possible for an MITM to intercept and replace its update installer. Based on our testing, however, the payload needs to be signed. Using our code signing certificate, we signed our payload program and then it ran successfully with administrator privilege. This suggests that despite checking the signature of the update installer, the WeGuardia implementation uses an unnecessarily wide trust anchor and does not check that the code signer indeed has the correct identity. This is particularly interesting as the VPN log-in part of WeGuardia is not found to be susceptible to credential theft in our previous test. For DPTech on Windows, although the MITM can replace the update installer with a malicious payload, the payload fails to run as the app complains about signature being illegal. We tried signing the payload with our code signing certificate and using the Wireshark installer (which comes signed) as the attack payload, but all failed to run. We suspect that the app might have key pinning in checking the installer signature, and thus do not consider it vulnerable in this case.

The Huawei Seco app on macOS fetches a tar ball containing a `.pkg` file during update, which the MITM can intercept and replace. The attack `.pkg` does not need to be signed, and its `preinstall` script will be run automatically with root privilege. For V5 on macOS, when updates happen, the app downloads a `.dmg` file and automatically mounts it. The MITM can replace this file, but since the system only mounts

Table 3: RCE and drive-by attack via in-app updates

App	Platform	Vuln?	Root?	App	Platform	Vuln?	Root?
EnAgent	Windows	RCE	No	Qianxin	Windows	DI	—
	Mac	No	—		iOS	No	—
DPTech	Windows	No	—		Android	No	—
	Mac	No	—	AhnLab	Windows	RCE	Yes
	Android	DI	—	Huawei Seco	Windows	RCE	Yes
	iOS	DI	—		Mac	RCE	Yes
V5	Windows	RCE	Yes	MotionPro	Windows	RCE	No
	Mac	DD	—		Mac	No	—
F5 Big-IP Edge	Windows	No	—	Sangfor	Windows	No	—
	Mac	No	—		Mac	No	—
Ruijie	Windows	RCE	No	WeGuardia	Windows	RCE	Yes
UniVPN	Windows	RCE	Yes	UniVPN	Mac	RCE	Yes

Table 4: VPN front-ends vulnerable due to name checking

app(mode)	OS	Out.	app(mode)	OS	Out.
Huawei Seco	W10	P	OpenVPN (Bootstrapping)	W10	R
				Mac	R

it without automatically installing or executing code, we consider it a case of drive-by download, though we suspect the user could be tricked into installing from the malicious `.dmg` file. Finally, for DPTech on Android and iOS, the attacker can intercept and replace the app package files (`.apk` for Android, `.ipa` for iOS) downloaded during update, which will then be installed on the victim device. For iOS, the `.ipa` file needs to be signed by a valid code signing certificate. For Android, the DPTech VPN app needs to be allowed as a source for installing unknown apps (also needed for genuine updates). Since the malicious apps are not executed automatically afterwards, we consider them cases of drive-by install.

(ii) trusted certificate with incorrect name For this test, we use a server certificate that we purchased from a commercial CA (Sectigo), for a toy domain that we control. We included this test case because previous works showed that some software systems, despite checking the validity of the server certificate chain, neglect to verify that the names on the certificate indeed match the expected identity of the server [27, 29, 33]. We performed this test on apps that are not reported as vulnerable in Table 1. To our surprise, we found three apps, shown in Table 4, to exhibit additional vulnerabilities to MITM under this test case. In the previous test, the Huawei Seco app on Windows, as well as the OpenVPN Connect apps on macOS and Windows all had a behavior of prompting the user to decide whether to accept the untrusted certificate. Interestingly, under this test case, those three apps no longer display any prompts and directly continue with the intercepted TLS, which suggests they have the aforementioned classic flaw of neglecting hostname verification. A MITM can thus easily mount an attack with certificates purchased from a commonly trusted CA to stealthily steal user passwords. For Huawei Seco, the password is sent directly inside the TLS (Out = P), and for OpenVPN Connect, the username and password are base64 encoded (Out = R) and sent as part of the HTTP header

(following the basic HTTP authentication scheme [51]).

4.2 Setup guides prescribed to users ⟨P2⟩

IPSec-PKI on Android. We first revisit the 4 PKI-based IPSec setup guides from Section 3.2, and found that all of them instruct users to leave the server certificate unverified, leaving them vulnerable to credential theft.

TLS-based VPNs. We then evaluate the setup guides of TLS-based VPNs. In particular, we wanted to better understand (i) the spread of vulnerable front-end apps (those in Table 1 or susceptible to RCE in Table 3), and (ii) whether users are taught to enforce certificate validation on apps that rely on human decisions (apps in Table 2). We also collect VPN gateway addresses for measurement in Section 4.3.

(i) Spread. Based on the setup guides we found, 306 academic units use Sangfor EasyConnect, and 99.7% of them are in China. For the other vulnerable VPN front-ends, their adoption in our data set ranges from single digit to tens of academic units. For instance, 17 schools in Korea use WeGuardia. Additionally, we found evidence that some vulnerable VPNs are also used by non-academic organizations. For example, Qianxin is also used by the Bank of Communications and SINOPEC in China, and Array MotionPro is recommended by IBM Cloud service [11]. These suggest the vulnerabilities can indeed be very lucrative to IABs.

(ii) Problematic setup guides. Based on the default settings and behaviors we determined in Table 2, we classify each setup guide into one of four possible outcomes: `Unknown`, `Insecure`, `User Insecure`, `Secure`. `Insecure` outcome could be due to not enabling certificate validation or instructing users to accept any certificates when prompted. `User Insecure` is a notion used in previous work [14], which we adopted to mean a setup guide does not explicitly instruct the user on how to handle the prompts of untrusted certificate, and users might perceivably accept without understanding the implications. `Unknown` refers to the case where an instruction for the app on a particular OS concerned is not found. Lastly, a setup guide is considered `Secure` if it leads to either programmatic or manual rejection of untrusted certificates.

In the end, we generated 6210 entries for all the academic units, OSes, and apps considered. Excluding `Unknown` (53.8%), `User Insecure` (24.4%) is the most common outcome observed. This suggests that IT admins often only focus on the benign case and neglect to educate users on proper *exception handling*. The good news is that `Insecure` occupies the lowest percentage (2.7%) among all the outcomes, though it is somewhat skewed by the Cisco apps, which we partly attribute to its robust default settings and behavior. A breakdown of outcomes by apps can be found in Table 5. Though not a lot of schools prescribe insecure setup guides, our findings suggest that credential theft attacks can still be profitable when attackers target specific schools.

