The Devil is in the Details: Hidden Problems of Client-Side Enterprise Wi-Fi Configurators

Ka Lok Wu
Department of Information Engineering
The Chinese University of Hong Kong
Sha Tin, Hong Kong
klwu@link.cuhk.edu.hk

Man Hong Hue*
Georgia Institute of Technology
Atlanta, GA, USA
hugohue@gatech.edu

Ka Fun Tang
Department of Information Engineering
The Chinese University of Hong Kong
Sha Tin, Hong Kong
1155126139@link.cuhk.edu.hk

Sze Yiu Chau
Department of Information Engineering
The Chinese University of Hong Kong
Sha Tin, Hong Kong
sychau@ie.cuhk.edu.hk

ABSTRACT

In the context of connecting to enterprise Wi-Fi, previous works show that relying on human users to manually configure or enforce server authentication often leads to insecure outcomes. Consequently, many user credentials can potentially be stolen by the so-called "Evil-Twin" (ET) attack. To ease the burden of human users, various easy-to-use Wi-Fi configurators have been released and deployed. In this work, we investigate whether such configurators can indeed protect users from variants of the ET attack. To our surprise, the results of our investigation show that all configurators considered in the study suffer from certain weaknesses due to their design, implementation, or deployment practices. Notable findings include a series of design flaws in the new trust-on-first-use (TOFU) configurator on Android (available since version 12), which can be exploited in tandem to achieve a stealthy ET attack. Moreover, we found that 2 open-source Android Wi-Fi configurators fail to properly enforce server authentication under specific situations. The cause of these could be partly attributed to the complexity stemmed from certificate name matching as well as the limitations of the Android API. Last but not least, we found that a commercial configurator not only allows insecure Wi-Fi configurations to be deployed, but also the covert injection of certificates on the user device to facilitate interception of other TLS traffic, posing yet another hidden security and privacy threat to its users. All in all, this study shows that despite years of research on the topic, developing a user-friendly yet reliable Wi-Fi configurator remains an elusive goal, and thus the threat of ET attacks continues to be relevant. As such, it is time to rethink whether the complexity of the standard certificate chain validation is actually good for enterprise Wi-Fi.

*Work done while the author was at The Chinese University of Hong Kong.

WiSec '23, May 29-June 1, 2023, Guildford, United Kingdom

ACM ISBN 978-1-4503-9859-6/23/05…$15.00
https://doi.org/10.1145/3558482.3590199

CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**; **Software security engineering**; *Authentication.*

KEYWORDS

WPA Enterprise, Evil-Twin, Authentication, TLS, Trust-on-first-use

ACM Reference Format:
Ka Lok Wu, Man Hong Hue, Ka Fun Tang, and Sze Yiu Chau. 2023. The Devil is in the Details: Hidden Problems of Client-Side Enterprise Wi-Fi Configurators. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '23), May 29-June 1, 2023, Guildford, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3558482.3590199

1 INTRODUCTION

Wi-Fi is a near-ubiquitous technology that enables Internet connectivity to countless devices. Depending on how the user gets authenticated, a Wi-Fi setup can be classified as the *personal* mode, where authentication is achieved using a pre-shared key (PSK), or the *enterprise* mode, where the authentication is done via the IEEE 802.1X standard. Comparatively, the enterprise mode enables a more fine-grained authorization and accounting, and is thus more commonly used by large companies and educational institutes. In fact, many organizations take advantage of IEEE 802.1X to reuse existing single sign-on (SSO) credentials for accessing their enterprise Wi-Fi. Unfortunately, this also makes enterprise Wi-Fi a high-value target for attackers, as stolen credentials can enable access to other resources of the victim organization.

Typical in such setups, the *client device*, also known as the *supplicant*, would establish a TLS tunnel with the *authentication server* to protect the password-based user authentication. Unfortunately, since the SSID itself is not cryptographically verifiable, an attacker can launch a Wi-Fi setup broadcasting the same SSID, known as an Evil Twin (ET), and trick nearby supplicants into connecting. In this case, it is crucial for the *supplicant* to properly validate the identity of the *authentication server*. Otherwise, the supplicant could inadvertently perform the user authentication exchange with an ET impersonator, and hand over the sensitive user credentials to an attacker. Once the attacker obtained the credentials, further access to and attacks against other organizational resources could

be possible (see for instance, [1]), especially when SSO is deployed at the victim's organization.

Similar to many other TLS applications, in most entperise Wi-Fi setups, the validation of server identity is based on the supplicant validating the certificate chain sent by the authentication server. However, previous work found that many organizations fail to prescribe secure Wi-Fi configurations that enforce proper certificate validation and are thus vulnerable to ET-style credential theft, in part due to the counter-intuitive designs and implementations of the configuration user interface (UI) found on mainstream OSes [4, 15, 23]. In light of this worrisome situation, many new configurators have been designed and deployed to help streamline the enterprise Wi-Fi configuration process for users. For instance, Android has introduced a new configurator since version 12, which is easier to use than its conventional UI. Similarly, several configurator apps exist, which can help users to configure enterprise Wi-Fi by loading some network profiles pre-configured by IT admins. Although previous work considered potential configuration issues in the profiles used by some configurators [3, 15], the robustness of the configurators themselves in terms of enforcing security policies and protecting user credentials remain unclear.

In this work, we consider several new or popular configurators that are already deployed in practice, and investigate if they can indeed achieve the security guarantees expected from them, that is, help users arrive at a secure enterprise Wi-Fi configuration and thus prevent ET-style credential theft. Specifically, we focus on the design and implementation issues hidden in the configurators considered. For this, we rely on adaptive dynamic testing, manual code review, and in some cases reverse engineering, to evaluate and understand different aspects of such configurators. Notable findings include several design flaws in the new trust-on-first-use (TOFU) configurator on Android that makes it susceptible to ET-style credential theft, as well as implementation issues that render other configurators ineffective in *enforcing* the policies stipulated by IT admins. Along the way of presenting our findings, we also discuss their root causes as well as the possible ways of avoiding pitfalls and improving the overall security of enterprise Wi-Fi. In particular, the numerous problems stemmed from conventional certificate validation prompts us to question whether that approach is really beneficial to and necessary for enterprise Wi-Fi.

Overall, this paper makes the following technical contributions:

(1) We provide the first security evaluation of the new TOFU configurator on Android 12, and explain its various design flaws that, when exploited in tandem, could enable a stealthy credential theft.
(2) We test two open-source Wi-Fi configurators used by thousands of organizations to setup eduroam and other enterprise Wi-Fi on client devices, and dissect the root causes of their failures in enforcing critical checks under certain conditions.
(3) We investigate the attributes supported by the ChromeOS built-in configurators, and provide a viable explanation to some of the insecure profiles observed by previous work [15].
(4) We evaluate the pre-configured profiles of a commercial Wi-Fi configurator not studied by previous work, and reveal their configuration problems and other concerning behaviors that can threaten user privacy.
(5) Based on empirical evidence, we discuss how the confusing and restrictive API design of the underlying OS also plays a part in spreading insecure Wi-Fi configurations, as well as some practical considerations on TOFU and standard certificate validation in the context of enterprise Wi-Fi.

