Assessing Certificate Validation User Interfaces of WPA Supplicants

Kailong Wang*
National University of Singapore

Yuwei Zheng*
ByteDance

Qing Zhang
ByteDance

Guangdong Bai*
The University of Queensland

Mingchuang Qin
ByteDance

Donghui Zhang
ByteDance

Jin Song Dong
National University of Singapore

ABSTRACT

WPA (Wi-Fi Protected Access) Enterprise is the *de facto* standard for safeguarding enterprise-level wireless networks. It relies on Transport Layer Security (TLS) to establish a secure tunnel during its authentication process, and thus the notoriously error-prone certificate validation may haunt it. Incorrect validation may lead to the SSL/TLS man-in-the-middle attack, or the *evil twin attack* in the context of wireless networking, where the supplicant connects and unwittingly sends authentication credentials to a fake access point.

We conduct an empirical study on the effectiveness of certificate validation user interfaces (UIs) in WPA supplicants. We focus on a broad variety of mobile devices and mainstream operating systems (OSes), and find that a vast majority of them are susceptible to the evil twin attack. Insecure configuration options and lack of visual security indicators have been found common. Besides, five severe vulnerabilities (four are listed by CVE and one is found in parallel with Google) are identified from their validation processes. By examining the source code of Android's Wi-Fi manager, we link the root causes of these vulnerabilities to the immature designs and implementations of WPA software modules. Our investigation, including a review of Wi-Fi configuration guidelines of the top 200 universities and a realistic experiment deployed in a company with over 50k employees, reveals the user susceptibility in practice. Our findings have been reported to Google, leading to a security enhancement in the WPA supplicant of Android's latest version 11.

ACM Reference Format:
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 ACM Mobile Computing & Networking Conference (Mobicom '22), October 24–28, 2022, Sydney, NSW, Australia.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3495243.3517026

*These authors contribute equally. Guangdong Bai (g.bai@uq.edu.au) is the corresponding author.

ACM MobiCom '22, October 24–28, 2022, Sydney, NSW, Australia

ACM ISBN 978-1-4503-9181-8/22/10.
https://doi.org/10.1145/3495243.3517026

1 INTRODUCTION

The enterprise mode of WPA, WPA Enterprise (also referred to as the WPA-802.1X mode), is the most commonly used security mechanism for safeguarding enterprise-level wireless networks. It empowers WPA with authentication capability through the EAP (Extensible Authentication Protocol) authentication framework [49], which is carried by the Remote Authentication Dial In User Service (RADIUS) protocol [51]. EAP supports a wide variety of authentication mechanisms without any pre-negotiation or involvement of the pass-through agents (e.g., access points), such that it is compatible with traditional password/token-based authentication methods.

EAP could directly utilize TLS as an authentication method by reusing its mutual authentication feature based on client and server certificates, leading to the EAP-TLS [59]. More commonly, EAP uses TLS to establish a tunnel between the supplicant and the RADIUS server (referred to as *phase 1*) to protect (legacy) inner authentication methods (referred to as *phase 2*) such as PAP (Password Authentication Protocol) and MSCHAP [41] (Microsoft Challenge-Handshake Authentication Protocol). This leads to other widely deployed protocols such as EAP-TTLS (Tunneled TLS) [21] and PEAP (Protected EAP) [20]. They eliminate the use of client certificates for phase-1 authentication, freeing the enterprise from the burden of deploying and managing a public key infrastructure (PKI).

Whenever TLS is involved, correctly validating server certificates is crucial for recognizing a legitimate server. Otherwise, failures could cause the access point impersonation attack, or the *evil twin attack* [6, 13, 16]. It occurs when the supplicant is tricked into connecting to a rogue access point which has the same SSID as any of the supplicant's previously-connected access points. The attacker could install a variety of attacks into the evil twin, for example, to steal the phase-2 authentication credentials. Given that many companies use the same authentication credentials (e.g., username and password) for Wi-Fi and the internal single sign-on (SSO) system, a single compromised account would threaten the confidentiality of their sensitive services and information such as trade secrets. This threat is further exacerbated by the increasingly popular BYOD (bring your own device) scheme—the great variety in employees' devices and installed OSes significantly enlarges the attack surface.

To defend against the evil twin attack, existing WPA supplicants have implemented security countermeasures to allow users to configure for mandatory validation, and to warn users once an evil twin is detected. Nonetheless, they may not be as effective as their counterparts in contemporary web browsers [24, 63], which display the padlock icon and actively block suspicious sites. For example,

some supplicants leave insecure "*do not validate* (certificate)" as the by-default option [6].

Our Work. We study the effectiveness of the certificate validation UIs in WPA supplicants. We focus primarily on the visual configuration options and security warnings when the supplicant is connected or reconnected to the enterprise wireless network, and analyze whether they are sufficient to defeat the evil twin attack. Our study involves a broad variety of device types (laptops and phones) and mainstream OSes (Android, iOS, MacOS and Windows) that affect at least 95% of global mobile device users according to their market share [56]. We enumerate and test all configuration options in each supplicant while connecting them to our evil twin testbed. Our study finds that *weaknesses in the UIs commonly exist in the vast majority of existing WPA supplicant implementations*. This is to our great surprise, given that the evil twin attack has been noticed for more than a decade [18] and many studies have been conducted on other non-browser user agents [23, 27].

By examining the related source code in Android, we link the root cause of these vulnerabilities to the immature designs and implementations of the WPA supplicants, as well as the insecure APIs exposed by them. In the `android.net.wifi` package in initial AOSP 10, we find that the `WifiManager.addNetwork` API permits the option of "*do not validate*" in the configuration UI. This is inherited by Original Equipment Manufacturers (OEMs) such as Vivo (CVE-2020-12484) and Huawei (CVE-2020-1836). We also find that the `WifiManager.addNetworkSuggestions` API confuses the identifiers of the suggested configurations, allowing the attacker to manipulate the certificates the `wpa_supplicant` accepts (the vulnerability A-150500247 acknowledged by Google). From the source code of `wpa_supplicant`, we find that `wpa_supplicant` accepts any method proposed by the authentication server when the phase-2 authentication method is not specified. This weakness allows the attacker to manipulate the phase-2 authentication into a less secure one and further obtain victims' login credentials (CVE-2020-0201 in AOSP and CVE-2020-9260 in Huawei OEM).

We conduct two studies to better understand the users' susceptibility to the evil twin attack on all OSes. First, we perform a review of the Wi-Fi configuration guidelines of the global top 200 universities, revealing that only 37% (74/200) of them manage to provide secure instructions to their users. Then, we deploy an evil twin in an Internet technology company in a confined scenario, following the legal and ethical guidelines from the company. The simulated attack harvests 166 login credentials within a 40-minute period, suggesting the severity of the weaknesses identified by our work. We re-conduct the experiment after the company has taken actions to secure the Wi-Fi, and still manage to gain 14 login credentials within 6 hours, showing the persistence of the weaknesses. We thus propose several recommendations and advice to mitigate the risk.

The security of WPA and certificate validation have been continually and actively studied in the literature [6, 13, 23, 27, 30, 36, 46, 54, 66–68]. Among the related works, Brenza et al. [13] leverage valid server certificates and exploit the `eap_workaround` compatibility setting in the phase-2 protocol to hijack the Eduroam connections. Bartoli et al. [6] survey Eduroam users and reveal prevalent incorrect network configurations used by them. As a comparison, our work targets the general certificate validation UIs, and thus is applicable to any WPA Enterprise network including Eduroam. Our work provides a specific consideration of the implementations of UI/UX during the client-side configuration. This complements existing studies on the certificate validation in non-browser user agents that mainly target the programmatic interfaces, such as those in SDKs [27, 46] and APIs [23, 30].

Contributions. We summarize our contributions as follows.

- **An approach to analyzing the effectiveness of certificate validation UIs in WPA supplicants and a large-scale study.** We take into account the implementation of UI/UX into the security assessment of certificate validation. We have analyzed 13 mobile device brands/models and 16 OSes/versions, and revealed that most of them suffer from the evil twin attack due to deficient configuration interfaces and inconspicuous security warnings. This study is the first of its kind in the literature, complementing the existing works in non-browser user agents, such as mobile apps [23, 27, 46] and IoT devices [4, 35, 38].
- **Vulnerabilities affecting billions of users.** Our work has identified and reported four CVE-listed vulnerabilities (CVE-2020-0201, CVE-2020-12484, CVE-2020-1836 and CVE-2020-9260) and one Google-confirmed vulnerability (found separately in parallel with Google). Our findings have led to Android's security enhancement in the WPA supplicant of its latest version 11.
- **Characterization of user susceptibility.** Our studies have revealed users' susceptibility to the weaknesses in certificate validation UIs, suggesting the necessity of enhancing them to mitigate impacts from the evil twin attack.

2 BACKGROUND

Our work focuses on client-side issues during the authentication process in WPA Enterprise, mainly the certificate validation in WPA supplicants. In this section, we briefly review the WPA Enterprise and its authentication process, covering several authentication protocols in the context of this paper. We also recap the WPA Enterprise configuration process in the mainstream OSes/devices.