Table 5: Security outcome of VPN setup guides (by apps)

VPN apps	Total (Perc.)	Security Grading	Count	Percentage
ZyXEL	2 (0.0323%)	Insecure	2	100%
WatchGuard	6 (0.0969%)	User Insecure	4	66.7%
SoftEther (RADIUS)	7 (0.113%)	User Insecure	1	14.3%
		Insecure	5	71.4%
SonicWall NetExt	24 (0.388%)	Insecure	1	4.17%
		User Insecure	12	50.0%
Aruba (fetch profile)	24 (0.388%)	User Insecure	13	54.2%
Check Point (Remote Access)	26 (0.42%)	Insecure	5	19.2%
		User Insecure	13	50.0%
Check Point Capsule Connect	26 (0.42%)	Insecure	2	7.69%
		User Insecure	4	15.4%
F5	32 (0.517%)	User Insecure	4	12.5%
F5 (SSO)	32 (0.517%)	User Insecure	8	25.0%
F5 BIG-IP Edge	32 (0.517%)	User Insecure	18	56.2%
SonicWall MobConn	48 (0.775%)	User Insecure	15	31.2%
SonicWall ConnTunn	48 (0.775%)	User Insecure	1	2.08%
Array MotionPro	48 (0.775%)	Insecure	5	10.4%
		User Insecure	25	52.1%
OpenVPN (fetch profile)	251 (4.05%)	User Insecure	30	12.0%
Pulse	588 (9.5%)	Insecure	7	1.19%
		User Insecure	358	60.9%
Forti new	634 (10.2%)	Insecure	38	5.99%
		User Insecure	237	37.4%
Forti old	844 (13.6%)	Insecure	52	6.16%
		User Insecure	62	7.35%
GlobalProtect (Bootstrapping)	1295 (20.9%)	Insecure	25	1.93%
		User Insecure	719	55.5%
Cisco	1964 (31.7%)	User Insecure	1	0.0509%
		Insecure	25	1.27%
		Secure	1185	60.3%

4.3 Configuration issues on back-ends ⟨P3⟩

Given the VPN gateways collected in Section 4.2, we can look for potential issues in server-side configurations. For this we use `TLS-Scanner` [3] and other open-source tools.

TLS-Scanner. We noticed that `TLS-Scanner` can initiate hundreds of connections to the server during each scan. To avoid inducing a significant load on each gateway, we configured `TLS-Scanner` to only use one connection at any time. We also note that there are possible compatibility issues between `TLS-Scanner` and some intolerant servers. For example, to determine whether a server supports a certain TLS version, `TLS-Scanner` might send a `Client Hello` message that the server is not willing to proceed, despite the latter actually supports the TLS version of concern. Upon closer inspection, we suspect this might be due to the initial `Client Hello` message having a long ciphersuites list containing 327 ciphersuites, which some gateways, especially the ones by GlobalProtect, refuse to handle. Because of this, `TLS-Scanner` could occasionally run into *false negatives*, incorrectly claiming a server does not support certain versions of TLS, and fails to collect the server certificate. Nevertheless, 1727 `TLS-Scanner` scans were at least partially successful, covering 1545 unique domains. If one hostname resolves to multiple IP addresses, we scan all of them. in this exercise.

The first interesting observation is that the scores reported by `TLS-Scanner`, which quantify and approximate the TLS

security level of the server (the higher the better), tend to be relatively low. Across all the successful scans, the mean score is 71 ($\sigma = 873, \min = -5250, \max = 3000, \text{median} = 150$). While the exact score scheme can be found on [4], here we highlight some of the recurring scoring criteria. For example, if a server supports TLS 1.3, the score will increase by 500, otherwise 50 points will be deducted. If a server supports TLS 1.0, 100 points will be deducted, otherwise 50 points will be added. We found that a large portion of servers lost points because of criteria (which is consistent with our later experiments presented in Table 7). Additionally, susceptibility to certain vulnerability could lead to `TLS-Scanner` imposing a score cap, further limiting the scores. For example, if a server is vulnerable to the Bleichenbacher's attack, in addition to deducting 800 points, a score cap of 500 is also imposed (the final score can at most be 500). This score cap affects more than half of the servers we scanned. Although one can argue that the scoring metric might be subjective, nevertheless the relatively low average suggest that many VPN gateways are not exhibiting good hygiene.

`TLS-Scanner` probes for a number of vulnerabilities related to TLS on the server side, such as various padding and compression oracles. Table 6 summarizes the vulnerabilities found by `TLS-Scanner`. Due to space constraints, only the 4 vendors contributing the most data points are shown. We note that the impact of a vulnerability might be limited by how the TLS is used. For example, if a vulnerability is only able to decrypt some TLS packets after a large amount of TLS traffic, then it might not apply to short-lived TLS sessions (*e.g.*, the ones used for bootstrapping). We also note that while some vulnerabilities might appear prominent, whether they are directly exploitable depends on other deployment practices and the actual TLS parameters proposed or chosen by the (vendor) specific front-ends. For example, CBC padding oracles present a threat if the client and server agree to use a ciphersuite with a block cipher in CBC mode. Even though the server might support such ciphersuites, however, if the client front-end never proposes such ciphers in the handshake, the padding oracle might not be exploitable. To better gauge the exploitability, we also collect the actual session parameters agreed by front-ends and gateways, and then cross-reference with the `TLS-Scanner` findings.

TLS Session Parameters. Since UI automation on different OSes can be tricky, and manually driving each app to probe the gateways is time-consuming, we instead emulate this by first collecting the `Client Hello` message in the TLS handshake sent by each front-end app, and then replay it to the gateways to obtain the `Server Hello` message, which allows us to better understand the session parameters actually used, and to correlate with the vulnerabilities found by `TLS-Scanner`. We first use `tshark` to collect the traffic from the TLS VPN front-ends on all 4 OSes. Then we extract the `Client Hello` message from the traffic. To replay the extracted message we use the packet manipulation tool `scapy` [2]. It is also necessary to modify and correct the Server Name Indication extension of the `Client Hello` before sending it to different gateways and collecting the `Server Hello` . Overall we collected 7991 `Server Hello` messages from 1581 gateways for all the front-end apps and OSes considered.

Table 6: Number of gateways that have a certain vulnerability as reported by `TLS-Scanner` and the correlated number after cross-referencing the `Server Hello` responses

Vendor	Platform†	ALPACA	Bleich. Attack	Padding Oracle	Raccoon	Direct Raccoon	SSL Poodle	TLS Poodle
Cisco	`Scanner`	485	421	74	0	3	5	17
	Windows	/	46	9	0	1	0	6
	Mac	/	46	9	0	1	0	6
	Android	/	46	9	0	1	0	6
	iOS	/	45	9	0	1	0	6
Pulse	`Scanner`	134	112	3	0	0	10	3
	Windows	/	102	3	0	0	0	3
	Mac	/	100	3	0	0	0	3
	Android	/	102	3	0	0	0	3
	iOS	/	102	3	0	0	0	3
Forti (new)	`Scanner`	192	188	2	0	0	4	1
	Windows	/	0	2	0	0	0	2
	Mac	/	0	2	0	0	0	2
	Android	/	0	2	0	0	0	2
	iOS	/	0	2	0	0	0	2
Sangfor	`Scanner`	281	273	0	0	0	7	0
	Windows	/	0	0	0	0	0	0
	Mac	/	3	0	0	0	0	0
	Android	/	3	0	0	0	0	0
	iOS	/	0	0	0	0	0	0

†: `Scanner` shows the number of gateways that have the vulnerability detected by `TLS-Scanner`, while the rest show the number of gateway-app pairs that choose TLS parameters which make the `TLS-Scanner` reported vulnerability applicable.