2 PRELIMINARIES

2.1 Authentications in enterprise Wi-Fi

Under the IEEE 802.1X standard, enterprise Wi-Fi uses the Extensible Authentication Protocol (EAP), which is a generic framework that allows one to build authentication methods. Historically, many such methods exist [13], however, most contemporary production deployments use TLS on top of EAP to provide the necessary security guarantees over the authentication exchange. Popular EAP methods such as *PEAP* and *TTLS* are both variants of this. Under this model, TLS provides end-to-end guarantees between the supplicant and the authentication server, with the wireless access point relaying messages between the two.

2.1.1 Server authentication. Just like any other applications of TLS, in enterprise Wi-Fi, sever authentication under the EAP with TLS model typically hinges on the validation of the server certificate. There are different approaches of how this validation can happen. For example, a system might implement the conventional certificate chain validation as outlined in [8]. To achieve the necessary authentication guarantees following this approach, it is critical that ❶ a *chain of trust* can be formed from some predefined trust anchors by verifying the digital signatures of each certificate on the chain, and ❷ the end-entity server certificate has a *correct name*, either in the `Subject` field or the `subjectAltName` extension of the certificate, that matches the hostname of the authentication server. Alternatively, a system can also use *pinning* to validate the server certificate. This is often done using a ❸ trust-on-first-use (TOFU) approach, where the server certificate might be *manually inspected* the first time it is seen or when a discrepancy is encountered, and will then be memorized for future use.

Vulnerability to the ET attacks can then be explained with respect to the approach used to validate the server certificate. Under the conventional chain validation approach, if ❶ is skipped, then any certificates, including the ones generated by an attacker, could be accepted by the system. And depending on the trust anchors used in ❶, failures in properly performing ❷ could allow the system to accept certificates from a trusted commercial CA issued to domains controlled by an attacker. Alternatively, if the manual inspection in ❸ is skipped, the outcome would effectively be the same as skipping ❶. So long as an ET attacker can obtain a certificate that will be accepted by the supplicant, the user could be tricked into performing client authentication with an impersonating ET setup, thus enabling to credential theft.

2.1.2 User authentication. The actual implication of the ET credential theft depends on how user authentication is performed. A typical enterprise Wi-Fi setup, especially when SSO is used, performs some password-based user authentication[1]. This is commonly known as the phase-2 authentication (whereas the establishment of

[1]In the EAP-TLS method, users can also be authenticated using certificates during the TLS handshake, but this has seen limited usage in practice [13].

TLS is the phase-1). Depending on the phase-1 EAP method, not all phase-2 methods are supported. Commonly used phase-2 methods include the PAP and GTC, as well as MSCHAPv2. When the server authentication is broken in phase-1, the use of PAP and GTC would allow the ET attacker to obtain the user password directly. For MSCHAPv2, the attacker can instead record the challenge-response transcript, and then either mount a dictionary attack to recover the password [18], or spend at most 2^{56} attempts to brute-force the MD4 hash of the password, which would allow the attacker to login as the user in future. Given that these commonly used phase-2 authentication methods fail to protect user credentials, the server certificate validation in phase-1 is of utmost importance in preventing ET-style credential compromise. Unfortunately, as we will show in later sections, despite their ease of use, many configurators fail to adequately enforce this critical validation.

2.2 Experiment setup

To evaluate the design and implementation of different Wi-Fi configurators, we built an ET-style test platform. Specifically, we used a decade-old laptop to run Ubuntu and the open-source `hostapd-wpe` software package, which is a patched version of `hostapd` that implements the ET attack. This allows us to configure various server parameters (*e.g.*, certificate, ciphersuites, phase-2 methods, etc.), and obtain user credentials if the attack succeeds. Given that this portable attack setup can be easily built with off-the-shelf commodity hardware and software components, the ET attack is a very realistic threat that cannot be ignored.

Our experiments are performed in a controlled environment. We did not collect any real user credentials, and only used made-up credentials in our testing. For the server certificates, depending on the experiment, we either use a self-signed certificate chain, or the ones that we purchased from a commercial CA and were issued to a domain under our control. Since the API related to enterprise Wi-Fi varies significantly across Android versions, we used several devices (running different OS versions) available at our disposal for testing. And unless explicitly stated otherwise, in the rest of this paper, all the testing for Windows and macOS was done on Windows 10 and macOS 11 Big Sur respectively.

3 NEW TOFU CONFIGURATOR ON ANDROID

Our first investigation concerns the new TOFU configurator on Android, introduced since version 12. Unlike the conventional configuration UI on older versions of Android where the user has to choose the trust anchor for ❶ and type in the server hostname for ❷, this new TOFU configurator follows approach ❸ and relies on human users to manually check the certificate information. A screenshot of it can be found in Figure 1. When a user first connects to an enterprise Wi-Fi network, a prompt showing some attributes of the certificate chain is shown to the user, and then the user has to decide whether to accept the certificate chain. If the user chooses to continue with the connection, certain attributes of the certificate chain will then be pinned. Although it appears to be a functional and reasonable design at first glance, we discovered several weaknesses in this TOFU configurator in publicly released versions of Android. We assume the goals of an ET-style attacker to be ① stealing user credentials at first use, and ② making the system memorize the correct information for future use, so that the attack becomes stealthy. By exploiting the Android weaknesses in tandem, both attack goals can be achieved.

Using the test platform described in Section 2.2 with several certificate chains strategically crafted in an adaptive manner, we black-box tested Android 13 on a Pixel 6, and constructed a state machine of the TOFU configurator, which is shown in Figure 2. For each transition in the state machine, the text before the / separator describes the triggering condition, and the text after shows the action taken by the Android system. The problematic transition actions are shown in red color.

Send first, verify later. The first major flaw in Android's new TOFU configurator is that when the user connects to an enterprise Wi-Fi network for the first time, the credentials are sent to the authentication server *before* the manual inspection prompt is shown to the user. This is a severe design oversight, as an ET attack setup that is already in place during the first connect could trivially achieve attack goal ①.

What you see is not what you expect. A second flaw in the TOFU configurator concerns the prompt shown to the user for manual inspection of information regarding the certificate chain. As shown in Figure 1, the prompt displays the following attributes of the *highest-level* certificate on the chain that the system received from the authentication server: (i) subject name, (ii) issuer name, (iii) organization name, and (iv) *prefix* of its *digital signature*.

