ANALYZING SECURITY AND USER PRIVACY IN NETWORK SECURITY PROTOCOLS

by

KAILONG WANG

(B.Eng., Nanyang Technological University)

A THESIS SUBMITTED FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

in the

SCHOOL OF COMPUTING

of the

NATIONAL UNIVERSITY OF SINGAPORE

2021

Supervisor:
Professor Jin Song Dong

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Kailong Wang

2021

Acknowledgments

First and foremost, I would like to express my sincerest gratitude to my supervisor Prof. Dong Jin Song for his unconditional support, wholehearted encouragement and indispensable guidance through the rewarding but adventurous journey. His unwavering enthusiasm for computer science and acknowledgement of my research interests have constantly kept me inspired, motivated and engaged during my pursuit of the PhD degree, which will undoubtedly continue to lead the way through my future career.

I would like to extend my deep and sincere thanks to my mentor and friend, Dr. Guangdong Bai for his continual support and immense encouragement throughout the dark and difficult times, and for his insightful and invaluable guidance in the area of security research. Without his mentorship, I would have never been able to complete this thesis. My appreciation also goes to Dr. Naipeng Dong for her kind support and guidance as my mentor.

I would like to thank all of my coauthors and collaborators for their valuable insights and precious contributions to the work of this thesis. Particularly, I would like to thank Miss Kulani Mahadewa, Dr. Shi Ling and Dr. Quanqi Ye for their continual support on my research. I would also express my appreciation to everyone in the Software Engineering lab for their friendship and companionship through the years I cherish most. At the same time, I am grateful for all the endless support from my best friends, especially Dr. L. G.

Finally, I am deeply indebted to my family, especially my parents, for their overwhelming love, tremendous support and understanding during this journey and during every moment in my life.

Contents

Abstract

With the ubiquity of network communications nowadays, network security protocols serve a key role in protecting users from the hostile Internet environment, safeguarding both security and privacy of the transmitted information. As a result of the complications in protocol development life cycle from design to implementation, the network security protocols, such as SSL/TLS and single sign-on (SSO) protocols, have been continually found error-prone. Given their critical role, vulnerabilities (such as logic flaws in the protocol design and certificate validation failure in the implementation) could easily compromise the end-to-end confidentiality, integrity and authentication. Additionally, those vulnerabilities also enable the adversaries to profile the network users based on their fine-grained online activity data. Hence, analyzing such security and user privacy properties in network security protocols become vital throughout the steps for deploying them in practice.

In this thesis, we focus on the security and privacy of the network security protocols by scrutinizing them on the design level (e.g., the protocol specifications, logics and control flows) and the implementation level (e.g., APIs, libraries, user interfaces and relevant applications).

The network security protocol designs are the basis to guarantee the desired security and privacy properties. Without a rigorous analysis, even trivial errors would easily propagate to the downstream implementations and applications. To address such problems, we propose a framework to facilitate the formal modeling and verification of the network security protocol designs. The framework covers the essential components for the analysis, including a formal model for web and network infrastructure, the attacker models and formal definitions of the security and privacy properties. To demonstrate its effectiveness and expressiveness, we have applied the framework to analyze four mainstream SSO protocols, which are network security protocols widely adopted for authentication among the user, the identity provider and the service provider. We have confirmed that they preserve the security property, and found one previously unknown privacy

attack.

In spite of the secure and privacy-respecting designs, flaws introduced throughout the network security protocol implementations could still be exploited and abused by the adversaries. In particular, the man-in-the-middle (MITM) attacks and the network traffic analysis are two of the notorious threats against the implemented protocols, targeting security and privacy properties respectively.

The evil twin attack is a type of MITM attack frequently found in the Wi-Fi Protected Access (WPA) Enterprise authentication scheme, commonly due to the failed certificate validation in the SSL/TLS based authentication protocols. To facilitate effective identification of such vulnerabilities and evaluate their security impacts, we propose a systematic analysis approach based on user interface testing. We apply the approach to comprehensively scan the laptops and smart phones installed with four mainstream operating systems, and have identified prevalent insecure user configuration options, among which five are severe vulnerabilities. They would compromise the server authentication and allow the victim to connect to rogue Wi-Fi hotspots. We further reveal that such vulnerabilities are rooted in the immature design and implementation of the WPA supplicants.

The pervasively deployed network traffic analysis have raised privacy concerns over the information leakage via the encrypted network communication channels such as HTTPS and Tor [141]. Even though the data integrity and confidentiality is protected by encryption, the network traffic patterns still inadvertently reveal information regarding user browsing activities. To gauge such information leakage and understand its privacy implications, we propose a novel web page fingerprinting technique based on features extracted from the characteristic traffic between a browser and a content delivery network (CDN) server during the web page rendering. Through a series of in-depth analysis and evaluations, we have demonstrated that the technique can achieve high accuracy in identifying similar web pages in the same web domain, which allows the adversary to further monitor the target user's browsing behavior.

List of Figures

List of Tables

Chapter 1

Introduction

Network protocols, which are established sets of rules standardizing and regulating the information exchange among different devices, serve as a pivotal part in governing and characterizing essential aspects of contemporary networks including network management (e.g., Simple Network Management Protocol (SNMP), Internet Control Message Protocol (ICMP), etc.), communication (e.g., TCP/IP, HTTP, FTP, etc.) and security (e.g., SSL/TLS, HTTPS, etc.). Among them, network security protocols have been pervasively utilized in countless scenarios ranging from messaging via social media and shopping on e-commerce websites, to access authentication in enterprises and communication among Internet of Things (IoT) devices. The protocols have been seamlessly incorporated into applications such as web browsers, identity authentication portals, secure digital transaction platforms, embedded software, etc., forming the cornerstone for safeguarding the security and privacy of billions of Internet users nowadays [162].

The network security protocols, SSL/TLS as representatives, are notoriously error prone [28, 84, 38, 56, 107]. During the protocol design stage, errors can easily be introduced due to the ambiguity in the semi-formally defined protocol specifications and standards. Such errors would consequently propagate to the protocol implementation stage or into other dependent protocols (e.g., HTTPS versus SSL/TLS). Even with the secure designs, vulnerabilities could still arise from the inevitable differences and deviations in the protocol implementations, which is further exacerbated by the emphasis on working code and interoperability across devices and platforms. Given

the critical role of network security protocols and the complexity of the Internet ecosystem, the security and privacy vulnerabilities in the protocol design and implementation could cause various catastrophic impacts.

On the one hand, insecure protocol designs and implementations would compromise the end-to-end confidentiality, integrity and authentication, directly exposing users to the hostile Internet and adversaries. Impacts of such vulnerabilities include personal data interception (e.g., reveal the private messages sent via social media), critical information disclosure (e.g., login credentials leakage), impersonation (i.e., the adversary pretend to be the legitimate communication participant) and even consequent financial frauds or involuntary cybercrimes.

On the other hand, the existing network security protocols, SSO protocols for instance, have mainly focused on the security properties (i.e., confidentiality, integrity and authentication), placing little emphasis on the user privacy. Therefore, the user's online activity information, such as the network services he/she has accessed over time, could be directly exposed to the adversary. Even if the communication is protected under strong network security protocols (e.g., Tor [141], OpenVPN [131] and WireGuard [205]), the fine-grained and use-specific network traffic patterns, including network packet sizes, packet arrival intervals, IP addresses and port numbers, could still be visible to the adversary enforcing network traffic analysis. In particular, encrypted network traffic fingerprinting [80, 41, 202] have demonstrated the possibility of tracking the websites accessed by the target user simply based on such traffic metadata.

To analyze and evaluate the security and privacy implications from such vulnerabilities in the context of network security protocols, three critical research questions should be raised and scrutinized.

- How to evaluate the security and privacy of the network security protocol designs?
- How effective are the network security protocol implementations in protecting against the security attacks such as MITM attacks?

- How effective are the network security protocol implementations in preserving the user-specific and privacy-invasive information against the monitoring techniques such as network traffic analysis?

These three questions have been studied by a wide range of research works in the literature [133, 78, 16, 170, 8, 44, 36, 22]. However, we have identified the following three gaps yet to be filled: the rigorous analysis framework on the security and privacy properties of the protocol designs; the systematic approach for detecting security vulnerabilities against MITM attacks among wireless authentication scheme implementations; the technique for evaluating the privacy leakage from encrypted network traffic of similar web pages.

Formal Analysis on the Security and Privacy of Network Security Protocol Designs. Network security protocols have been widely applied and incorporated for the authentication among network communication participants. One of the well-known examples is the SSO protocol, which is designed and built on top of the SSL/TLS protocols to enable the secure transmission of user authentication information. Unfortunately, vulnerabilities such as logic flaws have been continually found in the SSO designs [198, 65, 66]. Considering their importance in safeguarding users' sensitive and private data, a rigorous security assessment should be conducted before they are implemented and deployed for practical use. The existing studies mainly focus on analyzing the security property (i.e., the correct authentication/authorization between the protocol participants) [19, 18, 16], while the analysis on privacy property (i.e., unlinkability) has attract minimal attention.

The formal verification on the security and privacy properties for the protocols is challenging. The formal analysis demands an accurate formal model of the underlying web infrastructure which the protocols rely upon, which incorporates server side (e.g., web servers and SDKs), client side (e.g., web browsers) and various communication channels. Furthermore, due to the involvement of these multiple communication parties, the authentication/authorization systems in social media are naturally exposed to a large attack surface. To facilitate the analysis, it become imperative

to comprehensively formalize the behaviors of the adversaries and formally define the security and privacy properties.

Detecting the Security Vulnerabilities against MITM Attacks in Wireless Authentication Scheme Implementations. The ubiquity of mobile devices enables accessing the Internet with unprecedented convenience, which relies on the correct authentication between the user and the network service provider to secure subsequent wireless connection. Among the existing wireless authentication protocols (e.g., WEP, WPA), WPA Enterprise (also referred to as the WPA-802.1X mode), is the cutting-edge and the most commonly used security mechanism in enterprise-level wireless networks. Thus, it has been the focus of the wireless security analysis, and various vulnerabilities associated with its implementations have been revealed in the literature [44, 185, 150]. However, there is a lack of comprehensive and systematic studies targeting the security vulnerabilities against MITM attacks that cover diversified devices among users. Furthermore, well-structured complimentary user studies are demanded to characterize the real impacts of such vulnerabilities and the users' susceptibility.

The main obstacle to such comprehensive and systematic study is the heterogeneity of the mobile devices which are joined together by the increasingly popular BYOD (bring your own device) schemes. Considering the wide range of device types (i.e., laptops, smart phones), popular OSes (e.g., Android, iOS, Windows, etc.) and their versions, a universally applicable analysis approach should be proposed to facilitate the analysis. In addition, it is also challenging to fully understand the mechanisms leading to a certain vulnerability in the protocol implementations, as the source code from the device manufacturers is only partially available in many cases.

Analysis on the User's Privacy Leakage through Encrypted Network Traffic of Similar Web Pages. Even though deploying encrypted channels (e.g., established using HTTPS) has become a security principle for Internet communication [204] which preserves the integrity and confidentiality of the transmitted data, the metadata (e.g., IP addresses, network packet sizes and timestamps) is still inadvertently leaked and could be utilized to derive more privacy-intruding information contained in the encrypted

traffic (e.g., certain content from a specific website). Such attacks, known as inter-domain website fingerprinting (WSF) which aims to identify which web domain the victim user is accessing, has been proven a non-negligible threat to user's browsing privacy [78, 88, 91, 105, 133, 147, 203, 209]. Despite the trend of increasingly stringent monitor on social media [11, 165, 51] and the added emphasis of content-based network censorship [101], there is nevertheless few studies of web fingerprinting attacks in the domain of social media sites. We hereafter refer to such attacks as intra-domain web page fingerprinting (WPF) which aim to distinguish similar web pages within the same domain of a social media site.