Ciphersuites. Out of the 7991 `Server Hello` messages, 23.1% chose RSA for key exchange, and 11.5% use CBC mode of encryption, both are not recommended due to various padding oracle attacks [52, 54]. Additionally, 38 handshakes chose the ciphersuite `TLS_RSA_WITH_3DES_EDE_CBC_SHA` and 29 handshakes chose `TLS_RSA_WITH_RC4_128_SHA` or `TLS_RSA_WITH_RC4_128_MD5` ciphersuites. The first is weak against the SWEET32 attack [15] in part due to the small block size of 3DES, and the latter two are insecure due to various weaknesses of RC4 and are prohibited by standard [47].

TLS Version. In addition to the actual version being negotiated by the VPN client and the gateway, we also collect all the TLS versions supported by gateways via enumerating version-specific `Client Hello` messages. The result is summarized in Table 7. While most gateways support and *choose* TLS 1.2 or TLS 1.3, some gateways still support SSLv2 and SSLv3, and SSLv3 is still chosen in certain cases. While the shift to newer versions of TLS is very promising, continued support of old versions, as well as the unpatched vulnerabilities found by `TLS-Scanner` suggest that some gateways are lacking behind in terms of software updates and maintenance.

TLS Vulnerabilities. Here we discuss the TLS vulnerabilities based on the combined view of the two measurement attempts. Due to space constraints, the less frequently occurring vulnerabilities are discussed in Appendix C.

Table 7: TLS versions chosen and supported

		SSLv2	SSLv3	TLSv1	TLSv1.1	TLSv1.2	TLSv1.3
Default‡	**Count**	0	4	245	0	6513	1229
Total = 7991	**Percentage**	0%	0.1%	3.1%	0%	81.5%	15.4%
Supported‡	**Count**	3	94	851	1050	1515	246
Total = 1581	**Percentage**	0.2%	5.9%	53.8%	66.4%	95.8%	15.6%

‡: *Default* shows the negotiated version by the `Client Hello` replay measurement; *Supported* indicates the versions that the 1581 servers support

Table 8: Certificate Parameters

	Leaf Cert. Stats Uniq. Certs = 1547 Total Certs = 1763		CA Cert. Stats Uniq. Certs = 158 Total Certs = 2957	
Parameter	**Count**	**Percentage**	**Count**	**Percentage**
RSA Public Key = 1024 bits	37	2.4%	7	4.4%
RSA Public Key > 1024 bits	1487	96.1%	143	90.5%
Elliptic-curve Public Key	23	1.5%	8	5.0%
SHA1-RSA Signature	50	3.2%	30	19.0%
SHA256-RSA Signature	1367	88.4%	104	65.8%
SHA384-RSA Signature	103	6.7%	18	11.4%
SHA512-RSA Signature	8	0.5%	1	0.6%
SHA256-ECDSA Signature	14	0.9%	0	0%
SHA384-ECDSA Signature	5	0.3%	5	3.2%
Expired Certificates	61	3.9%	9	5.7%
Version 3 Certificate	1532	99.0%	158	100%
Version 1 Certificate	15	1.0%	0	0%
Chain Verification Success	1113	71.9%	N/A	
Chain Verification Failure	434	28.1%	N/A	

Bleichenbacher's Attack. `TLS-Scanner` checks if the server supports `TLS_RSA` ciphersuites and test if there exists a PKCS#1 v1.5 padding oracle. We then checked if the negotiated ciphersuite after replaying the `Client Hello` is indeed using RSA key exchange. If so, we consider the particular gateway-app pair to be vulnerable to a *direct* Bleichenbacher's attack. While `TLS-Scanner` reported a large number of Bleichenbacher's attack for Cisco gateways (see Table 6), after correlating with the ciphersuite chosen, we found only 46 out of the 421 reported to be a direct threat. Interestingly, things are much worse for Pulse gateways. We found that most (113 out of 146) gateways would choose `TLS_RSA` ciphersuites with Pulse client apps on the 4 OSes, and intersecting this finding with the reports from `TLS-Scanner`, we can see that more than a hundred gateway-app pairs could be susceptible to Bleichenbacher's attack. We note that the Bleichenbacher's attack could lead to a signing oracle [35], though due to the number of queries needed, it remains to be seen if this leads to a practical attack to break server authentication in TLS VPNs.

Padding Oracle. This describes the general CBC-mode padding oracle vulnerabilities, which allow the attacker to decrypt TLS traffic. In this case we check to see if the gateways reported vulnerable by `TLS-Scanner` indeed negotiates CBC-mode ciphersuites with the client. Table 6 shows that very few gateway-app pairs actually choose CBC ciphersuites, so such attacks might only be useful in some targeted cases.

POODLE [42]. For TLS Poodle, CBC-mode ciphers are required. SSL Poodle has the same requirement, and also needs the negotiated TLS version to be SSLv3. Any of the two Poodle variants could allow an attacker to decrypt TLS traffic. For the most popular 4 vendors, we did not observe any gateways that would agree on using SSLv3, thus SSL Poodle is not applicable in this case. For TLS Poodle, some gateway-app pairs do make use of CBC-mode ciphers and are thus considered exploitable.

Server Certificates. Additionally, we also collected the certificates presented by VPN gateways in TLS handshake using OpenSSL `s_client`. We then analyze their quality, since weak certificate parameters may also compromise the security of TLS connections. The results are shown in Table 8, where we collected 1547 unique (1763 total) leaf certificates and 158 unique (2957 total) CA certificates. Although we see 2.4% of leaf use short RSA modulus (1024 bits), 3.9% expired, and 3.2% use SHA1-RSA signature which may be susceptible to collision attacks [38], comparing with previous work on enterprise Wi-Fi [33], the overall quality of certificates for VPN gateways appears slightly better.

Chain Verification. We tried to verify the obtained X.509 certificate chain using `OpenSSL` and the default CA bundle on Ubuntu as the trust anchor. We found that 28.1% of the certificate chains failed to verify due to various reasons. Here we reference the `OpenSSL` error code, and found that 41.9% (182) of the failed chains are missing some issuer certificates (code 20), 37.1% (161) only contain a single self-signed certificate (code 18), and 11.5% (50) has a self-signed root that is not trusted (code 19). Correlating with the results of Section 4.2, we found that 59.3% of the gateways of `Insecure` setup guides have a certificate chain that failed to verify. Conversely, about 40% of the `Insecure` setup guides actually have chains that can verify, and thus it is unnecessary and inexcusable to instruct users to accept arbitrary certificates. Here we also remark that 30% (130) of the certificate chains that fail to verify can be attributed to Sangfor gateways, and 83.0% (108) of them use a self-signed certificate with the same subject name (`C=CN, ST=guangdong, L=shenzhen, O=sangfor, CN=sslvpn`) but different RSA moduli.