Interestingly, these attributes are not extracted from the end-entity server certificate, but from the highest-level certificate on the chain. We argue that this is a misguided design decision. In many production setups, the certificate chain are issued by a commercial CA. As such, the highest-level certificate on the chain sent by the authentication server would most likely be of some commercial intermediate CA (*e.g.*, *Sectigo RSA Domain Validation Secure Server CA* in Figure 1). In this case, it is impossible for users to properly enforce ❸. Because the attributes shown are not tied to the server in any meaningful ways, an attacker can simply go to the same intermediate CA and purchase a certificate chain for an arbitrary domain under control, and then use that chain in the ET attack setup. The user would then see the exact same attributes and be tricked into connecting, thus fulfilling attack goal ①. In other words, unless an organization chooses a dedicated private CA over commercial ones for its enterprise Wi-Fi, it is impossible for the IT admin to prescribe a Wi-Fi setup guide using this new TOFU configurator while protecting the users from ET-style attacks, even if the first flaw (*send first, verify later*) is fixed.

What you see is not necessarily authentic. More crucially, with our black-box testing, we found that the signatures on the certificate chain are *not verified* before the prompt is shown. Notice that all four attributes shown on the prompt are *directly* from the certificate chain, and none are computed locally by the Android system. As such, the lack of signature verification means that all fields and extensions on the certificates, including the four attributes shown on the prompt, are practically *unauthenticated inputs* that can be arbitrarily chosen by the attacker. In other words, an ET attacker can prepare an attack certificate chain by replicating the one used by the authentication server of the target organization, and then replacing the genuine public key on the server certificate with one

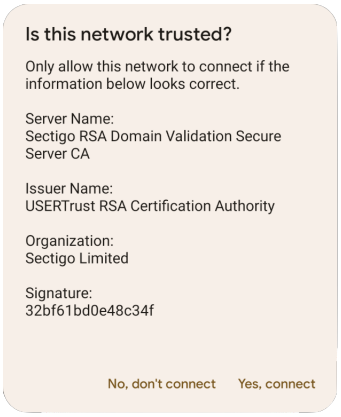

Figure 1: The TOFU prompt. All 4 attributes shown on this prompt can be spoofed by an attacker.

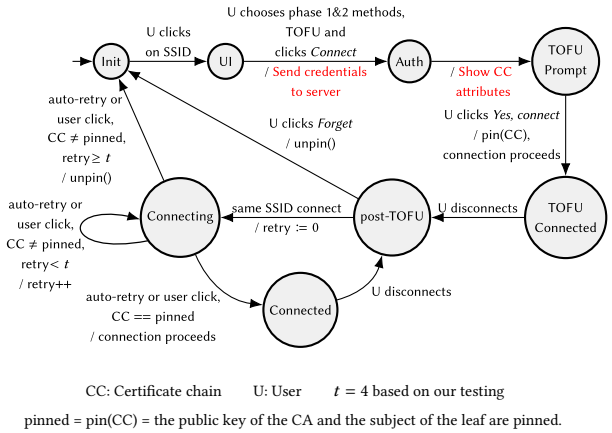

Figure 2: State machine for Android TOFU setup of one SSID

that has a known private key (*e.g.*, a key pair generated by the attacker). This way, even if the previous two flaws are fixed and the prompt actually shows attributes of the server certificate, the user could still be tricked into connecting to an ET attack setup *after* manually verifying the four attributes shown on the prompt, thus fulfilling attack goal ① once again.

What you see is not what it pins. Using our our test platform, we discovered another interesting finding that, when exploited together with the previous flaws, could allow an ET attacker to achieve both attack goals ① and ②.

We found that if the user continues to connect after seeing the prompt, the system would extract from the received chain and memorize only (i) the public key of the highest-level (CA) certificate, and (ii) the server name from the leaf certificate. Then, in future connection to the same Wi-Fi network, the system would use the conventional chain validation with the memorized information for ❶ and ❷. In other words, the server public key itself is not memorized. Because of this, the attack certificate chain described in the previous flaw not only tricks the users but also helps the system to memorize correct information for future use. Thus when the user connects to the legitimate Wi-Fi in the future, the connection will proceed as normal, and the first-use ET attack goes undetected.

Attack-induced first-use. The attacks described above assume the ET setup is already in place when the user performs the TOFU configuration. One might question whether this is a realistic assumption, or does it limit the practicality of the attacks. Based on our testing, it turns out to be relatively easy for an attacker to induce the Android system to go back to the first-use state. As shown in the left-hand side of Figure 2, the Android system automatically unpins an existing Wi-Fi TOFU setup after failing and retrying to connect for too many times. As such, an attacker can place an ET setup nearby, and once the connection fails for enough times, the user will be forced to perform TOFU again.

Reflection: Designing an enterprise Wi-Fi configurator is not only about the ease of use, and achieving the expected security outcome requires careful analysis and testing. Unfortunately, Android's new TOFU falls short in the latter.

4 OPEN-SOURCE WI-FI CONFIGURATORS

In this section, we discuss the findings on 2 popular open-source configurators that stemmed from the eduroam project, which is an international roaming service that provides Wi-Fi access to traveling users affiliated with educational institutions. Both configurators help users to configure their device for enterprise Wi-Fi by programmatically loading some pre-configured profiles uploaded by IT admins. After a successful configuration, server authentication would then happen through conventional chain validation, with ❶ and ❷ using the parameters stipulated in the profile. Apart from the eduroam network, some organizations also use these configurators for their own Wi-Fi networks.

4.1 Flaws in the eduroam CAT Android app

The first open-source configurator that we consider is the eduroam Configuration Assistant Tool (CAT), which is available on most mainstream operating systems. On Google Play alone, the eduroam CAT app has been downloaded for more than one million times. Previous work investigated and discovered several weaknesses in the *policy* (configuration parameters) stipulated in the pre-configured profiles [15], however, they did not consider whether the eduroam CAT configurator can actually *enforce* those policies. With our test platform and manual code review, we found a few implementation flaws in the current version of eduroam CAT, which could lead to improper enforcement of otherwise strong policies under certain situations, thus opening door to the ET-style credential theft.

We found that under certain situations, the eduroam CAT Android app could weaken the server name checking constraint used in ❷. The eduroam CAT app fetches and parses the pre-configured profile of an organization chosen by the user, and then invokes various methods of the `WifiEnterpriseConfig` class from the Android API [2] to actually setup the Wi-Fi connection on the client device. The main problem is that eduroam CAT uses the `setSubjectMatch()` method, which was introduced in Android API level 18 but deprecated since level 23 [2], to set the final server name matching constraint[2].

