All your Credentials are Belong to Us: On Insecure WPA2-Enterprise Configurations

Man Hong Hue
The Chinese University of Hong Kong
hugohue@link.cuhk.edu.hk

Joyanta Debnath
The University of Iowa
joyanta-debnath@uiowa.edu

Kin Man Leung
The University of British Columbia
kmleung@student.ubc.ca

Li Li
Syracuse University
lli101@syr.edu

Mohsen Minaei
Visa Research
mominaei@visa.com

M. Hammad Mazhar
The University of Iowa
muhammadhammad-mazhar@uiowa.edu

Kailiang Xian
The Chinese University of Hong Kong
1155133741@link.cuhk.edu.hk

Endadul Hoque
Syracuse University
enhoque@syr.edu

Omar Chowdhury
The University of Iowa
omar-chowdhury@uiowa.edu

Sze Yiu Chau*
The Chinese University of Hong Kong
sychau@ie.cuhk.edu.hk

ABSTRACT

In this paper, we perform the first multifaceted measurement study to investigate the widespread insecure practices employed by tertiary education institutes (TEIs) around the globe when offering WPA2-Enterprise Wi-Fi services. The security of such services critically hinges on two aspects: (1) the connection *configuration* on the client-side; and (2) the TLS setup on the authentication servers. Weaknesses in either can leave users susceptible to credential theft. Typically, TEIs prescribe to their users either *manual instructions* or pre-configured *profiles* (e.g., eduroam CAT). For studying the security of configurations, we present a framework in which each configuration is mapped to an *abstract security label* drawn from a strict partially ordered set. We first used this framework to evaluate the configurations supported by the user interfaces (UIs) of mainstream operating systems (OSs), and discovered many design weaknesses. We then considered 7045 TEIs in 54 countries/regions, and collected 7275 configuration instructions from 2061 TEIs. Our analysis showed that majority of these instructions lead to insecure configurations, and nearly 86% of those TEIs can suffer from credential thefts on at least one OS. We also analyzed a large corpus of pre-configured eduroam CAT profiles and discovered several misconfiguration issues that can negatively impact security. Finally, we evaluated the TLS parameters used by authentication servers of thousands of TEIs and discovered perilous practices, such as the use of expired certificates, deprecated versions of TLS, weak signature algorithms, and suspected cases of private key reuse among TEIs. Our long list of findings have been responsibly disclosed to the relevant stakeholders, many of which have already been positively acknowledged.

*Corresponding author of this paper.

CCS CONCEPTS

• **Networks** → **Mobile and wireless security**; • **Security and privacy** → *Public key (asymmetric) techniques*; • **Human-centered computing** → *Graphical user interfaces*; • **Social and professional topics** → *Identity theft*.

KEYWORDS

Network Security; Authentication; SSL/TLS; PKI; X.509 Certificate

ACM Reference Format:
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. 2021. All your Credentials are Belong to Us: On Insecure WPA2-Enterprise Configurations . In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3460120.3484569

1 INTRODUCTION

Wi-Fi is one of the cornerstones of digital communication, providing connectivity to a plethora of devices. The IEEE 802.11i Wi-Fi standard is commonly referred to as WPA2 (Wi-Fi Protected Access II) by consumer products. Despite the recent release of WPA3, WPA2 continues to be dominant due to the needs to support pre-existing devices. Authentication in IEEE 802.11i can be achieved through either a pre-shared key (PSK) or IEEE 802.1X. Many organizations including companies and tertiary educational institutions (TEIs) currently rely on IEEE 802.1X authentication (the so-called *WPA2-Enterprise mode*) for providing their users authenticated access to the Internet and other internal resources.

In a typical WPA2-Enterprise setup, a *TLS tunnel* is created, where the back-end authentication server authenticates itself to the user's device using X.509 certificates during the TLS handshake, and then a password-based user authentication happens inside the tunnel. In most cases, the user authentication reuses the same credential used by the user's other services (*e.g.*, email) at the organization through an existing single sign-on (SSO) service (*e.g.*, Microsoft AD FS). While this improves convenience and user experience, it makes WPA2-Enterprise a lucrative target for attacks, such as the "*Evil Twin*" (ET) attack [13, 23], where the adversary sets up a rogue access point to trick users into handing over their SSO credentials. To make matters worse, the ET attack can be easily performed with off-the-shelf hardware and software components costing around USD $100. Weaknesses allowing the ET attack have been observed in real-life. For instance, a recent report found that it was possible to use the ET attack to steal credentials of staffs working at the United States Department of the Interior, leading to access of its network and other internal systems [8].

Goal. In this paper, we investigate potential configuration weaknesses which could enable the ET and other attacks in the WPA2-Enterprise ecosystem, especially among TEIs. For our investigation, we resort to a multifaceted measurement study.

The security of a WPA2-Enterprise Wi-Fi connection relies on the robustness of the following two factors: (1) the user-side (supplicant) *configuration*; (2) the strength of the TLS parameters used by the back-end authentication server. A supplicant configuration can be viewed as assigning values to different connection attributes including server name, certificate authorities (CAs), secondary authentication method, etc. TEIs deliver to their staffs and students OS-specific supplicant configurations by prescribing either (a) step-by-step *manual instructions* that users are expected to follow in the configuration UI of the target OS, or (b) *pre-configured profiles* through installers such as eduroam CAT, which can be downloaded and installed in advance. Depending on the target OS, pre-configured profiles might be able to assign values to certain connection attributes that are otherwise impossible through the UI. In this study, we evaluate the security of WPA2-Enterprise deployment of TEIs by considering aspects (1)(a), (1)(b), and (2).

Approach. In order to enable fair assessment and comparison of different connection configurations induced by manual instructions and pre-configured profiles in a systematic way, we first develop a comparison framework. The main challenge of developing such a framework is to identify aspects of the connection configuration that play a critical part in terms of security. In our framework, each configuration is assigned an *abstract security label*, drawn from a strict partially ordered set of labels in which the relative security acts as the ordering relation. A strict partial order is used because it is not always clear how to order two insecure configurations.

Studying aspect (1)(a). Using our comparison framework, we first evaluate the configuration UIs of mainstream operating systems (OSs) based on their achievable configurations. We observed that many OSs contain subtle design weaknesses that hinder users from achieving a secure WPA2-Enterprise configuration. We reported our findings and recommendations to the corresponding vendors, and received an assortment of responses: some issues got fixed and new CVE IDs assigned, others were dismissed.

After analyzing the configurations supported by different OSs, we then evaluate the quality of manual configuration instructions prescribed by TEIs. We limit ourselves to TEIs because companies seldom make their Wi-Fi configuration instructions publicly accessible. In our large-scale study, we collected 7275 configuration instructions from 2061 TEIs in 54 countries/regions. Using our framework and knowledge of the OS behaviors, we produced 14602 labels for the resulting configurations on mainstream OSs. Our results show a grim state of affairs: majority of the instructions lead to insecure configurations, and 97.2% of TEIs that do not rely on profile installers prescribe at least one insecure instruction for the various OSs considered. In other words, countless credentials around the globe can be easily stolen by the ET attack.

Studying aspect (1)(b). After analyzing the instructions, we turn our attention to TEIs that provide pre-configured eduroam CAT profiles. We collected 3593 CAT configuration profiles for each mainstream OS. Although one can side-step some UI limitations by using pre-configured profiles, we found that some CAT profiles themselves suffer from issues including weak/no server name checking, CA store pollution with large number of certificates, and reliance on CA certificates with weak signature algorithms.

Studying aspect (2). To evaluate the strength of TLS connections induced by the back-end authentication servers, we utilize the roaming nature of *eduroam*, and conduct a large scale measurement study on the various parameters related to the trustworthiness of the TLS tunnel and X.509 certificates. We successfully measure the back-end setups of 3637 domains, and observe that many of them suffer from issues that can hinder proper user configuration and negatively impact security, including the use of deprecated versions of TLS (*e.g.*, 1.0 and 1.1), weak signature algorithms, certificates without meaningful names, expired and extremely long-lived certificates, and suspected cases of key reuse across institutions.

Finding summary. All in all, the results of our multifaceted study draw a dire picture of the WPA2-Enterprise ecosystem. Our findings suggest that the current alarming state of affairs is powered by 3 factors, much like the fire triangle, making deployments susceptible to the ET attack: 1) misguided UI designs on OSs that confuse and frustrate users; 2) low consideration of adversaries when prescribing configuration instructions; 3) subtleties of certificate validation and server name matching are sometimes misconfigured.

Contributions. We present the first comprehensive measurement study evaluating the security of WPA2-Enterprise deployments of TEIs around the globe. Our study consists of the following elements:

(1) A comparison framework for evaluating the security of WPA2-Enterprise supplicant configurations.
(2) A security evaluation of the UIs and achievable configurations of mainstream OSs, identifying design weaknesses that lead to insecure configurations and other attacks.
(3) A measurement study of 7275 instructions from 2061 TEIs, showing 85.7% of TEIs are susceptible to the ET attack.
(4) An evaluation of 3593 CAT profiles for each mainstream OS, revealing several oversights in pre-configured profiles.
(5) A measurement study evaluating the TLS setup on the back-end authentication server of 3637 domains, revealing weak parameters used by many TEIs.

2 TECHNICAL BACKGROUND

Here we give a brief summary of the terminologies relevant to this paper. Readers interested in a thorough review of the IEEE 802.11i and 802.1X standards can consult the literature (*e.g.*, [33] and [34]).

2.1 WPA2-Enterprise, IEEE 802.1X, and EAP

The authentication in WPA2-Enterprise follows the IEEE 802.1X standard, which provides an authentication mechanism by encapsulating the Extensible Authentication Protocol (EAP) over LAN (and WLAN). Three components are defined for the authentication process in 802.1X: *supplicant*, a component of the end-user device that is seeking to join the network; *authenticator* (sometimes known as the "front-end"), a component that is typically a part of the wireless access point; and *authentication server* (sometimes known as the "back-end"), which actually authenticates the user by checking the received credential. The authentication exchange happens logically between the supplicant and the authentication server, with the authenticator relaying messages between the two. Although 802.1X did not specify the type of authentication server to be used, in most cases, the back-end would be a RADIUS server running on a separate system located either in an organization's core network or remotely (as in the case of *eduroam*). RADIUS can integrate with existing identity providers such as Microsoft AD FS and LDAP servers, which enables the reuse of existing SSO credentials for Wi-Fi access.

2.2 EAP methods and TLS tunneling

EAP is a generic authentication framework that does not dictate a particular way of authenticating users. Instead, it enables protocol designers to build their own authentication methods. A variety of EAP methods can be used with 802.1X, but many of them lack inbuilt confidentiality protection for the messages being exchanged, and hence are susceptible to eavesdropping, especially in the wireless setting. One way of retrofitting crytographic protections to such EAP methods is to tunnel them through Transport Layer Security (TLS). In fact, taking advantage of the flexibility of the EAP framework, several EAP methods were proposed to do exactly that, with subtle technical differences.

There are two TLS tunneling proposals that have seen considerable deployments in the wild, namely, *PEAP* and *EAP-TTLS*. Both of their designs are quite similar, with the phase-1 of EAP aimed at establishing a TLS tunnel, and if certificates are not used for client authentication during TLS handshake, the protocol will then go to the so-called *phase-2 authentication* (also known as the inner authentication), where the user authenticates to the server through a different method, typically password-based. This is also how PEAP and EAP-TTLS differ: inside the TLS tunnel, PEAP simply executes a second EAP method, but EAP-TTLS exchanges attribute-value pairs, which allows it to use other non-EAP methods for the inner authentication [33].

2.3 Phase-2 authentication methods

Here we give an overview of some phase-2 authentication methods commonly used with PEAP and EAP-TTLS.

2.3.1 PAP and EAP-GTC. Password Authentication Protocol (PAP) is a simple password-based authentication protocol, and is often used with EAP-TTLS but not PEAP, since PAP itself is not an EAP method. Although it can be configured to transmit an obfuscated password for some deployment scenarios [36, 38], in its simplest form, the user identity and password are transmitted to the authentication server in *cleartext* [33], which is typically what happens when used as the inner authentication of EAP-TTLS [9]. Occasionally PAP is also used together with token cards, as the alphanumeric codes generated by those systems can be used to mimic passwords [33].

EAP Generic Token Card (GTC) is an EAP method based on exchanging *cleartext* credentials, which closely resembles PAP. As the name suggests, EAP-GTC was intended to be used with token cards, though in reality it is often *overloaded* for performing a password-based user authentication instead [33].

2.3.2 EAP-MSCHAP-V2. Another widely used EAP method, especially among organizations that rely on the Microsoft AD infrastructures, is the Microsoft Challenge-Handshake Authentication Protocol version 2 (MSCHAPv2). Unlike PAP and GTC, MSCHAPv2 does not transmit passwords in cleartext. Instead, the ciphertext of a challenge hash will be sent, which is encrypted using DES, with the MD4 hash of the user's password as the secret key. Given the captured transcript of a MSCHAPv2 handshake, revealing the secret key (password hash) is only as difficult as exhaustively searching for the key of a single DES encryption (a complexity of 2^{56}), which can then be used by an attacker to impersonate the victim in future MSCHAPv2 attempts [7]. Additionally, MSCHAPv2 also has some design weaknesses that can be exploited to speed up dictionary attacks [30, 50], if one wants to reveal the actual password. Despite its weaknesses, MSCHAPv2 is supported by most mainstream OSs, and has been widely used as the phase-2 authentication method.

2.4 The Evil Twin attack

One problem of WPA2 is that the access points are not authenticated by design, and therefore it is possible for an attacker to impersonate a known network by setting up a so-called "Evil Twin" (ET) [13, 23]. Depending on the attacker's goal, the ET attack does not need to follow a typical man-in-the-middle (MITM) model. In fact, if the objective is to steal user credentials, which is made particularly profitable by the reuse of SSO credentials in WPA2-Enterprise, the attack setup does not even need to provide legitimate connectivity and can simply terminate after receiving the victim's response during the phase-2 authentication. To make matters worse, the ET attack can be performed using low-cost and highly portable off-the-shelf hardware and software components. We purchased the 4GB model of Raspberry Pi 4 with less than USD $60, a protective case at USD $5, and a micro SD card at less than USD $6. We then install the Raspberry Pi version of Kali Linux and the `hostapd-wpe` software package (a configurable implementation of the ET attack), both of which are free of charge. It took us just a matter of minutes to set up. The Raspberry Pi 4 hardware works with the necessary software packages right out of the box, without the need of an extra USB Wi-Fi adapter, further driving down the cost. With a portable power bank at less than USD $20, one can even carry the attack setup in a backpack and move around to hunt down victims at strategic locations, at a total monetary cost of less than USD $100, well within reach for mere script kiddies.