Lifespan. We also analyzed the lifespan of the certificates collected. 88.0% of the leaf and 77.2% of the CA certificates have typical lifespans (leaf certificates being at most 5 years and CA for 5-20 years). However, we saw 4.7% of leaf having a lifespan of more than 20 years, and one certificate (which is self-signed) has a negative lifespan of -116 years.

Suspected Key Reuse. Finally, we also investigate the key reuse problem. It is known that software vendors might ship the same private key to different customers [23], and IT admins sometimes keep and use those default keys [33]. After filtering out cases of school alliances and legitimate wildcard names on certificates, we found five cases that might be reusing keys. In the first four cases, we found multiple schools using the same *generically named* certificates: (i) `(C=CN, CN=SWServer)`, (ii) `(C=CN, ST=zhejiang, L=hangzhou,`

O=H3C Hangzhou, OU=ts-security, CN=H3C_local),
(iii) (C=CN, O=Venus, OU=Venus VSG, CN=GateWay),
(iv) (C=CN, ST=BJ, L=BJ, O=HW, OU=VPN, CN=svn).
Based on their subject names, these certificates appear very likely to be originated from the gateway vendors. Finally, there is an interesting case of school A using an expired certificate of school B, and the current trusted certificate school B, issued by a commercial CA (DigiCert), has the exact same RSA modulus. In this case, school A and B can impersonate the VPN gateways of each other. Additionally, since the current trusted certificate school B has a wildcard name for all its subdomains, school A can also impersonate all other websites of school B with its private key.

Insecure SSO redirect. Finally, we also found that some SSO setups incorrectly redirect users to a login page that uses HTTP (without TLS). If users attempt to login, a passive MITM could easily observe the HTTP messages, possibly including user credentials. To investigate, we used the browser automation tool `selenium` to follow all the redirects from gateways that respond to HTTP requests. Redirects can happen by HTTP status code 3XX, by HTML `<meta>` tags, or by JavaScript. In the end we found 16 gateways that redirect users to an HTTP SSO login page. All 16 are from Chinese schools that use Sangfor VPNs. We believe this represents another back-end configuration issue. However, we are unable to confirm whether this is due to default settings, and whether the Central Authentication Service (CAS) that implements the SSO login page is also from the same vendor.

5 Ethical Matters and Discussions

Research ethics. Here we address some ethical concerns of our methodologies. We note that it is not unusual to programmatically query and collect results from search engines (as done in Section 3.2). For instance, it was used by [33] to locate Wi-Fi configuration guides. Additionally, it is also commonly used to facilitate the discovery and analysis of malicious websites and online scams [34, 48, 55]. Similar to previous work, we used Google as an oracle for discovering VPN setup guides. We manually inspected the top results of each query. We also performed manual searches on some school websites to locate setup guides. However, many schools lack a reliable search function on their homepage.

Production servers are often probed for discovering vulnerabilities (see for example [16, 32, 46]). In particular, TLS-Scanner has been used to evaluate the security of medical websites [53]. For probing live VPN gateways (Section 4.3), we minimize the server workload due to TLS-Scanner by configuring the tool to use only 1 connection at any given time. VPN back-ends are typically capable of handling many concurrent users, and thus the workload induced by this should be negligible. We also note that TLS-Scanner measures the possibility of an attack (e.g., existence of timing differentials) using its own TLS sessions, and doesn't mount a full attack on other TLS sessions. Thus our use of TLS-Scanner does not compromise the security of other VPN users.

Responsible disclosure. Google confirmed our report of the PPTP with MPPE issue on Android (CVE-2022-20145), and gave us a bug bounty. Additionally, we disclosed the MITM vulnerabilities (credential theft and RCE) to all the affected equipment vendors. Table 9 shows the latest status at the time of writing. While most have confirmed the vulnerabilities, some are still investigating at the moment. For some that did not respond to emails, we had to make IDD calls and send letters by post. We also encountered unwillingness from some vendors to provide technical support to non-customers, and based on our interactions, some seem to care much more about vulnerabilities affecting gateways than front-ends. For a few vendors, we had to show them demo videos to convince them. Aruba fixed the vulnerability in its Windows 10 app (CVE-2022-23678) in a recent release, though it is unclear to us if they have any plans for the hardcoded 3DES. Leagsoft released a new version of UniVPN to address the RCE vulnerability, which shows a prompt to user when an untrusted server certificate is encountered in the update channel, effectively an upgrade from `Insecure` to `User Insecure`. Whether this fix is adequate remains to be seen.

To our surprise, 2 vendors, Sangfor and Qianxin, confirmed our report of missing certificate validation in their front-ends, but said that they will not fix it. Sangfor in particular claimed (translated from Chinese): *"We don't think this is a vulnerability. MITM hijack and traffic manipulation is a generic attack known to the community, and we will not fix this problem."*

We also sent 235 emails to academic institutions concerning the issues in their VPN setup guides and gateway configurations, but unfortunately the rate of positive response (excluding replies that merely acknowledge receipt of email) is low. Among the issues being reported, 87 emails were sent to those who have PSK exposed, and we received 11 positive responses. For insecure ⟨`SA2`⟩ VPN setup guides, we sent 108 emails but only received 2 positive responses.

Limitations, threats to validity, and reproducibility. We believe the VPN protocols considered in this study to be comprehensive and representative, as we covered those natively supported by major OSes and other popular ones used by real schools. We note that our app discovery is limited to publicly accessible setup guides, and there could be other VPN front-ends (and protocols) used by some organizations but evaded our analysis. For app selection, our order of preference is official app store > vendor homepage > school homepage > school gateway. The latter two might not always yield the latest versions, but they at least represent what are being used by some schools. For simplicity, we only consider one version of each front-end, and some historical versions might behave differently from what we observed. Moreover, we only targeted a few gateways in testing the front-end behav-

Table 9: Disclosure status of MITM vulnerabilities as of October 4, 2022.