Despite the success of inter-domain WSF and the availability of advanced learning techniques, intra-domain WPF remains a challenging and non-trivial task. In a typical social media website, almost all its web pages (such as Facebook user profile pages) are generated from the same template, such that similar traffic patterns are exhibited in the encrypted network traffic. Consequently, those previously distinguishable features that classification algorithms heavily rely on in inter-domain WSF are blurred out. Some "higher-definition" fingerprints capable of differentiating among such similar web pages are demanded. Such features should be highly individualized and robust in characterizing the subtle differences among the pages.

1.1 Thesis Overview

The main objective for this thesis is to *analyze and enhance the users' security and privacy in network security protocol designs and implementations, leveraging various techniques including formal methods, software testing and traffic analysis.* In particular, this thesis focuses on the following three perspectives:

- We propose a formal framework for modeling and verifying the security and privacy of network security protocol designs. We further apply the framework to analyze the mainstream SSO protocols.
- We propose an approach based on UI testing to evaluate the security

of wireless authentication protocol implementations. In particular, we target the correctness of certificate validations in the configuration UIs for WPA Enterprise authentication.

- We propose a web fingerprinting technique to study the privacy leakage from the encrypted network channels when users browse on similar web pages, or intra-domain web pages.

1.1.1 A Formal Analysis Framework for Security and Privacy in Single Sign-on Protocols

To address the challenges and facilitate the formal analysis on network security protocol designs, we propose a framework which consists of a formal model of the web infrastructure, the attacker models, as well as formal definitions of the security and privacy properties. We abstract the whole infrastructure into three parts, i.e., web browser, network and web server, which are essential to network security protocols. The attacker model covers both security attackers and privacy attackers. The model of security attackers considers the network attacker which is a typical Dolev-Yao model; the model of privacy attackers includes three types of malicious IDPs (identity providers), i.e., honest-but-curious IDP server which infers user activity information based on the messages generated or received, malicious IDP server which is an IDP server capable of sending fake information and malicious IDP client which is capable of repetitively calling browser API (e.g., to request the browser to open a new window). We use the applied pi calculus [2] as our modeling language in the framework, since applied pi models can be automatically verified using the state-of-the-art verifier Proverif [29, 31]. The security property is formalized as correspondence and privacy property is formalized as observational equivalence [57] in applied pi.

We apply our framework to analyze four SSO protocols, including the OAuth 1.0a [207] and the OAuth 2.0 [135] which have been implemented by major SSO schemes (e.g., Twitter Login (OAuth 1.0a), Google Sign-In (OAuth 2.0), OpenID (OAuth 2.0), Facebook Connect (OAuth 2.0), etc.),

the BrowserID protocol [37] and SPRESSO protocol [67] which have specially taken into consideration privacy property in their designs. These four protocols are representative and suitable to test our framework as modeling them covers most of the web techniques, including server-to-server communication, HTML5's cross-domain communication, AJAX (Asynchronous JavaScript And XML) and so on. Our analysis confirms that the four protocols preserves authentication property, and that OAuth 1.0a and OAuth 2.0 do not provide privacy protection. In addition, we have found that BrowserID and SPRESSO suffer from a previously-unknown flaw which allows a malicious IDP to use two key pairs to learn which users have logged into a particular service provider, or namely relying party (RP).

1.1.2 Assessing Certificate Validation User Interfaces of WPA Supplicants

We propose a software testing-based approach to 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 covers mainstream device types (laptops and phones) and mainstream OSes (Android, iOS, MacOS and Windows). We enumerate 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, 5 of them are severe vulnerabilities (4 CVE listed and 1 acknowledged by Google).

Furthermore, we conduct three studies to better understand the users' susceptibility to the evil twin attack on all OSes. First, we survey 129 WPA Enterprise users to investigate their behavior during the configuration process. The majority (68%) of the participants choose to accept the prompted certificate without doubting its validity. Then, we perform a review of the Wi-Fi configuration guidelines of the global top 200 universities, revealing that only 38% (76/200) of them manage to provide secure instructions to

their users. Last, we deploy an evil twin in an Internet technology company in a confined scenario. Within a 40-minute period, the simulated attack obtains the login credentials of 166 individuals, indicating the severity of the weakness in the existing certificate validation UIs.

1.1.3 Intra-domain Fingerprinting Social Media Websites Through CDN Bursts

Considering the media-sharing nature in social media sites, the large-size content such as photos and videos are typically hosted in content delivery networks (CDNs) for efficient delivery. We reveal that when a browser renders a social web page, the traffic between the browser and the CDN server may exhibit patterns that can uniquely characterize the page. Based on this finding, we propose a web page fingerprinting technique based on the burst patterns of CDN traffic. In our approach, a CDN burst is defined as an aggregation of several temporally adjacent network packets originated from a certain CDN server (identified by its unique IP address); a sequence of bursts further characterize a web page. From the perspective of the application layer, the personalized social media contents diversify the Document Object Model (DOM) trees among pages, which shapes the network stream segmentation during a web page rendering and influences how the contents are fetched from CDN servers. As a result, the information conveyed by CDN bursts, such as the size of a retrieved content and the intervals of retrievals, may (partially) expose the DOM structure and further be utilized used to characterize similar but distinct intra-domain web pages.

We conduct a series of in-depth studies to evaluate the web page fingerprinting approach. We first design an algorithm to extract CDN bursts incorporating both temporal and volumetric factors from the network traces, and to derive a feature set for classification. To measure the effectiveness of each CDN burst as a classification feature, we quantify the information leakage [105] contained in each CDN burst as mutual information between the web pages and the CDN burst. Our experimental evaluation shows that intra-domain WPF can achieve a performance at least as accurate as that

of the state-of-the-art inter-domain WSF, indicating that the information contained in the feature set is sufficient for fingerprinting social media pages. Finally, we propose countermeasures by obscuring the temporal and volumetric patterns that the attacker utilizes to construct CDN bursts.

1.2 Thesis Structure

The structure of this thesis is organized and summarized as follows.

Chapter 2. Background and Literature Review. This chapter includes a brief introduction on the techniques and related works relevant to the analysis on the designs and implementations of network security protocols, including formal analysis of protocol designs, software testing for security vulnerability detection and encrypted traffic analysis of user privacy intrusion evaluation.

Chapter 3. A Formal Analysis Framework for Security and Privacy in Single Sign-on Protocols. This chapter details the our framework for facilitating formal modeling of SSO protocols and analysis of their crucial security and privacy properties including authentication and unlinkability. Through the case studies on four extensively-used SSO protocols, this work highlights the need to include the previously-neglected privacy property in the analysis and verification.

Chapter 4 . Assessing Certificate Validation User Interfaces of WPA Supplicants. This chapter presents a testing-based approach to analyze the effectiveness of certificate validation user interfaces for protecting the connections to WPA Enterprise guarded networks. From our systematic and comprehensive study, we show that immature designs and implementations of WPA software modules can lead to common insecurities in the wireless configurations. We also illustrate that users can be highly vulnerable to such insecurities in practice.

Chapter 5. Intra-domain Fingerprinting Social Media Web-sites Through CDN Bursts. This chapter investigates the intra-domain WPF against social media websites, utilizing the classification feature extracted from encrypted network traffic patterns generated during downloading large-

size web content from CDN servers. Through the evaluations, we have shown that intra-domain WPF can be a non-negligible threat to the user privacy during social media browsing activity.

Chapter 6. Conclusion. This chapter concludes this thesis and discuss potential future work following this thesis.

1.3 Acknowledgement of Published Works

The listed publications and paper submission under review are from this thesis.

- **A Formal Analysis Framework for Single Sign-on Protocols** [195]. This work is presented in Chapter 3 and it has been published at the 13th EAI International Conference on Security and Privacy in Communication Networks (SecureComm'17).
- **Intra-domain Fingerprinting Social Media Web-sites Through CDN Bursts** [196]. This work is presented in Chapter 4 and it has been published at the 30th The Web Conference (WWW'21).
- **Assessing Certificate Validation User Interfaces of WPA Supplicants** [197]. This work is presented in Chapter 5 and this work is current submitted to a conference for peer review.

In addition, we have two publications remotely related to but not presented in this thesis.

- **Scrutinizing Implementations of Smart Home Integrations** [112, 111]. In this work, we propose an approach for analyzing the security of communication among IoT (internet of things) devices. We first extract the IoT communication protocols using a combination of static analysis, dynamic testing, and traffic analysis. Then we model the protcols into Labelled Transition Systems (LTSs) and apply formal analysis to evaluate the security properties.

Chapter 2

Background and Literature Review

This chapter presents a brief survey on the techniques and prior studies for analyzing the security and privacy properties of the designs and implementations of network security protocols. We mainly focus on three types of analysis techniques including formal methods on the protocol designs, software testing and encrypted network traffic analysis on the protocol implementations.

2.1 Formal Analysis on Network Security Protocol Designs

Formal methods have been applied in the security protocol design verification in the literature [124, 113, 164], which typically includes *property formalization* (i.e., the security and privacy goals the designed protocols should satisfy), *model construction* (i.e., formally specify the protocol and its relying infrastructure into the pre-defined formalisms) and *verification* (i.e., use the constructed models to verify if the desired properties are satisfied or not).

2.1.1 Security and Privacy Properties

Data confidentiality or secrecy is defined as the property that the target data (such as user sensitive information) handled by the network security protocol cannot be obtained by the adversary. The data confidentiality

is verified by querying if the data is present in the adversary's knowledge set [58]. When the data confidentiality is compromised, the adversary, Dolev-Yao adversary for example, could further manipulate (such as intercept, modify and insert) the data, compromising its **data integrity** [171].

Authentication is defined as the property that the participants of the network security protocol are confident about the identities of each other. This property has been supported by many formal methods [188, 190]. In particular, it is captured as correspondence [2] in the applied pi calculus: $B \rightarrow A$ describes that event B can only be executed if event A has been executed before.

User privacy is defined as the property that the adversary cannot obtain more information regarding a user, simply based on observing the user's interactions with the network security protocol. The privacy property can be captured by the **unlinkability property** where the adversary is unable to identify the links between two or more objects in a system [138, 163]. For example, in the scenario of web browsing, the property checks if the links can be established between a particular user and the websites or web contents he has accessed. Particularly in the applied pi calculus, the unlinkability property is formalized as the observational equivalence relation between two parallel processes [2]. Intuitively, the unlinkability property is satisfied if these processes are indistinguishable, or equivalent by the adversary's observation.

2.1.2 Formal Modeling and Verification Approaches

To formally analyze whether a protocol design satisfies the desired security and privacy properties, the current approaches follow two stages: first, formally model or specify the target system which includes the protocol, the relying infrastructure and other components such as adversary and protocol participants; second, verify the properties against the system model.