3 OUR COMPARISON FRAMEWORK

In this section, we present a framework for comparing the relative robustness of supplicant configurations. We went through multiple rounds of refinements in order to accommodate all the options and subtleties of the mainstream OSs discussed in Section 4. We are not aware of any prior work giving such a framework. Due to space constraints, the implicit assumptions and exceptions of our framework are presented in Appendix A.

3.1 Insight of our comparison framework

For comparing configurations, we introduce the notion of an abstract security label ℓ that captures the abstracted connection attributes of a given configuration. Such labels ℓ are drawn from a strict partially ordered set $\mathcal{L}$ of labels. We resort to $\mathcal{L}$ being strict partially ordered because if two abstract security labels ℓ_a and ℓ_b both indicate susceptibility to attacks, in many cases it is not clear whether ℓ_a or ℓ_b leads to a better outcome for the victim.

Given two configurations c_1 and c_2 which are mapped to labels ℓ_1 and ℓ_2 correspondingly, we say c_1 is more secure than c_2 if and only if $\ell_2 <_s \ell_1$ (read, ℓ_2 is less than ℓ_1). Multiple configurations (stemmed from the various UIs on different OSs) can be mapped to the same abstract security label, so long as their abstracted connection attributes are equivalent.

3.2 Abstract security label

In our discussion, we consider $\ell \in \mathcal{L}$ to be of the form $\ell = (\alpha, \beta, \gamma, \delta)$ and use it to capture the possible PEAP and EAP-TTLS configurations that are considered in this work. The first element of ℓ, α, is used to capture how the **rejection** of invalid certificates is being performed. $\alpha \in \{$`P`, `AM`, `N`$\}$, which stands for `Programmatic`, `Assisted Manual`, and `None`, respectively. Some might question why α is not a boolean variable and why we need to introduce `AM`, besides `P` and `N`. Such a refinement is actually necessary because some OSs (*e.g.*, macOS and iOS) require the user to make the final decision on whether to *reject* the received certificate, as we will explain in Section 4. The second element, β, captures the trust anchor used by α to establish the validity of the certificate. $\beta \in \{$`Sp.`, `Sys.`, `n/a`$\}$, which stands for `Specific`, `System CA store`, and `not applicable`, respectively. Notice that $\alpha =$ `N` $\implies \beta =$ `n/a`, and not all OSs come with a UI that supports both the $\beta =$ `Spec.` and $\beta =$ `Sys.` options. The third element, γ, denotes the way that the server name is being checked, and $\gamma \in \{$`P`, `M`, `N`$\}$, which stands for `Programmatic`, `Manual`, and `None`. Once again, support for γ varies greatly among mainstream OSs. Finally, the fourth element, $\delta \in \{$`Ob.`, `Cl.`$\}$, denotes whether the phase-2 authentication method sends `Obfuscated` or `Cleartext` credentials. Not all OSs allow the user to choose the preferred phase-2 method.

3.3 Strict partial ordering of security labels

A Hasse diagram showing all the possible configurations captured by the abstract security label ℓ can be found in Figure 1. Nodes in the diagram represent abstract security labels that connection configurations can be mapped to, where the ones that might be susceptible to the ET credential theft are rectangular-shaped (red and purple in color), and the ones that are not immediately vulnerable are represented with rounded rectangles (blue in color). Nodes are ranked according to their relative robustness, and a directed edge from node A to node B means $B <_s A$. The styles (and colors) of the edges capture the degradation of security. A solid edge with solid arrow (black in color) captures the notion of using a `Specific` trust anchor is better than using the `System CA store`. This is related to the possibility of some CAs getting compromised/going rogue, which is a well documented concern when using the public key infrastructure (PKI) [42]. Abstractly, trusting only some specific CAs can help reduce exposure to potentially compromised/rogue CAs when compared to the case of trusting all the CAs included in the system CA store, since in the prior case, an ET attacker will need to target the specific CAs in order to launch a successful attack.

Meanwhile, a dotted edge with solid arrow (green in color) captures the notion that rejecting invalid certificates, for it be enforced programmatically ($\alpha =$ `P`) or manually with machine assistance ($\alpha =$ `AM`), is more secure than not rejecting invalid certificates at all ($\alpha =$ `N`). The rationale behind is intuitive, as accepting certificates that cannot be validated will enable the ET attack. Not properly validating certificates is one of the most critical flaws of many TLS-using systems, examples include Android banking apps [48] and a variety of Web and messaging applications [35].

Similarly, a dashed edge with hollow arrow (purple in color) captures the fact that some form of server name checking, whether it is programmatic ($\gamma =$ `P`) or manual ($\gamma =$ `M`), is better than not checking it at all ($\gamma =$ `N`). This is yet another classic issue that often gets overlooked in PKI-using systems [2, 31, 35, 39]: just because the certificate can be validated does not mean the peer is indeed the right entity. When $\gamma =$ `N`, an attacker can get a certificate chain for a domain under control from the same trust anchor, and then use it to launch a successful ET attack.

Finally, a solid edge with hollow arrow (blue in color) captures the idea of using a phase-2 authentication method that sends obfuscated credentials ($\delta =$ `Ob.`) is better than those that send cleartext credentials ($\delta =$ `Cl.`). Notice that the scenario where the supplicant configuration does not specify a particular phase-2 method is also captured by $\delta =$ `Cl`. This is because in WPA2-Enterprise mode, the authentication server gets to choose the phase-2 method. In other words, from an adversarial perspective, a rational ET attacker would definitely choose methods that are more advantageous (*i.e.*, EAP-GTC and PAP) to get the victims' passwords directly. This is especially important when using an already vulnerable configuration, as it can mark the difference between allowing an ET attacker to steal the password directly (rectangular-shaped nodes with diagonal cut corners in Figure 1), or some obfuscated credential (as in the case of MSCHAPv2 [1]), which needs further dictionary/brute-force attacks to reveal the actual password. Under this consideration, even (`N, n/a, N, Ob.`) can potentially lead to a better outcome than configurations like (`P, Sys., N, Cl.`) and (`N, n/a., P, Cl.`). In Figure 1, this special case is represented with a solid edge with hollow arrow that is red in color.

[1]As discussed in Section 2.3.2, one can find the password hash at the cost of 2^{56} [7], but the complexity of finding the password depends on its length and composition.

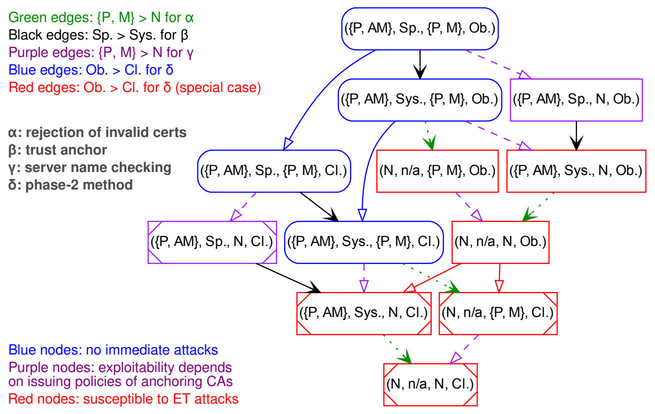
Green edges: {P, M} > N for α
Black edges: Sp. > Sys. for β
Purple edges: {P, M} > N for γ
Blue edges: Ob. > Cl. for δ
Red edges: Ob. > Cl. for δ (special case)
α: rejection of invalid certs
β: trust anchor
γ: server name checking
δ: phase-2 method
({P, AM}, Sp., {P, M}, Ob.)
({P, AM}, Sys., {P, M}, Ob.)
({P, AM}, Sp., N, Ob.)
({P, AM}, Sp., {P, M}, Cl.)
(N, n/a, {P, M}, Ob.)
({P, AM}, Sys., N, Ob.)
({P, AM}, Sp., N, Cl.)
({P, AM}, Sys., {P, M}, Cl.)
(N, n/a, N, Ob.)
({P, AM}, Sys., N, Cl.)
(N, n/a, {P, M}, Cl.)
(N, n/a, N, Cl.)
Blue nodes: no immediate attacks
Purple nodes: exploitability depends on issuing policies of anchoring CAs
Red nodes: susceptible to ET attacks

Figure 1: Possible configurations, ordered in terms of robustness. Rectangular nodes represent potential vulnerabilities.

4 MAKING SENSE OF THE OS BEHAVIORS

To the best of our knowledge, no previous work gave a systematization of what WPA2-Enterprise configurations are supported by the mainstream OSs. Hence, in order to be able to grade configuration instructions, it is necessary to first understand the UI behavior of each OS, as well as the actual configurations that they support. For this, we evaluate each realizable configuration by launching an ET attack using the set up discussed in Section 2.4.

We are primarily interested in how the *design of UIs* limit configuration possibilities, and how the OS behaves under specific configurations. All OSs tested in this section have native support for both PEAP and EAP-TTLS, except for Windows 7, which only supports PEAP out of the box, but third party plugins can be installed to add support for EAP-TTLS. A summary of the design weaknesses identified through testing is shown in Table 1. Though Linux is often not supported by TEIs, we discuss the issues of its `NetworkManager` GUI in Appendix C.

4.1 Android

4.1.1 Configurations supported. For Android, various vendors offer heavily customized UIs, so we base our discussion on the ones that offer near-stock UIs (*e.g.* participants of Android One). A comparison of the configurations supported by Android can be found in Fig. B2a in Appendix. In general, Android does not prompt the user to inspect the server certificate and hostname ($\alpha \neq \texttt{AM} \wedge \gamma \neq \texttt{M}$). We also make a distinction between Android 7+ (Nougat and newer) and 6- (Marshmallow and older), as the two have noticeably different UI options in terms of Wi-Fi configuration.

First of all, for trust anchor, while it is possible to choose “Use system certificates” ($\alpha = \texttt{P} \wedge \beta = \texttt{Sys.}$) from the dropdown menu on Android 7+, Android 6- simply does not offer that option ($\beta \neq \texttt{Sys.}$). However, Android 6- can actually perform certificate validation using one specific trust anchor ($\alpha = \texttt{P} \wedge \beta = \texttt{Sp.}$), given that the user installs a CA certificate and sets it for use with Wi-Fi prior to configuring. Also, the UI on Android 6- does not have an input box for the user to type in the name of the authentication server ($\gamma \neq \texttt{P}$), and given $\gamma \neq \texttt{M}$ as discussed above, it only supports configurations with $\gamma = \texttt{N}$. Both Android 6- and 7+ have a drop-down menu for choosing either one specific inner authentication method ($\delta = \texttt{Ob.}\ \texttt{XOR}\ \delta = \texttt{Cl.}$, depending on the method chosen), or to let the system decide ($\delta = \texttt{Cl.}$).

4.1.2 Design weaknesses. We observed an interesting flaw in the UI of Android 7+. When the user chooses “Use system certificates” for the trust anchor, the UI will then require the user to type in the expected name of the authentication server, before allowing the user to connect ($\beta = \texttt{Sys.} \implies \gamma = \texttt{P}$). However, if the user chooses a specific CA certificate as the trust anchor, the UI will allow the user to connect without typing in a server name ($\beta = \texttt{Sp.} \implies \gamma = \texttt{P}\ \texttt{XOR}\ \gamma = \texttt{N}$). We filed a bug report regarding this discrepancy and the Google Android team confirmed it with a moderate severity. CVE-2020-27055 has been assigned for this and a fix has been released.

4.2 Chrome OS

4.2.1 Configurations supported. We base our analysis of Chrome OS on v87, the latest at the time of writing. Most Chromebooks receive rolling updates, and across recent versions (from v83 to v87) we did not notice any meaningful differences in the configuration UIs. The configurations supported by Chrome OS can be found in Fig. B2b in Appendix. We found that for the purpose of configuring Wi-Fi networks, Chrome OS is highly similar to Android 6-, albeit with its own oddities. In general, Chrome OS also does not prompt the user for manual inspection ($\alpha \neq \texttt{AM} \wedge \gamma \neq \texttt{M}$), and it does not have an input box for configuring server name checking ($\gamma = \texttt{N}$) when either PEAP or EAP-TTLS is chosen as the phase-1 method.

Regarding certificate validation, specific CA certificates can be installed and chosen as the trust anchor in a dropdown menu ($\alpha = \texttt{P} \wedge \beta = \texttt{Sp.}$). There is an option in the dropdown menu *confusingly* named “Default”, which based on our testing, uses the system CA store as the trust anchor ($\alpha = \texttt{P} \wedge \beta = \texttt{Sys.}$). Another option available in the dropdown menu is “Do not check”, which disables the certificate validation completely ($\alpha = \texttt{N}$).

4.2.2 Design weaknesses. Our testing found 4 design weaknesses on Chrome OS. First, the lack of a server name input box is definitely a UI design flaw that will lead to insecure configurations. Another UI flaw that we found concerns the actual default option for the trust anchor of certificate validation. If the user chooses to join a WPA2-Enterprise network from the list of scanned SSIDs, then “Default” ($\alpha = \texttt{P} \wedge \beta = \texttt{Sys.}$) will be chosen by the UI as the default option. However, if one attempts to add a new network directly (by manually typing in the SSID), then the “Do not check” option ($\alpha = \texttt{N}$) will be chosen by the UI by default. A more severe flaw on Chrome OS is that, to our surprise, the enforcement of server name checking (currently only possible through importing pre-configured profiles), uses a *substring matching* logic. In other words, if one sets the server name checking constraint to `example.edu`, a certificate with the name `example.edu.attacker.com` will be considered by Chrome OS as a match. Consequently, the server name checking on Chrome OS is simply ineffective against impersonation attacks. Finally, another issue concerns networks configured through side-loading profiles. If a profile that relies on more than one embedded CA certificate as the trust anchors ($\alpha = \texttt{P} \wedge \beta = \texttt{Sp.}$) is imported,

Table 1: Design weaknesses across OSs

OS	Issues	Implications
Android 6-	The UI lacks an input box for specifying the correct server name of the certificate.	Configurations vulnerable to ET attacks if commercial CAs are used as the trust anchor.
Android 7+	Server name becomes optional when a specific CA is chosen as the trust anchor.	Possibility of insecure configurations if commercial CAs are used as trust anchor.
Chrome OS	The UI lacks an input box for specifying the correct server name of the certificate.	Configurations vulnerable to ET attacks if commercial CAs are used as the trust anchor.
Chrome OS	Domain name checking (possible only via profiles) uses a substring matching logic.	Attackers can easily obtain attack certificates which satisfy the name checking condition.
Chrome OS	Inconsistent default setting for CA certificate depending on the UI entry points.	Potential confusion and misconfiguration for users who do not understand the subtleties.
Chrome OS	The UI shows "Do not check" after importing a profile with multiple CA certificates.	Gives a misleading perception that the configuration is not validating the server certificate.
Windows 10	The short timeout hinders manual checking of the certificate thumbprint.	Frustrating the users into continuing the connection without checking the thumbprint.
Windows 10	The certificate thumbprint is the SHA1 hash digest of the certificate.	Resourceful attackers can compute and leverage hash collision for impersonation attacks.
Windows 7&10	By default, detailed configurations fall back to simple alerts and can be overridden.	Users might ignore the alerts, render the original configuration pointless and open doors to attacks.
iOS	The UI marks any server certificate as "Not Trusted" with a red-colored warning text.	Confusing the users into blindly trusting any server certificates.
macOS	Self-signed CA certificates in a pre-configured profile given full trusts after import.	Installing profiles can open doors to many other attacks, *e.g.*, MITM against HTTPS & IPSec.

and the user later wants to modify the settings of that connection, the configuration UI would display "Do not check", even though the (P, Sp., _, _) behavior still persists under the hood. This could create confusions with the users and mislead them into thinking that the "Do not check" option (α = N) is recommended. We filed 4 separate bug reports regarding these issues and all were confirmed by the Chrome OS developers. The missing input box issue has a ranking of low severity, and the other 3 issues all got a ranking of medium severity. CVE-2021-21212 has been assigned for the issue of inconsistent default setting.