Vendor	ACK	Investigating	Confirmed	Fixed
Aruba	Y	Y	Y	Fix released
Fortinet	Y	Y	Y	Fix planned
OpenVPN	Y	Y		
Sangfor	Y	Y	Y	Won't Fix
WeGuardia	Y	Y	Y	
Top	Y	(◇)		
Qianxin (log-in)	Y	Y	Y	Won't Fix
Qianxin (RCE)	Y	Y	Y	
MotionPro	Y	Y	Y	
H3C	Y	Y	Y	
V5	Y	Y		
Ruijie	Y	Y	Y	Fix planned
Enlink	Y	Y	Y	Fix planned
Ahnlab	Y	Y	Y	Fix planned
Huawei	Y	EoS		
Leagsoft	Y	Y	Y	Fix released
Hillstone	Y	Y	Y	Fix planned
DPTech	Y	Y		

(◇): vendor promised to follow-up through emails, but we have yet to hear from them
EoS: End of Support (for both AnyOffice and Seco)

iors, and in theory an app can change its behavior for specific gateways. We also acknowledge that setup guides in natural languages are subject to ambiguity. To improve interpretation consistency, two of the authors were in charge of resolving ambiguity encountered occasionally in the setup guides.

One might also question whether the use of second factor authentication (2FA) can thwart the attacks discussed in the paper. In our experience, some 2FA deployments that rely on passcodes, often piggyback the passcode with user password and transmit both together via the original communication channel (*e.g.*, IPSec and TLS), so that the 2FA can be retrofitted into existing systems. In such cases, compromising the secure channel would allow the attacker to also obtain the passcode, rendering 2FA moot. Since not all schools adopt 2FA and their 2FA policies are not always clear to us, we did not consider 2FA in our study.

Finally, we refrain from publicly releasing our artifacts, which include the impersonation setups, an archive of VPN front-end installers and setup guides, and a list of VPN gateways probed, due to concerns of copyright and sensitivity (as some schools have yet to fix their setups). However, we are willing to share with fellow researchers upon request.

6 Related Work

Given the importance and prominence of VPNs, recent work investigated different aspects of the security and privacy issues of VPNs, with much focus being put on personal (commercial) VPNs. Our effort can thus be seen as an orthogonal attempt to shed light on the relatively less scrutinized academic and organizational VPNs, and some of our findings echo with those reported in previous work. For example, many Android personal VPN apps based on OpenVPN are found to be vulnerable to MITM attacks due to issues in distributing configuration profiles and misconfiguration on the client side [58]. Other commercial VPNs have also been analyzed and found to suffer from similar issues of misconfiguration, insecure default settings, and poor setup instructions [19]. Researchers also empirically measured some commercial VPNs and found a multitude of issues, such as traffic leakage and reliance on vantage points in countries different from those advertised [36]. Recently, some researchers developed a tool to systematically test commercial desktop VPNs, and found various privacy and security concerns (*e.g.*, DNS leaks and malfunctioning "kill switch") [50]. Further on the privacy of VPN, it has been shown that fingerprinting OpenVPN-based services is possible, which enables state-level VPN blocking [57]. On IPsec IKE, researchers found ways to exploit the Bleichenbacher padding oracle, which is particularly damaging when the same RSA key pair is reused across different versions and modes of IKE [28].

On the other hand, implementing and deploying proper certificate validation is a classic problem in the security community. Previous work found poor implementations of certificate validation in various apps and TLS libraries [18,20,23,25,29]. In particular, many Android apps across categories were found to perform only weak or even no certificate validation [21,24,27,43,49], making them vulnerable to MITM attacks. Additionally, many client-side setups of enterprise Wi-Fi are configured to skip crucial certificate checks [13,14,33], opening doors to credential theft by the so-called evil twin attack. Due to difficulties in automating the front-end UIs, we refrain from using large scale testing to discover fine-grained implementation issues in certificate validation, such as mishandling of fields and extensions [18,20,25], and focus on a handful of high-level but critical test cases.

7 Conclusion

In this paper we present a comprehensive review of academic VPNs. Our study shows that many classic implementation and configuration issues can still be found in the academic VPN ecosystem, and as such, it is not difficult to imagine why credential theft and IABs proliferate. To improve the security of their VPNs, organizations are strongly recommended to make plans to retire their ⟨SA0⟩ and ⟨SA1⟩ setups, and move to PKI-based solutions. For ⟨SA2⟩ VPNs, it is critical that the front-ends enforce proper certificate validation for both VPN connect and software updates, and users should be instructed to make sure such enforcement will indeed happen. This requires both vendors and IT admins to pay close attentions to the implementation and deployment of the underlying protocols. Finally, the VPN gateways should be updated frequently to avoid known vulnerabilities, and certificates need to be properly maintained to facilitate validation.

Acknowledgments

We thank the anonymous reviewers and shepherd for helping us improve the overall quality of our paper. This work was supported in part by a grant from the Research Grants Council (RGC) of Hong Kong (Project No.: CUHK 24205021), Project Impact Enhancement Fund 3133292C and Direct Grant 4055125 from CUHK, as well as grants from the CUHK IE department (project code: NEW/SYC, GRF/20/SYC, and GRF/21/SYC).

References

[1] IKEv1 Aggressive Mode with PSK Authentication. https://docs.strongswan.org/docs/5.9/howtos/securityRecommendations.html#_ikev1_aggressive_mode_with_psk_authentication.

[2] Scapy. https://scapy.net/.

[3] Tls-scanner. https://github.com/tls-attacker/TLS-Scanner.

[4] Tls-scanner scoring criteria. https://github.com/tls-attacker/TLS-Scanner/blob/master/TLS-Server-Scanner/src/main/resources/rating/influencers.xml.

[5] GmSSL - 支持国密SM2/SM3/SM4/SM9/SSL的密码工具箱, 2017-2022. https://github.com/guanzhi/GmSSL.

[6] RDP and VPN use skyrocketed since coronavirus onset, 2020. https://www.zdnet.com/article/rdp-and-vpn-use-skyrocketed-since-coronavirus-onset/.

[7] All Access Pass: Five Trends with Initial Access Brokers, 2021. https://ke-la.com/all-access-pass-five-trends-with-initial-access-brokers/.

[8] Hackers Breached Colonial Pipeline Using Compromised Password, 2021. https://www.bloomberg.com/news/articles/2021-06-04/hackers-breached-colonial-pipeline-using-compromised-password.

[9] Initial Access Brokers Report, 2021. https://resources.digitalshadows.com/whitepapers-and-reports/initial-access-brokers-report.

[10] Investigating the Emerging Access-as-a-Service Market, 2021. https://www.trendmicro.com/vinfo/de/security/news/cybercrime-and-digital-threats/investigating-the-emerging-access-as-a-service-market.

[11] Connecting to ssl vpn from motionpro clients (windows, linux, and mac os x) | ibm cloud docs, 2022. https://cloud.ibm.com/docs/iaas-vpn?topic=iaas-vpn-standalone-vpn-clients.

[12] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. DROWN: Breaking TLS with SSLv2. In *USENIX Security*, 2016.

[13] Alberto Bartoli, Eric Medvet, Andrea De Lorenzo, and Fabiano Tarlao. (in) secure configuration practices of wpa2 enterprise supplicants. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pages 1–6, 2018.

[14] Alberto Bartoli, Eric Medvet, and Filippo Onesti. Evil twins and wpa2 enterprise: A coming security disaster? *Computers & Security*, 74:1–11, 2018.