In the modeling stage, the targeted system is commonly specified using three types of modeling/specification languages: **state machine lan-**

guages, **modal logics** and **process algebra**. **State machine languages**, such as Z [159], VDM [89], Moore machines [120], model the systems into finite state machines (FSMs). A FSM consists of a set of states, a set of inputs, a set of outputs, a transition function, an output function and an initial state. Starting from the initial state, the finite state machine transits its state based on the trigger input from environments (as specified by transition function) and return outputs to the environment (as specified by output function). **Modal logics**, such as CTL [48] and BAN logic [39], model the systems into a collection of logic formulae and utilize the inference rules to derive the proofs of satisfaction or violation of certain properties. **Process algebra**, such as CSP [81], CCS [119], LOTOS [33] and applied pi calculus [2], model the system as distributed or parallel processes, each of which is defined by a sequence of events. Processes are joined together by operators such as sequential composition, choice, etc. Thus they are suitable for modeling security systems where interactions, communications, and synchronizations among various independent processes are common.

In the verification stage, the desired property of the target system is reasoned using two types of approaches: **theorem proving** and **model checking**. **Theorem proving** determines if the formulated property (i.e., the desired theorem) is satisfied or violated against the model, based on the axioms and inference rules such as induction and rewriting rules [106, 114]. There are different theorem provers available, including ProVerif [31], CryptoVerif [30] and Tamarin Prover [116], to facilitate the automatic analysis of security and privacy properties. **Model checking** applies exhaustive search on the given system model, which includes finite number of states, to check if the desired property is satisfied. If a violation is found, a counterexample will be produced. Various model checking tools have been proposed and used in analyzing the security property, including PAT [168], AVISPA [176], UPPAAL [103], PRISM [99], and SPIN [83].

2.1.3 Formally Verifying Security and Privacy Properties in SSO Protocol Design

In the context of analyzing SSO protocols with formal methods, the protocol model together with its relying web infrastructure model (i.e., network communication channels, browser, etc.), the attack models and the formalization of the desired security and privacy properties are the required inputs from the specification stage to the verification stage. Then the security and privacy properties are analyzed against the attack models while the protocol is executed together with the web infrastructure model.

Formal Modeling of Web Infrastructure. An accurate and comprehensive web infrastructure model is important to guarantee the accurate delivery of SSO protocol verification. Prior works on web authentication protocol security analysis [8, 87, 93, 170] considers a very limited web model. Subsequent works includes more detailed modeling of essential web infrastructure [67, 6, 18] (such as the encrypted communication channels and web browser) and components (such as the attacker models [17, 77]) for SSO protocol analysis.

Verifying Security and Privacy properties. Typically, the SSO protocol is first formally modeled and checked against a certain set of attack models. Based on the flaws reported by the model, they are further mapped into the real world web implementations for vulnerability confirmation. A number of prior works analyzed the security of SSO protocols including the SAML-based Web Browser SSO protocol from Google [8, 9], Windows Live ID [199, 16], OAuth [18, 199]and BrowserID [16, 65]. Few work has been done on the SSO privacy checking and verification. Fett et al. [65, 66] have analyzed the privacy property of BrowserID and found that the promised privacy property in BrowserID can be compromised.

2.2 Security and Privacy Analysis in Network Security Protocols Implementations

Formal methods are effective for identifying violations of security and privacy properties in the early stage of the network security protocol de-

ployment. However, even based on correct designs, vulnerabilities can still be easily introduced during the implementation stage, plausibly owing to factors such as discrepancies in the developers' security and privacy awareness. Thus, **software engineering techniques**, especially *software testing*, and **network traffic analysis** have been widely adopted to detect security and privacy property violations in the network security protocol implementations.

2.2.1 Software Testing

Software testing, including white-box testing and black-box testing, is a well prevalent technique for validating the desired quality of software in its development life cycle [123]. The white-box testing, such as taint analysis [49, 183] and symbolic execution [95, 14], examines the logic, structure and coding of the software to ensure the expected flow of input-output information. The black-box testing, such as various fuzzing techniques [172, 23], checks the functionality of the software based on its observed behaviors during information exchanges.

In the literature, there is rich prior research [63, 62, 82, 59, 68] focusing on testing the implementations of network security protocols, including SSL/TLS and their variants (e.g., HTTPS and Extensible Authentication Protocols) which form the cornerstone for securing the contemporary network communications. In these protocols, implementing certificate validation securely is the key to the correct identification of the legitimate communicating end-points for the subsequent interactions. However, certificate validation has been found notoriously error-prone, owing to the design flaws in the APIs of SSL/TLS implementations (e.g., OpenSSL [129], wolfSSL [206] and GnuTLS [178]) and data-transport libraries (such as cURL [52]). Given the confusing settings and options available to the developer in the flawed APIs, vulnerabilities could compromise the overall security in the network security protocols. Thus, we briefly discuss the software testing approaches for detecting certificate validation flaws in the literature, among SSL/TLS implementations and their applications in web

and mobile platforms.

Testing Certificate Validation in SSL/TLS Libraries. To effectively identify the flawed certificate validation, common approaches include guided [180, 38] and input-format-agnostic [137] differential testing with synthetic certificates, and protocol state fuzzing [56, 68].

Testing Certificate Validation in HTTPS Implementations. Due to the diversity and variety of implementations among web service providers, the certificate validation in HTTPS between clients (i.e., web browsers) and servers (i.e., web service providers) are generally tested and evaluated through large-scale measurements and screenings [59, 7, 3, 84, 97, 46]. Their study identified that a range of 0.02% to 0,2% of the SSL/TLS connections are insecure. Common errors identified include adoption of weak encryption algorithm, acceptance of invalid certificates (such as self-signed certificates, expired certificates, name mismatches related to subdomains) and incorrect chain of trust (i.e., the X.509 certification path).

Testing Certificate Validation in Mobile Applications. With extensive adoption of encrypted channels using PKI (public key infrastructure), the security of communication on mobile applications hinges on the correct certificate validation in particular. Techniques and tools [71, 128, 145, 166] with a combination of white-box analysis and black-box analysis have been proposed to test the mobile applications incorporated with SSL/TLS. They have reveal that certificate validation flaws are prevalent among such applications.

2.2.2 Network Traffic Analysis

The Internet traffic, which is the superposition of the source-to-destination network traffic flows, is crucial to characterize and address problems in various aspects of the network, including traffic measurement and performance optimization (such as traffic forecasting and capacity planning [139], traffic matrix estimation [210, 115] and traffic engineering [26, 155]), network security monitoring (such as anomaly detection [27, 4], attack detection [121, 127]) and network user privacy analysis (such as traffic identification and

classification [43, 42]).

Among the above applications, network traffic classification has been increasingly adopted to study the privacy threats and attacks during the use of network security protocols, due to the mounting emphasis for user privacy over stringent Internet censorship [54]. Traffic classification is a class of techniques to associate traffic flows with the applications or the collections of contents (such as the homepage of a website) generating them. According to the types of inputs, network traffic classification can be categorized into port-based approaches [13] which infers the applications based on transport-layer (TCP or UDP) ports; payload-based approaches [125, 98, 35] which identifies application-specific byte strings inside the packets; pattern-based approaches [167, 10, 173] which identifies the applications or contents using the communication patterns of the transport level interactions.

Considering the decreasing reliability of the port-based approaches [94] and increasing difficulty to apply payload-based approaches due to circumvention methods such as encryption and encapsulation, the pattern-based approaches have dominated over the past decade. They have achieved results comparable to payload-based approaches without the need to inspect packet contents. Among these approaches, web fingerprinting [80], which is a non-intrusive analysis targeting the user's web browsing privacy against implementations of network security protocols (e.g., HTTPS, IPSec, and Tor), has gained visibility recently.

Web Fingerprinting. Web fingerprinting aims to identify the website or the web content the user has just browsed, utilizing the meta parameters of the encrypted network traffic such as the timestamps, direction and sizes of packets. Successful web fingerprinting attacks have emerged as a prominent threat to user privacy, as they are capable of profiling the user's preferences and fine-grained online activities based on passive observations.

In the literature, most prior works on web fingerprinting target at differentiating the website a user has visited, which are referred to as the inter-domain website fingerprinting (WSF) techniques. They were initially demonstrated feasible by a series of early studies [191, 45, 80, 169], which used the total volume of the data flow to differentiate encrypted network

traffic. More recently, the inter-domain WSF techniques [79, 41, 202, 73, 78, 88, 91, 105, 109, 133, 147, 203, 209] are still based on the encrypted traffic characteristics including traffic patterns and statistics, but they are typically evaluated under more stringent criteria. More specifically, the performance is analyzed against both traditional encrypted channels and anonymous channels such as Tor network. The datasets used for evaluation are also reasonably larger, containing at least thousands of unique websites.

Chapter 3

A Formal Framework for Single Sign-on Protocols

3.1 Introduction

Single Sign-on (SSO) protocols allow users to log into a website, i.e., the relying party (RP), using the accounts registered with another website, i.e., the identity provider (IDP). They have started forming the basis of managing user identities in contemporary commercial websites. For example, statistics shows that OpenID is used by nearly 1.5 million websites (0.65% of the entire Internet), and Facebook Connect is used by more than 10 million websites (2.17% of the Top 10K sites) [156]. These protocols serve as the safeguard of various security- and privacy-sensitive web services and users' online data. Nonetheless, they have been continually found vulnerable and insecure by previous research [198, 65, 66, 16, 170].

Given the critical role that SSO protocols are playing, they deserve a rigorous security assessment, and ideally formal verification, before they are implemented and deployed for practical use. However, the challenge on formally verifying SSO protocols is at least threefold. First, formal verification of SSO protocols requires an accurate formal model of the underlying web infrastructure which SSO protocols rely upon. The web infrastructure is complicated because it involves server side (e.g., web servers and SSO SDKs), client side (e.g., web browsers) and various communication channels. In addition, SSO protocols often rely on new techniques and

features (e.g., HTML5's `postMessage`) to fulfill their advanced requirements (e.g., cross-domain client-side communication). These features often play an important role in the security properties of SSO protocols. For example, misusing `postMessage` and HTML5's client-side storage may lead to credential leakage [198, 76].

The second challenge is regarding the comprehensiveness of the attacker behaviors. Since SSO protocols rely on web clients, web servers and various communication channels, they are naturally exposed to a large attack surface. The attacks targeting any component of the web infrastructure need to be taken into consideration when assessing an SSO protocol. In addition, behaviors of malicious parties also play crucial roles to particular properties of SSO protocols, such as the malicious RP to authentication and the malicious IDP to privacy.

The third challenge is on the formalization of the target properties with respect to SSO protocols. *Authentication* has been commonly regarded as a critical security property of SSO protocols, given that the authentication needs to be guaranteed among the multiple participants (i.e., the user, the IDP and the RP), and that a significant proportion of the attacks (such as replay attack and impersonation attack) aim at compromising the authentication [16, 170, 8]. Recently, user privacy has become a serious public concern. Particularly in SSO protocols, *unlinkability* property – whether an attacker is able to track which RP a user has logged into, can be used to determine whether the SSO preserves the user privacy [66].

In this work, we propose a framework for analyzing SSO protocols. Our framework consists of a formal model of the web infrastructure, the attacker models, as well as formal definitions of the security and privacy properties. We abstract the whole infrastructure into three parts, i.e., *web browser*, *network* and *web server*, which are essential in SSO. The attacker model covers both security attackers and privacy attackers. The model of security attackers considers the network attacker which is a typical Dolev-Yao model; the model of privacy attackers includes three types of malicious IDP, i.e., *honest-but-curious IDP server* which infers user activity information based on the messages generated or received, *malicious IDP server* which is an

IDP server capable of sending fake information and *malicious IDP client* which is capable of repetitively calling browser APIs (for example, to request the browser to open a new window). We use the *applied pi calculus* [2] as our modeling language in the framework, since applied pi models can be automatically verified using the state-of-the-art verifier Proverif [29, 31]. The security property is formalized as correspondence and privacy property is formalized as observational equivalence [57] in applied pi.