2.1 WPA Enterprise and its Authentication

As shown in Figure 1, there are three principal participants in the WPA Enterprise authentication model: a *supplicant* (STA) which indicates the user device that supports the IEEE 802.1X standard, an *authenticator* (AC) which is typically a network access point, and an *authentication server* (AS) that runs the RADIUS authentication protocol. For a supplicant to be admitted into the network, the supplicant and the RADIUS server have to be mutually authenticated using EAP with X.509 client/server certificates or/and inner authentication methods. Given that EAP is designed to function within the point-to-point protocols (PPP), EAP messages between the supplicant and the access point are encapsulated using the EAP over LAN (EAPOL) protocol in the wireless environment while those between the access point and the authentication server are encapsulated using RADIUS.

In this work, we focus on the three most widely-deployed EAP authentication protocols in WPA Enterprise [26], i.e., EAP-TLS, EAP-TTLS and PEAP, since others are either proven insecure (EAP-PWD [68]) or rarely used in practice (EAP-SIM and EAP-AKA). A generic EAP protocol flow is shown in Figure 1. The authentication

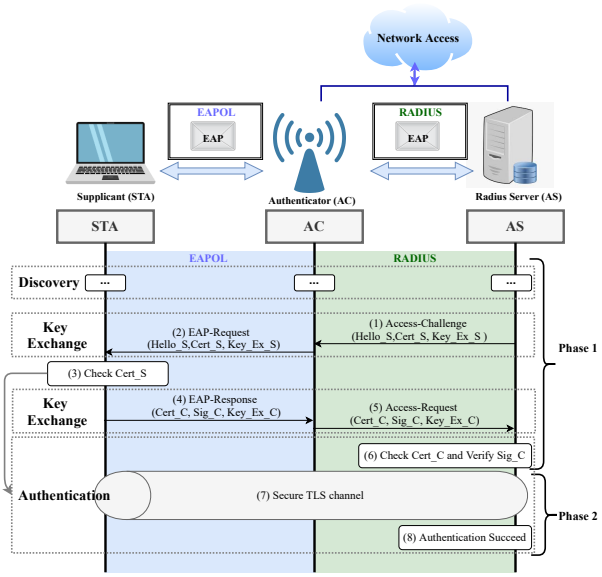


Figure 1: Participants and Workflow of WPA Enterprise

process incorporates three stages, i.e., *discovery*, *key exchange* and *authentication*. Phase 1 includes the discovery and the key exchange stages. In this phase, all three protocols follow the same process. In the discovery stage, the supplicant connects with the AC and initiates the authentication process, and in the key exchange stage, an encryption key is established between the supplicant and the AS via the TLS handshake (steps (1)-(5)). The authentication stage is slightly different among three protocols:

- **For EAP-TLS**, the certificate-based mutual authentication is completed within phase 1 where supplicant authenticates the AS at step (3) and AS authenticates the supplicant at step (6).
- **For EAP-TTLS and PEAP**, the supplicant authentication is completed in phase 2 at step (7) through the secure TLS tunnel, after completing the AS authentication in phase 1. The commonly supported phase-2 authentication protocols (i.e., the inner authentication protocols) include PAP which relies on username and password, MSCHAPv2 which extends MSCHAP, and GTC (Generic Token Card) which uses the password and a one-time token for authentication.

2.2 Certificate Validation UIs

During the discovery stage, the supplicant searches for available networks and then automatically connects to one it recognizes, i.e., one that has been previously connected to or is in the preferred network list. Thus, securely configuring the initial Wi-Fi connection is essential for safeguarding the subsequent connections. However, due to the complexity of WPA Enterprise and the diversity of its implementations among OSes and devices, the configuration process could be counter-intuitive for the users. Below we recap this process in the mainstream OSes and devices, and its security implications.

Validation UI of Android. After clicking on the target network SSID from the Wi-Fi Settings UI, the user is prompted to select options including the EAP method, the phase-2 protocol and the CA certificate validation method, as shown in Figure 2(a). The current design allows the user to select the option "*do not validate*" or "*unspecified*". If either is selected, the user's credentials could be obtained by the attacker, as our work reveals (see Section 4.2).

Validation UI of MacOS and iOS. After clicking on the SSID, the user is prompted to decide whether to trust the network or the pre-configured profile before being connected, as is shown in Figure 3(a) and (b). Generally, the detailed configuration is "hidden" inside the profile and the user can manually generate this profile via the Apple Configurator app shown in Figure 2(b). This configuration process turns out to be complex and non-intuitive such that users are prone to trust malicious access points.

Validation UI of Windows. After clicking on the SSID, the user is prompted for their consent to connect to the network, as shown in Figure 3(c). The user would have to manually configure the setting if the by-default one is incompatible with the server-selected options. To do this, the user opens the Network and Sharing Center and selects "*Set up a new connection or network*", followed by "*Manually connect to a wireless network*". The available options for the EAP properties are shown in Figure 2(c). With such a UI design, the users also tend to blindly trust any certificate and proceed with the connection.

3 CHARACTERIZING WEAKNESSES IN CERTIFICATE VALIDATION UI

The complexity of configuring the certificate validation may lead to mis-configurations that is subject to the evil twin attack. In this section, we present this attack, and characterize the desirable security properties in certificate validation UIs.

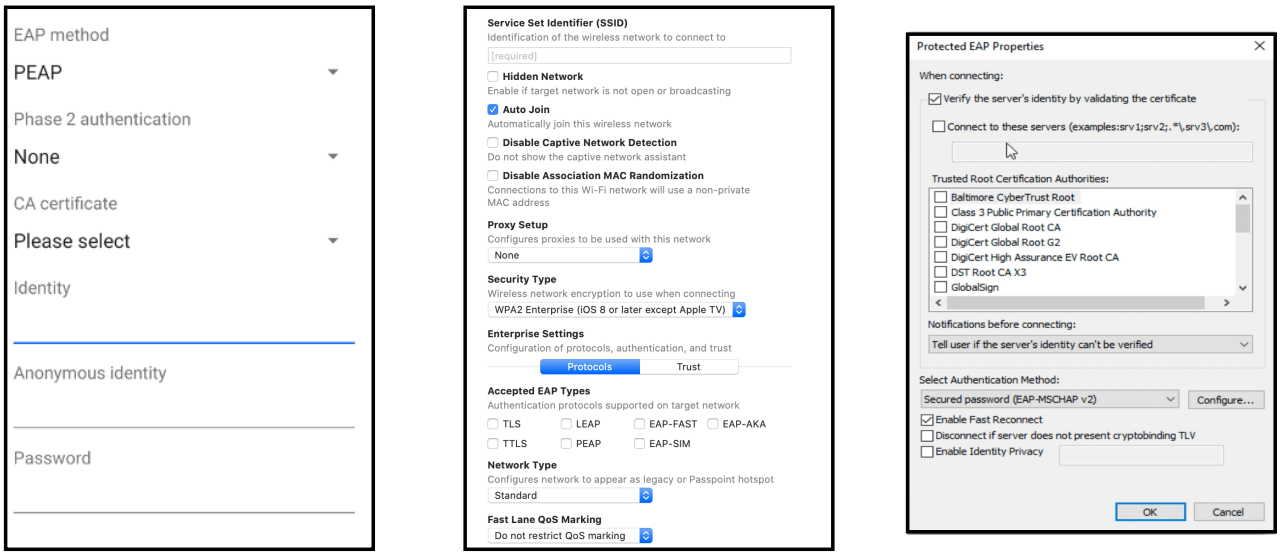
3.1 The Evil Twin Attack

3.1.1 Attacker Model. Our work considers the *active, on-path* evil twin attacker model [13, 16, 23], where the attacker deploys an access point in the vicinity of the victim. It advertises the identical SSID (thus named the evil twin) as the target legitimate wireless network. Assume the victim supplicant has previously connected to the target network. The attacker attempts to automatically reconnect the victim supplicant or misleads the victim user to authorize the reconnection. It impersonates the authenticator and RADIUS server of the target network to complete the TLS handshake (phase 1) and establish a rogue TLS tunnel with the supplicant. Through the rogue tunnel, it obtains phase-2 authentication credentials.

To be practical, we assume the attacker exploits the insecure configurations (e.g., missing CA certificate validation) set over the initial connection to the target network, and attempts to steal the victim's authentication credentials in the subsequent attempts to connect with the evil twin. This means the phishing attack that advertises a visually identical SSID by including tailing non-printable characters (e.g., "`SSID␣`" against "`SSID`") to deceive the newly-connected users is out of the scope of this work.

3.1.2 Attacker Capabilities. Through manually examining the WPA Enterprise authentication process, we summarize the required capabilities to launch the evil twin attack. Below we list them and briefly discuss their feasibility.