[15] Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-)security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In *ACM CCS*, 2016.

[16] Hanno Böck, Juraj Somorovsky, and Craig Young. Return Of Bleichenbacher's Oracle Threat (ROBOT). In *USENIX Security*, 2018.

[17] Marcus Brinkmann, Christian Dresen, Robert Merget, Damian Poddebniak, Jens Müller, Juraj Somorovsky, Jörg Schwenk, and Sebastian Schinzel. ALPACA: Application layer protocol confusion - analyzing and mitigating cracks in TLS authentication. In *USENIX Security*, 2021.

[18] Chad Brubaker, Suman Jana, Baishakhi Ray, Sarfraz Khurshid, and Vitaly Shmatikov. Using frankencerts for automated adversarial testing of certificate validation in ssl/tls implementations. In *IEEE S&P*, 2014.

[19] Thanh Bui, Siddharth Rao, Markku Antikainen, and Tuomas Aura. Client-side vulnerabilities in commercial vpns. In *Nordic Conference on Secure IT Systems*, pages 103–119. Springer, 2019.

[20] Sze Yiu Chau, Omar Chowdhury, Endadul Hoque, Huangyi Ge, Aniket Kate, Cristina Nita-Rotaru, and Ninghui Li. Symcerts: Practical symbolic execution for exposing noncompliance in X.509 certificate validation implementations. In *IEEE S&P*, 2017.

[21] Sze Yiu Chau, Bincheng Wang, Jianxiong Wang, Omar Chowdhury, Aniket Kate, and Ninghui Li. Why Johnny Can't Make Money With His Contents: Pitfalls of Designing and Implementing Content Delivery Apps. In *ACSAC*, 2018.

[22] Aldo Cortesi, Maximilian Hils, Thomas Kriechbaumer, and contributors. mitmproxy: A free and open source interactive HTTPS proxy, 2010–.

[23] X de Carné de Carnavalet and Mohammad Mannan. Killed by proxy: Analyzing client-end tls interception software. In *NDSS*, 2016.

[24] Joyanta Debnath, Sze Yiu Chau, and Omar Chowdhury. When TLS meets proxy on mobile. In *International Conference on Applied Cryptography and Network Security*, 2020.

[25] Joyanta Debnath, Sze Yiu Chau, and Omar Chowdhury. On Re-engineering the X.509 PKI with Executable Specification for Better Implementation Guarantees. In *ACM CCS*, 2021.

[26] P. Eronen (Ed.) and H. Tschofenig (Ed.). Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279 (Proposed Standard), December 2005.

[27] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. Why Eve and Mallory love Android: An analysis of Android SSL (in) security. In *ACM CCS*, 2012.

[28] Dennis Felsch, Martin Grothe, Jörg Schwenk, Adam Czubak, and Marcin Szymanek. The Dangers of Key Reuse: Practical Attacks on IPsec IKE. In *USENIX Security*, 2018.

[29] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: validating ssl certificates in non-browser software. In *ACM CCS*, 2012.

[30] Yoel Gluck, Neal Harris, and Angelo Prado. SSL, gone in 30 seconds - a BREACH beyond CRIME. In *Black Hat USA*, 2013.

[31] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, and G. Zorn. Point-to-Point Tunneling Protocol (PPTP). RFC 2637 (Informational), July 1999.

[32] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *USENIX Security*, 2012.

[33] Man Hong Hue, Joyanta Debnath, Kin Man Leung, Li Li, Mohsen Minaei, M Hammad Mazhar, Kailiang Xian, Endadul Hoque, Omar Chowdhury, and Sze Yiu Chau. All your Credentials are Belong to Us: On Insecure WPA2-Enterprise Configurations. In *ACM CCS*, 2021.

[34] Luca Invernizzi, Paolo Milani Comparetti, Stefano Benvenuti, Christopher Kruegel, Marco Cova, and Giovanni Vigna. Evilseed: A guided approach to finding malicious web pages. In *IEEE S&P*, 2012.

[35] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. On the security of tls 1.3 and quic against weaknesses in pkcs# 1 v1. 5 encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1185–1196, 2015.

[36] Mohammad Taha Khan, Joe DeBlasio, Geoffrey M Voelker, Alex C Snoeren, Chris Kanich, and Narseo Vallina-Rodriguez. An empirical analysis of the commercial vpn ecosystem. In *ACM IMC*, 2018.

[37] Masashi Kikuchi. Ccs injection vulnerability, 2014. http://ccsinjection.lepidum.co.jp/.

[38] Gaëtan Leurent and Thomas Peyrin. From collisions to chosen-prefix collisions application to full sha-1. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 527–555. Springer, 2019.

[39] Moxie Marlinspike. Divide and Conquer: Cracking MS-CHAPv2 with a 100% success rate, 2012. https://web.archive.org/web/20160316174007/https://www.cloudcracker.com/blog/2012/07/29/cracking-ms-chap-v2/.

[40] D. McDonald. A Convention for Human-Readable 128-bit Keys. RFC 1751 (Informational), December 1994.

[41] Robert Merget, Marcus Brinkmann, Nimrod Aviram, Juraj Somorovsky, Johannes Mittmann, and Jörg Schwenk. Raccoon attack: Finding and exploiting Most-Significant-Bit-Oracles in TLS-DH(E). In *USENIX Security*, 2021.

[42] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This poodle bites: exploiting the ssl 3.0 fallback. *Security Advisory*, 21:34–58, 2014.

[43] Marten Oltrogge, Nicolas Huaman, Sabrina Amft, Yasemin Acar, Michael Backes, and Sascha Fahl. Why Eve and Mallory Still Love Android: Revisiting TLS (In)Security in Android Applications. In *USENIX Security*, 2021.

[44] G. Pall and G. Zorn. Microsoft Point-To-Point Encryption (MPPE) Protocol. RFC 3078 (Informational), March 2001.

[45] B. Patel, B. Aboba, W. Dixon, G. Zorn, and S. Booth. Securing L2TP using IPsec. RFC 3193 (Proposed Standard), November 2001.

[46] Damian Poddebniak, Fabian Ising, Hanno Böck, and Sebastian Schinzel. Why TLS is better without STARTTLS: A security analysis of STARTTLS in the email context. In *USENIX Security*, 2021.

[47] A. Popov. Prohibiting RC4 Cipher Suites. RFC 7465 (Proposed Standard), February 2015.

[48] M Zubair Rafique, Tom Van Goethem, Wouter Joosen, Christophe Huygens, and Nick Nikiforakis. It's free for a reason: Exploring the ecosystem of free live streaming services. In *NDSS*, 2016.

[49] Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Murat Kantarcioglu, and Danfeng Yao. Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized java projects. In *ACM CCS*, 2019.

[50] Reethika Ramesh, Leonid Evdokimov, Diwen Xue, and Roya Ensafi. VPNalyzer: Systematic Investigation of the VPN Ecosystem. In *NDSS*, 2022.