[2]https://github.com/GEANT/CAT-Android/blob/f1053d54/src/uk/ac/swansea/eduroamcat/WifiConfigAPI18.java#L204

Unlike the newer `setAltSubjectMatch()` method introduced since API level 23 which can handle multiple server names separated by semicolons, the `setSubjectMatch()` method can only handle one single server name. Because of that, when a pre-configured profile stipulates multiple server names, eduroam CAT uses an improvised algorithm to compute a *common suffix* that can be set with `setSubjectMatch()`. The code for this algorithm can be found in line 176–210 of the `WifiConfigAPI18.java` file[3] in the eduroam CAT source tree, which is also shown in Listing 1.

```
176 if (aAuth.getServerIDs().size()>0)
177 {
178   String subjectMatch="";
179   if (aAuth.getServerIDs().size()>1)
180   {
181     String subjectMatch_next=aAuth.getServerIDs().get(0);
182     String subjectMatch_new="";
183     if (subjectMatch_next.indexOf(".")>0){
184       subjectMatch_new=subjectMatch_next.substring(
          subjectMatch_next.indexOf("."));
185       subjectMatch=subjectMatch_new;
186     }
187     for (int serverCount=0; serverCount<aAuth.getServerIDs()
          .size(); serverCount++)
188     {
189       subjectMatch_next=aAuth.getServerIDs().get(serverCount
          );
190       if (subjectMatch_next.indexOf(".")>0)
191       subjectMatch_new=subjectMatch_next.substring(
          subjectMatch_next.indexOf("."));
192       if (subjectMatch.equals(subjectMatch_new)) continue;
193       else {
194         //error with serverIDs
195         StatusFragment.setDebug("ServerID error with profile
          :"+subjectMatch+" and "+subjectMatch_new);
196         eduroamCAT.debug("ServerID error with profile:"+
          subjectMatch+" and "+subjectMatch_new);
197         subjectMatch="";
198         break;
199       }
200     }
201   }
202   else subjectMatch=aAuth.getServerIDs().get(0);
203
204   enterpriseConfig.setSubjectMatch(subjectMatch);
205   eduroamCAT.debug("subjectMatch="+subjectMatch);
206 }
```

Listing 1: algorithm for computing the name matching constraint in eduroam CAT (`WifiConfigAPI18.java`)

When multiple servers are specified in the pre-configured profile, this algorithm always peels off the leftmost subdomain (line 184 and 191), and then try to see if the remaining suffix is shared by all the server names (line 192). If so, use this suffix for name matching. If not, an empty string will be given to `setSubjectMatch()` (line 197). Because of this, if the same server name `a.b.c` and `a.b.c` is given more than once in a profile, the servername `b.c`, instead of `a.b.c`, would be set and matched. Even more interestingly, if an organization has two authentication servers on different second-level domains, for instance, `a.b.c` and `a.d.c`, then an empty string would be set and matched.

[3]https://github.com/GEANT/CAT-Android/blob/f1053d54/src/uk/ac/swansea/eduroamcat/WifiConfigAPI18.java#L176

Proof-of-concept attack. At the time of testing, we targeted the pre-configured profile of TU Graz[4], which stipulates multiple server names to be matched (`zar.tugraz.at`, `zar1.tu-graz.ac.at`, `zar1.tugraz.at`, `zar2.tu-graz.ac.at`, `zar2.tugraz.at`), and specifies the certificate of a commercial CA, Sectigo, as the trust anchor. Because of the bug outlined above, the actual server name matching constraint given to `setSubjectMatch()` would become `.at`. We then purchased a certificate from Sectigo for a `.at` domain under our control, and used it in our test platform, which runs an ET eduroam Wi-Fi. Then, we used the eduroam CAT app on an Android 11 device to load the pre-configured profile of TU Graz, and found that after configuring, the system accepts our certificate as expected, and the user credentials are indeed sent to the ET setup.

Android's loose matching logic. Worst yet, the API method `setSubjectMatch()` on Android actually performs *substring* matching instead of suffix matching. This is similar to the name matching problem on ChromeOS observed by a previous work [15]. As such, when eduroam CAT is used, any pre-configured profiles that use commercial CAs as trust anchors could be attacked by an ET attacker. The attacker could purchase a domain, then get a subdomain satisfying the server name matching constraint stipulated by a target profile, and go to the trusted commercial CA and purchase a certificate for that subdomain. Then this certificate can be used in an ET attack setup and satisfy both ❶ and the substring matching logic of ❷. We successfully reproduced this substring matching weakness with our test platform and a certificate from the Let's Encrypt CA, targeting our own pre-configured profile.

4.2 A flaw in the geteduroam Android app

The second open-source configurator that we consider is geteduroam. Similar to eduroam CAT, geteduroam also helps users to setup connection to eduroam and other Wi-Fi by fetching and loading pre-configured profiles. In fact, it shares the same profile repository with eduroam CAT, and can be regarded as the next generation of CAT. According to its official Website[5], the geteduroam Android app is intended to replace the eduroamCAT Android app.

Similar to eduroam CAT, with our test platform and manual code review, we found a flaw in geteduroam that makes it fail to properly enforce the policy stipulated in the profile under certain conditions. Specifically, in the stable release of geteduroam version `1.0.21`, the application sets different strings for server name validation according to the Android version of the devices. Listing 2, which captures line 554–573 of `WifiProfile.java` in the geteduroam source tree[6], shows the developer comment which says Android 9 only supports a single name given to the `setDomainSuffixMatch()` method. As such, an improvised algorithm is used on Android 9 to compute the longest suffix match of all the server names given in the profile. For Android versions 10 or above, the `setDomainSuffixMatch()` method supports multiple server names separated by semicolon, so this computation is not needed.

Interestingly, the common suffix computation algorithm, which is in line 287–312 of `WifiProfile.java` in the geteduroam source

[4]Only in a controlled environment, and TU Graz have since updated their profile.
[5]https://www.geteduroam.app/about/faq/
[6]https://github.com/geteduroam/ionic-app/blob/d8e08d48/geteduroam/plugins/wifi-eap-configurator/android/src/main/java/com/emergya/wifieapconfigurator/config/WifiProfile.java#L554

tree[7] and also shown in Listing 3, can output an empty string in 3 cases. First, if the multiple server names given in the profile are of different forms, for example, one is an IP address and the other is a domain name, then unsurprisingly, no common suffix can be found. Second, even if all the server names are in the profile are domain names, if they are on different top-level domains (TLDs), then no common suffix can be found. Finally and more subtly, because the algorithm does not normalize the names into lowercase before computing the suffix, domain names of different cases (*e.g.*, `A.B.C` and `p.b.c`) would also lead to an empty common suffix.