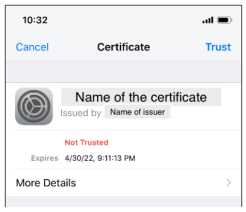
Evil Twin Setup. The attacker is able to set up a mobile hotspot, which could be launched using a laptop combined with a wireless



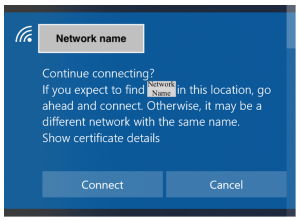
(a) Configuration Page in Android (b) Configuration Page in MacOS/iOS (c) Configuration Page in Windows

Figure 2: Configuration Pages in Four OSes

(a) Connection Warning in MacOS



(b) Connection Warning in iOS



(c) Connection Warning in Windows

Figure 3: Connection Warnings in Three OSes

Table 1: Evil Twin Attack Scenarios

Attack Scenario	Deauthentication Needed	Stealthiness[2]	Effectiveness[3]
Inner twin	✓ STA⇔[1]target net	★★★ non-suspicious	★☆ likely disconnected
Perimeter twin	✓ when STA⇔[1]target net	★★☆ fairly suspicious	★★ possibly disconnected
	✗ when STA⇎[1]target net	★★ fairly suspicious	★★☆ unlikely disconnected
Outer twin	✗ STA⇎[1]target net	★☆ suspicious	★★★ unlikely disconnected

[1] ⇔(⇎): The supplicant is (not) connected with the target network when attack is launched.
[2] The stealthiness measures the degree to which the user becomes suspicious of the evil twin signal. The signal from evil twin is considered non-suspicious (highly stealthy) within the enterprise perimeter and vice versa.
[3] The effectiveness measures the degree to which the evil twin attack can be completed without interference. The supplicant may be disconnected from the evil twin before the completion of phase-2 authentication due to the stronger signal from the target network.

network adapter. The attacker is also able to set up a RADIUS server to manipulate the handshake messages exchanged with the supplicant. This is achievable through open-source or free software, e.g., hostapd [61] or FreeRADIUS [60]. Section 4.1 presents the setup in our experiments, demonstrating the setup is feasible and inexpensive.

To launch the attack, the victims need to be within the evil twin's signal range, under three possible scenarios depending on its setup locations: the *inner twin* where the evil twin is within the enterprise, e.g., inside an office building, the *perimeter twin* where the evil twin is in the surrounding area near the enterprise, e.g., inside the car park of the enterprise, and the *outer twin* where the evil twin is away from the enterprise, e.g., by the street 2 km away from the enterprise. Table 1 lists the characteristics of each attack scenario.

Physical Channel. The attacker is able to deploy the evil twin in a radio frequency different from that used by the target network to avoid signal interference.

Communication Relay. The attacker is able to communicate with the victim supplicant and the target network. In particular, the evil twin could act as either a *forwarder* or a *black hole* after obtaining the victim's credentials. In the former case, it relays communication between the supplicant and the target RADIUS server; in the latter case, it simply drops the connection to avoid suspicion. It can record

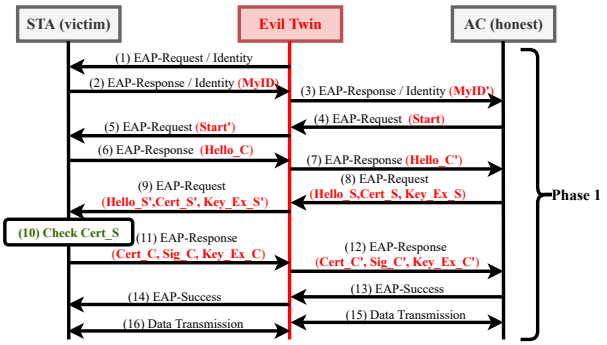


Figure 4: Evil Twin Attack Steps against Flawed Phase-1 Server Certificate Validation

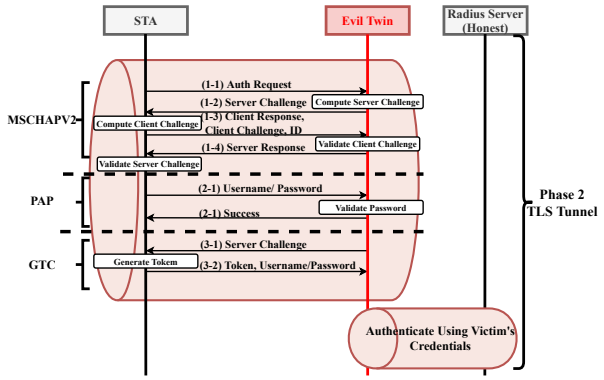


Figure 5: Evil Twin Attack Steps against Flawed Phase-2 Authentication Protocols

every frame exchanged from the beginning of the authentication process, and modify the forwarded frames.

Deauthentication. The attacker is capable of forcing the victim supplicant to disconnect from the target network by sending a deauthentication frame with a spoofed address [10]. The attacker could repeat this action until the victim supplicant connects with the evil twin. To serve this purpose, accessible tools such as aircrack-ng [1] could be applied.

3.1.3 System Prerequisites. We make the following two assumptions regarding the target network. First, its infrastructure, including the authenticator and the RADIUS server, is securely implemented. Otherwise the attacker is able to trivially compromise the network. Second, it supports the popular WPA Enterprise authentication protocols, including EAP-TLS, EAP-TTLS and PEAP. For EAP-TTLS and PEAP, it supports the commonly used phase-2 authentication protocols such as MSCHAPv2, GTC and PAP.

3.2 Desirable Security Property in Certificate Validation UIs

To the best of our knowledge, there is no available guideline on client-side vulnerability assessment to follow when analyzing certificate validation UIs. We thus resort to an manual inspection on the WPA Enterprise authentication protocols. We focus on the steps of certificate validation and credential exchanging in the protocol, and examine how UI elements affect these steps.

3.2.1 Desirable Security Property in Phase 1. Figure 4 zooms into the steps of the evil twin attack against phase 1 of WPA Enterprise. In step (9), the attacker's server certificate could be accepted by the supplicant (step (10)) if certificate validation is mis-configured. As of EAP-TLS, the supplicant would mistake the evil twin as the target network. It then sends its client certificate (step (11)) and establishes a TLS tunnel (steps (12)-(13)) with the attacker. For EAP-TTLS and PEAP, the client certificate validation is optional during the TLS handshake, as specified in [49]. The phase 2 would use the established TLS tunnel for user authentication, such that the user authentication credentials may be leaked (detailed in phase-2 vulnerabilities in Section 3.2.2). We have identified the following security properties in phase 1 to prevent the above attack steps.

P1-1: ***Freedom from insecure options in configuration of phase 1***. When first connecting to a WPA Enterprise network, user interactions in phase 1, such as selecting the EAP protocol and the certificate validation method, are crucial for the security of the connection. The design of the GUIs thus must be intuitive and secure-by-default, so that lay users without security expertise are able to provide secure responses.

Examples of violation. Insecure options are listed in the configuration of certificate validation (e.g., "*Unspecified*" or "*Do not validate*"). The by-default option of certificate validation is "*unspecified*". Users are prompted to decide whether to trust a network without sufficient information displayed.

P1-2: ***Presence of conspicuous visual cues when any error occurs***. The user would have a chance of rectifying the misconfigurations or terminating insecure connections, if conspicuous warnings against insecure options or untrusted certificates are given.

Examples of violation. The supplicant is automatically connected to the evil twin that has the identical SSID as the target network without any warnings prompted to the user. The user is allowed to proceed with an insecure certificate validation option of "*unspecified*" without any warning.

3.2.2 Desirable Security Property in Phase 2. The legacy authentication protocols, such as MSCHAPv2, PAP and GTC, assume a securely established TLS tunnel. Therefore, they may be at risk once the TLS connection in phase 1 is compromised, as shown in Figure 5. Among the existing phase-2 authentication protocols, MSCHAPv2 provides relatively strong user authentication, as it encrypts the username/password. However, there have been well-documented approaches [11, 34, 55] and tools (e.g., Openwall [62]) to retrieve the user password from the challenge-response messages exchanged in the handshake conversation (step (1-1) to (1-4)), regardless of whether it is carried by EAP-TTLS or PEAP. For PAP and GTC, the username and password/token are transmitted in plaintext (step (2-1) and (3-2)) such that they can be trivially obtained by the attacker. In view of these risks, the following property is expected to secure the phase-2 authentication.

P2-1: ***Secure by default configuration of phase 2***. The supplicant should not set the by-default option of the phase-2 authentication

Figure 6: A Photo of Our Experimental Setup

protocol as "*Unspecified*" or "*None*", as this triggers the adaptive setting that the client accepts the protocol type determined by the server. The evil twin is thus able to manipulate it into PAP or GTC.

4 ASSESSING CERTIFICATE VALIDATION UI IN ANDROID 10

Based on the recognized security properties in certificate validation UIs, we propose a systematic testing approach to examining the connection processes and identifying potential weaknesses. Given that the open-source nature of Android enables in-depth scrutiny and analysis, we first apply our approach to Android, and defer our analysis on other OSes to Section 5.

4.1 Our Approach

Environment Setup. We aim to check whether the evil twin of a previously connected network can hijack the supplicant's connection or steal the login credentials. We deploy a simplified WPA Enterprise authentication system as our confined experiment environment, to avoid posing security threats to any legitimate networks. The target network setup includes an access point and a RADIUS server, both installed on the same laptop (shown in Figure 6). We use hostapd, which is a user space daemon software, to transform a `TP-Link TL-WN722N` network interface card into an access point. We use FreeRADIUS to implement the RADIUS server. Considering the lightweightness and simplicity during the implementation, we use hostapd to deploy RADIUS server for evaluating EAP-TLS and PEAP. For EAP-TTLS, we use FreeRADIUS for better protocol compatibility. We have registered the domain name `anonymous.domain`, and purchased a certificate for `radius.anonymous.domain` which is trusted by all devices. Following the common practice, we set PEAP (phase 1) and MSCHAPV2 (phase 2) as the default protocols.