4.3 Windows

4.3.1 Configurations supported. We mainly consider Windows 10 and 7, as both of them currently have a sizable market share. Interestingly, their UI behaviors are quite different. The configurations supported by Windows 10 and 7 are shown in Fig. B1a and Fig. B1b in Appendix. For both versions of Windows, there are 2 possible UIs that can be used to setup Wi-Fi, which we refer to as the *Simple UI* and *Traditional UI*. In both cases, the inner authentication method defaults to MSCHAPv2 (δ = Ob.). From the Simple UI, it is not possible to choose a different phase-2 method, but on the Traditional UI, additional methods like PAP and EAP-GTC can be chosen if one first installs some third party plugins (δ = Cl.). Such plugins are used quite extensively in certain countries/regions.

On Windows 10, the Simple UI can be launched directly from the Wi-Fi icon in the system tray. There are no checkboxes, dropdown menus, or input boxes for the user to indicate the preferred trust anchors and server names. Instead, it will ask if the user wants to continue connecting through an in place prompt. Blindly continuing means no server name checking would be performed (γ = N) and no invalid certificates would be rejected (α = N). Alternatively, the server identity can be confirmed by first clicking the "Show certificate details" link in the prompt and then matching the displayed SHA-1 digest of the certificate against some known values (SHA1(AM, Sp., M, Sp.)). Newer versions of Windows 10 (since version 2004) also display the issuer and subject name of the server certificate along side with the SHA-1 digest. While this makes the UI more informative, however, without a prior validity check, those names cannot be relied upon as they can be chosen arbitrarily by an attacker when crafting certificates for impersonation attacks.

On the other hand, the simple UI of Windows 7 will not display the hash digest of the server certificate, but instead has an implicit α = P $\wedge$ β = Sys. logic, that is, if the server certificate cannot be validated by any of the trust anchors in the system CA store, it will be automatically rejected. Only if the server certificate passed through the initial validation check, then further alert windows will be shown to allow the user to possibly check the name of the server ceritifcate (γ = M) and the anchor of the chain of trust (α = AM$\wedge$$\beta$ = Sp.). It is also possible to just blindly continue without inspecting the server name (γ = N) or the trust anchor shown on the alert windows.

The traditional UI of Windows 10 and 7 can be invoked from "Control Panel" and they are very similar. Both offer an optional input box for the expected server name to be checked programatically (γ = P), one checkbox for each trusted CA in the system CA store for selecting the preferred trust anchors (β = Sp.), and a checkbox for disabling certificate validation completely, which would also disable hostname checking (α = N $\wedge$ γ = N).

4.3.2 Design weaknesses. There are several issues that we identified with the design of the Windows 10 UI. First, regarding the Simple UI, the reliance of SHA-1 digest alone as the basis for confirming server identity provides a questionable level of security, as a collision attack against SHA-1 has already been demonstrated by previous work [55]. Moreover, we found that the user is given a meagre 20-second period to decide whether to continue with the connection when using the Simple UI. When the system times out, the user will have to reenter the username and password, and get another 20-second opportunity to decide. Based on the personal experience of our team members, it is nearly impossible to match the SHA-1 digest (20 bytes each, encoded into 40 hexadecimal characters for display) before timeout. This could lead to frustrations and the user may end up blindly continuing without performing the verification. We reported these 2 issues to the Microsoft Security Response Center, but they dismissed our report on the grounds that those appear to be product suggestions and are not vulnerabilities.

Finally, we note that on both Windows 10 and 7, there is a subtle option on the Traditional UI that needs to be explicitly disabled by the user in order to enforce the validation behavior specified in the Traditional UI (*e.g.*, α = P, γ = P, *etc.*). Otherwise, when the OS receives a server certificate that is untrusted or contains unexpected names, it will by default *fall back* to the Simple UI. This is particularly detrimental for Windows 10, as unlike Windows 7, it does not have an implicit α = P $\wedge$ β = Sys. logic, and the SHA-1 digest on the Simple UI is prone to collision attacks. Unfortunately, this subtle option is sometimes overlooked by instructions that we evaluate in Section 5, leading to insecure configurations.

4.4 macOS and iOS

4.4.1 Configurations supported. For macOS and iOS, we consider version 10.15 and 13.6, respectively, which are the latest versions at the time of writing. Despite aesthetic differences, the two are quite similar in spirit when it comes to configuring WPA2-Enterprise. The configuration and verification possibilities supported by the two are basically the same, as shown in Figure B1c in Appendix. Without the use of profiles, both macOS and iOS rely heavily on human intervention for certificate validation and server name checking. No drop-down menus or checkboxes are provided for indicating the preferred phase-2 method, although both macOS and iOS support a variety of methods under the hood ($\delta = \texttt{Cl.}$). When a TLS tunnel is established with a particular authentication server, an alert will pop out prompting the user to verify the server certificate and the user has to decide whether to continue or not.

On macOS, the alert window by default only shows the common name of the server certificate, and this can be manually inspected and considered in a rejection decision ($\gamma = \texttt{M}$), or completely ignored ($\gamma = \texttt{N}$) before continuing. Additional information, including a *predetermined certificate validity* and the CA certificates used to form the chain of trust, will be shown only after the user clicks the "Show Certificate" button. Interestingly, even if the system has already determined a server certificate to be invalid, it will not be programatically rejected ($\alpha \neq \texttt{P}$), and the alert window will still be shown, effectively delegating the rejection decision to the user. The user can then decide to continue/stop the connection based on the predetermined (in)validity ($\alpha = \texttt{AM} \land \beta = \texttt{Sys.}$), or blindly continue without inspecting the validity information on the alert window, in which case no invalid certificates will be rejected ($\alpha = \texttt{N}$).

Moreover, there are two ways to upgrade the configuration. First, the user can check both the predetermined validity and the issuer name in the detailed information given on the alert window, and then continues to connect only if the certificate is found to be valid and the issuer name fits that of an expected CA ($\alpha = \texttt{AM} \land \beta = \texttt{Sp.}$). Another way to upgrade the configuration is to inspect the SHA-256 (and SHA-1) digests of the certificate, both of which can be found when the user clicks on the "Details" button and scroll all the way down to the bottom of the alert window.

4.4.2 Design weaknesses. We have identified 1 issue for iOS and 1 issue for macOS. For iOS, the determination of certificate validity with system CA store seems to be broken, and it appears that all server certificates, including those that are determined to be valid on macOS, will always be marked as invalid. Consequently, the only way to confirm the server identity is through verifying hash digests of the server certificate. This bug has been mentioned in some of the instructions that we surveyed in Section 5, and we filed a report to Apple regarding this, but at the time of writing we have not received any responses from Apple.

For macOS, we found that the CA certificates imported through pre-configured profiles will be included in the system CA store, which will affect the determination of certificate validity for other WPA2-Enterprise networks in the future. More severely, *self-signed root CA* certificates imported this way will be trusted *by default* for all purposes beyond Wi-Fi, including secure mails (S/MIME), Web browsing (TLS), IPSec, and code signing. In other words, importing pre-configured Wi-Fi profiles embedded with self-signed root CA certificates provides a covert pathway for injecting trusted certificates useful for other attacks (*e.g.*, TLS interception for breaking HTTPS). We successfully used a Linux wireless AP running mitmproxy [22] to intercept HTTPS traffic from Safari and Firefox on a MacBook that imported a Wi-Fi profile containing our own self-signed root CA certificate. We reported this elevation of trust to Apple but their product security team dismissed our report, claiming that there is no security risk to users.

5 INSECURE WPA2-ENTERPRISE CONFIGURATION INSTRUCTIONS

To evaluate the spread of problematic configuration instructions, we conduct a large-scale study based on publicly accessible ones that TEIs prescribe to their staffs and students. We also establish links between the misguided designs of mainstream OSs and the poor instructions found in the wild.

5.1 Instruction gathering and labeling

We first collected lists of TEIs from Wikipedia and lists of participants from national/regional eduroam homepages (see [40]), and consolidated a list of 7045 TEIs covering 54 countries/regions. We then used Selenium WebDriver to crawl the domains of these TEIs from Google. We crawled the top 8 Google search results for `"eduroam OR wifi OR WI-FI OR WLAN site:<domain>"`, in an attempt to automatically discover URLs to Wi-Fi configuration instructions. For each TEI, if none of those results seem relevant, we would manually navigate to its homepage and use the internal search there as a last resort. We only crawled results of Google search but never scrapped any TEI websites programatically. If any *applicable* Wi-Fi configuration instructions can be found, we would then manually capture and archive them using a custom Chrome extension, and then interpret them and assign security labels, based on the framework introduced in Section 3 as well as the supported configurations determined in Section 4. Out of the 7045 TEI considered, more than 5000 were investigated by 3 authors, and another 4 authors each investigated hundreds of TEIs. Separation of labor was partly based on authors' language abilities. Notice that not all TEIs make their configuration instructions publicly accessible, and not all instructions describe the configuration of WPA2-Enterprise. We consider an instruction applicable if it mentions terminologies like `PEAP`, `TTLS`, phase-2 methods, and certificates.

Our data gathering and labeling effort happened between July 2020 to January 2021. In the end we collected and graded 7275 applicable configuration instructions from 2061 TEIs (around 30% of the TEIs considered). 3096 of the 7275 instructions came from non-anglophone countries, though some might be written in English. We follow instructions in local languages if possible, since those tend to be more informative. Top non-English languages in this set include Chinese, Japanese, Korean, German, French, and Italian. We consult Google Translate when necessary. The top 10 regions contributing the largest number of TEIs and instructions/grades can be found in Tables G5 and G6 in Appendix. Overall, US contributed the most for both campus Wi-Fi and eduroam, but numerous schools there still do not support eduroam. Many European schools use eduroam as their sole Wi-Fi service, while TEIs in Asian regions tend to have better support for dedicated campus Wi-Fi than eduroam.

5.1.1 *Tie-breaking and extrapolation.*

For each TEI, if multiple versions of instructions exist, we in general only consider the latest one. Occasionally, a TEI might provide multiple instructions that are endorsed equivalently for a specific OS (*e.g.*, one based on Windows 10 Simple UI, another based on Traditional UI), and in those cases we keep only the worst labels, because some users might choose to follow the bad ones without understanding the intricacies. Also, many instructions use screenshots to illustrate the steps that users should follow, although sometimes the texts might contradict the images. In such cases, we follow the textual description.

When we assign labels to instructions, we respect the OS (and version) that each instruction is explicitly specified to be targeting, however, in some cases it is necessary to extrapolate. The first such case concerns different versions of Android. While we made a distinction between Android 6- and 7+, many TEIs do not take that into consideration when they prepare instructions, and oftentimes they just provide one instruction for Android (without specifying which version). Given such instructions, we would grade them for both Android 6- and 7+, based on the UI restrictions discussed in Section 4.1. Another case where we perform extrapolation is when dealing with generic instructions. We assign labels based on OS-specific instructions if they exist, however, sometimes TEIs prescribe very generic instructions that are non-OS-specific. Given such instructions, we extrapolate based on the default OS behavior discussed in Section 4.

5.1.2 *Ambiguity and threats to validity.*

The main challenge of evaluating instructions is the ambiguity of natural languages. While the OS-specific instructions tend to be relatively straightforward to grade, the generic ones can sometimes be incredibly vague and confusing. For example, some non-OS-specific instructions might implicitly base their terminologies on one OS (*e.g.*, "`Unspecific`", an UI option only valid for Android 6-), and claim that the users can configure similarly on other OSs. In those cases, we try our best to match the given information, and imagine what a user would do, given the default behaviors and UI restrictions discussed in Section 4. This, however, is not a definitive prediction of what might actually happen in real life. Also, some instructions provide a link for downloading a CA certificate, without actually instructing the users how to install it. For these cases, we give them the benefit of the doubt and imagine the users would be able to properly install the CA certificate (and use it as the trust anchor if applicable).

Moreover, some instructions casually mentioned information that might be useful in performing certain checks, *e.g.*, *"When prompted, click **Trust**, to trust the <server name> certificate."* We interpret this as clicking the *Trust* button directly (assuming the *<server name>* clause is merely informational and not a precondition to the click), but some might argue that this means the server name is being checked (γ = `M`). Although we cannot guarantee perfect labeling of instructions, to improve the overall consistency and reliability of our results, one of our team members randomly sampled and verified thousands of security labels against their corresponding instructions, and the lead author independently checked all the security labels assigned. Most of the labeling conflicts are due to language ambiguity (similar to the examples given above), and confusing cases were discussed among authors, with the corresponding author making the final decision in conflict resolution.