We apply our framework to analyze four SSO protocols – the OAuth 1.0a [207], the OAuth 2.0 [135], the BrowserID protocol [37] and SPRESSO protocol [67]. The OAuth 1.0a and the OAuth 2.0 protocols have been implemented by major SSO schemes like Twitter Login (OAuth 1.0a), Google Sign-In (OAuth 2.0), OpenID (OAuth 2.0), Facebook Connect (OAuth 2.0), etc. The BrowserID and SPRESSO protocols have specially taken into consideration privacy property in their designs. These four protocols are representative and suitable to test our framework, because modeling them covers most of the web techniques, including server-to-server communication, HTML5's cross-domain communication, AJAX (Asynchronous JavaScript And XML) and so on. Our analysis confirms that the four protocols preserves authentication property against the security attackers. Our analysis has also revealed that OAuth 1.0a and OAuth 2.0 do not provide privacy protection. In addition, we have found that BrowserID and SPRESSO suffer from a previously-unknown flaw which allows a malicious IDP to use two key pairs to learn which users have logged into a particular RP.

3.2 Preliminaries

3.2.1 The General SSO Flow

A typical flow of SSO protocols is briefly described as follows in Fig. 3.1. In a web browser, a user visits the RP website that has an SSO login button. The user clicks the button, which triggers the browser to open a new window named *RPdoc* and load the RP login page from RP server. We call the window RPdoc the RP client, which is responsible for the communication

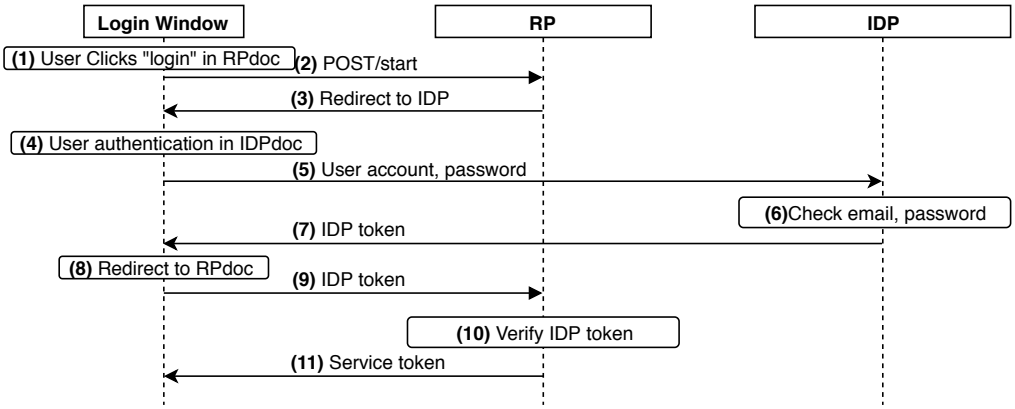


Figure 3.1: A General SSO Process

P,Q	:=	plain process 1.5cm
	0	null process
	P | Q	parallel composition
	!P	replication
	new n; P	name restriction
	in(u, x); P	message input
	out(u, M); P	message output
	if M $=_E$ N **then** P **else** Q	conditional
	let x = M **in** P **else** Q	term evaluation
A,B	:=	extended process
	P	plain process
	A | B	parallel
	new x; A	variable restriction
	new n; A	name restriction
	{M/x}	active substitution

Figure 3.2: Applied pi syntax

between the browser and the corresponding RP server. Then the RPdoc triggers the browser to open or redirect to an IDP client named *IDPdoc* and then load the IDP login page. The user is prompted to enter his account and password registered at the IDP. Next, the IDPdoc sends the user credentials to the IDP server which will issue a token back to the IDPdoc upon successful user credentials verification. Upon receiving the IDP token, the IDPdoc forwards it via the RPdoc to the RP server for verification. If the IDP token is valid, the RP server issues a service token and sends it back to the RPdoc denoting a successful user login session.

3.2.2 Modeling Language

We use a variation of the applied pi calculus [31, 2] for modeling protocols, attackers and properties. This calculus assumes an infinite set of names, which is used for modeling communication channels and atomic data, an infinite set of variables and a signature Σ consisting of finite number of symbols with arities, which are used for modeling cryptographic primitives. Terms are defined as names, variables as well as function symbols applied to terms. An equational theory E is defined to capture equations on terms. Key word **fun** is used to denote function symbols and key word **reduc** is used to denote equations on terms. The applied pi calculus assumes a type system for terms generated by a set of base types.

A system is modeled as processes, whose syntax is defined in Fig. 3.2. Null process 0 does nothing. Process $P|Q$ represents two processes P and Q running in parallel. Process $!P$ models infinite number of process P running in parallel, capturing unbounded number of sessions. Name restriction $\mathbf{new}\ n; P$ binds the name n in process P, capturing both fresh random numbers and private names and channels. Message input $\mathbf{in}(u, x); P$ describes that the process reads a message from channel u and binds received message to x in process P. Message output $\mathbf{out}(u, M); P$ describes that the process sends a message M on channel u and runs P afterwards. The conditional evaluation $\mathbf{if}\ M =_E N\ \mathbf{then}\ P\ \mathbf{else}\ Q$ runs P when equation $M =_E N$ is true under equational theory E otherwise runs Q. If M or N fails (can not be solved), the process stops. If Q is null, this process can be reduced to $\mathbf{if}\ M =_E N\ \mathbf{then}\ P$. The term evaluation $\mathbf{let}\ x = M\ \mathbf{in}\ P\ \mathbf{else}\ Q$ bounds x to M and takes process branch P, otherwise, Q is taken. If Q is null, the term evaluation can be simplified to $\mathbf{let}\ x = M\ \mathbf{in}\ P$. We abbreviate $\mathbf{new}\ n_1; \cdots; \mathbf{new}\ n_m$ as $\mathbf{new}\ \tilde{n}$. We write $\sigma = \{M/x\}$ as the substitution of variable x with term M. A process is closed if all variables are either bound by restriction or input, or defined by an active substitution. The reasoning on the models in applied pi is with respect to the Dolev-Yao attacker [58] who can block, obtain, tamper and/or insert messages over public channels.

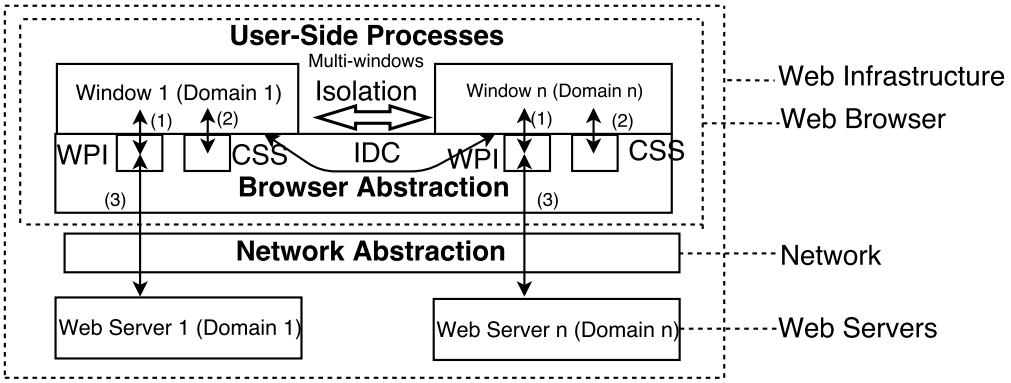


Figure 3.3: Web Infrastructure Abstraction

3.3 A Verification Framework for SSO

In this section, we present our verification framework for formally analyzing SSO protocols. First, we explain our web infrastructure model, followed by the security and privacy attacker models. Finally, we present the formalization of the authentication and unlinkability properties of SSO protocols.

3.3.1 Web Infrastructure Model

Fig. 3.3 shows our abstraction of the web infrastructure. In the abstraction, the web infrastructure consists of three components: the *web browsers*, the *network* and the *web servers*. It represents a common scenario where users use the browsers to download documents and communicate with the web servers via the network.

3.3.1.1 Web Browser Model

The web browser model has a list of *windows/iframes* (denoted by *window* $1, \ldots,$ *window* n in Fig. 3.3) that are containers for the client-side documents of the websites. In addition, the model includes the *webpage parser/interpreter* (denoted by *WPI*), the *client-side storage* (denoted by *CSS*), the *inter-domain communication* (denoted by *IDC*) and the *isolation*. They are important features for analyzing the security and privacy properties of SSO protocols.

$$
\begin{array}{lll}
WPI := & (\mathbf{in}(\mathtt{priv}, (= \mathtt{OW}, w_1)); \mathbf{let}\ w_2 = \mathsf{child}(w_1)\ \mathbf{in} & k_1 \\
& \ \mathbf{out}(\mathtt{priv}, (\mathtt{OW}, w_1, w_2)); & k_2 \\
& \ !\mathbf{in}(\mathtt{priv}, (= \mathtt{parentOf}, = w_2)); & k_3 \\
& \mathbf{out}(\mathtt{priv}, (\mathtt{parentOf}, w_2, w_1)))| & k_4 \\
& (\mathbf{in}(\mathtt{priv}, (= \mathtt{http(s)Connect}, e_1, e_2)); & k_5 \\
& \ \mathbf{out}(\mathtt{priv}', (\mathtt{http(s)Connect}, e_1, e_2)))| & k_6 \\
& (\mathbf{in}(\mathtt{priv}, (= \mathtt{http(s)Send}, msg_1, e_3, e_4)); & k_7 \\
& \ \mathbf{out}(\mathtt{priv}', (\mathtt{http(s)Send}, msg_1, e_3, e_4)))| & k_8 \\
& (\mathbf{in}(\mathtt{priv}', (= \mathtt{http(s)Receive}, msg_2, e_5, e_6)); & k_9 \\
& \ \mathbf{out}(\mathtt{priv}, (\mathtt{http(s)Receive}, msg_2, e_5, e_6))). & k_{10}
\end{array}
$$

WPI. The WPI parses and interprets the programs downloaded from the web servers. The WPI includes complex functions which may not be relevant to the SSO protocols, such as page rendering. Therefore, our framework only models the part that processes the SSO-relevant commands, as shown in the following model. These commands include open windows `OW` (line k_1-k_2), get the parent window `parentOf` (line k_3-k_4), establish HTTP(S) connections `http(s)Connect` (line k_5-k_6), send HTTP(S) messages `http(s)Send` (line k_7-k_8) and receive HTTP(S) messages `http(s)Receive` (line k_9-k_{10}). Note that terms w_1 and w_2 denote window names, terms $e_1, \cdots, e_m$ denote participants interacting with each other, such as windows and web servers, and terms $msg_1, \cdots, msg_l$ denote messages. Name `priv` denotes the private channel to call the browser commands, and name $\mathtt{priv}'$ denotes the private channel to send and receive the HTTP(S) messages through the *network* which is defined later.

CSS. The CSS includes both short-term storage (i.e., cookies and SessionStorage) and long-term storage (i.e., LocalStorage) that can only be accessed by the client-side web page of the same URL domain. In particular, we explicitly model LocalStorage because it may store data relevant to the privacy property. For example, compromising the LocalStorage may disclose the user's login status at a certain RP [66]. As shown in the following model, the LocalStorage is modeled as a process LS where `LSS` denotes the command of storing messages and `LSR` denotes that of retrieving messages.