Proof-of-concept attack. At the time of testing, we targeted the pre-configured profile of Baskent Universitesi[8], and ran geteduroam on an Android 9 device. The profile specifies a commercial CA, GoDaddy, as the trust anchor, and stipulates two server names that can be accepted: `gunes.baskent.edu.tr` and `193.255.45.5`. Following the discussion above, in this case, the longest common suffix given to the `setDomainSuffixMatch()` method will simply be an empty string, effectively disabling ❷. We purchased a certificate from GoDaddy for a domain that we control, and used it in our test platform. After loading the profile, the Android system accepted our certificate, and sent the user credential to our ET setup.

```
/**
* Get the string that Android uses for server name validation.
*
* Server names are treated as suffix, but exact string match is also accepted.
*
* On Android 9, only a single name is supported.
* Thus, for Android 9, we will calculate the longest suffix match.
*
* On Android 10 and onwards, the string can be semicolon-separated,
* which is what we will do for these platforms.
*
* @return The server name
*/
private String getServerNamesDomainString() {
  if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
    return String.join(";", serverNames);
  } else {
    return getLongestSuffix(serverNames);
  }
}
```

Listing 2: geteduroam adjusts the server name string based on the Android version (`WifiProfile.java`)

```
/**
* Get the longest common suffix domain components from a list of hostnames
*
* @param strings A list of host names
* @return The longest common suffix for all given host names
*/
static String getLongestSuffix(String[] strings) {
  if (strings.length == 0) return "";
  if (strings.length == 1) return strings[0];
  String longest = strings[0];
  for (String candidate : strings) {
    int pos = candidate.length();
    do {
      pos = candidate.lastIndexOf('.', pos - 2) + 1;
    } while (pos > 0 && longest.endsWith(candidate.substring(pos)));
    if (!longest.endsWith(candidate.substring(pos))) {
      pos = candidate.indexOf('.', pos);
    }
    if (pos == -1) {
      longest = "";
    } else if (longest.endsWith(candidate.substring(pos))) {
      longest = candidate.substring(pos == 0 ? 0 : pos + 1);
    }
  }
  return longest;
}
```

Listing 3: algorithm used by geteduroam to calculate the longest common suffix of multiple hostnames (`WifiProfile.java`)

4.3 How many are affected?

To better evaluate the impact of the aforementioned flaws, we crawled 3854 pre-configured profiles from the eduroam CAT repository in Oct 2022. To decide whether a profile is vulnerable, we consider two aspects: (i) whether it specifies a commercial CA as the trust anchor, and (ii) whether the final server name matching constraint to be enforced will be loose (*e.g.*, empty strings and TLDs). For (i), we parse all the CA certificates in a profile, and if any of them can be matched to the trusted root CA certificates in `/etc/ssl/certs` on Ubuntu 20.04, then we consider it to be using a commercial CA. For (ii), we extracted the corresponding suffix computation code from eduroam CAT and geteduroam, and ran them through all the profiles to determine the outcome. In the end, we found 1759 profiles that use commercial CAs as trust anchors, which could be affected by Android's substring matching weakness. Concerning the problem in the common suffix algorithm of eduroam CAT, we found 108 profiles to result in loose server name constraints, 52 (1.35%) of which also use commercial CAs as trust anchors, and are thus vulnerable. For geteduroam on Android 9, we found 57 profiles to result in loose server name suffixes, 15 (0.39%) of which have commercial CAs as trust anchors and are thus considered vulnerable.

4.4 API design and usage are both important

Although the flaws were found in the apps, we argue that to a certain extent, Google's Android API design also needs to take some blame. Looking back, before API level 18 (Android version 4.3), there were simply no ways for a configurator app to even setup the server name matching. Although newer API methods have since been introduced, certain Android versions still suffer from the bitter limitation of not being able to handle multiple server names, which is occasionally needed in enterprise Wi-Fi, as organizations might deploy multiple authentication servers for load-balancing reasons. A restrictive API that cannot accommodate such realistic needs then forces developers to improvise backward-compatible solutions, which gives up on enforcing correctly stipulated security policies, as demonstrated above.

[7] https://github.com/geteduroam/ionic-app/blob/d8e08d48/geteduroam/plugins/wifi-eap-configurator/android/src/main/java/com/emergya/wifieapconfigurator/config/WifiProfile.java#L287

[8] Only in a controlled environment without any real users.

On the other hand, instead of a one-size-fits all solution, developers should try to use the best API methods available for a particular platform. For instance, eduroam CAT should use the `setAltSubjectMatch()` and `setDomainSuffixMatch()` methods, both introduced since API level 23, on Android version 6 or above. This is particularly important when one considers the inherent weakness of the substring matching logic used by the `setSubjectMatch()` method. Newer devices with better APIs should not be suffering together with the older, more restrictive versions.

5 ISSUES OF CHROMEOS

5.1 ChromeOS built-in configurators

Unlike Android where apps can serve as Wi-Fi configurators to load pre-configured profiles, ChromeOS has this capability built-in and does not require a standalone third-party app. Specifically, one can manually load a network configuration profile (.onc file) through the system portal. While discussing the issue of policy enforcement with an engineer of the eduroam project, we realized that some of the pre-configured profiles for ChromeOS are also limited by the OS API design, which could partly explain the loose server name matching constraints observed by a previous work [15]. This situation is reminiscent of the improvised suffix computation algorithms discussed in Section 4. The crux of the problem is that when it comes to specifying server name matching constraints, older versions of ChromeOS only supports the `SubjectMatch` attribute, which cannot accommodate multiple common names, just like its Android counterpart of `setSubjectMatch()` discussed above. Because of this, it is not surprising that some ChromeOS profiles are found to be using loose name matching constraints (*e.g.*, TLDs).

Two new attributes, in addition to the original `SubjectMatch`, have since been added as part of the profile specification: `SubjectAlternativeNameMatch` and `DomainSuffixMatch`, which will match the Subject Alternative Name (SAN) in the certificate extension against the list of provided SANs, and any suffix of certificate DNS SAN against the list of suffixes, respectively[9]. This is again reminiscent of the new API methods introduced on Android since API level 23 (`setDomainSuffixMatch()` and `setAltSubjectMatch()`). However, an engineer from eduroam stated that all three attributes on ChromeOS pose problems of their own, since the documentation states that `SubjectAlternativeNameMatch` and `DomainSuffixMatch` will only look at the SAN, but not the subject name. Section 5.2 of RFC5216 states that if both the subject name and SAN are non-empty, both should be considered in the server name checking [19]. Thus, none of the three attributes for name checking, as described in the documentation, can fulfill the consideration prescribed by RFC5216. At the same time, setting both `SubjectMatch` and the SAN-checking attributes will require *both* the subject name and SAN to be matched (enforcing a logical AND, instead of the desirable logical OR).

5.2 Lenient configuration UI

While testing the enterprise Wi-Fi functionality of ChromeOS (Flex version 107.0.5359.172), we noticed that the Wi-Fi configuration UI makes inputs that are critical to checking certificate names optional to users. Specifically, none of the *"Subject match"*, *"Subject alternative name match"* and *"Domain suffix match"* inputs are required by the UI. This is highly reminiscent of the similar UI design issues found on older versions of Android [15], which have since been fixed. Because of this UI leniency, a careless user could switch on ❶ the chain of trust validation but leave ❷ name checking disabled. Under such a configuration, an ET attacker could attempt the attacks discussed in Section 4.