[51] J. Reschke. The 'Basic' HTTP Authentication Scheme. RFC 7617 (Proposed Standard), September 2015.

[52] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard), August 2018.

[53] R Röhrig et al. The security state of the german health web: An exploratory study. In *Proceedings of the Joint Conference of the 66th Annual Meeting of the German Association of Medical Informatics, Biometry, and Epidemiology EV (gmds) and the 13th Annual Meeting of the TMF-Technology, Methods, and Infrastructure for Networked Medical*, volume 283, page 180. IOS Press, 2021.

[54] Y. Sheffer, R. Holz, and P. Saint-Andre. Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). RFC 7457 (Informational), February 2015.

[55] Bharat Srinivasan, Athanasios Kountouras, Najmeh Miramirkhani, Monjur Alam, Nick Nikiforakis, Manos Antonakakis, and Mustaque Ahamad. Exposing search and advertisement abuse tactics and infrastructure of technical support scammers. In *Proceedings of the 2018 World Wide Web Conference*, pages 319–328, 2018.

[56] Soner Tari. SSLproxy - transparent SSL/TLS proxy for decrypting and diverting network traffic to other programs for deep SSL inspection, 2017-2022. https://github.com/sonertari/SSLproxy.

[57] Diwen Xue, Reethika Ramesh, Arham Jain, Michalis Kallitsis, J. Alex Halderman, Jedidiah R. Crandall, and Roya Ensafi. OpenVPN is Open to VPN Fingerprinting. In *USENIX Security*, 2022.

[58] Qi Zhang, Juanru Li, Yuanyuan Zhang, Hui Wang, and Dawu Gu. Oh-pwn-vpn! security analysis of openvpn-based android apps. In *International Conference on Cryptology and Network Security*, pages 373–389. Springer, 2017.

[59] G. Zorn. Deriving Keys for use with Microsoft Point-to-Point Encryption (MPPE). RFC 3079 (Informational), March 2001.

A Extra notes on testing TLS-based VPNs

Compatibility issues. Although `mitmproxy` suffices in most cases, there were a few compatibility issues that we had to work around during our testing. The most interesting case concerns the Sangfor EasyConnect app on iOS, which experiences an unexplained loss of connection when the TLS gets intercepted by `mitmproxy`. Finally we modified `SSLproxy` to implement the RSA exponent rewrite attack, and were able to show that the app is similarly vulnerable as its counterparts on other OSes.

Moreover, we also noticed that apps from some Chinese vendors have an alternative mode of operation known as GmSSL, which uses a modified TLS and ciphersuites from certain Chinese national standards, instead of the conventional ones defined for standard TLS. As such, existing setups including the likes of `mitmproxy` simply do not understand and fail to intercept the protocol. Instead, for testing the certificate validation behavior under this mode, we used an open-source implementation of GmSSL [5], which is a modified version of OpenSSL. Specifically, instead of trying to intercept the TLS, we modified the GmSSL version of the `s_server` program, run it as an impersonating VPN back-end, and redirect the VPN traffic to it. This setup allowed us to demonstrate the lack of certificate validation as well as possible credential theft of a few apps under the GmSSL mode, but for the rest we ran into cryptic low-level handshake errors. This experience suggests that implementations of the GmSSL mode is not very stable at the current stage.

Other interesting findings. We have noticed some special designs in some of the apps tested. For Check Point apps, to show the fingerprint of a certificate to the user, they present a series of 12 English words instead of the regular hash digest. Upon investigation we found that the app first computes the SHA1 hash of the certificate, takes the leading 128 bits of the 160-bit digest, and encodes it using the method described in RFC1751, which includes an algorithm to represent 128-bit keys as 12 English words [40]. Since only 128 bits are used and presented to the user, we speculate it might make some forms of hash collision slightly easier, but leave the cryptanalysis for future work. Another interesting finding concerns the mobile versions of the Check Point apps, where we found that the username and passwords are XOR-ed with a static key. We reverse engineered this via simple differential analysis after observing several pairs of plaintexts and ciphertexts.

Finally, we note that for the open-source Softether, the 2 modes of operation would lead to different attack outcome. Under the RADIUS mode, which is common for implementing SSO, the username and password will be sent directly to the server. Under its default mode, however, we noticed that the password cannot be found directly. After reading and testing its source code, we found that the client computes and sends the SHA0(SHA0(username || password) || conn.random) to the server, where conn.random is a nonce chosen by the server and sent earlier in TLS. As such, a successful MITM can observe both the SHA0 hash and server-chosen nonce, and attempt an offline dictionary attack to recover the user password (Out = D). Additionally, since the client contributes no randomness into the computation of the SHA0 hash, it is also possible for an active MITM to attempt real-time login as the user (Out = M). The MITM could obtain a conn.random from the target server first, then when the victim connects, use the exact same conn.random with the victim. Upon receiving the SHA0 hash from the victim, the MITM can then forward it to the server and login successfully as the victim.

B More on FortiClient under SSO mode

For FortiClient, when the user tries to authenticate using SSO, a built-in browser will be invoked to fetch policy files from `<gateway>/remote/info` as mentioned in Section 4.1.2. For Windows and Mac, the application uses one of the policy attributes `remoteauthtimeout` to set a timer for the SSO login, and the browser will be closed after the timeout. We found that the timeout is actually an IEEE 754 floating point type, so after we feed an extremely large number (like `1e10000`), the internal browser tries to decrement the counter, which will turn the timeout to `infinity`, and timeout will never occur. Figure B1 shows the screenshot of FortiClient in SSO mode, with infinite timeout and redirected to other webpages by active man-in-the-middle on Mac. It also demonstrates the phishing attack described in Section 4.1.2, with `http://www.attacker.com/login` instantiated with `http://www.github.com/login`, and `https://attacker.com/login` instantiated with `https://github.com/login`. Figure B2 shows the CA certificate injection possibility discussed in Section 4.1.2.

Additionally, during our testing, we found that some gateways explicitly specify a port for SSO, *e.g.*, `sso_port=8020`, in the response of `<gateway>/remote/info`. And if this specification is present, on Android and iOS, the subsequent request for `<gateway>/remote/saml/start` as well as the actual SSO login page will then happen with an external browser (*e.g.*, Chrome on Android and Safari on iOS) instead of the aforementioned built-in one. The external browser shows the address bar and performs proper certificate validation, which renders the above attacks ineffective. However, since the TLS for requesting and receiving `<gateway>/remote/info` can also be intercepted silently as discussed before, the MITM can simply remove the port specification in `<gateway>/remote/info` to make the Android and iOS apps consistently using their built-in browsers instead of the external ones.