In our setup, the evil twin is implemented as a replica of the target network with the following three modifications. First, the evil twin uses a self-signed CA certificate under `test.domain`, and a fake server certificate signed by it. Second, the evil twin uses a different channel to avoid signal interference. For example, if the target network occupies the 5GHz channel, then the evil twin is deployed on 2.4GHz. Third, the RADIUS server is modified to allow connections from any devices. To this end, we disable client validation by modifying the `eap_server_tls_ssl_init` function in `src/eap_server/eap_server_tls.c`.

Table 2: Fields and Options on Android WPA Enterprise Configuration UI

Field	Available Options
EAP Method	PEAP, TLS, TTLS
Phase 2 authentication	Null, PAP, MSCHAP, MSCHAPV2, GTC
CA Certificate	Null, Use system certificate, Do not validate
Domain	Null, anonymous.domain, malicious.domain
User Certificate	Null, client.crt, Do not provide
Identity	A
Password	P(A)

Testing Procedure. We enumerate all possible combinations of the options listed in Table 2, when setting the UI fields (recall this in Figure 2(a)). For the "*User Certificate*" field, we generate a client certificate (denoted by `client.crt`) as the input. We also create a username/password pair *A/P(A)* for authentication. To test the correctness of hostname validation, we purchase a certificate trusted by all devices for the `malicious.domain` which is an input to the "*Domain*" field. Each combination is regarded as a test case. A test case can be represented as a tuple, for example, ⟨"PEAP", "PAP","*Use system certificate*","*anonymous.domain*" ,"*client.crt*", *"A"*, *"P(A)"*⟩.

Besides the GUI, Android also supports in-application Wi-Fi configuration via the `WifiManager.addNetworSuggestions` programmatic API. It is extensively adopted by enterprises to implement their BYOD schemes. Therefore, we also build an application which incorporates our test cases to test this API. Our testing application is released at [25].

We execute the test cases in two steps using Pixel 4 installed with Android 10 (AOSP). In the first step, we turn on the target network while keeping the evil twin off, and then configure the connection using one test case. If the supplicant is successfully connected, we proceed to the second step; otherwise, we discard the current test case and set the connection with a new test case. In the second step, we turn on the evil twin and turn off the target network, and check if the supplicant switches to the evil twin (connection hijacked) or if the username/password is leaked to the evil twin. To be practical, the user is not required to re-configure the subsequent connections after the initial connection. Nevertheless, the user may re-click on the network SSID or authorize reconnection if prompted.

4.2 Testing Results on Android 10

During the experiments with the generated test cases, we have observed the following weaknesses from the Wi-Fi configuration of Android 10.

Observation #1: When the CA certificate is not validated (i.e., the test cases ⟨ *, *, "*Do not validate*", "*Null*",*, *"A"*, *"P(A)"* ⟩), the supplicant is directly connected to the evil twin in EAP-TLS. In EAP-TTLS and PEAP, the username/password pair (i.e., *A/P(A)*) for PAP

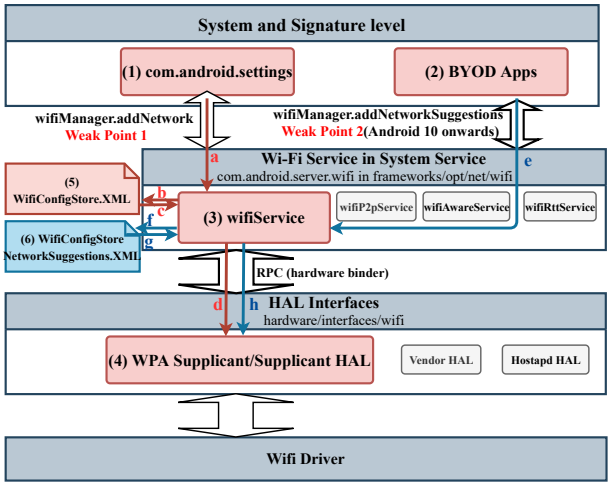


Figure 7: Architecture for WPA Supplicant in Android

and GTC, and the challenge-response hash value for MSCHAPV2 are sent to the evil twin.

Observation #2: When adding the network configuration automatically through our application tester using seemingly secure test cases ⟨"PEAP"/"TTLS", "MSCHAPV2", "*Use system certificate*",*, "`client.crt`", *"A"*, *"P(A)"*⟩, the supplicant can still connect to the evil twin.

Observation #3: When both CA certificate and phase-2 protocol are not specified for EAP-TTLS/EAP-PEAP (i.e., the test cases ⟨"PEAP"/"TTLS", *"Null"*, "*Do not validate*", "*Null*",*, *"A"*, *"P(A)"*⟩), the supplicant uses the phase-2 protocol specified by the evil twin (we set it to MSCHAPV2 in our experiment), and the challenge-response hash is obtained by the evil twin.

4.3 Investigating Causes of the Observations

We review the source code of the `android.net.wifi` package and the available source code from the other manufacturers to confirm the three observations and investigate their root causes. Through this, we have located three weak points. To facilitate the understanding of our investigation, we first brief Android's Wi-Fi management in Section 4.3.1. Then, we detail the identified weak points in Section 4.3.2.

4.3.1 Internals of Android Wi-Fi Manager. Figure 7 shows the components in each layer of Android OS that are involved in Wi-Fi management. As mentioned in Section 4.1, the enterprise network for Android devices can be configured manually through the Settings UI, or automatically through applications built on the Wi-Fi suggestion API, e.g., applications developed by BYOD-supporting enterprises (referred to as BYOD apps hereafter).

Configuration through Settings UI. The Android Settings application invokes the `WifiManager.addNetwork` API to interact with the system service `WifiService` through AIDL (Android interface description language). The WPA supplicant is further configured for Wi-Fi connection by `WifiService` through hardware binder (the configuration path is represented as (1)-a-(3)-d-(4) in Figure 7). Upon the initial connection to a wireless network, `WifiService` generates `WifiConfigureStore.XML` which is a configuration file containing the settings from the connection (the generation path is (1)-a-(3)-b-(5) in Figure 7). It is saved to automatically restore Wi-Fi configurations upon device rebooting (the configuration path is (5)-b-(3)-d-(4) in Figure 7).

Configuration through BYOD Applications. Before Android 10, BYOD applications utilize the same configuration path as the Android Settings. From Android 10 onwards, as part of the effort to standardize services available to the third-party applications through the Wi-Fi suggestion API, a distinct configuration path is adopted. The `WifiConfigStoreNetworkSuggestions.XML` configuration file is created by the `WifiNetworkSuggestion` class in `WifiService` through a BYOD application (the generation path is (2)-e-f-(6) in Figure 7). The specified configuration is then assessed and transferred to the WPA supplicant through `WifiService` (the configuration path is (6)-g-(3)-h-(4) in Figure 7).

4.3.2 Vulnerabilities Identified. By analyzing the configuration paths and the relevant source codes for the APIs and interfaces, we locate the weak points that lead to the three observations. Their locations are labelled in Figure 7. We also detail the function call flows of the identified weak points in our technical report [25].

Weak point #1: *insecure settings in `addNetwork` API*. This weak point leads to the Observation #1. The insecure settings in class `WifiEnterpriseConfig` allow vulnerable options in the configuration UI. In particular, `setCaCertificate` can be set as "*Null*" (default setting until Android 7), which leads to the "*Do not validate*" option in the UI. Under such a configuration, certificate validation is disabled because no certificate path is given in the file `WifiConfigureStore.XML`. As a result, any sever certificate is accepted.

Weak point #2: *flawed implementation in `addNetwork-Suggestions` API*. This weak point leads to the Observation #2. We have revealed that the original Wi-Fi configuration is prone to be "overwritten" without proper authorization. When the victim is tricked into connecting to the evil twin (e.g., upon rebooting), the `addOrUpdateNetwork` method in `WifiConfigManager` class is invoked. The method will update the Wi-Fi configuration since the victim has connected to the target network before. During the update, the evil twin is able to replace the originally specified CA certificates with a compromised one.

Weak point #3: *adaptive setting on the phase-2 protocol*. This weak point leads to the Observation #3. From the source code file `eap_tls_common.c`, we find that when "*Null*" is set to the function `eap_peer_select_phase2_methods` (i.e., the phase-2 protocol is set to "*None*" or "*Unspecified*" in the configuration UI), the supplicant would query the RADIUS server for its supported protocols and automatically adapt to one of them. If the default protocol from RADIUS server is also supported by the supplicant, it will be selected. To confirm this weakness, we change the evil twin's default phase-2 protocol from MSCHAPV2 to PAP and re-run the test case. As expected, we are able to derive the user's login credentials.