Table 2: Instruction security (excluding profile installers)

Campus Wi-Fi				*eduroam*			
OS	*Insecure Labels*	Total Labels	*Insecure Perc.*	OS	*Insecure Labels*	Total Labels	*Insecure Perc.*
Windows 10	*774*	801	*96.6%*	Windows 10	*780*	814	*95.8%*
Windows 7	*663*	725	*91.4%*	Windows 7	*624*	692	*90.2%*
Android 7+	*888*	990	*89.7%*	Android 7+	*761*	941	*80.9%*
Android 6-	*961*	992	*96.9%*	Android 6-	*840*	941	*89.3%*
macOS	*797*	809	*98.5%*	macOS	*749*	766	*97.8%*
iOS	*908*	916	*99.1%*	iOS	*799*	812	*98.4%*
Chrome OS	*243*	251	*96.8%*	Chrome OS	*207*	247	*83.8%*
Others	*402*	454	*88.5%*	Others	*368*	484	*76.0%*
Total	*5636*	5938	*94.9%*	Total	*5128*	5697	*90.0%*

Table 3: Majorities of labels assigned for each OS

Campus Wi-Fi					
OS	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
Windows 10	*923*	N, n/a, N, Ob.	*81.6%*	Installer only	*13.2%*
Windows 7	*863*	P, Sys., N, Ob.	*49.6%*	N, n/a, N, Ob.	*27.2%*
Android 7+	*1040*	N, n/a, N, Ob.	*44.0%*	N, n/a, N, Cl.	*41.3%*
Android 6-	*1043*	N, n/a, N, Ob.	*49.1%*	N, n/a, N, Cl.	*43.0%*
macOS	*893*	N, n/a, N, Cl.	*80.9%*	Installer only	*9.41%*
iOS	*993*	N, n/a, N, Cl.	*81.1%*	N, n/a, M, Cl.	*10.3%*
Chrome OS	*276*	N, n/a, N, Ob.	*34.8%*	P, Sys., N, Ob.	*27.5%*
eduroam					
OS	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
Windows 10	*1187*	N, n/a, N, Ob.	*63.2%*	eduroam CAT only	*23.6%*
Windows 7	*1054*	P, Sys., N, Ob.	*45.4%*	eduroam CAT only	*25.6%*
Android 7+	*1227*	N, n/a, N, Ob.	*33.9%*	N, n/a, N, Cl.	*28.1%*
Android 6-	*1223*	N, n/a, N, Ob.	*38.4%*	N, n/a, N, Cl.	*30.3%*
macOS	*1148*	N, n/a, N, Cl.	*58.4%*	eduroam CAT only	*24.9%*
iOS	*1195*	N, n/a, N, Cl.	*57.0%*	eduroam CAT only	*24.1%*
Chrome OS	*487*	eduroam CAT only	*42.1%*	P, Sys., N, Ob.	*20.1%*

5.2 Analysis of results

Each applicable instruction (*e.g.*, a single Webpage/PDF) can cover one or more OSs and can thus lead to multiple labels being assigned. For the 7275 applicable instructions collected, we assigned a total of 14602 labels, 2967 of which indicate profile installers, and 11635 are security labels for manual configurations following our framework presented in Section 3. 243 TEIs mandate the use of profile installers, and 1818 TEIs endorsed manual configurations, 97.2% of which prescribed at least one insecure instruction for one of the mainstream OSs. An OS-specific break down of the security label assigned for manual configurations can be found in Table 2. Not all schools have both a dedicated campus Wi-Fi and eduroam, hence we separate the configuration instructions accordingly. Overall, campus Wi-Fi instructions are slightly less secure than their eduroam counterparts. Nevertheless, a vast majority of both rely on manual configurations, and the percentages of insecure labels are overwhelming. Another highlight from Table 2 is that Android 7+ performed slightly better than the other OSs, and we attribute this to its UI design, which gives the users a slightly higher chance of ending up with a (`P`, `Sys.`, `P`, `_`) configuration. The rows of “`Others`” concern instructions for various OSs that have relatively low market shares, including Windows 8, Linux distributions, Blackberry and Symbian. Chrome OS is only sporadically supported among TEIs, primarily by schools in the US. For all OSs considered, however, the majority of labels are rather insecure, as shown in Table 3. Some examples of misguided instructions can be found in Appendix D.

We found that about 29% of the labels across all OSs are based on generic, non-OS-specific instructions. Also, among all the Android labels, more than 70% of them were from non-version-specific instructions. These numbers suggest that authors of instructions often want a one-size-fits-all solution, without carefully considering the technical subtleties of different versions of OSs. This is particularly bad for Android, where an insecure instruction based on old versions (*e.g.*, without server name checking) could limit the security of newer, better versions.

Further analysis of the data shows interesting patterns. We see that only a tiny portion of instructions actually mandate checking the hash digest of server certificates (less than 3% for Windows 10 and less than 2% for macOS and iOS). We attribute this to a low awareness of its positive implications among authors of instructions and the UI design of the OSs, as in all three cases the hash digest is not shown to the user by default, hindering adoption. In fact, on macOS and iOS, the digests are buried in a pile of auxiliary information regarding the certificate, which might have contributed to the even lower adoption percentage than Windows 10. Moreover, the percentage of profile installers for eduroam is more than doubled on each OS when comparing to that of campus Wi-Fi. We attribute this to the success of the CAT project. However, for both categories, Android sees the lowest percentage of profile installers.

Also, we found that the purple nodes in Figure 1 are not very common in the wild. Since Android 6- and Chrome OS do not have an input box for server name, those are the best security labels that they can achieve from the UI. Unfortunately, the percentage observed is rather low (less than 3% for campus Wi-Fi, and less than 9% for eduroam). Regarding the choice of phase-2 method on Android and Chrome OS, MSCHAPv2 is much more prominent than the other options, but a significant portion (more than 30% for Android) of instructions left it unspecified. Together with the fact that most do not configure proper certificate validation, this leaves an enormous amount of cleartext passwords world-wide to be stealable by an ET attacker.

The regional statistics for the various OSs considered can be found in Table G7 to G9 in Appendix. The overall trends still hold, though several regions, primarily European countries, rely more on eduroam CAT and are less prone to insecure configurations. For Android, the choice for phase-2 method also differs, with some Asian countries/regions tend to favor $\delta = \texttt{Cl.}$ over $\delta = \texttt{Ob}$.

6 ANALYSIS OF EDUROAM CAT PROFILES

To get a more complete picture of supplicant configurations, we also collect and analyze the official eduroam CAT profiles that are available to users. We choose to focus on eduroam CAT because a significant number of TEIs recommend/mandate the use of CAT, and its pre-configured profiles are publicly accessible online (https://cat.eduroam.org/). Some TEIs might employ similar tools from other vendors for their campus Wi-Fi, but in those cases the users would first need to connect to a guest network in order to download the installers and profiles, making them difficult for us to obtain at scale. Another advantage of focusing on CAT is that for each OS that it supports, a standard format would be used to encapsulate the profile. We can thus use standard text processing tools for the analysis, and consult the corresponding documentation to obtain the syntax and semantic meanings of the contents [1, 4–6].

We performed the data collection in January 2021, and in the end we successfully crawled 3593 CAT configuration profiles for each of the mainstream OSs considered in this study (Windows 10 has only 3592 downloadable profiles because one school does not offer CAT profiles for it). These profiles are from 3300 unique identity providers (IdPs)[2]. Sometimes a CAT profile can contain the configuration of multiple Wi-Fi (and wired) networks. In most cases, each organization runs an IdP for its users, but occasionally several organizations share the same IdP (and thus one IdP can be used in multiple profiles). Since a precise mapping between organizations and IdPs is unavailable to us, in what follows we use IdPs to approximate the organizations behind profiles.

A quick inspection shows that iOS and macOS profiles are mostly the same, with only minor differences in some internal payload identifiers (which do not affect the security of the configurations), and that macOS profiles sometimes have additional LAN configurations (on top of the WLAN ones), and thus we only consider macOS (but not iOS) profiles. Android CAT profiles appear to be quite similar across different versions of Android, and for simplicity we focus on profiles for Android 10. Likewise, we only consider profiles for Windows 10 as the ones for Windows 7 are quite similar.

We found that although CAT profiles in general lead to more secure connection attributes (*e.g.*, certificate validation is always enabled) than their manual configuration counterparts, they are not completely free from configuration problems. Specifically, there are interesting discrepancies across profiles for different OSs, some of which can lead to vulnerabilities and other usability issues. Names of TEIs have been redacted to protect them from potential attacks.

6.1 No server name checking (γ = N)

The first issue we discovered is that not all CAT profiles contain the information necessary for server name checking to happen. For Windows 10, Android 10, and macOS, we found only one profile without server name checking information. Since the resulting configurations rely on an internal CA as the trust anchor, depending on its issuing policy, it might not be possible for an ET attacker to obtain a workable certificate.

For Chrome OS, however, the situation is more bleak. In total we found 30 profiles without server name checking information. To determine whether they would be vulnerable to an ET attacker, we cross-referenced the anchoring CA certificates in such profiles with the trusted root CA certificates in `/etc/ssl/certs` on a recent installation of Ubuntu 20.04. 12 out of the 30 profiles rely on at least one commonly trusted root CA certificate as the trust anchor (Table E1 in Appendix). In other words, an attacker can simply purchase a certificate chain from the corresponding anchoring CAs for a domain under control, and then use it to launch the ET attack to exploit these profiles and steal user credentials. For the other 18 profiles, the success of the ET attack once again depends upon the issuing policies of their corresponding anchoring CAs.

Our findings on profiles without server name checking seem to correlate with the fact that Chrome OS does not have a hostname input box on its UI (Section 4.2.2), but short of interviewing the

[2]For the purpose of this paper, IdPs are logically equivalent to authentication servers.

actual creator of those problematic Chrome OS profiles, we cannot definitively prove causality.

6.2 Unspecific server names

Another requirement for server name checking is that the matching constraint needs to be hierarchically specific. This is usually achieved by either adding more subdomain labels if a hostname is preferred, or taking the directory attributes (*e.g.*, country, locality, organization, *etc.*) into consideration. Using only a single string without any directory or domain components as the server name matching constraint is prone to collision, as an attacker can try to obtain a certificate chain with the same server name from a different country, and then use that to satisfy the matching constraints. Some profiles contain multiple name matching constraints, and a certificate name is considered to be a match when at least one of the constraints is satisfied. Thus we only need to consider the most permissive constraint as we focus on potential impersonation attacks. Our analysis revealed several profiles with unspecific server names that rely on commonly trusted CAs as the trust anchor (see Table E2 in Appendix). Since the expected names are rather generic for those profiles, an attacker can set up a similarly named entity in a different country and obtain a workable certificate chain from the same anchoring CAs, and then use that to launch the ET attack against staffs and students of these TEIs.

6.3 Permissive hostname constraints

Another way to render server name checking ineffective is to use overly permissive hostname constraints. For this, we searched for hostname constraints that are less than 8 characters long, which resulted in only tens of profiles for each OS. As some organizations happen to own short domains, we manually filtered out those cases.

In the end, we found 20 Chrome OS profiles to have exceptionally short and permissive hostname matching constraints (*e.g.*, TLDs like `.at`, `.br`, `.cz`, `.dk`, *etc.*). After cross-referencing the anchoring CAs in those profiles with the CA certificates in `/etc/ssl/certs`, we determined 7 of them to be vulnerable (see Table E3 in Appendix). Because their hostname matching constraints are quite permissive, by purchasing a strategically chosen domain, an attacker can obtain a certificate chain from the corresponding anchoring CA while satisfying the hostname constraint relatively easily, and then a targeted ET attack can be launched using that certificate. This attack is made even easier if one considers the substring hostname matching logic used by Chrome OS (Section 4.2.2). However, even if Chrome OS fixes its matching logic in the future, these hostname constraints still need to be tightened to secure the supplicant configurations.

For the other 13 Chrome OS profiles with permissive hostname constraints, the difficulty of mounting the attack once again depends upon the issuing policies of their anchoring CAs (which are not part of `/etc/ssl/certs`, and most of them are internal CAs). The same also applies to the macOS, Windows 10, and Android 10 profiles with permissive hostname constraints, as none of them use commonly trusted root CA certificates as trust anchors.

6.4 Large number of certificates in profiles

Another interesting observation is that some profiles contain a large number of certificates. About two-thirds of the profiles embed only one anchoring certificate, and about 30% of the profiles embed $2-5$ certificates, but there are a few cases where the total number of certificates embedded in a profile goes beyond a hundred.

Depending on the OS concerned, importing certificates from profiles would have different effects. For Chrome OS, these certificates will be offered in a drop-down menu as possible trust anchors if one configures another WPA2-Enterprise network in the future, so the large number of imported certificates would cause a UI pollution but not detrimental to security, as they *will not* be counted in the system CA store. On macOS, however, this is a major cause of concern, because the imported CA certificates will be counted in the system CA store, affecting the certificate validation of other WPA2-Enterprise networks in the future. More importantly, the self-signed root CA certificates imported would be given full trusts for many different purposes, which can then be used to launch other attacks, as discussed in Section 4.4.2.

Table 4: Certificates in CAT profiles

	mac / Chrome / Android 10 Unique = 1469 (Overall = 6060)		*Windows 10 Unique = 1355 (Overall = 5734)*	
Parameter	***Count***	**Perc.**	***Count***	**Perc.**
RSA Public Key = 1024 bits	*188 (331)*	12.8%	*186 (320)*	13.7%
RSA Public Key > 1024 bits	*1255 (5682)*	85.4%	*1161 (5399)*	85.7%
Elliptic-curve Public Key	*26 (47)*	1.8%	*8 (15)*	0.6%
MD5-RSA Signature	*29 (35)*	2.0%	*29 (30)*	2.1%
SHA1-RSA Signature	*676 (3032)*	46.0%	*636 (2860)*	46.9%
SHA256-RSA Signature	*678 (2207)*	46.2%	*628 (2102)*	46.4%
SHA384-RSA Signature	*31 (704)*	2.1%	*28 (695)*	2.1%
SHA512-RSA Signature	*32 (41)*	2.1%	*29 (38)*	2.1%
SHA256-ECDSA Signature	*23 (41)*	1.6%	*5 (9)*	0.4%
Expired Certificates	*210 (563)*	14.3%	*200 (528)*	14.7%
Version 3 Certificates	*1430 (6007)*	97.3%	*1318 (5693)*	97.3%
Version 1 Certificates	*39 (53)*	2.7%	*37 (41)*	2.7%
CA Certificate	*1298 (5868)*	88.4%	*1186 (5554)*	87.5%
Non-CA Certificate	*171 (192)*	11.6%	*169 (180)*	12.5%

6.5 Quality of certificates in CAT profiles

Finally, we also analyze the quality of the certificates embedded in the CAT profiles. The results can be found in Table 4. We found that short RSA modulus (*e.g.* $\leq$ 1024 bits) is not yet extinct in the WPA2-Enterprise ecosystem, with about 5% of the certificates embedded in CAT profiles still use one. Interestingly, despite the known weakness in collision resistance [54, 57, 63], MD5-based signatures can still be found on a few certificates. Similarly, SHA1 is also collision-prone under various settings [44, 55, 56], but nearly half of the certificates embedded in the CAT profiles have SHA1-based signatures. The use of SHA256-based signatures is also prominent, but SHA512-based signatures are very rare. Moreover, RSA is still the overwhelmingly popular choice when it comes to digital signature, and only a tiny portion of the certificates are signed using ECDSA. Curiously, there are a few Version 1 certificates, and given that certificate extensions were only introduced since version 3 of X.509, whether these can actually be used as CA certificates would require verification through out-of-band means [21]. Finally, about 3% of the certificates are non-CA, and to our surprise, about 10% of the certificates are expired. Embedding non-CA or expired certificates in the profiles is counterproductive, as they do not help in building a valid chain of trust.