We do not explicitly model the short-term storage since it can be recorded in the local variables.

$$LS := \quad \mathbf{in}(\mathtt{priv}, (= \mathtt{LSS}, (index, msg))); !\mathbf{out}(\mathtt{priv}, (\mathtt{LSR}, (index, msg))).$$

IDC. The IDC is mostly achieved by an API called `postMessage` in HTML5. It is extensively used in the SSO protocols since the involved participants (at least RP and IDP) which have to communicate with each other are typically from different domains. As shown in the following model, the `postMessage` is modeled as a process PM where `PMS` and `PMR` denote sending and receiving messages respectively. The sender and the receiver window identities (i.e. w_1 and w_2) are required to indicate the two endpoints of the `postMessage`.

$$PM := \quad \mathbf{in}(\mathtt{priv}, (= \mathtt{PMS}, w_1, w_2, msg)); \mathbf{out}(\mathtt{priv}, (\mathtt{PMR}, w_1, w_2, msg)).$$

$$\begin{array}{lcll}
HTTPconnect & := & \mathbf{in}(\mathtt{priv}', (= \mathtt{httpConnect}, e_1, e_2)); & l_1 \\
 & & \mathbf{out}(\mathtt{c}, (e_1, e_2)). & l_2 \\
HTTPsend & := & \mathbf{in}(\mathtt{priv}', (= \mathtt{httpSend}, Msg_1, e_3, e_4)); & l_3 \\
 & & \mathbf{out}(\mathtt{c}, Msg_1, e_3, e_4). & l_4 \\
HTTPreceive & := & \mathbf{in}(\mathtt{c}, (Msg_2, e_5, e_6)); & l_5 \\
 & & \mathbf{out}(\mathtt{priv}', (\mathtt{httpReceive}, Msg_2, e_5, e_6)). & l_6 \\
HTTPSconnect & := & \mathbf{in}(\mathtt{priv}', (= \mathtt{httpsConnect}, e_7, e_8)); & l_7 \\
 & & \mathbf{let}\ k = \mathsf{httpskey}(e_7, e_8)\ \mathbf{in}\ \mathbf{out}(\mathtt{c}, (e_7, e_8)). & l_8 \\
HTTPSsend & := & \mathbf{in}(\mathtt{priv}', (= \mathtt{httpsSend}, Msg_3, e_9, e_{10})); & l_9 \\
 & & \mathbf{new}\ \mathtt{nonce};\ \mathbf{let}\ key = \mathsf{httpskey}(e_9, e_{10})\ \mathbf{in} & l_{10} \\
 & & \mathbf{out}(\mathtt{c}, (\mathsf{enc}((\mathtt{nonce}, Msg_3), key), e_9, e_{10})). & l_{11} \\
HTTPSreceive & := & \mathbf{in}(\mathtt{c}, (EncMsg, e_{11}, e_{12})); & l_{12} \\
 & & \mathbf{let}\ key = \mathsf{httpskey}(e_{11}, e_{12})\ \mathbf{in} & l_{13} \\
 & & \mathbf{let}\ (Nonce, Msg_4) = \mathsf{dec}(EncMsg, key)\ \mathbf{in} & l_{14} \\
 & & \mathbf{out}(\mathtt{priv}', (\mathtt{httpsReceive}, Msg_4, e_{11}, e_{12})). & l_{15}
\end{array}$$

Isolation. The isolation among domains is a security feature (i.e., the *same origin policy*) provided by the web browsers. This feature ensures the

domains at the client side are isolated such that scripts from one domain cannot access data belonging to other domains. Since we model the windows as individual and parallel processes, the documents received by a window cannot be accessed by others. In addition, cross-domain messaging between different windows is via private channels, restraining messages only to the intended processes according to the protocol. Thus, isolation property is implicitly retained in our web browser model.

3.3.1.2 Network Model

The network model covers both HTTP and HTTPS channels which are the basis for data transmission in the SSO protocols. The network model in our framework is shown below. The terms $e_1, \cdots, e_m$ denote communicating participants, while the terms $Msg_1, \cdots, Msg_l$ denote exchanged messages.

The *HTTP channels* are not encrypted. Hence, we simply model the HTTP messages to be sent and received on the public channel `c` (line l_1-l_6). The *HTTPS channels* include two parts: *session key establishment* which sets up a session key between the two communicating participants using handshake protocols (line l_7-l_8), and *message exchange* which uses the established session key to protect the messages. In particular, the message is encrypted when it is sent out (line l_9-l_{11}) and decrypted when it is received (line l_{12}-l_{15}).

3.3.1.3 Web Servers

Web servers are the server-side SSO participants such as the RPs and IDPs. Their behaviors need to be manually modeled according to the protocol specifications.

In summary, we provide the interfaces listed in Table 3.1 to facilitate modeling of the SSO protocols using our web infrastructure. Each interface includes an **out** message representing the command from a client-side process to the web infrastructure and an **in** message representing the response from the web infrastructure to the client-side process. Overall, the web infrastructure is defined as a process where all the above processes run

Table 3.1: Interfaces: infrastructure inputs

Interfaces	**Functionality**
out(priv,(OW, w_1))	open window request from w_1
in(priv,(=OW, w_1, w_2))	return child window w_2 of w_1
out(priv,(parentOf, w_1))	request parent window of w_1
in(priv,(=parentOf, $=w_1$, w_2))	return parent window w_2 of w_1
out(priv,(PMS, w_1, w_2, m)	send postMessage from w_1 to w_2
in(priv,(=PMR, $=w_1$, $=w_2$, m)	receive postMessage from w_1 to w_2
out(priv,(LSS,(index, m)))	store m to LocalStorage
in(priv,(=LSR,(=index, m)))	retrieve m from LocalStorage
out(priv,(http(s)Connect, e_1, e_2))	http(s) connection over e_1 and e_2
in(c,(e_1, e_2)	request http(s) connection over e_1 and e_2
out(priv,(http(s)Send, m, e_1, e_2)	send http(s) message m from e_1 to e_2
in(priv,(=http(s)Receive, m, e_1, e_2))	receive http(s) message m from e_1 by e_2

in parallel, as shown below.

$$WebInfra = WPI|LS|PM|HTTPconnect|HTTPsend|HTTPreceive \\ |HTTPSconnect|HTTPSsend|HTTPSreceive$$

3.3.2 Attacker Models

3.3.2.1 Security Attackers

For the analysis of SSO security property (i.e., authentication property), we consider both the network attacker and the malicious participants.

The network attacker. This attacker controls the public communication channels between two endpoints. It is modeled by the active Dolev-Yao model [58] that is capable of recording and intercepting all the messages sent on public channels between any two endpoints and sending messages that can be derived from these recorded messages. For example, the attacker can decrypt or encrypt messages if it possesses the cryptographic keys. However, the attacker cannot obtain the message when the cryptographic key is unknown to it. Therefore the attacker can control the HTTP messages but cannot read the HTTPS messages which is encrypted. In addition, the attacker cannot access or guess the data sent via private channels.

The malicious participants.These attackers, including malicious RPs, malicious RP/IDP clients and compromised web browsers, are modeled based on the attacker models described in prior works [16, 20, 193]. The malicious RP aims to impersonate the victim user to login to another benign RP/IDP. The malicious RP can result in attacks including spoofing. It is modeled by sending the user- and session- specific data (e.g. user account and session token) to the public channel. The malicious RP/IDP clients and compromised browsers aim to steal the user credentials and/or gain unauthorized access to user profile and sensitive information. The malicious RP/IDP client can result in attacks including the CSRF (cross-domain request forgery) attacks. They are modeled by sending the browser generated or received messages to the public channel.

3.3.2.2 Privacy Attackers

For the analysis of SSO privacy properties, we consider malicious IDP as privacy attacker (privacy properties would be trivially violated if either user or RP is malicious). The goal of malicious IDP is to find out which RP a particular user is trying to login based on the information under his control. According to the attacker capabilities, we define three distinct attack models namely *honest-but-curious IDP server*, *malicious IDP server* and *malicious IDP client*. Below, we first detail the capability of each attacker model, followed by the definition of the transformation that converts a benign IDP process to the corresponding attacker process.

Honest-but-curious IDP Server. This attacker tries to break user privacy based on his own knowledge. It records messages generated and received by the IDP and derives user login information from those recorded messages. We use the transformation defined by [57] to model honest-but-curious IDP, shown in Definition 1. The main idea of the transformation is to let a benign IDP process send all the generated and received names of base type (i.e. not channel type) to the public channel. In this way, the honest-but-curious IDP attacker can be simulated by the built-in attacker model in applied pi calculus since they have the same knowledge and capability.

Definition 1 (Honest-But-Curious IDP Server Transformation)**.** Let P be a plain process representing an honest IDP process and $\mathtt{chc}$ be a public channel name that does not appear in P. We define the honest-but-curious IDP process $P^{\mathtt{chc}}$ which sends all the generated and received messages to the public channel $\mathtt{chc}$ as below:

- $0^{\mathtt{chc}} := 0$,
- $(P|Q)^{\mathtt{chc}} := P^{\mathtt{chc}}|Q^{\mathtt{chc}}$,
- $(!P)^{\mathtt{chc}} :=!P^{\mathtt{chc}}$,
- $(\mathbf{new}\ \mathtt{n}; P^{\mathtt{chc}} := \mathbf{new}\ \mathtt{n}; \mathbf{out}(\mathtt{chc}, \mathtt{n}); P^{\mathtt{chc}}$ *where* $\mathtt{n}$ *is a base type name*,
- $(\mathbf{new}\ \mathtt{n}; P)^{\mathtt{chc}} := \mathbf{new}\ \mathtt{n}; P^{\mathtt{chc}}$ *otherwise*,
- $(\mathbf{in}(\mathtt{u}, x); P)^{\mathtt{chc}} := \mathbf{in}(\mathtt{u}, x); \mathbf{out}(\mathtt{chc}, x); P^{\mathtt{chc}}$ *x is a variable of base type*,
- $(\mathbf{in}(\mathtt{u}, x); P)^{\mathtt{chc}} := \mathbf{in}(\mathtt{u}, x); P^{\mathtt{chc}}$ *otherwise*,
- $(\mathbf{out}(\mathtt{u}, M); P)^{\mathtt{chc}} := \mathbf{out}(\mathtt{u}, M); P^{\mathtt{chc}}$,
- $(\mathbf{if}\ M =_E N\ \mathbf{then}\ P\ \mathbf{else}\ Q)^{\mathtt{chc}} := \mathbf{if}\ M =_E N\ \mathbf{then}\ P^{\mathtt{chc}}\ \mathbf{else}\ Q^{\mathtt{chc}}$,
- $(\mathbf{let}\ x = M\ \mathbf{in}\ P\ \mathbf{else}\ Q)^{\mathtt{chc}} := \mathbf{let}\ x = M\ \mathbf{in}\ P^{\mathtt{chc}}\ \mathbf{else}\ Q^{\mathtt{chc}}$.

Malicious IDP Server. This attacker tries to break user privacy by creating a fake message and sending it out in place of the authentic message at the server side of IDP. Given an honest IDP process, we provide the following transformation which generates the malicious IDP server process.