4.4 Responsible Disclosure

We have promptly reported our findings to Google and the three weak points are acknowledged. In particular, weak point #1 has been fixed by the update in Android 11, weak points #2 and #3

Table 3: Summary of Reported Vulnerabilities

CVE/Vulnerability ID	Weakness	Affected OS	Root Cause	Patch
CVE-2020-12484	Weak point #1	Vivo OEM 10 and earlier	`setCaCertificate` in `WifiEnterpriseConfig` can be null	Set `setCaCertificate` and `loadCertificates` as non-null
CVE-2020-1836	Weak point #1	Huawei OEM 10 and earlier	`setCaCertificate` in `WifiEnterpriseConfig` can be null	Set `setCaCertificate` and `loadCertificates` as non-null
CVE-2020-0201*	Weak point #3	Google AOSP 10 and earlier	Phase-2 protocol is allowed to be set as null in `eap_tls_common.c`	Remove the option `WIFI_PEAP_PHASE2_NONE`
CVE-2020-9260	Weak point #3	Huawei OEM 10 and earlier	Phase-2 protocol is allowed to be set as null in `eap_tls_common.c`	Remove the option `WIFI_PEAP_PHASE2_NONE`
A-150500247** (CVE-2020-0119)	Weak point #2	Google AOSP 10	Confused alias for saved Wi-Fi configurations	Create different aliases for suggested and the previously saved Wi-Fi configurations respectively

* The severity level for this vulnerability is rated by NVD [44] as critical with a base score of 9.8.
** This was found separately in parallel with Google.

are acknowledged as vulnerabilities listed on CVE. With the aim of minimizing possible security impacts, we also have tested other Android-based devices and reported to the respective manufacturers if the vulnerabilities also affect their devices. In particular, we are acknowledged with CVE-listed vulnerabilities corresponding to weak points #1 and #3. We ensure that the reported vulnerabilities have been securely fixed in their updates, before disclosing details of the vulnerabilities in this paper. All reported vulnerabilities have been fixed in Android latest version 11 in response to our findings. The available detailed bug reports and acknowledgment documents are uploaded to our online repository [25]. In summary, Table 3 lists the vulnerabilities together with their corresponding CVE/vulnerability IDs, root causes and patches.

5 WEAKNESS PREVALENCE IN CERTIFICATE VALIDATION UI

Fueled by the popularity of BYOD, WPA Enterprise has been supported among a great variety of devices: from laptops to smart phones, from Windows to iOS, and from obsolete OS versions to the latest ones. The diverse configuration options available in their certificate validation UIs may lead to security weaknesses discussed in Section 3.2. To better understand and further characterize them, we extend our analysis from Android to the mainstream OSes and devices, following the same methodology detailed in Section 4.1.

We examine laptops and smartphones which are the main devices for enterprise network access. To be representative, we select OSes and their most popular versions that cover more than 95% of the global active users based on their market share [56], as listed in Table 4. We find that the weaknesses in the certificate validation UIs are ubiquitous among these OSes and devices. In addition, we also notice that when secure options are configured, all of them manage to correctly validate the certificate, including the certificate chain and hostname. This suggests that the SSL/TLS libraries have been securely implemented.

Windows. All versions of Windows (i.e., Windows 7, 8 and 10) prompt for users' permission before continuing with the authentication in Phase 1 (P1-1), as demonstrated in Figure 3(c). As part of a secure procedure, the user should check the fingerprints of the server certificate by clicking on the button "*Show certificate detail*".

Table 4: Evaluation of Security Property in Various OSes

Properties	OS/Versions				
	Windows 7, 8 and 10	MacOS 10 and 11	iOS 9-14	Android (AOSP) 6-10	Android (OEM) 7-10
P1-1 (phase 1)	✗	✗	✗	✗	✗
P1-2 (phase 1)	✗	✗	✗	✗	✗
P2-1 (phase 2)	✓	✓	✓	✗	✗

✓ denotes the property is satisfied, and ✗ denotes otherwise.

Taking Windows 8.1, whose default phase-1 protocol is EAP-TTLS and phase-2 protocol is PAP, as an example, the user's username/-password pair would be sent to the evil twin immediately after the user clicks "*Connect*". We have reported this vulnerability to Microsoft (MSRC Case number 62537), who responds that it is the responsibility of the network administrator to disable this prompt so that the certificate invalidity would lead to connection failure.

Windows 7, 8 and 10 suffer from the lack of conspicuous warnings when the certificate validation setting is insecure (P1-2). For example, no warning is issued when the user disables the server certificate validation during manual configuration (all connections including the initial connection) in PEAP, as is shown in Figure 2(c). In addition, there is also no warning when a suspicious CA certificate (e.g., issued by an untrusted CA or unrecognized hostname) is received.

iOS and MacOS. Similar to Windows, all latest versions of iOS (v9-14) and MacOS (v10-11) seek the user's authorization to accept the certificate from the network being connected in phase 1 (P1-1). A user needs to verify that both server certificate name and issuing CA correctly match those adopted by the authentic network before trusting it. Otherwise, the connection would be hijacked and credentials would be disclosed. There is no warning when the user selects to trust a suspicious certificate and proceeds with the connection (P1-2). iOS/MacOS users are not given the choice for phase-2 protocols since the detailed configuration for the network is provided by the configuration profile crafted by the network administrator.

Android (AOSP) before V10. All Android versions up to version 10 have the option of voiding or skipping the CA certificate validation for phase 1 (P1-1), which remains a notable threat to the wireless connection security. More specifically, Android 6 or earlier has the CA certificate option "*Unspecified*" by default, while Android 7 and versions onwards remove this choice but add the "*Do not validate*" option. There is a warning on the insecure connection during the initial configuration if CA certificate is not validated. However, there are no warnings during subsequent connections, and there is also no warning when a suspicious CA certificate is received (P1-2). Versions before Android 10 allow the phase-2 authentication protocol to be left blank (P2-1), giving the attacker chance to manipulate the protocol selection.

Android (OEM). It is well known that Android has been suffering from the notorious *fragmentation* problem—due to the openness of AOSP (Android Open Source Project), Android OEM versions deployed by different device manufacturers have seen a great variety of unique features and functionalities. The security issues caused by fragmentation have been raised in a previous study [70]. Thus, we conduct an investigation into 9 popular Android phones available in

the market. They are from 8 distinct phone manufacturers, including Samsung, Xiaomi, Huawei, Vivo, Oppo, Lenovo, Smartisian and OnePlus.

We have found vulnerabilities similar to those in Android AOSP, with a few exceptions. Oppo Android 7 and Vivo Android 8 provide the option of "*Unspecified*" for CA certificate, which has been fixed since Android 6 in AOSP. These two brands/versions, together with Samsung Android 9 and Lenovo Android 9, fail to provide any warning against the initial insecure configurations. All our findings have been reported to the manufacturers (see Section 4.4).

6 ASSESSING USER SUSCEPTIBILITY

A study by CISCO in 2020 [28] shows that manual configuration remains the dominant means for configuring enterprise network connections, indicating that weaknesses related to certificate validation UIs have profound impacts on users' wireless security in practice. Due to the wider discrepancies in security awareness levels, rich compatibility of versions and types of devices, these weaknesses raise complications that possibly lead to attacks in reality. In this section, we aim to better understand the user susceptibility to weaknesses in certificate validation UIs through two user-related studies, from the perspectives of both organizations and users.

6.1 Examining Universities' Wi-Fi Guidelines

The complexity of the configuration process could cause misinterpretations of enterprise network administrators. Thus, the Wi-Fi connection guidelines made by them are an ideal source that reflects how they interpret the security of WPA Enterprise. Therefore, we conduct a study regarding the security of the guidelines provided by universities. To find publicly accessible guidelines, we resort to the websites of universities. For each of the top 200 universities listed in QS World University Rankings [50], we query Google with "*wifi setup site:*(university domain name)" to retrieve the web pages and/or PDF documents that contain Wi-Fi configuration guidelines. We review them to identify the weaknesses discussed in this paper.

Among the 200 universities, 176 of them have made their guidelines publicly available. Through reviewing those available ones, we find that nearly half of the universities have documented insecure configurations, as is shown in Figure 8. The mistakes found from the insecure guidelines are similar. For Windows, iOS and MacOS, users are instructed to skip the server and CA certificate validation when they are prompted with "*Continue* (to connect)". The insecure guidelines miss reminding the user of clicking "*Show the details*" and checking both certificates before proceeding to connect. In addition, for Windows, the majority of the insecure guidelines have instructed the users to deselect the option of "*Verify the server's identity by validating the certificate*". For Android, many guidelines tend to neglect CA certificate validation and instruct the users to choose "*Unspecified*" or "*Do not validate*", instead of "*Use system certificate*".

6.2 A Practical Study on Attack Feasibility

To study the evil twin attacks in real-world enterprise networks, we conduct a two-step confined experiment within ByteDance (an IT company with over 50k employees), prior and after the implementation of Wi-Fi security enhancement. Our experiment simulates an attack that aims to harvest the employees' login credentials, through a stealthily deployed evil twin in a spot where the mobility of employees is high. Even most employees have relevant experience and expertise in computer networking in this company, the success of the evil twin attack implies its high feasibility and effectiveness.