Table 5: TLS Session and Certificate parameters

Session Stats Total Domains = 3637				***Leaf Cert. Stats Unique = 2701 (Overall = 3637)***		***CA Cert. Stats Unique = 652 (Overall = 5300)***	
EAP	***Count***	**Perc.**	**Parameter**	***Count ‡***	**Perc.**	***Count ‡***	**Perc.**
PEAP	*3430*	94.3%	RSA Public Key = 1024 bits	*158 (223)*	5.9%	*25 (87)*	3.8%
TTLS	*2451*	67.4%	RSA Public Key > 1024 bits	*2535 (3406)*	93.8%	*621 (5201)*	95.3%
Both	*2202*	60.5%	Elliptic-curve Public Key	*8 (8)*	0.3%	*6 (12)*	0.9%
TLS Support							
Version	***Count***	**Perc.**	MD5-RSA Signature	*7 (8)*	0.3%	*3 (3)*	0.5%
SSLv3	*15*	0.4%	SHA1-RSA Signature	*324 (412)*	12.0%	*204 (1320)*	31.3%
TLSv1	*3577*	98.3%	SHA256-RSA Signature	*2092 (2791)*	77.4%	*401 (2252)*	61.5%
TLSv1.1	*2579*	70.9%	SHA384-RSA Signature	*200 (341)*	7.4%	*21 (1698)*	3.2%
TLSv1.2	*2643*	72.7%	SHA512-RSA Signature	*71 (78)*	2.6%	*17 (18)*	2.6%
TLSv1.3	*59*	1.6%	SHA256-ECDSA Signature	*7 (7)*	0.3%	*6 (9)*	0.9%
TLS by Default							
TLSv1	*979*	26.9%	Expired Certificate	*118 (162)*	4.4%	*67 (161)*	10.3%
TLSv1.1	*13*	0.4%					
TLSv1.2	*2586*	71.1%	Version 3 Certificate	*2669 (3598)*	98.8%	*650 (5298)*	99.7%
TLSv1.3	*59*	1.6%	Version 1 Certificate	*32 (39)*	1.2%	*2 (2)*	0.3%
Only TLSv1 Support = 968							
Only TLSv1.1 Support = 1			Chain Verification Success	*1635 (2280)*	60.5%	*N/A*	
Only TLSv1.2 Support = 47			Chain Verification Failure	*1066 (1357)*	39.5%	*N/A*	

‡ Count for overall certificates are in ()

7 ANALYZING THE TLS PARAMETERS USED BY AUTHENTICATION SERVERS

In this section, we collect and evaluate the different TLS parameters used by the authentication servers in the context of eduroam.

7.1 Data Collection

Taking advantage of the roaming nature of eduroam, we used the local eduroam access offered by our universities to perform this measurement study. From multiple sources (see [40]) including the public list of IdPs, the national/regional eduroam list of participants, and results of Section 5, we curated a list of domains conjectured to be providing authentication services for eduroam. We then tried a collection of generic usernames (*e.g.*, `anonymous`, `admissions`, *etc.*) combined with possible subdomain prefixes (*e.g.*, `ad`, `mail`), and in a few cases actual email addresses. In the end, we found usable usernames that allowed us to establish TLS tunnels via PEAP or TTLS for 3637 domains. Since we are only interested in the TLS parameters, we modified `wpa_supplicant v2.9` [3] so that it **terminates** the TLS handshake after receiving and archiving the `ServerHello` and `Certificate` messages, which is also how we determine whether an attempt was successful. Notice that for this data collection, **no actual user log-ins** (phase-2 authentication) **were performed**. Modifying `wpa_supplicant` also allows us to extract and archive the encrypted certificate chains in TLSv1.3. In order to test for deprecated versions of SSL, we also used older versions of `wpa_supplicant` and `OpenSSL` whenever necessary.

7.2 Traffic Evaluation

We captured the TLS traffic from the authentication servers of 3637 eduroam domains worldwide (see Table F4 in Appendix for the top 20 contributing TLDs). Table 5 shows the overview of our results.

7.2.1 Analysis of Session parameters. We found that PEAP has better support than EAP-TTLS (see the left side of Table 5). 94.3% of the 3637 domains support PEAP whereas only 67.4% support EAP-TTLS. We also thoroughly investigated the list of TLS versions supported by the authentication servers of eduroam domains by enforcing different TLS versions in the `Client Hello` message (see *TLS Support* in Table 5). Legacy versions of TLS are susceptible to various attacks under different models [15, 28, 51], and many of them have been proposed to be deprecated by standardization bodies [11, 46, 47, 59]. Much to our surprise, we encountered many domains that still support older and/or deprecated versions of TLS (*e.g.*, from SSLv3 to TLSv1.1). However, when we offered all the TLS versions in `Client Hello` packet (see *TLS by Default* in Table 5), 71.1% selected TLSv1.2 and 26.9% (979 domains) selected TLSv1 for their sessions. Further investigation revealed 26.6% (968) domains support *only* TLSv1. These statistics revealed that many of the authentication servers of eduroam domains surveyed in our analysis are still relying on old and weak versions of TLS.

7.2.2 Analysis of Certificate parameters. We also checked the parameters of X.509 certificates to evaluate the quality of the certificate chains used in the eduroam ecosystem. The rationale behind this analysis is that if a remote authentication server sends an X.509 certificate chain consisting of weak/insecure parameters, it might be possible for an attacker to compromise the all-important TLS tunnel and steal user credentials. We collected in total 3637 X.509 certificate chains, which include 3637 leaf and 5300 (intermediate and root) CA certificates. While many of them are reused across different domains, we identified some certificates that are unique: 2701 leaf and 652 CA. Surprisingly, a significant number of these certificates rely on some weak parameters that are prone to exploitation (see Table 5). For instance, 5.9% of the unique leaf certificates use short RSA moduli (*e.g.*, 1024 bits), which are not recommended [10]. 12% of the unique leaf certificates are signed with SHA1-RSA, and 7 leaf certificates are signed with MD5-RSA, both of which are susceptible to potential collision attacks [43, 44, 54–57, 63, 64]. Additionally, we found that a few domains use expired or version 1 certificates, both of which cannot provide an adequate level of trust to the given certificate chains.

Chain verification status: We verified each X.509 certificate chain using `OpenSSL` and observed that 39.5% of them were rejected with different reason codes. Out of the rejected chains, 5.3% have expired certificates, and 1% have CA certificates that are invalid due to incorrect X.509 extension values. 35.2% of the rejected chains are missing some issuer certificates (code 20), 50.8% were built correctly but the root is self-signed (code 19), and 7.7% are single-certificate chains that are self-signed (code 18). The fact that so many chains cannot be easily verified might be a reason why many TEIs instruct their users to disable certificate validation completely.

Certificate lifespan: We also investigated the lifespan (in years) of the 2701 unique leaf certificates and 652 unique CA certificates. While most of the lifespans are typical (53.2% of the leaf certificates are valid for 2-5 years and 69.2% of the CA certificates are valid for 5-20 years), much to our surprise, we discovered a few domains (1.5%) using leaf certificates with more than 20 years of validity and 1 domain with a negative lifespan. Similarly, 2.1% of CA certificates have unusually long validity of more than 50 years, most of which appears to be from self-signed CAs used by Identity and Access Management (IAM) products.

7.2.3 Suspected cases of key reuse. By examining certificates and public keys that are used by multiple TEIs, we identified several cases of suspected key reuse, which are reminiscent of the practice embraced by some TLS-intercepting anti-virus software [24]. We found clusters of TEIs in 3 countries (*i.e.*, KR, CN, FR) that appear to be using certificates or public keys generated by vendors of IAM products. If the corresponding private keys are indeed shared across different instances of the same product, an attacker might also purchase such products and abuse the shared keys for launching targeted ET attacks. To protect the organizations involved, we redact the names of TEIs and vendors in the rest of this section.

First, we computed clusters based on identical certificate chains. The largest cluster we found has 219 universities in Korea, all sent the exact same chain, with the leaf certificate issued to `*.`[redacted]`.ac.kr`. It could be that [redacted] University ([redacted]`.ac.kr`) is in charge of providing authentication service to all these universities. The other large clusters with more than 20 domains are all made of sibling/child subdomains that belong to the same organization (*e.g.*, different academic departments or affiliated colleges of a university, or different universities in a national alliance), and thus they might reasonably share the same authentication server. However, we found a cluster made of 18 universities in China, with the same server certificate issued to `C=FR;ST=Radius;O=Example Inc.;CN=Example Server Certificate`, and both the CA and server certificates had expired back in 2017. It is not clear to us whether these schools are simply sharing the same authentication server or reusing the same private key.

Additionally, we noticed a cluster made of 17 universities in China, with the leaf certificate issued to [redacted]`.com.cn`, which is a domain owned by a Chinese vendor of networking equipment. Based on the vendor's documentation, this certificate seems to be coming from one of its IAM software packages. Similarly, there is a cluster made of 12 universities in Korea, with the leaf certificate issued to `C=KR;ST=Seoul;L=`[redacted]`;O=`[redacted]`;OU=network;CN=`[redacted]` Root CA`, which seems to be coming from an IAM server product developed by a Korean vendor. We suspect these are cases of key reuse, but cannot prove them definitively at this point. If our suspicion turns out to be correct, the staffs and students of these 29 schools could be susceptible to ET attacks, regardless of their supplicant configurations.

Finally, we also computed clusters of domains that had the same public key on the leaf certificate, and compared them with the clusters that had identical certificate chains. Different certificate chains that have the same leaf public key is highly indicative of possible cases of key reuse, as the same authentication server would most likely send the same chain every time. In the end, we found 3 interesting cases. The largest cluster among the 3 is made of 10 Korean universities, where the CA certificate remains the same (which lasts till year 2111), and the leaf certificates were all issued to the same entity (`C=KR;ST=Seoul;L==Seoul;O=`[redacted]`;OU=`[redacted]`;CN=`[redacted]` CA SERVER`), but they have different expiration dates (all with a 20-year validity period). Those seem to be coming from IAM products of another Korean vendor, which we believe could be an additional instance of private key reuse. Furthermore, we found 4 universities in China to be using the same server public key. Their chains differ because some also sent the CA certificate during TLS handshake, but some did not. Interestingly the name of the leaf certificates is once again [redacted]`.com.cn` (same as the cluster of 17 universities discussed above), but these are all expired in September 2020. Nevertheless, this suggests that our suspicions might be correct. Another cluster that gained new membership is made of 3 French schools, where the name of the leaf certificate is [redacted]`.network`, which seems to be a domain owned by a French Wi-Fi solution provider. The chains differ because both the leaf and CA certificates were different, but the server public key remains the same, which we believe could be another case of key reuse.

8 DISCUSSION

8.1 Responsible disclosure

As discussed in Section 4, we have responsibly reported the OS design weaknesses to the corresponding vendors. While Microsoft and Apple dismissed our reports, Google already has fixes planned for both Chrome OS and Android. The CAT profile issues discussed in Section 6 have also been disclosed to the affected TEIs. Some replied saying that they fixed the issues following our reports, while some said they are ditching eduroam CAT and have removed the profiles completely. Some acknowledged but dismissed the reported weaknesses claiming the probability of exploit is low. Some took our reports more seriously and had further discussions with us and an engineer of eduroam. A problem faced by some TEIs is that Chrome OS can only handle a set of names when matching subject alternative names but not for subject names. However, those TEIs have multiple servers with subject names that belong to different third/second-level domains, and thus they resort to using a very permissive hostname constraint (*e.g.*, a TLD) in their profiles, which contributed to the findings discussed in Section 6.2 and 6.3. This is yet another example of poor OS designs hurting user security. We have relayed these deployment challenges back to the Chrome OS team and they are now considering adding new parameters in the design of their profile format to facilitate name checking. For the suspected cases of key reuse (Section 7.2.3), we contacted the IAM vendors but received no responses. After 30 days, we then contacted the affected TEIs. Two schools acknowledged our findings and changed their keys and certificates following our reports. For the problematic configuration instructions (Section 5), we prepared sample instructions recommending good practices that should be adopted, and have already shared with the affected TEIs. To improve the potential impact of our reports, we manually locate the contact methods of the IT staffs in charge of the instructions/profiles. Each report is tailor-made in accordance with the security labels assigned to the instructions (Section 5), including the explanation of issues as well as the suggested fixes. We finished the disclosure process in early August 2021. We managed to contact 1732 TEIs which prescribed insecure manual configuration instructions, and received 48 acknowledgments a month later. Some TEIs have updated their instructions following our reports.

8.2 Suggestions for improving security

Based on our results, we make the following suggestions for improving the security of WPA2-Enterprise. First, the use of pre-configured profiles should be encouraged, as the overall quality of the resulting configurations tend to be significantly higher (see Section 6), and it helps to prevent human errors from Wi-Fi users. Second, OS vendors need to revamp their designs of the Wi-Fi configuration UI.

Insecure options should be made difficult to apply or even removed completely. Relying on human intervention (*e.g.*, iOS and macOS) is generally a bad idea, and the design of Android 7+ has merits as it increases the chance of arriving at a secure configuration (see Section 5.2). Third, to reduce the impact of ET attacks, TEIs should consider to use multi-factor authentication to protect services that share SSO credentials with Wi-Fi, or use separate (instead of SSO) credentials for Wi-Fi. Finally, we question the merits of using the X.509 PKI in WPA2-Enterprise. We argue that Wi-Fi does not need a scalable authentication solution like the X.509 PKI, which is known to be difficult to implement and deploy correctly. The current design seems to be inheriting all of the PKI's complexity and deployment challenges for no apparent benefits. We conjecture that one might be better off with a simpler means of server authentication, but we leave the exploration of that design space for future work.

8.3 Ethical considerations

We take the following ethical considerations into account during our study. First, we manually retrieve configuration instructions from TEIs without programmatic scraping of their websites, thus `robots.txt` does not apply. Second, we redact names of TEIs in our results, as some of the weaknesses might still persist. Third, when collecting TLS parameters from eduroam authentication servers, we do not complete the TLS handshakes and never attempt any actual user log-ins. For 223 out of the 3637 successfully probed eduroam domains, we had to use actual user email addresses, which we collected from Google Scholar and IEEE Xplorer. We conclude this use of email addresses is highly unlikely to cause harms to the users based on the following considerations. When authenticating to eduroam, the phase-1 traffic is not encrypted, and the outer identity can be snooped by the host institute (if roaming) and eavesdroppers within range. Some schools encourage the use of anonymous outer identity because of this. In an ideal world, our data collection should succeed with only anonymous identities. However, we noticed that some servers perform a legitimacy check on the outer identity before proceeding to TLS handshake. Thus in an attempt to increase coverage, we try real email addresses in cases where generic accounts did not work. Since the outer identity is an *unauthenticated input*, the account owners can repudiate ever initiating the phase-1 traffic when phase-2 authentication is unsuccessful, which is guaranteed by our aborted TLS handshakes. This should pose little to no risk to the corresponding users. Over the data collection period, our probing only incurs an infrequent, negligible amount of traffic (a few partial TLS handshakes) to the TEIs' authentication servers, which should not cause any disruption to their benign services.