Table A1: VPN front-ends that reject untrusted certificates

app (mode)	OS	CertChkOpt	Prompt	Block
Cisco (SAML)	Mac	T#	T#	T
	W10 (UWP)		NA	
	W10	T#	T#	T
	And	T#	T#	T
	iOS	T#	T#	T
Citrix SSO	Mac	F	F	T
	iOS	F	F	T
Citrix Workspace	Mac	F	F	T
	W10	F	F	T
GlobalProtect (after bootstrapping)	Mac	F	F	T
	W10	F	F	T
	And	F	F	T
	iOS	F	F	T
F5	Mac	F	F	T
	W10	F	F	T
	iOS	F	F	T
F5 (SSO)	Mac	F	F	T
	iOS	F	F	T
Forti new	Mac	Tϕ	Tϕ	Tϕ
Forti Fabric	Mac	Tϕ	Tϕ	Tϕ
Clavister OneConn	Mac	F	F	T
	iOS	F	F	T
	W10	F	F	T
Clavister OneConn Classic	Mac	F	F	T
Forcepoint	W10	F	F	T
	Mac	F	F	T
	And	F	F	T
W10 SSTP	W10	F	F	T
OpenVPN (bootstrapping)	W10 (GUI)	F	F	T
WeGuardia	W10	F	F	T
OpenVPN (after bootstraping)	Mac	F	F	T
	W10 (Connect)	F	F	T
	W10 (GUI)	F	F	T
	And	F	F	T
	iOS	F	F	T

CertChkOpt = certificate validation is a configurable option (can be enabled/disabled)
# has no effect for SSO as the invoked browser enforces its own cert validation
ϕ cert validation enforced by `trustd` on macOS regardless of settings and prompts

C Other TLS vulnerabilities

Here we briefly discuss the other TLS vulnerabilities not covered in Section 4.3:

Raccoon [41]. Raccoon is an attack that allows the attacker to obtain the premaster secret of a TLS session that uses (Ephemeral) Diffie-Hellman key exchange, assuming that the server reuses the private key for a certain period of time.

Figure B1: A screenshot of FortiClient in SSO mode with infinite timeout and redirected to other webpages on Mac

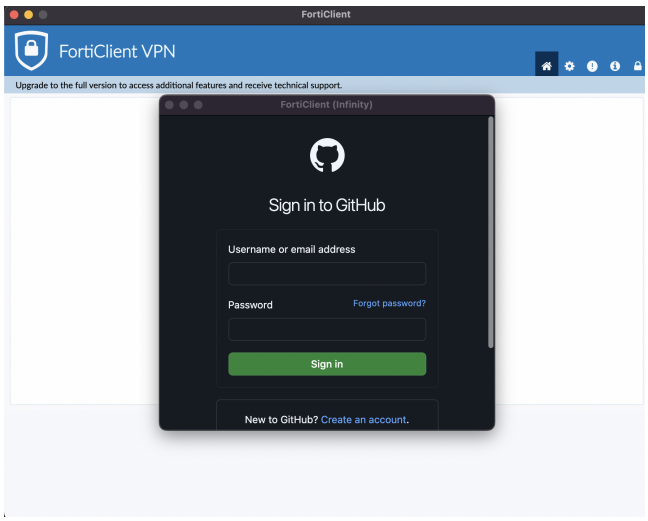
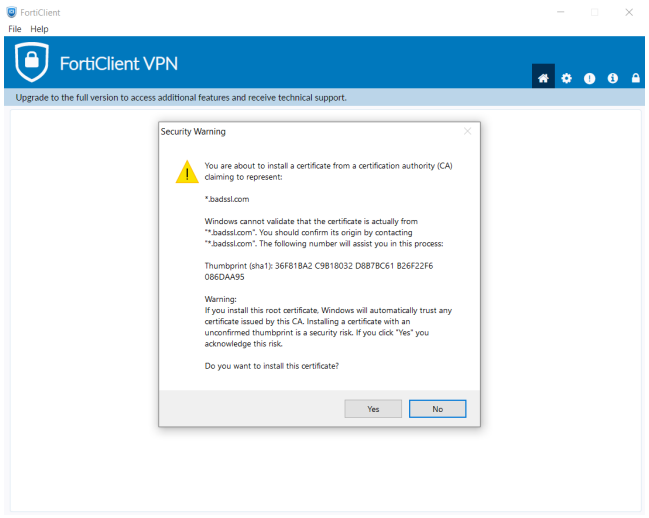


Figure B2: A screenshot of Windows FortiClient in SSO mode prompting the user to import an invalid certificate into the trusted CA store on Windows.



Thus we check if the gateway negotiates `TLS_DHE` or `TLS_DH` ciphersuites with the client. In this case, very few gateways choose DH(E) ciphersuites, which suggests that the Raccoon attack might not be very profitable against VPN setups. To exploit Raccoon, a passive MITM will sniff a TLS session, then interacts with the server to recover the premaster secret of the sniffed session, and finally decrypt traffic of that session.

ALPACA [17]. This requires that the existence of other services (on the same domain) with a certificate subject name that can match the VPN gateway. Since we only considered and measured VPN services, we cannot confirm if the ALPACA attack conditions can be satisfied. This attack requires an active MITM. Even if ALPACA is possible, the precise impact on TLS VPNs depends on the other services that the exploit interacts with.

EarlyCcs [37]. This requires the underlying TLS implementation of the client and the server to be using some old versions of OpenSSL. While it might be possible to fingerprint the client TLS implementation, we refrain from investigating further as the number of affected gateways is small. A successful exploit may result in an undetectable active TLS MITM interception.

CRIME [54]. We have observed 3 cases of CRIME over all the VPN gateways. CRIME requires the client to offer TLS compression, in addition to using CBC-mode ciphers. We manually examined each case and confirmed that none of the corresponding clients propose to use TLS compression. Even when the server is vulnerable, the impact on VPNs depends on how the TLS is being used. For some the impact could be limited, but if HTTP cookies are used and can be stolen, this might enable session hijack.

DROWN [12]. We only observed 1 gateway of Enlink VPN that might be vulnerable to DROWN directly, as it supports the use of SSLv2 and by default, the EnAgent client and server agree on using `TLS_RSA` ciphersuites, which could allow the TLS v1.0 traffic (default choice of the app) to be passively collected and decrypted. However, we note that thousands of handshakes need to be sniffed by the attacker in order to decrypt one of the handshakes, and whether that is practical remains to be seen.

BREACH [30]. This vulnerability requires HTTP compression to be negotiated for both the client and the server. We observed that not all the VPN clients use HTTP inside TLS, and we leave the investigation of clients proposing HTTP compression as future work. Even when the server is vulnerable, the impact on VPNs depends on how the TLS is being used. For some the impact could be limited, but if HTTP cookies are used and can be stolen, this vulnerability could enable session hijack.

D Additional tables

We show the full list of TLS-based VPN apps tested in Section 4.1.2, including their full product names, version numbers, and sources, on https://github.com/vpn-test-2022/VPN-test-2022/blob/main/sslvpn-application-ref.pdf.