> Reflection: Designing an API (and UI) that can both accommodate flexible deployments and enforce secure configurations, require understanding of actual deployment needs and making careful design decisions. Unfortunately, the legacy APIs for enterprise Wi-Fi on both Android and ChromeOS are not the best examples of that.

6 A PROPRIETARY WI-FI CONFIGURATOR

In this section, we discuss SecureW2 JoinNow, a commercial Wi-Fi configurator which provides services to businesses and schools that use eduroam and other enterprise Wi-Fi. On Android, the SecureW2 JoinNow app is in charge of fetching and applying the pre-configured profile on the client device, which is somewhat similar to the open-source ones discussed in Section 4. On macOS and Windows, to get a pre-configured profile, the user is expected to download and run an executable, which has the actual profile encapsulated inside and will be applied during run time. For iOS and ChromeOS, the pre-configured profiles are in their corresponding OS-specific formats (*i.e.*, `.mobileconfig` and `.onc`), which are downloaded directly from Web without a dedicated app. As we are more interested in the configurator apps and their own profiles, iOS and ChromeOS are not considered in the rest of this section.

To collect the profiles used in this study, we determined how the SecureW2 JoinNow app on various OSes obtain their profiles, by using a man-in-the-middle (MitM) setup with `mitmproxy` [9] to monitor the network traffic, and reverse engineering the apps if necessary. After getting the profile repository URL, we then enumerated the domain IDs to obtain the corresponding pre-configured profiles for Android, and executables for macOS and Windows. We then used 7zip to extract the actual profile files from those executables. For all 3 OSes, the profiles are PKCS#7 signed data, with the payload being an XML and signed by the SecureW2 CA. We then parsed the XML for subsequent analysis. In the end, we successfully crawled more than 470 unexpired profiles for each of the 3 OSes (for the exact number, see the second column of Table 1).

Interesting Attributes. For the XML payload, we found several interesting tags related to enterprise Wi-Fi connections: `enableServerValidation` and `certificate`. Inside the `certificate` tag, we also find the tags `useSystemStore`, `useDpiSSL` and `useFirefoxCertStore`. In order to understand the meaning of these tags, we used two methods: (1) parse all the profiles and find different values in the tags, and test the profiles empirically with our test platform to see if the differing tags have any effects on the Wi-Fi connection; (2) reverse engineer the application to see if and how certain tags are used in the code. Furthermore, to allow us to test arbitrary profiles, we also patched the Windows executable to bypass the code that verifies the signature of the profile. Additionally, some of the profiles specify the use of Web-based single sign-on (SSO)

[9] https://chromium.googlesource.com/chromium/src/+/refs/heads/main/components/onc/docs/onc_spec.md

OS\ Numbers	Total	Vulnerable
Windows	474	11 (2.32%)
Android	471	12 (2.55%)
macOS	475	10 (2.11%)

Table 1: The number of valid SecureW2 profiles crawled and vulnerable on each mainstream OS.

login that invokes a browser to authenticate the user. In that case, the browser would be in charge of performing the authentication exchange, and thus we do not analyze the security of these profiles in this study.

Vulnerable Wi-Fi Profiles. For the `enableServerValidation` attribute, its name suggests that setting the attribute to `false` will disable the conventional certificate validation ❶ and ❷. That is indeed the case for Windows and Android before version 10, and we tested that the ET attack will be successful.

To understand why the same attack outcome does not directly apply on Android 10 or above, we have to look at the Android API once again. Since Android 10 (API level 29), the `WifiConfiguration` API for setting up a Wi-Fi connection, was deprecated in favor of the new `WifiNetworkSuggestion` API. For the `WifiNetworkSuggestion` API on Android 10, the trust anchor for ❶ and the server name constraint for ❷ in a configuration can remain empty, though the system always double checks with the user through a prompt before actually loading the configuration. Following the terminology used by a previous work [4], this could be classified as an "User-Insecure" behavior, in the sense that an insecure configuration needs explicit user confirmation before it gets loaded. However, since Android 11 (API level 30), the `WifiNetworkSuggestion` API mandates that both trust anchor for ❶ and the server name constraint for ❷ must be set. Otherwise, the Wi-Fi configuration will be ignored[10]. Since the SecureW2 JoinNow app uses the `WifiNetworkSuggestion` API on Android version 10 or above, pre-configured profiles with `enableServerValidation` set to false, which are insecure on Android 9 or below, are only User-Insecure on Android 10, and simply not functional on Android 11 or above.

For SecureW2 JoinNow on macOS, when the `enableServerValidation` attribute is set to false, then it takes approach ❸, and asks the user to decide whether to accept the server certificate (and memorize its information for future use). As such, it is also User-Insecure. When the `enableServerValidation` attribute is true, then the system programmatically enforces certificate validation without relying on manual inspection.

With this knowledge in mind, we can then evaluate how many profiles are outright insecure or User-Insecure. The numbers can be found in Table 1. Notice that in order to be accurate, we only count the unexpired profiles that are not for setting up test or guest networks. Finally, in order to validate our result, we also used our ET setup to test all the profiles that have `enableServerValidation` set to `false`. For all of the outright insecure and User-Insecure profiles across Windows, macOS and Android, we were able to perform the ET attack and get the user credentials.

[10]https://developer.android.com/guide/topics/connectivity/wifi-suggest

Discrepancies on different OSes. One interesting observation is that the XML configurations across different OSes for the same domain (organization) can differ. We found cases where the security-critical attributes take different values on different OSes. In one particular case, the value of `enableServerValidation` is set to `true` in the Windows profile (with some CA certificates pinned), but the corresponding Android profile has `enableServerValidation` set to `false`. It is not clear to us why such discrepancies exist.

Injecting certificates on user device. Through experimentation, we also found that the SecureW2 JoinNow app could inject certificates on the client device to facilitate TLS interception beyond enterprise Wi-Fi.

For all three OSs, if the `useSystemStore` is set to `true`, the certificate will be installed into the system certificate store.

On Windows, the app always asks for administrative right via the UAC prompt. Then, if the `useSystemStore` is set to `true`, the certificates from the profile will be installed in the "Trusted Root Certification Authority". This allows a MitM with the private key of one of those installed certificates to intercept other TLS applications that rely on this compromised root store. In our experiment with a custom profile, we were able to intercept traffic to and from the Edge browser. Alternatively, if `useSystemStore` is `false` but `useFirefoxCertStore` is true, then the `NSS certutil` will be invoked to inject the profile certificates into the Firefox trust store, which would then allow a MitM to intercept TLS traffic to and from Firefox. Once again, we tested this with a custom profile. The `useDpiSSL` attribute set to `true` appears to be effectively the same as `useSystemStore` set to `true`, but shows an additional prompt warning user about the injection of trusted CA certificate.