9 RELATED WORK

Given its importance, implementations and deployments of TLS have gone through extensive scrutiny over the years [14, 15, 25, 27, 32, 37, 45, 53]. TLS typically relies on X.509 certificates for server authentication, which is itself tricky to implement and configure correctly [17, 19, 20, 35, 41, 58]. Various appliances intercept TLS for different reasons, though many were found to be hurting security due to the use of broken ciphers and improper validation of certificates [24, 26, 29, 29, 62]. Some of these results inspired the design of our measurement study presented in Sections 6 and 7. Although enterprise Wi-Fi was shown to be vulnerable to different attacks under various settings [12, 13, 16, 18, 49, 60, 61], to the best of our knowledge, this is the most extensive measurement study to date covering different aspects of supplicant and server configurations, and the first to establish links between the misguided UI designs of mainstream OSs and poor supplicant configurations. Our results also suggest that awareness of the ET attack remains low among practitioners, despite years of academic research on the topic.

10 CONCLUSION

In this paper, we presented a multifaceted study of the WPA2-Enterprise ecosystem. We first proposed a framework for comparing the security of WPA2-Enterprise configurations. We then utilized this framework to evaluate the realizable configurations supported by the UIs of mainstream OSs and discovered many design weaknesses that can negatively impact security. Moreover, we conducted a large-scale evaluation of 7275 configuration instructions from 2061 TEIs and observed that 85.7% TEIs were prone to credential thefts on at least one OS. We also analyzed 3593 eduroam CAT profiles and discovered additional configuration issues. Finally, we collected and evaluated the TLS parameters used by the authentication servers of 3637 eduroam domains, and identified numerous security issues including the use of weak signature algorithms and suspected cases of key reuse. The results of our study show that the WPA2-Enterprise ecosystem has numerous security holes caused by poor supplicant and server configurations. Vendors, IT administrators and users all need to do their part in order to fully realize the security promises of WPA2-Enterprise.

ACKNOWLEDGMENTS

We thank the reviewers, especially our shepherd Frank Li, for their insightful comments and suggestions on how to improve this paper. We would also like to thank the engineers at eduroam, particularly Stefan Winter, for taking the time to discuss with us some of the problems found in CAT profiles and the deployment challenges faced by certain TEIs. This work was supported in part by the departmental startup budget NEW/SYC, GRF matching fund GRF/20/SYC, and Project Impact Enhancement Fund 3133292C from The Chinese University of Hong Kong (CUHK), as well as US Department of Defense (DARPA) Grant D19AP00039, and US National Science Foundation (NSF) grants CNS-2007512 and CNS-2006556. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the policies or endorsements of the funding agencies.

REFERENCES

[1] [n. d.]. A Configuration File Format for Extensible Authentication Protocol (EAP) Deployments. https://tools.ietf.org/id/draft-winter-opsawg-eap-metadata-00.html.

[2] [n. d.]. CWE-297: Improper Validation of Certificate with Host Mismatch. https://cwe.mitre.org/data/definitions/297.html.

[3] [n. d.]. Linux WPA/WPA2/IEEE 802.1X Supplicant. https://w1.fi/wpa_supplicant/.

[4] [n. d.]. Open Network Configuration. https://chromium.googlesource.com/chromium/src/+/main/components/onc/docs/onc_spec.md#EAP-configurations.

[5] [n. d.]. WiFi CSP - Windows Client Management | Microsoft Docs. https://docs.microsoft.com/en-us/windows/client-management/mdm/wifi-csp.

[6] [n. d.]. WiFi.EAPClientConfiguration | Apple Developer Documentation. https://developer.apple.com/documentation/devicemanagement/wifi/eapclientconfiguration.

[7] 2012. Divide and Conquer: Cracking MS-CHAPv2 with a 100% success rate. https://web.archive.org/web/20160316174007/https://www.cloudcracker.com/blog/2012/07/29/cracking-ms-chap-v2/.

[8] 2020. Evil Twins, Eavesdropping, and Password Cracking: How the Office of Inspector General Successfully Attacked the U.S. Department of the Interior's Wireless Networks. https://www.doioig.gov/sites/doioig.gov/files/FinalAudit_WirelessNetworkSecurity_Public.pdf.

[9] Brad Antoniewicz. 2015. 802.11 Attacks.

[10] Elaine Barker and Allen Roginsk. 2019. Transitioning the Use of Cryptographic Algorithms and Key Lengths. *NIST special publication 800-131A Rev. 2* (2019).

[11] R. Barnes, M. Thomson, A. Pironti, and A. Langley. 2015. Deprecating Secure Sockets Layer Version 3.0. https://tools.ietf.org/html/rfc7568.

[12] Alberto Bartoli, Eric Medvet, Andrea De Lorenzo, and Fabiano Tarlao. 2018. (In) Secure Configuration Practices of WPA2 Enterprise Supplicants. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*. 1–6.

[13] Alberto Bartoli, Eric Medvet, and Filippo Onesti. 2018. Evil twins and WPA2 Enterprise: A coming security disaster? *Computers & Security* 74 (2018), 1–11.

[14] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. 2015. A messy state of the union: Taming the composite state machines of TLS. In *IEEE Symposium on Security and Privacy (S&P)*. 535–552.

[15] Karthikeyan Bhargavan and Gaëtan Leurent. 2016. Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH. In *NDSS*.

[16] Sebastian Brenza, Andre Pawlowski, and Christina Pöpper. 2015. A practical investigation of identity theft vulnerabilities in eduroam. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. 1–11.

[17] Chad Brubaker, Suman Jana, Baishakhi Ray, Sarfraz Khurshid, and Vitaly Shmatikov. 2014. Using frankencerts for automated adversarial testing of certificate validation in SSL/TLS implementations. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 114–129.

[18] Aldo Cassola, William K Robertson, Engin Kirda, and Guevara Noubir. 2013. A Practical, Targeted, and Stealthy Attack Against WPA Enterprise Authentication.. In *NDSS*.

[19] Sze Yiu Chau, Omar Chowdhury, Endadul Hoque, Huangyi Ge, Aniket Kate, Cristina Nita-Rotaru, and Ninghui Li. 2017. Symcerts: Practical symbolic execution for exposing noncompliance in X. 509 certificate validation implementations. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 503–520.

[20] Yuting Chen and Zhendong Su. 2015. Guided differential testing of certificate validation in SSL/TLS implementations. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 793–804.

[21] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. 2008. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. https://tools.ietf.org/html/rfc5280.

[22] Aldo Cortesi, Maximilian Hils, Thomas Kriechbaumer, and contributors. 2010–. mitmproxy: A free and open source interactive HTTPS proxy. https://mitmproxy.org/ [Version 6.0].

[23] Dino A Dai Zovi and Shane A Macaulay. 2005. Attacking automatic wireless network selection. In *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*. IEEE, 365–372.

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

[25] Joeri de Ruiter and Erik Poll. 2015. Protocol State Fuzzing of TLS Implementations. In *USENIX Security*.

[26] Joyanta Debnath, Sze Yiu Chau, and Omar Chowdhury. 2020. When TLS Meets Proxy on Mobile. In *International Conference on Applied Cryptography and Network Security*. Springer, 387–407.

[27] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella-Béguelin, Karthikeyan Bhargavan, Jianyang Pan, and Jean Karim Zinzindohoué. 2017. Implementing and proving the TLS 1.3 record layer. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 463–482.

[28] Thai Duong and Juliano Rizzo. 2011. *Here Come The ⊕ Ninjas*. Technical Report.

[29] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J Alex Halderman, and Vern Paxson. 2017. The Security Impact of HTTPS Interception. In *NDSS*.

[30] Jochen Eisinger. 2001. Exploiting known security holes in Microsoft's PPTP Authentication Extensions (MS-CHAPv2). *University of Freiburg,[cit. 2008-27-05] Dostupné* (2001).

[31] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. 2012. Why Eve and Mallory love Android: An analysis of Android SSL (in) security. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 50–61.

[32] Paul Fiterau-Brostean, Bengt Jonsson, Robert Merget, Joeri de Ruiter, Konstantinos Sagonas, and Juraj Somorovsky. 2020. Analysis of DTLS Implementations Using Protocol State Fuzzing. In *29th USENIX Security Symposium (Security)*.

[33] Matthew Gast. 2005. *802.11 wireless networks: the definitive guide*. O'Reilly Media, Inc.

[34] Jim Geier. 2008. *Implementing 802.1X security solutions for wired and wireless networks*. John Wiley & Sons.

[35] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. 2012. The most dangerous code in the world: validating SSL certificates in non-browser software. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 38–49.

[36] Jonathan Hassell. 2002. *RADIUS: securing public access to private resources*. O'Reilly Media, Inc.

[37] Boyuan He, Vaibhav Rastogi, Yinzhi Cao, Yan Chen, VN Venkatakrishnan, Runqing Yang, and Zhenrui Zhang. 2015. Vetting SSL usage in applications with SSLint. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 519–534.

[38] Joshua Hill. 2001. An analysis of the RADIUS authentication protocol. (2001).

[39] Michael Howard, David LeBlanc, and John Viega. 2010. *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*. McGraw-Hill.

[40] Man Hong Hue. 2021. List of URLs used as references. https://gist.github.com/hugohue/66a45b16bd444f73e757b65eba858113

[41] Deepak Kumar, Zhengping Wang, Matthew Hyder, Joseph Dickinson, Gabrielle Beck, David Adrian, Joshua Mason, Zakir Durumeric, J Alex Halderman, and Michael Bailey. 2018. Tracking certificate misissuance in the wild. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 785–798.

[42] B. Laurie, A. Langley, and E. Kasper. 2013. Certificate Transparency. RFC 6962 (Experimental).

[43] Arjen Lenstra and Benne De Weger. 2005. On the possibility of constructing meaningful hash collisions for public keys. In *Australasian Conference on Information Security and Privacy*.

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

[45] Christopher Meyer and Jörg Schwenk. 2013. SoK: Lessons learned from SSL/TLS attacks. In *International Workshop on Information Security Applications*. Springer, 189–209.

[46] K. Moriarty and S. Farrell. 2021. Deprecating TLSv1.0 and TLSv1.1. https://tools.ietf.org/html/draft-ietf-tls-oldversions-deprecate-12.

[47] PCI Security Standards Council. 2015. *Migrating from SSL and Early TLS*. Technical Report.

[48] Bradley Reaves, Jasmine Bowers, Nolen Scaife, Adam Bates, Arnav Bhartiya, Patrick Traynor, and Kevin RB Butler. 2017. Mo (bile) money, mo (bile) problems: Analysis of branchless banking applications. *ACM Transactions on Privacy and Security (TOPS)* 20, 3 (2017), 1–31.

[49] Pieter Robyns, Bram Bonné, Peter Quax, and Wim Lamotte. 2014. Short paper: exploiting WPA2-enterprise vendor implementation weaknesses through challenge response oracles. In *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*. 189–194.

[50] Bruce Schneier, David Wagner, et al. 1999. Cryptanalysis of microsoft's PPTP authentication extensions (MS-CHAPv2). In *International Exhibition and Congress on Network Security*. Springer, 192–203.

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

[52] Suphannee Sivakorn, George Argyros, Kexin Pei, Angelos D Keromytis, and Suman Jana. 2017. HVLearn: Automated black-box analysis of hostname verification in SSL/TLS implementations. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 521–538.

[53] Juraj Somorovsky. 2016. Systematic fuzzing and testing of TLS libraries. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1492–1504.

[54] Alexander Sotirov, Marc Stevens, Jacob Appelbaum, Arjen K Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. 2008. MD5 considered harmful today, creating a rogue CA certificate. In *Annual Chaos Communication Congress*.

[55] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. 2017. The first collision for full SHA-1. In *Annual International Cryptology Conference*. Springer, 570–596.

[56] Marc Stevens, Pierre Karpman, and Thomas Peyrin. 2016. Freestart collision for full SHA-1. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*.

[57] Marc Stevens, Arjen Lenstra, and Benne Weger. 2007. Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In *Annual International Conference on Advances in Cryptology*.

[58] Cong Tian, Chu Chen, Zhenhua Duan, and Liang Zhao. 2019. Differential testing of certificate validation in SSL/TLS implementations: an RFC-guided approach. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 28, 4 (2019).

[59] S. Turner and T. Polk. 2011. Prohibiting Secure Sockets Layer (SSL) Version 2.0. https://tools.ietf.org/html/rfc6176.

[60] Mathy Vanhoef and Frank Piessens. 2017. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. In *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*. ACM.

[61] Mathy Vanhoef and Frank Piessens. 2018. Release the Kraken: new KRACKs in the 802.11 Standard. In *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS)*. ACM.

[62] Louis Waked, Mohammad Mannan, and Amr Youssef. 2018. To intercept or not to intercept: Analyzing TLS interception in network appliances. In *ACM AsiaCCS*.
[63] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. 2004. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. *IACR Cryptology ePrint Archive* (2004).
[64] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. 2005. Finding Collisions in the Full SHA-1. In *Annual International Cryptology Conference*.

A ASSUMPTIONS AND CONSIDERATIONS OF OUR COMPARISON FRAMEWORK

A.1 Trust anchors and issuance policies

Note that in order to keep our framework concise, we do not enumerate the number of trust anchors, and we do not compare the trustworthiness of different CAs. We acknowledge that not all CAs follow the same set of issuance policies, and some CAs might be easier to compromise than the others. Also because of this, the policies that a CA adhere to in its daily operations will have a significant influence over the actual security of configurations with β = Sp.. Certificate validation can be rendered pointless if the specified trust anchors contain a CA that will issue any certificates to arbitrary entities, including potential attackers. On the other extreme, if one unique CA is specified as the sole trust anchor, and this turns out to be a dedicated CA that will *only* sign the certificate used by the authentication server but nothing else, then even supposedly vulnerable configurations (the four of (_, Sp., N, _)) might turn out to be safe from the ET attack, as the attacker will be unable to obtain a workable certificate. This is also why those 4 nodes are colored purple in Figure 1, to denote the fact that the actual success of the attack depends on the policies and practices of the specified trust anchors. This consideration is particularly important for OSs that do not support server name checking (γ = N), as a dedicated CA could be the only means for achieving secure supplicant configurations.

A.2 Phase-2 methods

Similarly, when we rank (δ = Ob.) to be better than (δ = Cl.), we only make the distinction on whether the credentials are sent in cleartext, and refrain from having a fine-grained comparison on the security of possible obfuscation algorithms (e.g. different hash digests and ciphers), as doing so will make the framework unnecessarily complex, especially when only a handful of phase-2 methods are actually getting deployed at the time of writing.