Definition 2 (Malicious IDP Server Transformation)**.** Let P be a plain process modeling an honest IDP, $\mathtt{chc}$ be a public channel name that does not appear in P, ℓ be a subset of terms in P. Particularly, we specify ℓ={pk(M), sign(M,N), $\mathsf{AuthTokens}$(M,N), $\mathsf{AuthCodes}$(M,N)} where function pk(M) represents the public key corresponding to the private key M, sign(M,N) generates a cryptographic signature on M with private key N, $\mathsf{AuthTokens}$(M,N) and $\mathsf{AuthCodes}$(M,N) generate fresh authentication token and authorization token respectively between entities M and N. ℓ We define malicious IDP server process $P^{\ell,\mathtt{chc}}$ as below:

- $0^{\ell,\mathtt{chc}} := 0$,
- $(P|Q)^{\ell,\mathtt{chc}} := P^{\ell,\mathtt{chc}}|Q^{\ell,\mathtt{chc}}$,
- $(!P)^{\ell,\mathtt{chc}} :=!P^{\ell,\mathtt{chc}}$,
- $(\mathbf{new}\ \mathtt{n}; P)^{\ell,\mathtt{chc}} := \mathbf{new}\ \mathtt{n}; \mathbf{out}(\mathtt{chc}, \mathtt{n}); P^{\ell,\mathtt{chc}}$ *where* $\mathtt{n}$ *is a base type name*,

– $(\mathbf{in}(\mathtt{u}, x); P)^{\ell,\mathtt{chc}} := \mathbf{in}(\mathtt{u}, x); \mathbf{out}(\mathtt{chc}, x); P^{\ell,\mathtt{chc}}$ *x is a base type variable*,
– $(\mathbf{out}(\mathtt{u}, M); P)^{\ell,\mathtt{chc}} := \mathbf{in}(\mathtt{chc}, M'); \mathbf{out}(\mathtt{u}, M'); P^{\ell,\mathtt{chc}}$ *when* $M \in \ell$,
– $(\mathbf{out}(\mathtt{u}, M); P)^{\ell,\mathtt{chc}} := \mathbf{out}(\mathtt{u}, M); P^{\ell,\mathtt{chc}}$*otherwise*,
– $(\mathbf{if}\ M =_E N\ \mathbf{then}\ P\ \mathbf{else}\ Q)^{\ell,\mathtt{chc}} := \mathbf{if}\ M =_E N\ \mathbf{then}\ P^{\ell,\mathtt{chc}}\ \mathbf{else}\ Q^{\ell,\mathtt{chc}}$,
– $(\mathbf{let}\ x = M\ \mathbf{in}\ P\ \mathbf{else}\ Q)^{\ell,\mathtt{chc}} := \mathbf{let}\ x = M\ \mathbf{in}\ P^{\ell,\mathtt{chc}}\ \mathbf{else}\ Q^{\ell,\mathtt{chc}}$.

In the definition, whenever a message in ℓ is generated, a corresponding fake one is generated; whenever a message in ℓ is sent out on a channel, the corresponding fake one is sent out instead. Note that for now we restrict set ℓ to include four types of IDP generated messages as shown since they are strongly relevant to user privacy.

We follow the same methodology to formalize a malicious IDP client. First, we define the capability of the malicious IDP client in Definition 3. Then we define the transformation to convert a benign IDP client process into the modeled malicious IDP client in Definition 4.

Malicious IDP Client. This attacker mainly follow the client-side IDP process but contains a malicious iframe. The malicious iframe has the capability of repetitively calling web infrastructure interfaces. For example, Browser ID suffers from this attack, because when a user logs in, the malicious window can be triggered to inform the IDP that the user is logged in to a particular RP [65]. Before we can define the transformation from an honest IDP client to malicious IDP client, we first define the capabilities of the malicious IDP client.

Definition 3 (Malicious IDP Client Capabilities)**.** Let MC be a plain process modeling the capabilities of malicious IDP client

$MC :=$

$(!\mathbf{out}(\mathtt{priv}, (\mathtt{OW}, w_1)); \mathit{where}\ w_1 \in \wp$

$\mathbf{in}(\mathtt{priv}, (= \mathtt{OW}, w_1, w_2))\ \mathit{where}\ \wp = \wp \cup \{w_2\})|$

$(!\mathbf{out}(\mathtt{priv}, (\mathtt{ParentOf}, w_3)); \mathit{where}\ w_3 \in \wp$

$\mathbf{in}(\mathtt{priv}, (= \mathtt{ParentOf}, w_3, w_4))\ \wp = \wp \cup \{w_4\})|$

$!\mathbf{out}(\mathtt{priv}, (\mathtt{PMS}, m_1, w_5, w_6))\ \mathit{where}\ m_1 \in \chi, w_5, w_6 \in \wp|$

$!\mathbf{in}(\mathtt{priv}, (= \mathtt{PMR}, m_2, w_7, w_8))\ \mathit{where}\ \chi = \chi \cup \{m_2\}, w_7, w_8 \in \wp|$

$!\mathbf{out}(\mathtt{priv}, (\mathtt{LSS}, m_3))$ *where* $m_3 \in \chi|$

$!\mathbf{in}(\mathtt{priv}, (= \mathtt{LSR}, m_4))$ *where* $\chi = \chi \cup \{m_4\}|$

$!\mathbf{out}(\mathtt{priv}, (\mathtt{http(s)Connect}, w_9, s_1))$ *where* $w_9 \in \wp, s_1 \in \zeta|$

$!\mathbf{out}(\mathtt{priv}, (\mathtt{http(s)Send}, m_3, w_{10}, s_2)$ *where* $m_3 \in \chi, w_{10} \in \wp, s_1 \in \zeta|$

$!\mathbf{in}(\mathtt{priv}, (= \mathtt{http(s)Receive}, m_4, w_{11}, s_3)$ *where* $\chi = \chi \cup \{m_4\}, w_{11} \in \wp, S_3 \in \zeta$

where $w_i \in \wp$ be a subset of names representing the identities of windows in a browser; $s_j \in \zeta$ be a subset of names representing the identities of web servers that browser windows communicate with; $m_k \in \chi$ be a subset of terms representing the messages known by the malicious IDP client. Initially, we assume $\wp$={root} where name root represents the state before any windows are opened; ζ={$rpdomain_1, ..., rpdomain_m, idpdomain$} where $rpdomain_i$ and idpdomain represent the corresponding domain names of RPs and the IDP servers in an SSO system; χ=$\emptyset$.

Given an honest IDP client-side process and the above malicious IDP client capabilities, we provide the following transformation which generates the malicious IDP client process.

Definition 4 (Malicious IDP Client Transformation)**.** Let P be a plain process modeling an honest IDP client, MC be a plain process representing the capabilities of malicious IDP client. We define malicious IDP client process P^{MC} as below:

- $0^{MC} := MC$,
- $(P|Q)^{MC} := P^{MC}|Q^{MC}$,
- $(!P)^{MC} :=!P^{MC}$,
- $(\mathbf{new}\ \mathtt{n}; P)^{MC} := \mathbf{new}\ \mathtt{n}; MC|P^{MC}\ \chi = \chi \cup \{\mathtt{n}\}$,
- $(\mathbf{in}(\mathtt{u}, x); P)^{MC} := \mathbf{in}(\mathtt{u}, x); MC|P^{MC}\ \chi = \chi \cup \{x\}$,
- $(\mathbf{out}(\mathtt{u}, M); P)^{MC} := \mathbf{out}(\mathtt{u}, M); MC|P^{MC}\ \chi = \chi \cup \{M\}$,
- $(\mathbf{if}\ M =_E N\ \mathbf{then}\ P\ \mathbf{else}\ Q)^{MC} := \mathbf{if}\ M =_E N\ \mathbf{then}\ P^{MC}\ \mathbf{else}\ Q^{MC}$,
- $(\mathbf{let}\ x = M\ \mathbf{in}\ P\ \mathbf{else}\ Q)^{MC} := \mathbf{let}\ x = M\ \mathbf{in}\ P^{MC}\ \mathbf{else}\ Q^{MC}$.

where χ= $\emptyset$.

3.3.3 Formalization of Authentication and Unlinkability Properties w.r.t. SSO

Authentication Property. The authentication property of security protocols is commonly captured by the correspondence property [149]. Correspondence $B \rightarrow A$ describes that event B can only be executed if event A has been executed before. Therefore, authentication property in SSO protocol is formally defined using correspondence in Definition 5.

Definition 5 (SSO Authentication Property)**.** An SSO protocol satisfies authentication property if the following correspondence properties hold:

$ev : user\_e(account, rp, idp) \rightarrow ev : rp\_b(account, rp, idp)$.

$ev : rp\_e(account, rp, idp) \rightarrow ev : user\_b(account, rp, idp)$.

- Event $user\_b(account, rp, idp)$ represents the beginning the user login session and event $user\_e(account, rp, idp)$ represents the end of the user login session at the web browser side.

- Event $rp\_b(account, rp, idp)$ represents the beginning of the RP process and event $rp\_e(account, rp, idp)$ represents the end the RP process.

Intuitively, the first correspondence query ensures that if the user successfully login to the RP, then the targeted RP must have initiated login process for the user. The second correspondence query ensures that if the RP logs the user in, then the user must have initiated this login session. Note that this definition captures mutual authentication in SSO protocols: *a user verifies an RP* and *an RP verifies a user*, respectively.

Unlinkability Property. Unlinkability captures the privacy property that satisfies when the attacker is unable to identify the links between two or more objects in a system [138]. In the applied pi calculus, the unlinkability property can be formalized as the observational equivalence relation [2]. Intuitively, two processes are observationally equivalent if the Dolve-Yao attacker cannot distinguish the two processes. In the following part, we first define the generalized evaluation context of SSO processes in

Definition 6 in order to further define the unlinkability property in SSO protocols in Definition 7.

Definition 6 (Evaluation Context of General SSO Processes)**.** We define an evaluation context D as the following SSO process with a hole ([_]).

$D :=$ **new** $\tilde{\mathtt{n}}$;
$C(account, rp)\sigma_{11}|C(account, rp)\sigma_{12}|\cdots|[\_]|\cdots|$
$C(account, rp)\sigma_{nm}|!RP_1|\cdots|!RP_m|!IDP|!WebInfra,$

- $\tilde{\mathtt{n}}$ indicates private channel names and data in this process.
- Process $C(account, rp)$ models the client-side login process including the behaviors of the client-side RP, the client-side IDP, etc., together with the user's behaviors (e.g., input the password). The *account* and the *rp* are two free variables denoting the user account and the RP domain which are instantiated by $\mathtt{Account}_i$ and $\mathtt{RPname}_j$ respectively using the substitution σ_{ij} where $\sigma_{ij} = \{\mathtt{Account}_i/account, \mathtt{RPname}_j/rp\}$. We assume there are totally n accounts and m RP domains, therefore σ is a set $\sigma = \{\sigma_{11}, \cdots, \sigma_{nm}\}$. The $RP_1, \cdots, RP_m$ and the IDP are honest RPs and IDP. Any sub-process can be null except $WebInfra$.
- The hole [_] can be filled with a process $C(account, rp)\sigma_{ij} \mid C(account, rp)$ σ_{lk} with σ_{ij}, $\sigma_{lk} \in \sigma$.

With the evaluation context, we can formally define the privacy property as follows.