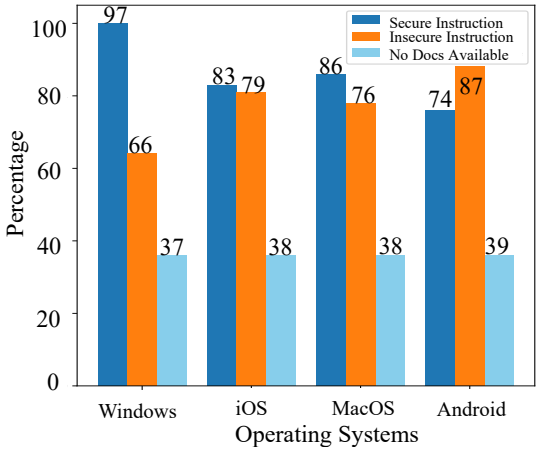


Figure 8: Results from Analyzing Wi-Fi Configuration Guidelines from Top Universities

6.2.1 Ethical Considerations. We have given careful ethical considerations on the user-related studies in our work, aiming to minimize the potential impact to the participants. Prior to the experiments, we obtained clearance from the company's legal department. The study and experiment were guided by them, ensuring compliance with privacy laws. We have carefully designed our experiment to be responsible and harmless. Any data collected throughout the experiment only serves the purpose for exploring the feasibility of the evil twin attack, and for the internal evaluation of the security vulnerabilities within the company. The study serves a part in the optimal goal to enhance the overall network security within the company. Upon completion, the company debriefed the participants with security advice (e.g., adopting secure configurations), and erased all collected data.

6.2.2 Evil Twin Experiment prior to Security Enhancement. We deploy the evil twin at the entrance of an office belonging to the R&D department to cover a sufficient number of employees. The setup of the evil twin follows the system configuration detailed in Section 4.1. Because PEAP is supported by the company wireless network (whose default protocol is PEAP-MSCHAPv2), we configure the evil twin to support only PEAP-GTC so that it can trigger phase-2 protocol downgrading (see week point #3 in Section 4.3.2) to obtain the user's credentials in plaintext. The evil twin is configured to automatically initiate the attack once the employees move into the effective range. To confine the impact from this study, we restrict the experiment time to a total of continuous 40 minutes.

During this experiment period, there are around 400 employees in the vicinity. The evil twin manages to collect passwords from the mobile devices of 166 employees without being noticed by any of them. This alarming number during the short period raises an alert on the severity of the security threat from the evil twin attack. We

note that we have thoroughly and strictly followed the responsible conduct detailed in Section 6.2.1. The collected information has been handled by the dedicated specialists from the company, and the authors have no access to these credentials. The "victims" have been debriefed by the specialists, after the completion of the experiment.

We have also requested the specialists to investigate the causes of these leakages. Based on their feedback, we summarized the following findings. For "victims" using Android devices, the leakage is because they have chosen not to validate the CA certificate in their initial connection to the company network, prior to our experiment. Therefore, the credentials are automatically sent to the evil twin as elaborated in our Observation #1 (see Section 4.2). For "victims" using iOS, they are prompted with a request to trust the fake certificate when their devices are connected with the evil twin. As most of them choose to trust the certificate, their credentials are automatically sent to the evil twin.

6.2.3 Evil Twin Experiment after Security Enhancement. In response to the results of our initial experiment, the company has implemented security countermeasures, including replacing PEAP with EAP-TLS as the default protocol, eliminating insecure options from their BYOD apps, conducting network security training through seminars and courses. To evaluate the effectiveness of those countermeasures, we conduct a similar experiment one year later after the initial one. We deploy the evil twin at the main entrance located in one of the company's office buildings for 6 hours which covers the morning rush hour. Around 8,000 connection attempts from over 1,400 unique devices are captured. The simulated attack manages to collect 14 distinct credentials in plaintext during the experiment.

Apparently, the overall Wi-Fi security has been drastically improved over the year, with much fewer employees affected by the attack. Nevertheless, it still has not been completely eliminated even after the security countermeasures are implemented, suggesting its high susceptibility. The specialists attributes the 14 compromised credentials to the backward compatibility of the earlier devices/OSes. They either do not support EAP-TLS or still allow insecure manual certificate validation settings.

```
diff --git
a/src/com/android/settings/wifi/WifiConfigController.java
b/src/com/android/settings/wifi/WifiConfigController.java
index 48c9e54..27ac69d 100644
--- a/src/com/android/settings/wifi/WifiConfigController.java
+++ b/src/com/android/settings/wifi/WifiConfigController.java
...
@@ -662,9 +666,6 @@
switch(phase2Method) {
- case WIFI_PEAP_PHASE2_NONE:
- config.enterpriseConfig.setPhase2Method(Phase2.NONE);
- break;
case WIFI_PEAP_PHASE2_MSCHAPV2:
config.enterpriseConfig.setPhase2Method(Phase2.MSCHAPV2);
break;
...
```

Figure 9: Patch for `WifiConfigController.java`

7 DISCUSSION

7.1 Mitigation

Our studies in Section 6 suggest that it is nearly impossible to completely eradicate the threat from evil twin attacks in large organizations, due to the complexity such as the diversity of devices, the continuously maintained backward compatibility, and the evolving functionality and interoperability. In this section, we discuss possible countermeasures to mitigate the security threats and reduce the potential attack surface.

Securing Network Configuration from OS providers. We recommend the OS providers to eliminate the insecure configuration options, such as the option of disabling certificate validation or selection of weak phase-2 authentication protocols including PAP and GTC. They should also maintain consistent security mechanisms in both of their certificate validation UIs (e.g., Android Settings) and programmatic APIs (e.g., `android.net.wifi.WifiManager`). Take the patch shown in Figure 9, which is Google's patch [48] for our reported CVE-2020-0201, as an example. The user should not be presented with the option `NONE` when choosing the phase-2 authentication protocol. Additionally, for the certificate validation, the domain of the authentication server should be made compulsory. For the adoption of password-based phase-2 protocols, PAKE (Password-Authenticated Key Agreement) authentication methods [47, 52] should be used to provide better security.

Furthermore, access to security critical assets such as the Wi-Fi KeyStore should be rigorously controlled so that no unauthorized manipulation is allowed. For example, the patch for our reported vulnerability A-150500247 fixes `WifiConfiguration.java`, as shown in Figure 10. It creates different aliases for suggested and the previously saved Wi-Fi configurations respectively to avoid the unauthorized manipulation on the saved ones.

WPA supplicants should also be implemented with strategies for mandatory certificate validation and displaying conspicuous warnings to users once suspicious CA certificate are received. Their certificate validation UIs are recommended to follow the design in web browsers' certificate UI [24, 63], which displays a conspicuous padlock icon and actively blocks suspicious access points. Unlike in the web browsers where the URLs are associated with the SSL/TLS certificates, the network SSIDs and the certificates are not correlated in Wi-Fi networks, rendering it a challenge to effectively detect the suspicious CA certificates. One possible solution is to use the trust-on-first-use method to pin the certificate upon the initial connection and issue warnings when the certificate fingerprint mismatches the saved one.

Securing Network Configurations on User Devices. To prevent their employees from crafting insecure Wi-Fi configurations, enterprises should provide standardized BYOD Wi-Fi configuration applications, such as the Eduroam Configuration Assistant Tool (CAT) used in universities. In such applications, the developers should ensure the correct certificate validation and the secure settings of the EAP methods and phase-2 authentication protocols. In addition to the more secured EAP-TLS, when implementing EAP-TTLS and PEAP protocols, MSCHAPv2 should be mandated as the phase-2 authentication protocol to provide better password secrecy. Note that the Wi-Fi access is configured through configuration profiles, the enterprise must ensure such secure settings are correctly specified when generating the profiles.

```
diff --git
a/wifi/java/android/net/wifi/WifiConfiguration.java
b/wifi/java/android/net/wifi/WifiConfiguration.java
index ed41642..88f2bb2 100644
--- a/wifi/java/android/net/wifi/WifiConfiguration.java
+++ b/wifi/java/android/net/wifi/WifiConfiguration.java
...
@@ -2113,15 +2113,23 @@
-  return trimStringForKeyId(SSID) + "_" + keyMgmt + "_" +
       trimStringForKeyId(enterpriseConfig.getKeyId(current != null
       ? current.enterpriseConfig : null));
+  String keyId = trimStringForKeyId(SSID) + "_" + keyMgmt + "_" +
        trimStringForKeyId(enterpriseConfig.getKeyId(current != null
       ? current.enterpriseConfig : null));
+  if (!fromWifiNetworkSuggestion) {
+  return keyId;
+  }
+  return keyId + "_" + trimStringForKeyId(BSSID) + "_" +
       trimStringForKeyId(creatorName);
...
```

Figure 10: Patch for `WifiConfiguration.java`

7.2 Implication on Future Research

Our findings in this work shed light on the importance of securing the emerging non-browser user-agents, e.g., IoT devices, for their certificate validation that involves error-prone human interactions. We have identified three research directions following this work.

Comprehensive Taxonomy of Certificate Validation Weaknesses. As discussed in Section 3.2, we reveal the need for taking into consideration the UI/UX design to secure a certificate validation process. This complements existing studies that mainly target the programmatic interfaces. A comprehensive taxonomy of the certificate validation weakness can subsequently encourage more systematic security assessment in the existing and future certificate validation schemes. Furthermore, it can benefit industry standards of secure use of SSL/TLS, e.g., NIST.SP.800-52 [42] and FIPS.140-2 [43].

Designing Secure Certificate Validation Involving Human Factors. Our work also calls for the expertise from researchers of human-centric security. We find that current certificate validation implementations trade off security for usability—the by-default options (see Section 4.2) save the users from specifying a server domain or maintaining a client certificate, but downgrade the security. We envision a secure yet user-friendly design of GUIs to facilitate the users' decision making in certificate validation processes.