For Android 10 or below, if, in addition to `useSystemStore` being true, `useDpiSSL` is also true, then a prompt will pop up, asking the user to install the certificate to the user "Trusted credentials". Otherwise, the certificate will be installed to the "User credentials" store without prompt, for use with Wi-Fi. We note that certificate in the "Trusted credentials" store could be trusted by other apps if they opt-in via the Network Security Configuration (NSC). A recent work [16] found that around 8% Android apps have this opt-in, which could be intercepted after the injection. Since Android 11, apps can no longer invoke the user prompt to install the certificate to the "Trusted credentials" store[11]. Instead, the user will have to manually install certificates in system settings, thus making it harder to inject and intercept. Based on our reverse engineering, the `useFirefoxCertStore` tag is not used on Android.

For macOS, if `useSystemStore` is set to `true`, then the certificate will be installed in the System Keychains after the user authenticates at the "Make changes on certificate trust settings" prompt, and will be trusted for all purposes, thus enabling interception of other TLS applications. The `useDpiSSL` attribute set to `true` appears to be effectively the same as `useSystemStore` set to `true`. If `useFirefoxCertStore` is set to true, then the Firefox autoconfig `security.enterprise_roots.enabled` is used to trust certificates from the macOS system store, and the Firefox settings will be locked (grayed out).

[11]https://developer.android.com/reference/android/security/KeyChain#createInstallIntent()

It is conceivable that such certificate injection features can be useful and convenient for IT admins to intercept and monitor traffic in an enterprise setting, especially when the Bring Your Own Device (BYOD) policy is becoming increasingly popular. However, since we cannot see the user agreements and documentation from organizations that attempt such injection, it is not clear to us whether the users are notified of the implications of installing the profiles (and its certificates), and whether that might create potential legal issues concerning user privacy in certain jurisdictions. We leave the user agreement analysis and potential legal arguments for future work.

Forcing weak ciphersuites. Apart from experimenting with the various interesting attributes, we also observed an interesting case that, for SecureW2 JoinNow on Windows, if a profile specifies the use of TTLS as the EAP method and PAP for phase2 authentication, the client will only offer 2 ciphersuites in its TLS `Client Hello`: `TLS_RSA_WITH_3DES_EDE_CBC_SHA` and `TLS_RSA_WITH_AES_128_CBC_SHA` (in this order). Our original ET attack setup failed against such profiles, because the default build of our OpenSSL did not support these weak ciphersuites. After compiling a custom build of OpenSSL to enable support for these ciphersuites, we were able to successfully perform the attacks and get the credentials in cleartext. The SecureW2 JoinNow apps on Android and macOS do not have the same behavior of forcing weak ciphersuites, and it is not clear to us why this is needed on Windows.

CA Certificates. We also extracted all the certificates embedded in the profiles and look for similarities based on public keys and common names. Overall, we extracted 1230 certificates from Andriod profiles, 1786 certificates from Windows profiles, and 1782 certificates from macOS profiles. Over 90% of the certificates are the same in Windows profiles and macOS profiles. For each OS, around 14% of the certificates are self-signed certificates. The most popular CA certificate has been used for approximately 280 times across profiles of different OSes. Previous work noted that some misconfigured setups might reuse default private keys in different instances of the same appliance [10, 15]. We also tried to detect cases where certificates are of different common names and share the same public key, but we did not find any such cases.

Reflection: Although the overall percentages of vulnerable profiles are small, the use of proprietary configurators does not completely mitigate insecure enterprise Wi-Fi configurations. One possible improvement is for the configurator provider to detect and prohibit vulnerable profiles, similar to what the new `WifiNetworkSuggestion` API on Android 11 does. Moreover, some organizations take advantage of the configurator to also inject CA certificates on client devices, raising additional security and privacy concerns. Ideally, this should be clearly communicated to the users.

7 DISCUSSION

7.1 Responsible Disclosure

The flaws of Android's new TOFU (Section 3) and the leniency of ChromeOS's UI (Section 5.2) have all been responsibly reported to Google, which confirmed our findings and gave us some bug bounties. Fixes are already planned for upcoming versions of Android and ChromeOS.

We reported the issues in eduroam CAT (Section 4.1) to the corresponding maintainers, and received positive acknowledgment confirming our findings. Instead of fixing the eduroam CAT app, however, the maintainers mentioned that for Android 8 or above, they recommend the use of the new geteduroam app, and both Android 6 and 7 have long passed the "End-of-life" (EOL) status and are used by only a small portion of users. In particular, they also argued that the substring matching of `setSubjectMatch()` is a known problem, and the attacks outlined in Section 4.1 require the affected organizations to be using commercial CAs as trust anchors, which is against their official guideline of using a dedicated CA[12]. As such, there is currently no plan to fix the eduroam CAT app. We note that the eduroam CAT app continues to be available on Google Play store and can still work on newer versions of Android (we tested it on both Android 11 and 13). Without inspecting the Wi-Fi setup guides from organizations, it is not clear whether they are indeed migrating from eduroam CAT to geteduroam. Also, based on our measurements in Section 4.3, around 45% (1759/3854) profiles use commercial CAs as trust anchors, which suggest that the official guideline of using dedicated CA might not be closely followed by many organizations.

We also reported to the maintainers the geteduroam issue that affects Android 9 users (Section 4.2). Fortunately, the issue was found in a stable release (`v1.0.21`) on GitHub, and the current release on the Google Play Store for Android 9 devices is of `v1.0.16`, which uses both `setDomainSuffixMatch()` and `setAltSubjectMatch()` for checking the server certificate names, and is thus not vulnerable based on our testing. The maintainers said that the problematic version will not be deployed, and we noticed that the next stable release on GitHub (`v1.0.23`) have since changed its minimum SDK version to API level 30, meaning that it can no longer be built for Android 9.

With respect to the vulnerable profiles of SecureW2 JoinNow, we have reported their problems to the 15 organizations that prescribed them. At the time of writing, we have received 2 replies stating that the ticket is closed as the issue is already resolved.

Finally, we have reported the problem of weak ciphersuites for TTLS on Windows (Section 6) to SecureW2, and also made a suggestion on helping organizations to tighten the security of their pre-configured profiles. We will continue to discuss with them on how to mitigate these two problems.

7.2 TOFU versus TOOFU

Based on the state machine of the new TOFU on Android that we observed, as well as the similar TOFU behaviors observed on other OSes [4], an interesting discussion is on what it actually means for a configurator to be adopting the TOFU approach. As discussed in Section 3, the current TOFU configurator on Android allows an attacker to repeatedly unpin the memorized information and induce the first-use case. The same also applies to macOS, iOS and Windows, which was deemed User-Insecure by previous work [4]. An arguably better alternative might be trust-only-on-first-use (TOOFU), in the sense that once pinning is successful after first-use, the subsequent validation enforcement should be

[12] https://wiki.geant.org/display/H2eduroam/EAP+Server+Certificate+considerations

strict and programmatic. The onus should then be on the user to take extra steps to unpin and reset the TOFU. This is actually doable and has been adopted by some productions systems. For instance, with OpenSSH, the first time that a user connects to a host, the fingerprint of the key will be shown and is supposed to be checked. Then the key fingerprint will be pinned by the system, and checked programmatically in the future. If a known host sends a different public key in a subsequent connection, the connection will be stopped automatically and a vibrant warning message about possible MitM attack will be shown to the user. To accommodate benign key rotation events, the user would have to manually reset the pinned information by removing the corresponding entries in `.ssh/known_hosts`. In other words, an attacker cannot induce an unpin behavior under this model. An open question is then whether TOOFU is less usable than TOFU, which could be an interesting research for future work.