Definition 7 (Privacy w.r.t. SSO Protocols)**.** An SSO protocol preserves user's privacy, or unlinkability, if the following observational equivalence query is true

$D[C\{\mathtt{A}_1/account, \mathtt{RP}_1/rp\}|C\{\mathtt{A}_2/account, \mathtt{RP}_2/rp\}] \approx$
$D[C\{\mathtt{A}_1/account, \mathtt{RP}_2/rp\}|C\{\mathtt{A}_2/account, \mathtt{RP}_1/rp\}]$

for accounts $\mathtt{A}_1$ and $\mathtt{A}_2$ and RPs $\mathtt{RP}_1$ and $\mathtt{RP}_2$.

In this definition, $\mathtt{A}_1$ and $\mathtt{A}_2$ represent two user accounts, and $\mathtt{RP}_1$ and $\mathtt{RP}_2$ represent two RP domains. Intuitively, the definition indicates that an

Table 3.2: SSO Modeling Information Summary

Protocol	Total Line No. with/without Framework	% of Size Reduction	% of Lines Use Framework Interfaces
OAuth 1.0a	75/160	53.1%	54.3%
OAuth 2.0	88/208	57.7%	54.2%
BrowserID	97/170	42.9%	50.0%
SPRESSO	102/198	48.5%	53.4%

SSO protocol respects user's privacy when $\mathtt{A}_1$ logs in to $\mathtt{RP}_1$ and $\mathtt{A}_2$ logs in to $\mathtt{RP}_2$ cannot be differentiated from (i.e., observationally equivalent to) $\mathtt{A}_1$ logs in to $\mathtt{RP}_2$ and $\mathtt{A}_2$ logs in to $\mathtt{RP}_1$. Note that two account and two RPs are required in order to define privacy, given that if there is only one account or RP, the malicious IDP can trivially know who is logging in to the RP based on the RP-IDP or user-IDP communication.

3.4 Evaluation

This section presents the statistics regarding the framework and the vulnerabilities found in the four SSO protocols through our analysis. The detailed protocol modeling with the help of our framework will be presented in Section 3.5. In this section, we first show that our framework can efficiently facilitate and ease the protocol modeling process. Then, we also show that our framework is comprehensive enough in identifying both existing and previously-unknown vulnerabilities.

3.4.1 Facilitating in SSO Modeling

The modeling of SSO protocols is greatly simplified by using the interfaces provided in our framework to handle standard browser APIs and network functions. We determine the size of each model (in terms of the total number of lines of code) before and after using our framework respectively, and further calculate the percentage of reduction in the model sizes. We also summarize the percentage of lines in a model that calls the interfaces provided in our framework. The information is summarized in Table 3.2, where it is shown that a minimum of 42.9% of reduction in model size can be

Table 3.3: Evaluation Results of SSO Protocols

Protocol	Property			
	Security	Privacy		
		Honest-But-Curious IDP Server	Malicious IDP Server	Malicious IDP Client
OAuth 1.0a	✓	×	×	−
OAuth 2.0	✓	×	×	−
BrowserID	✓	✓	×	×
SPRESSO	✓	✓	×	✓

achieved and that around 50% of each model constructed in the case studies can be handled using our framework. Therefore, the security analysts can focus better on the SSO protocol flows and the protocol-specific functions, such as the functions for generating and verifying digital signatures.

3.4.2 Comprehensiveness in Attack Identification

After analyzing the four well-known SSO protocols, we have found that all the four protocols preserve the authentication property against the security attackers and no protocol can fully protect user privacy against all the privacy attackers. We briefly present the analysis results as follows:

OAuth1.0a and OAuth2.0. The privacy attackers (i.e., the Honest-But-Curious IDP Server and the malicious IDP server) are able to distinguish which user account is logging in to which RP since the messages containing the RP and the user account information can be sent to the IDP server at the same time (for example, at step (9) in Fig. 3.12, the user account and the request token corresponding to the RP server are sent to the IDP server at the same time), thus violating the unlinkability property. Note that we did not verify OAuth1.0a and OAuth2.0 against the malicious IDP client attacker model (marked by '-' in Table 3.3) since the two protocols are designed without privacy in mind and the unlinkability property has been violated using the other two attacker models.

BrowserID and SPRESSO. We confirmed the privacy attack on BrowserID presented in [65] with respect to the malicious IDP client attacker model. We identified a novel attack that violates the unlinkability property against

the malicious IDP server attacker model. Both BrowserID and SPRESSO does not preserve unlinkability property against this attacker model. We present the novel privacy attack in Section 3.5.4.

3.5 Case Studies

In this section, we first demonstrate the modeling of four well-known SSO protocols using our framework: the SPRESSO protocol [67], the BrowserID protocol [37], the OAuth 1.0a protocol [207] and the OAuth 2.0 protocol [135]. Then we discuss how to run verification of the SSO protocol models against the attacker models, followed by the discussion of the novel privacy attack found using our framework. The complete SSO protocol models in our case studies together with our framework are available at [194] for reference.

3.5.1 SPRESSO

The general process of the SPRESSO protocol is shown in Fig. 3.4. Following this process, the SPRESSO protocol is modeled, as shown in Fig. 3.5, Fig. 3.6 and Fig. 3.7.

Fig. 3.5 shows the overall model of the SPRESSO protocol. The client-side process is modeled as $C(\texttt{Account}, \texttt{RPname})$ where `Account` and `RPname` are instantiations for the free variables *account* and *rp* in the process $C(account, rp)$. The process $C(\texttt{Account}, \texttt{RPname})$ is comprised of sub-processes namely the $RPdoc\_proc$, the $IDPdoc\_proc$ and the $FWDdoc\_proc$, which represent the behaviors of the client-side web pages (i.e., the RPdoc, the IDPdoc and the FWDdoc) of the corresponding web servers. The IDP and the RP servers are modeled in the $IDP\_proc(\texttt{IDPname})$ and the $RP\_proc(\texttt{RPname})$. In the rest of this section, we detail the modeling of the client-side process and the server-side processes.

3.5.1.1 CLIENT-SIDE PROCESS

The model of the client-side process of the SPRESSO protocol is shown in Fig. 3.6. We assume the user has an account (*account*) from the IDP

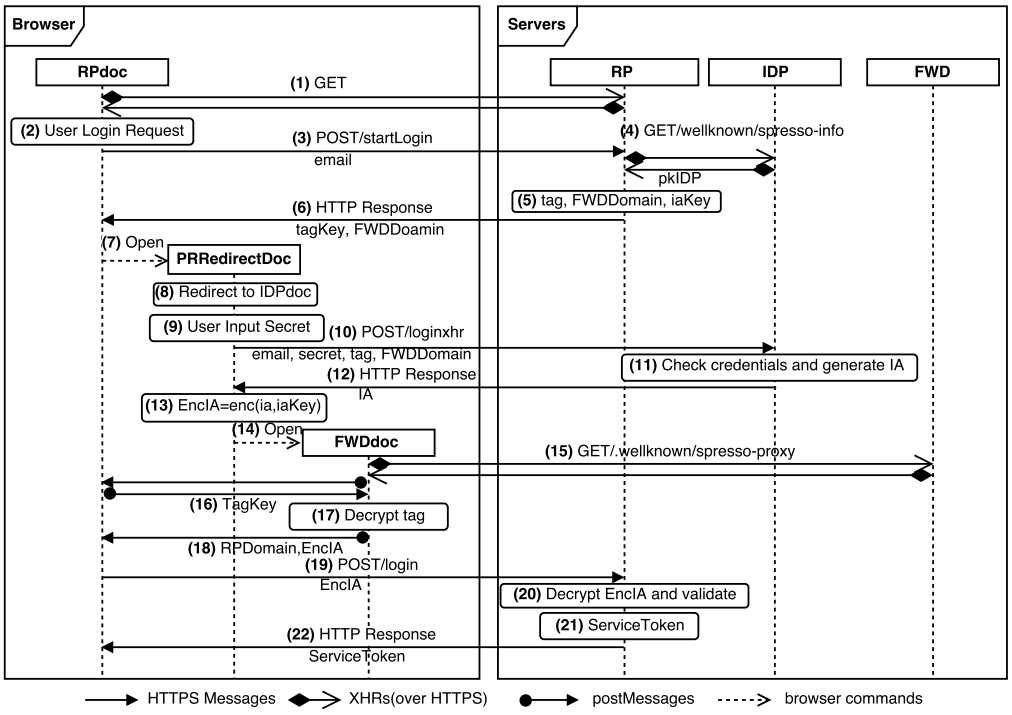


Figure 3.4: SPRESSO Protocol Flow Chart [67]

$$
\begin{array}{ll}
SPRESSO\_proc & := \\
s_1 & \textbf{new}\ \texttt{skidp}; \textbf{let}\ pkidp = \texttt{pk}(\texttt{skidp})\ \textbf{in} \\
s_2 & \textbf{new}\ \texttt{IDPname}; \textbf{new}\ \texttt{RPname}; \textbf{new}\ \texttt{Account}; \\
s_3 & (!\textbf{out}(\texttt{c}, \texttt{IDPname}) | !\textbf{out}(\texttt{c}, \texttt{RPname}) | C(\texttt{Account}, \texttt{RPname}) | \\
s_4 & !IDP\_proc(\texttt{IDPname}) | !RP\_proc(\texttt{RPname}) | !WebInfra)
\end{array}
$$

Figure 3.5: SPRESSO Protocol

($IDPname$) (line q_1-q_2 in Fig. 3.6)[1]. The three windows opened during the client-side login process are modeled by three individual sub-processes run in parallel: the $RPdoc\_proc$, the $IDPdoc\_proc$ and the $FWDdoc\_proc$ (q_6). Following the actual SSO login process, we model the SPRESSO protocol utilizing the standard browser APIs defined in our framework and customized functions defined to capture the specific function performed by the protocol participants.

[1]For simple reference to the same information in different figures, we use the following format ((k), line x_j) to represent the step k in Fig. 3.4, line x_j in Fig. 3.6 (when x_j is a q_j) or Fig. 3.7 (when x_j is a p_j).