Rigorous Analysis on Certificate Validation Implementations. Another possible research direction is on the rigorous analysis of the certificate validation implementations, especially when humans are in the loop. Considering the diverse combinations of available settings/options among various software versions, and the complexity in usage scenarios (e.g., initial and subsequent connections) and attack scenarios, it is infeasible to rely on manual testing or empirical studies for a thorough analysis. Thus, future work can utilize and extend the existing automated formal verification techniques to identify the potential security weaknesses/vulnerabilities in certificate validation.

8 RELATED WORK

To our best knowledge, security vulnerabilities in WPA Enterprise have not been systematically and comprehensively studied. Vulnerabilities found in our work are closely related to flawed certificate validation in the PKI which is extensively studied in the context of *wireless authentication* (for example, Wi-Fi authentication) and *encrypted secure communication* (i.e., SSL/TLS, HTTPS, etc.).

8.1 Analysis on Wi-Fi Authentication

Authentication Attacks. Various security protocols and standards have been developed to secure the wireless access, including WEP, WPA, WPA2 and WPA3. Despite providing better security compared to WEP, the state-of-the-art WPA standards have been continually found vulnerable [54, 66]. Cassola et al. [16] presented a novel and practical attack against WPA Enterprise, leveraging the combination of active jamming, design deficiencies in wireless management user interfaces and insecure trust model for wireless authentication. Vanhoef et al. [67] presented a MITM attack by reinstalling an already-in-use key during the 4-way handshake in WPA2 which leads to potential hijack of the user's Wi-Fi connection. In their latest work, Vanhoef et al. [68] conducted a systematic evaluation on the Dragonfly handshake protocol, and found vulnerabilities that result in downgrade, DoS attack and side-channel password leakage.

Compared to the existing works which focus on identifying vulnerabilities in the cryptosystem of WPA standards, we focus on the weaknesses originated from the certificate validation UIs. To the best of our knowledge, only several works in the literature studied WPA Enterprise in a systematic manner. Bartoli et al. [6] investigated the prevalence of WPA2 Enterprise vulnerabilities arises from incorrect client devices. Brenza et al. [13] demonstrated a similar MITM attack as discussed in our paper. The attack exploited the default settings in the client devices such that the user connection is hijacked without necessity of cracking the user password. In comparison, our work utilizes the adaptive settings on the client device to launch the attack, which can further retrieve the user password in plaintext.

Defences in Wi-Fi Authentication. Vanhoef et al. recommended that access points should disable WPA-TKIP, which is known as a weak encryption mechanism [65], to avoid the downgrade attack [66]. They also proposed countermeasures against key reinstallation attack [67], including disabling the reinstallation of already-in-use keys and implementing single-use keys during handshakes. Several works [53] have proposed secure device pairing to prevent the evil twin attacks, using properties in or around the communicating devices to establish their identities.

8.2 Certificate Validation in Web Apps

Analysis on HTTPS. Numerous prior works [2, 22] have revealed the certificate validation vulnerabilities. Kumar et al. [37] presents Zlint, a tool to systematically analyze the correctness of certificates issued by CAs. They examined over 240 million browser trusted certificates and identified an error rate of 0.02% which could lead to failed certificate validation. With a focus on the chain of trust (i.e., the X.509 certification path), Chuat et al. [17] analyzed the existing solutions to certificate revoking problems in HTTPS, including latency, availability and privacy.

Analysis on SSL/TLS. As the cornerstone for higher level communication protocols (e.g., HTTPS), SSL/TLS implementations have been extensively analyzed [14, 19, 32, 39]. Among them, a recent study from Tian et al. [64] proposed a differential testing approach named RFCcert to examine the SSL/TLS certificate validation implementations, based on the standard RFCs. They deployed RFCcert to test 6 popular SSL/TLS implementations and 3 web browsers.

8.3 Certificate Validation in Mobile Apps

Identifying Certificate Validation Vulnerabilities. With ubiquitous adoption of encrypted channels using PKI, the security of communication on mobile applications hinges on the correct certificate validation in particular. Poorly designed APIs and flawed SSL implementations have been found to be the main source for the certificate validation vulnerabilities [23, 27, 46]. More recently, Liu et al. [40] investigated a particular SSL vulnerability that originated from the error-handling code in the hybrid mobile apps which uses both native Android UIs and web UIs. This vulnerability will enable the continued communication regardless of SSL certificate validation failures. Stute et al. [57] analyzed the security and privacy of Apple Wireless Direct Link (AWDL) and its integration with BLE. The authors identified potential MITM attack of AirDrop service in around 40% of the surveyed users. In addition to the listed works on certificate validation in web and mobile apps, there are works proposing attacks on the cryptosystem of SSL/TLS [5, 12], causing broken certificate validation in PKI.

Defenses in Certificate Validation. Various mechanisms have been proposed against the failed certificate validation [7–9]. Recently, Nguyen et al. [45] developed FixDroid which is an IDE plugin to help developers adhere to best security practices, and reduce certificate validation vulnerabilities.

8.4 Exploitation of User Interfaces

The attacks presented in our work exploit the vulnerabilities residing in the user interfaces where security related information fails to be fully understood or aware by the users. In particular, an evil twin abuses the legitimate SSID to deceivingly appear as a benign access point to the victim user. Similarly, a number of browser-based attacks such as clickjacking [3, 31, 69] and phishing [33] trick user into clicking the malicious counterparts bearing a similar URL or name. To mitigate the security impacts from the attacks, various countermeasures and detection techniques have been proposed and evaluated for clickjacking [15, 31] and phishing [29, 58] respectively in the literature.

9 CONCLUSION

In this work, we present a comprehensive study on the effectiveness of certificate validation UIs in WPA supplicants. Our analysis has covered a wide variety of 13 mobile devices and 16 versions of the mainstream OSes. We have revealed that the majority of WPA supplicant implementations are susceptible to evil twin attacks due to deficient configuration interfaces and inconspicuous security warnings in the certificate validation UIs. We have investigated the root causes of the vulnerabilities in Android, and identified four CVEs and one Google-confirmed vulnerability. We have further confirmed users' susceptibility to such weaknesses through two user-related studies. We have also recommended mitigation to minimize the threats due to evil twin attacks. To the best of our knowledge, this work is the first of its kind in the literature. We intend to raise the awareness towards the easily-neglected enterprise security threat originated from the insecure certificate validation UIs.

ACKNOWLEDGMENTS

We would like to thank anonymous reviewers for improving this manuscript. This research is supported by the University of Queensland under Global Strategy and Partnerships Seed Funding, the NSRSG grant 4018264-617225, and UQ Cyber Seed Funding. This research is also supported by Singapore Ministry of Education Academic Research Fund Tier 3 under MOE's official grant number MOE2017-T3-1-007.

REFERENCES

[1] aircrack ng. 2008. https://www.aircrack-ng.org, visited in Feb 2022.
[2] Devdatta Akhawe, Bernhard Amann, Matthias Vallentin, and Robin Sommer. 2013. Here's My Cert, so Trust Me, Maybe? Understanding TLS Errors on the Web. In *Proceedings of the 22nd International Conference on World Wide Web*. 59–70.
[3] Devdatta Akhawe, Warren He, Zhiwei Li, Reza Moazzezi, and Dawn Song. 2014. Clickjacking Revisited: A Perceptual View of UI Security. In *WOOT*.
[4] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. SoK: Security Evaluation of Home-Based IoT Deployments. In *2019 IEEE Symposium on Security and Privacy (SP)*. 1362–1380.
[5] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. 2016. DROWN: Breaking TLS Using SSLv2. In *25th USENIX Security Symposium (USENIX Security 16)*. 689–706.
[6] 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*. Article 37, 6 pages.
[7] David Basin, Cas Cremers, Tiffany Hyuni-jin, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. 2016. Design, Analysis, and Implementation of ARPKI: An Attack-Resilient Public-Key Infrastructure. *IEEE Transactions on Dependable and Secure Computing* PP (08 2016).
[8] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. 2014. ARPKI: Attack Resilient Public-Key Infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 382–393.
[9] Adam Bates, Joe Pletcher, Tyler Nichols, Braden Hollembaek, Dave Tian, Kevin R.B. Butler, and Abdulrahman Alkhelaifi. 2014. Securing SSL Certificate Verification through Dynamic Linking. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 394–405.
[10] John Bellardo and Stefan Savage. 2003. 802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions *(SSYM'03)*. 2.
[11] Benjamin, Charles. Visited in Feb 2022. mschapv2acc. https://github.com/polkaned/mschapv2acc.
[12] Karthikeyan Bhargavan and Gaëtan Leurent. 2016. On the Practical (In-)Security of 64-Bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 456–467.
[13] 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*. Article 14, 11 pages.
[14] 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 *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. 114–129.
[15] Stefano Calzavara, Sebastian Roth, Alvise Rabitti, Michael Backes, and Ben Stock. 2020. A Tale of Two Headers: A Formal Analysis of Inconsistent Click-Jacking Protection on the Web. In *USENIX Security*.
[16] Aldo Cassola, William K. Robertson, Engin Kirda, and Guevara Noubir. 2013. A Practical, Targeted, and Stealthy Attack Against WPA Enterprise Authentication. In *20th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society.
[17] Laurent Chuat, AbdelRahman Abdou, Ralf Sasse, Christoph Sprenger, David Basin, and Adrian Perrig. 2019. SoK: Delegation and Revocation, the Missing