A.3 Usability

Also note that in Figure 1, nodes are ranked only in terms of security, not usability. For certificate validation, while programmatically rejecting invalid ones (α = P) might put less burdens on the user, given the same set of trust anchors and a *careful* user, we assume that the assisted manual mode of rejection (α = AM) can be just as secure. Similarly, given a user who faithfully and correctly carry out the necessary string comparison, we assume that checking server-names manually (γ = M) can be as secure as programmatically (γ = P), even though the latter one might be more user friendly. We reckon that proper support for Unicode and wildcard characters in name matching can be tricky even when it is done programmatically [17, 19, 52], and in the most direct form of γ = M where a human is required to perform the string matching, it might be susceptible to script spoofing attacks through confusing homoglyphs (*e.g.*, with Cyrillic characters), though this depends on the font of the UI and how the strings are preprocessed for display, and can potentially be ameliorated by showing encoded derivatives or hash digests of the strings instead. In the spirit of keeping the framework generic, we do not enumerate or compare such details in how server name matching can be enforced.

A.4 Different shades of (AM, Sp., M, _)

Some OSs might show the hash digests (also known as the fingerprints or thumbprints) of the server certificate computed using various hash algorithms, and configuration instructions might request the users to verify some or all of them. Since each of the hash digests is computed by the OS over the entire server certificate including the names and signature, manually matching the hash digest against an expected values fits our definition of $\alpha = \text{AM} \wedge \beta = \text{Sp.} \wedge \gamma = \text{M}$, as the rejection of invalid server certificates is performed manually with some programmatic assistance (computing the hash), the trust anchor is specific (directly at the server certificate level), and the server name is also checked manually. Thus, matching the hash digests can be seen as a relaxed version of (AM, Sp., M, _), where instead of an *exact match* of the validity, trust anchor and name of the server certificate, a relaxed matching logic (inexact due to potential hash collision) is being used. To keep our framework simple, we do not enumerate all possible combinations of hash algorithms. Instead, we allow the configurations of (AM, Sp., M, _) to be qualified further by the hash digests being matched, for instance, SHA1(AM, Sp., M, _) if only a SHA1 digest is being matched, and SHA1&SHA256(AM, Sp., M, _) if both SHA1 and SHA256 digests are being matched. The possible combinations of hash algorithms and choice of phase-2 method vary across versions of mainstream OSs.

B CONFIGURATIONS SUPPORTED BY THE UI OF DIFFERENT OPERATING SYSTEMS

The abstract security labels of configurations supported by the *UIs* of Android, Chrome OS, Windows 10, Windows 7, and macOS & iOS are given as Hasse diagrams in Figures B2a, B2b, B1a, B1b, and B1c, respectively. At the time of testing, configurations supported by Android 6- are also directly configurable on Android 7+. For Windows 10 and 7, configurations in Simple UI with α = AM or γ = M are not directly configurable in the Traditional UI, but can be implicitly inherited when fall back happens (Section 4.3.2). Configurations in the Simple UI of Windows 10 and 7 with $\alpha \neq \text{AM} \wedge \gamma \neq \text{M}$ are directly configurable in the Traditional UI. Currently, the best configuration of Windows 10 Simple UI (AM, Sp., M, Ob.) is implicitly qualified by SHA1(), and the best configuration of macOS & iOS (AM, Sp., M, Cl.) can be further qualified by the combination of hash digests being checked (see Appendix A.4 for details). Since the experiments for determining the supported configurations were conducted in 2020, new OS updates might not exactly follow these diagrams (*e.g.*, fixing CVE-2020-27055 would imply Android no longer supports (P, Sp., N, _)). Also, the use of pre-configured profiles, for example through eduroam CAT (Section 6), can sometimes unlock additional configurations that are not directly configurable via the UIs.

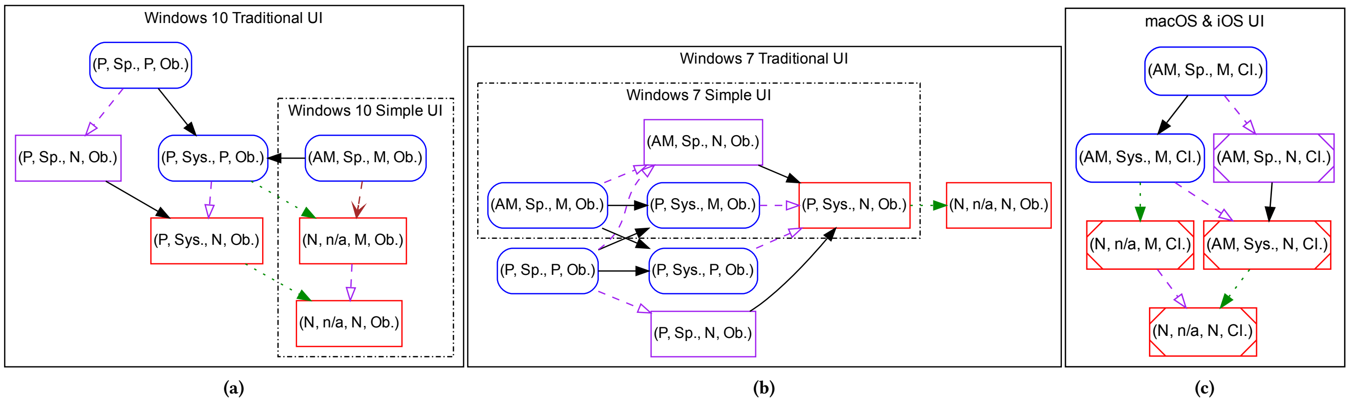

Figure B1: Possible configurations on Windows 10, Windows 7 and macOS & iOS

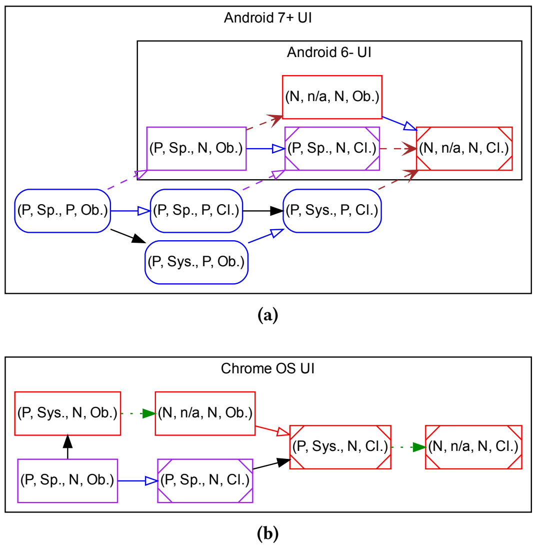

Figure B2: Possible configurations on Android, Chrome OS

C EVALUATING CONFIGURATION UI OF LINUX DISTRIBUTIONS

For Linux, there are many distributions and supplicant implementations with different GUIs available, and some users might prefer to write configuration files directly with a text editor, so it is difficult to have a general discussion covering all cases. Collectively Linux owns a tiny share of the consumer market, thus most schools do not offer technical support to Linux users. Based on the limited number of configuration instructions for Linux that we have found (Section 5), the most popular Wi-Fi configuration GUI on Linux seems to be `nm-connection-editor`, which is part of `network-manager-applet`, a GTK front end of `NetworkManager` that can run on various desktop environments (*e.g.*, Xfce, GNOME, Budgie, *etc.*). We refrain from a detailed discussion on the configurations supported by `nm-connection-editor` due to space constraint, however, there are two points worth highlighting. First, just like Android 6- and Chrome OS, old versions of `nm-connection-editor` do not have the input box for user to enter the expected hostname, which still affects certain long-term support distributions in production environments (*e.g.*, Ubuntu 16.04 LTS). Second, although that input box was introduced since version 1.8.2 (released in July 2017), for all versions up to and including 1.18.0 (the most recent one at the time of writing), it has always been an *optional* input that users can simply ignore, without getting any warnings or error messages from the UI. Consequently, when it comes to WPA2-Enterprise connections, it is possible that many Linux users might also be susceptible to ET attacks.

D SAMPLES OF MISGUIDED INSTRUCTIONS

Here we give a few samples of misguided instructions. Depending on which OS they are targeting, all of these instructions are mapped to a security label of either (`N, n/a, N, Ob.`) or (`N, n/a, N, Cl.`). In many cases, instructions are focused on only benign scenarios, instead of educating users proper exception handling. Sometimes the instruction also casually dismiss the threats due to insecure configurations, with absolutely no regards to the possibility of ET attacks. The iOS examples below show evidence of the broken certificate validity alert leads to bad instructions and hurts the security of its users. One example below apparently confused serial number with the hash digest (fingerprint) of certificate.

[redacted]*.edu, campus Wi-Fi on Android: "CA certificate - Don't validate (the network is providing the certificate. Ignore messages about the connection not being private; you are connecting to a known, trusted network)"*

[redacted]*.edu, campus Wi-Fi on Android: "For CA Certificate: Click the drop-down and choose "Do not Validate". Your connection WILL BE secure!"*

[redacted]*.edu, campus Wi-Fi on Android: "For CA certificate leave as N/A. Note: Android 9/Pie users will see a notice indicating the connection is not secure. This may be ignored as a certificate is provided by the wireless controllers."*

[redacted]*.edu, campus Wi-Fi on Windows 10: "Windows will notify you that it can't verify the server's identity. It is safe to click connect."*

[redacted]*.edu, campus Wi-Fi on Windows 10: "Joining the* [redacted] *Wireless for the first time will present a certificate warning that reads as rather benign on Apple iOS, Mac OS X and Android devices, but worded in a very dire manner for Windows clients. The key is to connect. Accept the certificate and log in using your* [redacted] *eID and password."*

[redacted].edu, eduroam on macOS and iOS *“ Next you will see a Certificate, press Accept to accept the certificate even though it is unverified. [redacted] University provides this certificate to allow you to connect and it has been verified. ”*

[redacted].edu, campus Wi-Fi on iOS: *“ Step 5: Select ‘Trust’ in the upper right hand corner for the Certificate. Note: iOS will think this certificate is not trusted – it is ok to trust this! ”*

[redacted].sg, campus Wi-Fi on iOS: *“ Verify the certificate server ends in nus.edu.sg and the certificate is from thawte Primary Root CA. Don’t be alarmed if the page shows certificate Not Verified. Accept the certificate and proceed. ”*

[redacted].za, eduroam on iOS: *“ When asked to accept the certificate, tap View Certificate Details then More Details. Check that the certificate was issued by DigiCert and that the serial number is 0[redacted]D ”*

E FINDINGS ON WEAK CAT PROFILES

Table E1 to E3 show the findings discussed in Section 6.1 to 6.3. The names of the IdP have been redacted to protect the corresponding TEIs from potential attacks.

Table E1: Vulnerable Chrome OS CAT profiles due to γ = N

IdP (ISO 3166-1-Alpha-2)	Anchoring CA(s)
[redacted] (AU)	VeriSign Class 3 Public Primary CA - G5 QuoVadis Root CA 2 G3
[redacted] (UK)	QuoVadis Root CA 2 G3
[redacted] (CA)	Entrust Root CA - G2
[redacted] (IE)	DigiCert Assured ID Root CA
[redacted] (IT)	DigiCert Assured ID Root CA
[redacted] (AT)	Comodo AAA Certificate Services DigiCert High Assurance EV Root CA DigiCert Assured ID Root CA
[redacted] (FR)	DigiCert Assured ID Root CA
[redacted] (TR)	Go Daddy Class 2 CA
[redacted] (UK)	QuoVadis Root CA 2 G3
[redacted] (US)	GlobalSign Root CA - R3
[redacted] (US)	DigiCert High Assurance EV Root CA
[redacted] (FR)	Comodo AAA Certificate Services

Table E2: CAT profiles with unspecific server names

IdP (ISO 3166-1-Alpha-2)	matching	Anchoring CA(s)
[redacted] (GR) †	“[redacted] Server Certificate”	DigiCert Assured ID Root CA
[redacted] (US) ‡	“radius1”	GlobalSign Root CA
[redacted] (AU) ‡	“cit-ias-ml1”	VeriSign Class 3 Public Primary CA - G5 QuoVadis Root CA 2 G3 QuoVadis Root CA 2

† Applicable to Chrome OS, Windows 10, macOS, and Android 10
‡ Applicable to Windows 10, macOS, and Android 10

Table E3: Vulnerable Chrome OS CAT profiles due to permissive hostname constraints

IdP (ISO 3166-1-Alpha2)	matching	Anchoring CA(s)
[redacted] (BE) †	t.be	DigiCert Assured ID Root CA
[redacted] (NO)	.no	Comodo AAA Certificate Services
[redacted] (FR)	.fr	DigiCert Assured ID Root CA
[redacted] (UK)	.ac.uk	QuoVadis Root CA 2 G3
[redacted] (AT)	.at	Comodo AAA Certificate Services

† 3 profiles under the same IdP

F STATISTICS OF THE EDUROAM IDP SERVERS MEASURED

Table F4 shows the top 20 TLDs of the 3637 eduroam domains that we successfully measured during the data collection discussed in Section 7.

Table F4: Top 20 TLDs of eduroam domains measured

TLD	*Count*	TLD	*Count*	TLD	*Count*	TLD	*Count*
edu	*602*	de	*184*	ca	*119*	be	*72*
kr	*277*	hr	*145*	si	*107*	ch	*66*
uk	*261*	nl	*142*	it	*80*	cz	*65*
jp	*242*	es	*133*	br	*78*	pt	*59*
fr	*212*	cn	*129*	gr	*75*	at	*46*

G REGIONAL STATISTICS OF INSTRUCTIONS COLLECTED

Table G5 shows the top 10 regions with the most number of TEIs having applicable instructions, and Table G6 shows the top 10 regions that contributed the most number of instructions collected and labels assigned in Section 5. Each TEI might have multiple instructions, to cover all of its supported OSs. Each instruction can lead to multiple labels being assigned, depending on the OSs that it covers. Table G7 to G9 shows the majorities of labels assigned for the mainstream OSs considered, broken down based on each of their own top 10 contributing regions. We refrain from showing the table of regional majorities for Chrome OS for the sake of space. Most of its campus Wi-Fi instructions came from US schools, and most of its eduroam instructions came from European schools recommending the use of eduroam CAT, thus in both cases, the majorities for Chrome OS basically follow the results shown in Table 3.