7.3 The complexity of certificate validation

In their 1999 paper on evaluating IPSec [12], Ferguson *et al.* coined the term *"complexity trap"*, which states that complexity is the worst enemy of security. The authors went on to argue that the IPSec standard has too many options and too much flexibility, and such complexity is bad for a security standard. After observing the findings from previous work [3, 4, 15, 23] as well as the flaws discussed in previous sections, we cannot help but wonder if the same criticism also applies to the conventional certificate validation with ❶ and ❷, and whether this approach actually benefits or hurts enterprise Wi-Fi. In the Web TLS case, since it is difficult to know ahead of time what hosts will be visited by the browser, and that there are billions of potential hosts, the flexibility and scalability offered by conventional certificate validation are useful in server authentication. That is, trusting a handful of root CAs allows a browser to also establish trust on the identity of billions of Websites on the Internet. However, in enterprise Wi-Fi, such a scalability challenge does not exist, as one enterprise Wi-Fi SSID is usually served by a very limited number of (usually one) authentication servers. There does not seem to be a technical reason why approach ❸ with an SSH-style TOOFU key pinning cannot work. We conjecture that a carefully designed approach ❸ could be both more usable and more secure for the users.

8 RELATED WORK

The ET attack is a classic problem known to the community, and has garnered the attention from the research community in recent years, with a series of work dedicated to understanding the status quo, evaluating the native UI designs and setup guides prescribed by organizations [4, 15, 23] at various scales. Additionally, previous work [3, 15] also considered configuration profiles and revealed several problems in them. In contrast, this work focuses on various forms of user-friendly configurators, and investigate how and why they fail to protect users.

With many moving parts in the certificate standards, conventional certificate validation is known to be very tricky to implement correctly [5, 6, 14, 21]. In particular, the task of name matching is also known to be a potential source of weakness [14, 20]. These existing research greatly echo with the enforcement flaws presented in Section 4 as well as the complexity discussion in Section 7.3.

Finally, although some apps are found to be inherently vulnerable to MitM interception due to improper implementation of certificate validation [7, 11, 17], injecting CA certificates on the client device could allow more apps to be susceptible to TLS interception [16], which is why the discussion in Section 6 could be concerning to user privacy. Outside of Wi-Fi configurators, many enterprise filters, antivirus and parental-control software also perform similar injection and TLS interception for scanning harmful contents, though in some case such interception opens up additional attack surface [10, 22].

9 CONCLUSION

In this paper, we investigate the robustness of enterprise Wi-Fi configurators and see how they protect users from ET credential theft and beyond. Our investigation found several design and implementation issues that are concerning. In particular, the new TOFU configurator on Android is vulnerable to stealthy attacks and thus fails to protect user credentials. We also dissected the implementation issues found on two open-source configurators that render them incapable of properly enforcing the security policies stipulated in pre-configured profiles, which are in part caused by the confusing and restrictive API design from Android. Similar problems also haunt ChromeOS, which might explain the insecure profiles observed by a previous work. Finally, we considered a commercial configurator, showed that some of its pre-configured profiles are also vulnerable to the ET attack, and discussed its other hidden behaviors that could damage user privacy. All in all, the findings of this paper suggest that the threat of ET is far from over, and it is perhaps time to rethink the merits of conventional certificate validation in the context of enterprise Wi-Fi.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for helping us improve the overall quality of our paper. This work was supported in part by a grant from the Research Grants Council (RGC) of Hong Kong (Project No.: CUHK 24205021), a grant from CUHK (Award No.: 4930961), as well as grants from the CUHK IE department (project code: NEW/SYC and GRF/21/SYC).

REFERENCES

[1] 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.

[2] 2022. WifiEnterpriseConfig | Android Developers. https://developer.android.com/reference/android/net/wifi/WifiEnterpriseConfig.

[3] 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.

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

[5] 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.

[6] 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.

[7] Sze Yiu Chau, Bincheng Wang, Jianxiong Wang, Omar Chowdhury, Aniket Kate, and Ninghui Li. 2018. Why Johnny Can't Make Money With His Contents: Pitfalls of Designing and Implementing Content Delivery Apps. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 236–251.

[8] 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. RFC 5280 (Proposed Standard). https://doi.org/10.17487/RFC5280 Updated by RFCs 6818, 8398, 8399.

[9] Aldo Cortesi, Maximilian Hils, Thomas Kriechbaumer, and contributors. 2010–. mitmproxy: A free and open source interactive HTTPS proxy. https://mitmproxy.org/ [Version 9.0].

[10] X de Carné de Carnavalet and Mohammad Mannan. 2016. Killed by proxy: Analyzing client-end TLS interception software. In *Network and Distributed System Security Symposium*.

[11] 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.

[12] Niels Ferguson and Bruce Schneier. 1999. A cryptographic evaluation of IPsec. (1999).

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

[14] 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.

[15] 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* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 1100–1117. https://doi.org/10.1145/3460120.3484569

[16] Marten Oltrogge, Nicolas Huaman, Sabrina Amft, Yasemin Acar, Michael Backes, and Sascha Fahl. 2021. Why Eve and Mallory Still Love Android: Revisiting TLS (In) Security in Android Applications.. In *USENIX Security Symposium*. 4347–4364.

[17] Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Fahad Shaon, Ke Tian, Miles Frantz, Murat Kantarcioglu, and Danfeng Yao. 2019. Cryptoguard: High precision detection of cryptographic vulnerabilities in massive-sized java projects. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2455–2472.

[18] 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.

[19] D. Simon, B. Aboba, and R. Hurst. 2008. The EAP-TLS Authentication Protocol. RFC 5216 (Proposed Standard). https://doi.org/10.17487/RFC5216

[20] 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.

[21] 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), 1–37.

[22] Louis Waked, Mohammad Mannan, and Amr Youssef. 2020. The sorry state of TLS security in enterprise interception appliances. *Digital Threats: Research and Practice* 1, 2 (2020), 1–26.

[23] Kailong Wang, Yuwei Zheng, Qing Zhang, Guangdong Bai, Mingchuang Qin, Donghui Zhang, and Jin Song Dong. 2022. Assessing certificate validation user interfaces of WPA supplicants. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 501–513.