Links in the Web's Chain of Trust.
[18] D. A. Dai Zovi and Shane Macaulay. 2005. Attacking automatic wireless network selection. In *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*. 365–372.
[19] Joeri De Ruiter and Erik Poll. 2015. Protocol State Fuzzing of TLS Implementations. In *Proceedings of the 24th USENIX Conference on Security Symposium*. 193–206.
[20] Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs. 2005. https://tools.ietf.org/html/rfc4017, visited in Feb 2022.
[21] Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0). 2008. https://tools.ietf.org/html/rfc5281, visited in Feb 2022.
[22] Sascha Fahl, Yasemin Acar, Henning Perl, and Matthew Smith. 2014. Why Eve and Mallory (Also) Love Webmasters: A Study on the Root Causes of SSL Misconfigurations. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*. 507–512.
[23] 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.
[24] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. 2017. Measuring HTTPS Adoption on the Web. In *USENIX Security*. 1323–1338.
[25] Repository for supplementary material. 2021. https://github.com/anonymous-research-security/Supplementary-Document.git, visited in Feb 2022.
[26] Eric Geier. 2006. Wi-Fi Hotspots: Setting Up Public Wireless Internet Access. *Networking Technology* (2006).
[27] 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.
[28] Global Networking Trends Report. 2020. https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/global-networking-trends-report-executive-brief.pdf, visited in Feb 2022.
[29] Eder S. Gualberto, Rafael Timóteo de Sousa, Thiago P. De B. Vieira, João Paulo C. L. Da Costa, and Cláudio Duque. 2020. From Feature Engineering and Topics Models to Enhanced Prediction Rates in Phishing Detection. *IEEE Access* 8 (2020), 76368–76385.
[30] Boyuan He, Vaibhav Rastogi, Yinzhi Cao, Yan Chen, V.N. Venkatakrishnan, Runqing Yang, and Zhenrui Zhang. 2015. Vetting SSL Usage in Applications with SSLINT. In *2015 IEEE Symposium on Security and Privacy*. 519–534.
[31] Lin-Shung Huang, Alex Moshchuk, Helen J. Wang, Stuart Schecter, and Collin Jackson. 2012. Clickjacking: Attacks and Defenses. In *USENIX Security*. 413–428.
[32] Lin Shung Huang, Alex Rice, Erling Ellingsen, and Collin Jackson. 2014. Analyzing Forged SSL Certificates in the Wild. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. 83–97.
[33] Collin Jackson, Daniel R. Simon, Desney S. Tan, and Adam Barth. 2007. An Evaluation of Extended Validation and Picture-in-Picture Phishing Attacks. In *Financial Cryptography and Data Security*, Sven Dietrich and Rachna Dhamija (Eds.). 281–293.
[34] Wright Joshua. Visited in Feb 2022. ASLEEP - recovers weak LEAP password. https://github.com/joswr1ght/asleap.
[35] Kaushal Kafle, Kevin Moran, Sunil Manandhar, Adwait Nadkarni, and Denys Poshyvanyk. 2019. *A Study of Data Store-Based Home Automation*. 73–84.
[36] Christopher Kohlios and Thaier Hayajneh. 2018. A Comprehensive Attack Flow Model and Security Analysis for Wi-Fi and WPA3. *Electronics* 7 (10 2018), 284.
[37] 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 *IEEE Symposium on Security and Privacy*. 785–798.
[38] Yan Li, Yao Cheng, Weizhi Meng, Yingjiu Li, and Robert H. Deng. 2021. Designing Leakage-Resilient Password Entry on Head-Mounted Smart Wearable Glass Devices. *IEEE Transactions on Information Forensics and Security* 16 (2021), 307–321.
[39] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. 2015. An End-to-End Measurement of Certificate Revocation in the Web's PKI. In *Proceedings of the 2015 Internet Measurement Conference*. 183–196.
[40] Yang Liu, Chaoshun Zuo, Zonghua Zhang, Shanqing Guo, and Xinshun Xu. 2018. An Automatically Vetting Mechanism for SSL Error-Handling Vulnerability in Android Hybrid Web Apps. *World Wide Web* 21, 1 (2018), 127–150.
[41] Microsoft PPP CHAP Extensions. 1998. https://tools.ietf.org/html/rfc2433, visited in Feb 2022.
[42] National Institute of Standards and Technology. 2019. Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf,visited in Feb 2022.
[43] National Institute of Standards and Technology. 2021. Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program. https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/fips140-2/fips1402ig.pdf,visited in Feb 2022.
[44] National Vulnerability Database. Visited in Feb 2022. https://nvd.nist.gov.
[45] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. 2017. A Stitch in Time: Supporting Android Developers in WritingSecure Code. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1065–1077.
[46] Lucky Onwuzurike and Emiliano De Cristofaro. 2015. Danger is My Middle Name: Experimenting with SSL Vulnerabilities in Android Apps. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. 6 pages.
[47] PAKE Selection. 2019. https://github.com/cfrg/pake-selection#reviews-by-crypto-review-panel-round-2, visited in Feb 2022.
[48] Patch for Reported Vulnerability CVE-2020-0201. Visited in Feb 2022. https://android.googlesource.com/platform/packages/apps/Settings/+/f3848729b416a20a3d5d4b1a8e5a8794f727cbdc%5E%21/#F2.
[49] PPP Extensible Authentication Protocol (EAP). 1998. https://tools.ietf.org/rfc/rfc2284.txt, visited in Feb 2022.
[50] QS World University Rankings 2020. 2020. https://www.topuniversities.com/university-rankings/world-university-rankings/2020, visited in Feb 2022.
[51] RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP). 2003. https://www.ietf.org/rfc/rfc3579.txt, visited in Feb 2022.
[52] Requirements for Password-Authenticated Key Agreement (PAKE) Schemes. 2017. https://tools.ietf.org/html/rfc8125.
[53] Volker Roth, Wolfgang Polak, Eleanor Rieffel, and Thea Turner. 2008. Simple and Effective Defense against Evil Twin Access Points. In *Proceedings of the First ACM Conference on Wireless Network Security*. 220–235.
[54] Domien Schepers, Aanjhan Ranganathan, and Mathy Vanhoef. 2019. Practical Side-Channel Attacks against WPA-TKIP. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 415–426.
[55] Bruce Schneier, Mudge, and David Wagner. 1999. Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2). In *Secure Networking — CQRE [Secure]*. 192–203.
[56] StatCounter. Visited in Feb 2022. Operating System Version Market Share. https://gs.statcounter.com/os-market-share.
[57] Milan Stute, Sashank Narain, Alex Mariotto, Alexander Heinrich, David Kreitschmann, Guevara Noubir, and Matthias Hollick. 2019. A Billion Open Interfaces for Eve and Mallory: MitM, DoS, and Tracking Attacks on iOS and macOS Through Apple Wireless Direct Link. In *28th USENIX Security Symposium (USENIX Security 19)*. 37–54.
[58] Hiroaki Suzuki, Daiki Chiba, Yoshiro Yoneya, Tatsuya Mori, and Shigeki Goto. 2019. ShamFinder: An Automated Framework for Detecting IDN Homographs. In *IMC*. 449–462.
[59] The EAP-TLS Authentication Protocol. 2008. https://tools.ietf.org/html/rfc5216, visited in Feb 2022.
[60] The FreeRADIUS Server Project. Visited in Feb 2022. https://freeradius.org.
[61] The Host Access Point Daemon. Visited in Feb 2022. https://w1.fi/hostapd.
[62] The Openwall Project. Visited in Feb 2022. https://www.openwall.com/john.
[63] Christopher Thompson, Martin Shelton, Emily Stark, Maximilian Walker, Emily Schechter, and Adrienne Porter Felt. 2019. The Web's Identity Crisis: Understanding the Effectiveness of Website Identity Indicators. In *USENIX Security*. 1715–1732.
[64] Cong Tian, Chu Chen, Zhenhua Duan, and Liang Zhao. 2019. Differential Testing of Certificate Validation in SSL/TLS Implementations: An RFC-Guided Approach. *ACM Trans. Softw. Eng. Methodol.* 28, 4, Article 24 (Oct. 2019), 37 pages.
[65] Mathy Vanhoef and Frank Piessens. 2013. Practical Verification of WPA-TKIP Vulnerabilities. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*. 427–436.
[66] Mathy Vanhoef and Frank Piessens. 2016. Predicting, Decrypting, and Abusing WPA2/802.11 Group Keys. In *Proceedings of the 25th USENIX Conference on Security Symposium*. 673–688.
[67] Mathy Vanhoef and Frank Piessens. 2017. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1313–1328.
[68] Mathy Vanhoef and Eyal Ronen. 2020. Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd. In *IEEE Symposium on Security & Privacy (SP)*.
[69] Mingxue Zhang, Wei Meng, Sangho Lee, Byoungyoung Lee, and Xinyu Xing. 2019. All Your Clicks Belong to Me: Investigating Click Interception on the Web. In *USENIX Security*.
[70] Xiao-yong Zhou, Yeonjoon Lee, Nan Zhang, Muhammad Naveed, and XiaoFeng Wang. 2014. The Peril of Fragmentation: Security Hazards in Android Device Driver Customizations. In *2014 IEEE Symposium on Security and Privacy*. 409–423.