Table G5: Top 10 regions with the highest number of TEIs having applicable instructions

ISO 3166- 1-Alpha-2	*No. of TEIs*	ISO 3166- 1-Alpha-2	*No. of TEIs*
US	*845*	KR	*94*
DE	*163*	FR	*69*
GB	*130*	IT	*66*
JP	*115*	CA	*66*
CN	*108*	TW	*59*

Table G6: Top 10 regions with the highest number of instructions/labels

Campus Wi-Fi		*eduroam*		*Campus Wi-Fi*		*eduroam*	
ISO 3166-1-Alpha-2	*No. of Instructions*	ISO 3166-1-Alpha-2	*No. of Instructions*	ISO 3166-1-Alpha-2	*No. of Labels*	ISO 3166-1-Alpha-2	*No. of Labels*
US	*1907*	US	*823*	US	*3689*	US	*2030*
JP	*242*	DE	*612*	KR	*494*	DE	*1052*
KR	*240*	GB	*405*	JP	*383*	GB	*817*
CA	*197*	CA	*195*	CA	*298*	JP	*507*
TW	*166*	FR	*193*	TW	*267*	FR	*464*
DE	*113*	JP	*162*	CN	*203*	CN	*419*
IT	*103*	CN	*125*	DE	*174*	CA	*383*
TH	*77*	IT	*125*	IT	*147*	IT	*343*
HK	*71*	AU	*101*	TH	*107*	AU	*214*
CN	*60*	AT	*85*	HK	*104*	IE	*182*

Table G7: Majorities of labels assigned for Windows 10 and Windows 7 by region

Windows 10 Campus Wi-Fi

ISO 3166-1-Alpha-2	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
US	*531*	N, n/a, N, Ob.	*82.5%*	Installer only	*13.2%*
KR	*75*	N, n/a, N, Ob.	*57.3%*	Installer only	*42.7%*
JP	*58*	N, n/a, N, Ob.	*86.2%*	SHA-1 FP	*12.1%*
CA	*39*	N, n/a, N, Ob.	*87.2%*	Installer only	*12.8%*
TW	*36*	N, n/a, N, Ob.	*86.1%*	Installer only	*13.9%*
CN	*28*	N, n/a, N, Ob.	*96.4%*	Installer only	*3.57%*
DE	*25*	N, n/a, N, Ob.	*84.0%*	SHA-1 FP	*12.0%*
IT	*18*	N, n/a, N, Ob.	*83.3%*	Installer only	*11.1%*
HK	*16*	N, n/a, N, Ob.	*87.5%*	Installer only	*6.25%*
AU	*13*	N, n/a, N, Ob.	*69.2%*	N, n/a, M, Ob.	*23.1%*

Windows 10 eduroam

ISO 3166-1-Alpha-2	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
US	*311*	N, n/a, N, Ob.	*67.2%*	Installer (not CAT) only	*15.1%*
DE	*152*	eduroam CAT only	*44.1%*	N, n/a, N, Ob.	*41.4%*
GB	*116*	N, n/a, N, Ob.	*54.3%*	eduroam CAT only	*32.8%*
JP	*85*	N, n/a, N, Ob.	*85.9%*	SHA-1 FP	*5.88%*
FR	*64*	eduroam CAT only	*50.0%*	N, n/a, N, Ob.	*43.8%*
CN	*59*	N, n/a, N, Ob.	*96.6%*	Installer (not CAT) only	*3.39%*
CA	*56*	N, n/a, N, Ob.	*76.8%*	eduroam CAT only	*17.9%*
IT	*50*	N, n/a, N, Ob.	*50.0%*	eduroam CAT only	*48.0%*
AU	*29*	N, n/a, N, Ob.	*82.8%*	N, n/a, M, Ob.	*10.3%*
IE	*26*	N, n/a, N, Ob.	*53.8%*	eduroam CAT only	*42.3%*

Windows 7 Campus Wi-Fi

ISO 3166-1-Alpha-2	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
US	*492*	P, Sys., N, Ob.	*61.4%*	N, n/a, N, Ob.	*19.5%*
KR	*79*	Installer only	*58.2%*	N, n/a, N, Ob.	*34.2%*
TW	*37*	N, n/a, N, Ob.	*56.8%*	P, Sys., N, Ob.	*32.4%*
JP	*36*	N, n/a, N, Ob.	*41.7%*	P, Sys., N, Ob.	*38.9%*
CA	*36*	P, Sys., N, Ob.	*44.4%*	N, n/a, N, Ob.	*22.2%*
CN	*30*	N, n/a, N, Ob.	*53.3%*	P, Sys., N, Ob.	*43.3%*
DE	*26*	P, Sys., N, Ob.	*65.4%*	N, n/a, N, Ob.	*23.1%*
IT	*20*	N, n/a, N, Ob.	*40.0%*	P, Sys., N, Ob.	*35.0%*
TH	*12*	N, n/a, N, Ob.	*75.0%*	Installer only	*16.7%*
HK	*11*	AM, Sp., M, Ob.	*54.5%*	P, Sys., N, Ob.	*18.2%*

Windows 7 eduroam

ISO 3166-1-Alpha-2	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
US	*267*	P, Sys., N, Ob.	*59.9%*	Installer (not CAT) only	*16.5%*
DE	*136*	eduroam CAT only	*51.5%*	P, Sys., N, Ob.	*37.5%*
GB	*100*	P, Sys., N, Ob.	*43.0%*	eduroam CAT only	*36.0%*
CN	*67*	N, n/a, N, Ob.	*67.2%*	P, Sys., N, Ob.	*26.9%*
FR	*61*	eduroam CAT only	*44.3%*	P, Sys., N, Ob.	*31.1%*
CA	*59*	P, Sys., N, Ob.	*50.8%*	eduroam CAT only	*16.9%*
IT	*48*	eduroam CAT only	*52.1%*	P, Sys., N, Ob.	*31.2%*
AU	*29*	P, Sys., N, Ob.	*62.1%*	N, n/a, N, Ob.	*10.3%*
IE	*26*	P, Sys., N, Ob.	*42.3%*	eduroam CAT only	*34.6%*
NO	*25*	P, Sys., N, Ob.	*52.0%*	eduroam CAT only	*28.0%*

Table G8: Majorities of labels assigned for Android 7+ and Android 6- by region

Android 7+ Campus Wi-Fi

ISO 3166-1-Alpha-2	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
US	*591*	N, n/a, N, Ob.	*43.8%*	N, n/a, N, Cl.	*39.4%*
KR	*86*	N, n/a, N, Cl.	*76.7%*	N, n/a, N, Ob.	*23.3%*
JP	*59*	N, n/a, N, Ob.	*47.5%*	N, n/a, N, Cl.	*35.6%*
TW	*49*	N, n/a, N, Cl.	*59.2%*	N, n/a, N, Ob.	*38.8%*
CA	*42*	N, n/a, N, Ob.	*50.0%*	N, n/a, N, Cl.	*31.0%*
CN	*36*	N, n/a, N, Cl.	*61.1%*	N, n/a, N, Ob.	*38.9%*
DE	*28*	P, Sp., N, Ob.	*39.3%*	N, n/a, N, Ob.	*21.4%*
IT	*20*	N, n/a, N, Ob.	*60.0%*	N, n/a, N, Cl.	*25.0%*
TH	*20*	N, n/a, N, Cl.	*50.0%*	N, n/a, N, Ob.	*45.0%*
HK	*17*	N, n/a, N, Ob.	*70.6%*	P, Sys., P, Ob.	*17.6%*

Android 7+ eduroam

ISO 3166-1-Alpha-2	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
US	*313*	N, n/a, N, Ob.	*32.6%*	N, n/a, N, Cl.	*29.1%*
DE	*148*	eduroam CAT only	*40.5%*	P, Sp., N, Ob.	*20.9%*
GB	*118*	N, n/a, N, Ob.	*45.8%*	eduroam CAT only	*25.4%*
JP	*80*	N, n/a, N, Ob.	*71.2%*	P, Sys., P, Ob.	*10.0%*
CN	*80*	N, n/a, N, Cl.	*80.0%*	N, n/a, N, Ob.	*20.0%*
FR	*61*	eduroam CAT only	*47.5%*	N, n/a, N, Cl.	*27.9%*
CA	*57*	N, n/a, N, Cl.	*38.6%*	N, n/a, N, Ob.	*35.1%*
IT	*47*	eduroam CAT only	*38.3%*	N, n/a, N, Ob.	*36.2%*
AU	*31*	N, n/a, N, Ob.	*58.1%*	N, n/a, N, Cl.	*29.0%*
IE	*27*	N, n/a, N, Ob.	*44.4%*	eduroam CAT only	*25.9%*

Android 6- Campus Wi-Fi

ISO 3166-1-Alpha-2	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
US	*588*	N, n/a, N, Ob.	*49.3%*	N, n/a, N, Cl.	*42.0%*
KR	*86*	N, n/a, N, Cl.	*76.7%*	N, n/a, N, Ob.	*23.3%*
JP	*60*	N, n/a, N, Ob.	*56.7%*	N, n/a, N, Cl.	*38.3%*
TW	*49*	N, n/a, N, Cl.	*59.2%*	N, n/a, N, Ob.	*40.8%*
CA	*42*	N, n/a, N, Ob.	*57.1%*	N, n/a, N, Cl.	*33.3%*
CN	*37*	N, n/a, N, Cl.	*59.5%*	N, n/a, N, Ob.	*40.5%*
DE	*27*	P, Sp., N, Ob.	*51.9%*	N, n/a, N, Ob.	*25.9%*
IT	*23*	N, n/a, N, Ob.	*65.2%*	N, n/a, N, Cl.	*26.1%*
TH	*20*	N, n/a, N, Cl.	*50.0%*	N, n/a, N, Ob.	*45.0%*
HK	*18*	N, n/a, N, Ob.	*94.4%*	N, n/a, N, Cl.	*5.56%*

Android 6- eduroam

ISO 3166-1-Alpha-2	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
US	*310*	N, n/a, N, Ob.	*41.0%*	N, n/a, N, Cl.	*34.2%*
DE	*146*	eduroam CAT only	*41.1%*	P, Sp., N, Ob.	*34.2%*
GB	*117*	N, n/a, N, Ob.	*49.6%*	eduroam CAT only	*28.2%*
CN	*80*	N, n/a, N, Cl.	*80.0%*	N, n/a, N, Ob.	*20.0%*
JP	*79*	N, n/a, N, Ob.	*81.0%*	N, n/a, N, Cl.	*10.1%*
FR	*63*	eduroam CAT only	*41.3%*	N, n/a, N, Cl.	*28.6%*
CA	*54*	N, n/a, N, Cl.	*42.6%*	N, n/a, N, Ob.	*37.0%*
IT	*48*	N, n/a, N, Ob.	*37.5%*	eduroam CAT only	*37.5%*
AU	*31*	N, n/a, N, Ob.	*64.5%*	N, n/a, N, Cl.	*32.3%*
IE	*27*	N, n/a, N, Ob.	*44.4%*	eduroam CAT only	*22.2%*

Table G9: Majorities of labels assigned for macOS and iOS by region

macOS Campus Wi-Fi

ISO 3166-1-Alpha-2	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
US	*551*	N, n/a, N, Cl.	*80.4%*	Installer only	*11.3%*
JP	*54*	N, n/a, N, Cl.	*75.9%*	N, n/a, M, Cl.	*16.7%*
KR	*47*	N, n/a, N, Cl.	*95.7%*	Installer only	*4.26%*
CA	*42*	N, n/a, N, Cl.	*78.6%*	N, n/a, M, Cl.	*16.7%*
TW	*23*	N, n/a, N, Cl.	*82.6%*	N, n/a, M, Cl.	*8.7%*
IT	*22*	N, n/a, N, Cl.	*86.4%*	Installer only	*9.09%*
DE	*22*	N, n/a, N, Cl.	*72.7%*	N, n/a, M, Cl.	*13.6%*
CN	*18*	N, n/a, N, Cl.	*100%*	n/a	*n/a*
HK	*16*	N, n/a, N, Cl.	*75.0%*	N, n/a, M, Cl.	*25.0%*
GB	*16*	N, n/a, N, Cl.	*81.2%*	N, n/a, M, Cl.	*12.5%*

macOS eduroam

ISO 3166-1-Alpha-2	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
US	*308*	N, n/a, N, Cl.	*65.9%*	Installer (not CAT) only	*15.9%*
DE	*143*	eduroam CAT only	*58.0%*	N, n/a, N, Cl.	*23.8%*
GB	*116*	N, n/a, N, Cl.	*51.7%*	eduroam CAT only	*31.0%*
JP	*82*	N, n/a, N, Cl.	*82.9%*	N, n/a, M, Cl.	*6.1%*
FR	*63*	eduroam CAT only	*55.6%*	N, n/a, N, Cl.	*30.2%*
CA	*58*	N, n/a, N, Cl.	*70.7%*	eduroam CAT only	*13.8%*
IT	*49*	eduroam CAT only	*51.0%*	N, n/a, N, Cl.	*42.9%*
CN	*42*	N, n/a, N, Cl.	*100%*	n/a	*n/a*
AU	*32*	N, n/a, N, Cl.	*81.2%*	N, n/a, M, Cl.	*12.5%*
IE	*27*	N, n/a, N, Cl.	*51.9%*	eduroam CAT only	*37.0%*

iOS Campus Wi-Fi

ISO 3166-1-Alpha-2	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
US	*562*	N, n/a, N, Cl.	*80.2%*	Installer only	*9.61%*
KR	*84*	N, n/a, N, Cl.	*100%*	n/a	*n/a*
JP	*61*	N, n/a, N, Cl.	*72.1%*	N, n/a, M, Cl.	*19.7%*
TW	*50*	N, n/a, N, Cl.	*96.0%*	Installer only	*2.0%*
CA	*42*	N, n/a, N, Cl.	*71.4%*	N, n/a, M, Cl.	*21.4%*
CN	*33*	N, n/a, N, Cl.	*97.0%*	N, n/a, M, Cl.	*3.03%*
DE	*19*	N, n/a, N, Cl.	*68.4%*	Installer only	*15.8%*
TH	*19*	N, n/a, N, Cl.	*89.5%*	N, n/a, M, Cl.	*5.26%*
HK	*18*	N, n/a, N, Cl.	*55.6%*	N, n/a, M, Cl.	*44.4%*
GB	*16*	N, n/a, N, Cl.	*75.0%*	N, n/a, M, Cl.	*25.0%*

iOS eduroam

ISO 3166-1-Alpha-2	*Total*	1st Majority	*Perc.*	2nd Majority	*Perc.*
US	*315*	N, n/a, N, Cl.	*61.9%*	Installer (not CAT) only	*14.6%*
DE	*144*	eduroam CAT only	*58.3%*	N, n/a, N, Cl.	*19.4%*
GB	*119*	N, n/a, N, Cl.	*52.1%*	eduroam CAT only	*27.7%*
JP	*80*	N, n/a, N, Cl.	*78.8%*	N, n/a, M, Cl.	*11.2%*
CN	*70*	N, n/a, N, Cl.	*100%*	n/a	*n/a*
FR	*60*	eduroam CAT only	*63.3%*	N, n/a, N, Cl.	*20.0%*
CA	*57*	N, n/a, N, Cl.	*68.4%*	eduroam CAT only	*17.5%*
IT	*43*	eduroam CAT only	*55.8%*	N, n/a, N, Cl.	*34.9%*
AU	*30*	N, n/a, N, Cl.	*80.0%*	N, n/a, M, Cl.	*13.3%*
IE	*27*	N, n/a, N, Cl.	*51.9%*	eduroam CAT only	*37.0%*