C(account, rp) :=

q_1 $\mathbf{in}(\mathtt{c}, IDPname_1);$
q_2 $\mathbf{let}\ email = (account, IDPname_1)\ \mathbf{in}$
q_3 $\mathbf{let}\ RPname_1 = rp\ \mathbf{in}$
q_4 $\mathbf{new}\ \mathtt{root}; \mathbf{out}(\mathtt{priv}, (\mathtt{OW}, \mathtt{root}));$
q_5 $\mathbf{in}(\mathtt{priv}, (= \mathtt{root}, rpdoc));$
q_6 $RPdoc\_proc|IDPdoc\_proc|FWDdoc\_proc$

RPdoc_proc :=

q_7 $\mathbf{out}(\mathtt{priv}, (\mathtt{httpsConnect}, rpdoc, RPname_1));$
q_8 $\mathbf{out}(\mathtt{priv}, (\mathtt{httpsSend}, email, rpdoc, RPname_1));$
q_9 $\mathbf{in}(\mathtt{priv}, (= \mathtt{httpsReceive}, (tagkey, fwdomain,$
$logsesstoken), = rpdoc, = RPname_1));$
q_{10} $\mathbf{out}(\mathtt{priv}, (\mathtt{OW}, rpdoc)); \mathbf{in}(\mathtt{priv}, (= rpdoc, rddoc));$
q_{11} $\mathbf{out}(\mathtt{priv}_{\mathtt{rd}}, (logsesstoken, rddoc));$
q_{12} $\mathbf{in}(\mathtt{priv}, (= \mathtt{PMR}, fwdoc, = rpdoc, = \mathtt{ready}));$
q_{13} $\mathbf{out}(\mathtt{priv}, (\mathtt{PMS}, rpdoc, fwdoc, tagkey);$
q_{14} $\mathbf{in}(\mathtt{priv}, (= \mathtt{PMR}, = fwdoc, = rpdoc, (EncIA,$
$= \mathsf{getrpdomain}(RPname_1))));$
q_{15} $\mathbf{out}(\mathtt{priv}, (\mathtt{httpsSend}, (EncIA, logsesstoken), rpdoc, RPname_1));$
q_{16} $\mathbf{in}(\mathtt{priv}, (= \mathtt{httpsReceive}, = \mathtt{success}, rpdoc, RPname_1)).$

IDPdoc_proc :=

q_{17} $\mathbf{in}(\mathtt{priv}_{\mathtt{rd}}, (logsesstoken_1, rddoc_1));$
q_{18} $\mathbf{out}(\mathtt{priv}, (\mathtt{httpsConnect}, rddoc_1, RPname_1));$
q_{19} $\mathbf{out}(\mathtt{priv}, (\mathtt{httpsSend}, logsesstoken_1, rddoc_1, RPname_1));$
q_{20} $\mathbf{in}(\mathtt{priv}, (= \mathtt{httpsReceive}, (= rddoc_1, tag, fwdomain1,$
$= email, iakey), = rddoc_1, = RPname_1));$
q_{21} $\mathbf{new}\ \mathtt{idpdoc};\ \mathbf{out}(\mathtt{priv}, (\mathtt{httpsConnect}, \mathtt{idpdoc}, IDPname_1));$
q_{22} $\mathbf{let}\ password = \mathsf{getpss}(email)\ \mathbf{in}$
q_{23} $\mathbf{out}(\mathtt{priv}, (\mathtt{httpsSend}, (email, password, fwdomain_1, tag),$
$\mathtt{idpdoc}, IDPname_1));$
q_{24} $\mathbf{in}(\mathtt{priv}, (= \mathtt{httpsReceive}, ia, = \mathtt{idpdoc}, = IDPname_1));$
q_{25} $\mathbf{let}\ EncIA = \mathsf{enc}(ia, iakey)\ \mathbf{in}$
q_{26} $\mathbf{out}(\mathtt{priv}, (\mathtt{OW}, rddoc_1));$
q_{27} $\mathbf{in}(\mathtt{priv}, (= rddoc_1, fwdoc_1));$
q_{28} $(\mathbf{out}(\mathtt{priv}_{\mathtt{fw}}, (EncIA, tag, fwdoc_1))).$

FWDdoc_proc :=

q_{29} $\mathbf{in}(\mathtt{priv}_{\mathtt{fw}}, (EncIA_1, tag_1, fwdoc_2));$
q_{30} $\mathbf{out}(\mathtt{priv}, (\mathtt{parentOf}, fwdoc_2));$
q_{31} $\mathbf{in}(\mathtt{priv}, (= \mathtt{parentOf}, rddoc_2, = fwdoc_2);$
q_{32} $\mathbf{out}(\mathtt{priv}, (\mathtt{parentOf}, rddoc_2));$
q_{33} $\mathbf{in}(\mathtt{priv}, (= \mathtt{parentOf}, rpdoc_1, = rddoc_2);$
q_{34} $\mathbf{out}(\mathtt{priv}, (\mathtt{PMS}, fwdoc_2, rpdoc_1, \mathtt{ready}));$
q_{35} $\mathbf{in}(\mathtt{priv}, (= \mathtt{PMR}, = rpdoc_1, = fwdoc_2, tagkey_1));$
q_{36} $\mathbf{let}\ (RPdomain_1, nonce_4) = \mathsf{dec}(tag_1, tagkey_1)\ \mathbf{in}$
q_{37} $\mathbf{out}(\mathtt{priv}, (\mathtt{PMS}, fwdoc_2, rpdoc_1, (EncIA_1, RPdomain_1))).$

Figure 3.6: SPRESSO Client-side Process

Browser API calls. They can be modeled by utilizing the interfaces provided by our framework, which are listed in Table. 3.1.

- Open window. This command can be triggered when a new window or enbedded iframe is prompted from the current window. For example, when the user initiates the SSO login process, the login page hereby named RPdoc is loaded from the RP server. We model the RPdoc opening and initialization by sending the `OW` command to our framework and receiving the identity of the opened window back from the framework (line q_4-q_5). Note that we use the identity `root` to represent the initial browser status before the SSO login process starts (q_4). Similarly, the opening of the IDP client named IDPdoc ((7), line q_{10}) and FWDdoc((14), line q_{26}) are modeled. Note that the parameters passed to the newly opened pages can be modeled as private messages shown in q_{11} via channel $\texttt{priv}_{\texttt{rd}}$ for IDPdoc and in q_{27} via channel $\texttt{priv}_{\texttt{fw}}$ for FWDdoc.

- HTTP(s) messaging. This command is triggered for browser-server and server-server message delivery. In order to deliver an HTTP(S) message, a channel needs to be established. The channel establishing process can be modeled by sending the `http(s)Connect` command to our framework which are shown in the model ((1,2), line q_7) for RPdoc, ((8), line q_{18} and q_{21}) for IDPdoc. After the HTTP(S) channel is established, the messages can be sent out by `http(s)Send` command and received by `http(s)Receive` command. For example, the RPdoc sends out the user email address to the RP server via HTTPS in ((3), line q_8). Similar HTTPS message sending requests can be found in ((19), line q_{15} ; (8), line q_{19}; and (10), line q_{23}). The RPdoc receives three arguments from the RP server after sending out the user email ((6), line q_9). Similar HTTPS message receiving requests can be found in ((22), line q_{16}; (8), line q_{20}; and (12), line q_{24}).

- Cross-domain messaging. As explained earlier, the cross-domain messaging is achieved using the browser function `postMessage`. We

model `postMessage` as a message sent over a private channel, marked by key words `PMS` and `PMR` for sending and receiving respectively. For example, FWDdoc sends the RPdoc a `ready` signal at ((16), line q_{34}) and RPdoc receives this message at the same time at ((16), line q_{12}). Similar `postMessage` pairs can be found at ((16), q_{13} and q_{35}) and ((18), line q_{37} and q_{14})).

- Parent-child relationship in windows. This information is usually maintained by the browser upon window opening and closure. We model the request of identifying the parent window of the current window by sending the `parentOf` command to our framework. For example, the window FWDdoc use this property to locate and message its grand parent (line q_{30}-q_{33}) by sending the `parentOf` command twice.

Customized functions. They can be modeled by defining customized function symbols together with the involved parameters. In particular, cryptographic functions including encryption/ decryption, digital signature and digital signature validation can be easily defined, utilizing the equational theory provided by the applied pi calculus. An example of the customized function is $getrpdomain(RPname_1)$ in line q_{14} will extract the RP domain of the parameter, which in this case will be filled in by argument RPname_1. Another example of the customized function is $\mathsf{getpss}(\mathsf{email})$ in line q_{22} which extracts the password associated with the email account. An example of the cryptographic function is $\mathsf{enc}(\mathsf{ia}, \mathsf{iakey})$ in line q_{25} which outputs an symmetrically encrypted message $EncIA$ generated by encrypting the message ia with the key iakey. $EncIA$ can only be decrypted with the same key used for encryption. Another example of the cryptographic function is $\mathsf{enc}(\mathsf{tag}_1, \mathsf{tagkey}_1)$ in line q_{36} which symmetrically decrypts the encrypted tag_1 using the key tagkey_1

3.5.1.2 SERVER-SIDE PROCESSES

Compared to the client process modeling described earlier, the the server process modeling usually includes simpler API calls that are only

$$
\begin{array}{ll}
 & \boldsymbol{RP\_proc(RPname_2)} := \\
p_1 & \mathbf{in}(\mathtt{c}, (rpdoc_2, = RPname_2)); \\
p_2 & \mathbf{in}(\mathtt{priv}, (= \mathtt{httpsReceive}, (account_2, IDPname_2), \\
 & = rpdoc_2, = RPname_2)); \\
p_3 & \mathbf{out}(\mathtt{priv}, (\mathtt{httpsConnect}, RPname_2, IDPname_2)); \\
p_4 & \mathbf{in}(\mathtt{priv}, (= \mathtt{httpsReceive}, pkidp, = RPname_2, = IDPname_2)); \\
p_5 & \mathbf{new}\ \mathtt{nonce}_3;\ \mathbf{new}\ \mathtt{iakey}_1;\ \mathbf{new}\ \mathtt{tagkey}_2; \\
p_6 & \mathbf{new}\ \mathtt{logsesstoken}_2; \mathbf{new}\ \mathtt{fwdomain}_2; \\
p_7 & \mathbf{let}\ RPdomain_2 = \mathsf{getrpdomain}(RPname_2)\ \mathbf{in} \\
p_8 & \mathbf{let}\ tag_2 = \mathsf{enc}((RPdomain_2, \mathtt{nonce}_3), \mathtt{tagkey}_2)\ \mathbf{in} \\
p_9 & \mathbf{out}(\mathtt{priv}, (\mathtt{httpsSend}, (\mathtt{tagkey}_2, \mathtt{fwdomain}_2, \mathtt{logsesstoken}_2), \\
 & rpdoc_2, RPname_2)); \\
p_{10} & \mathbf{in}(\mathtt{c}, (rddoc_2, = RPname_2)); \\
p_{11} & \mathbf{in}(\mathtt{priv}, (= \mathtt{httpsReceive}, = \mathtt{logsesstoken}_2, \\
 & = rddoc_2, = RPname_2)); \\
p_{12} & \mathbf{out}(\mathtt{priv}, (\mathtt{httpsSend}, (rddoc_2, tag_2, \mathtt{fwdomain}_2, \\
 & (account_2, IDPname_2), \mathtt{iakey}_1), rddoc_2, RPname_2)); \\
p_{13} & \mathbf{in}(\mathtt{priv}, (= \mathtt{httpsReceive}, (EncIA_2, = \mathtt{logsesstoken}_2), \\
 & = rpdoc_2, = RPname_2)); \\
p_{14} & \mathbf{let}\ ia_2 = \mathsf{dec}(EncIA_2, \mathtt{iakey}_1)\ \mathbf{in} \\
p_{15} & \mathbf{let}\ (= tag_2, = (account_2, IDPname_2), = \mathtt{fwdomain}_2) = \\
 & \mathsf{getmsg}(ia_2, pkidp)\ \mathbf{in} \\
p_{16} & (\mathbf{out}(\mathtt{priv}, (\mathtt{httpsSend}, \mathtt{success}, rpdoc_2, RPname_2))) \\
p_{17} & \mathbf{else}(\mathbf{out}(\mathtt{priv}, (\mathtt{httpsSend}, \mathtt{retry}, rpdoc_2, RPname_2))). \\
 & \\
 & \boldsymbol{IDP\_proc(IDPname_3)} := \\
p_{18} & \mathbf{in}(\mathtt{c}, (RPname_3, = IDPname_3)); \\
p_{19} & \mathbf{out}(\mathtt{priv}, (\mathtt{httpsSend}, pkidp, RPnanme_3, IDPname_3)); \\
p_{20} & \mathbf{in}(\mathtt{c}, (idpdoc_1, = IDPname_3)); \\
p_{21} & \mathbf{in}(\mathtt{priv}, (= \mathtt{httpsReceive}, (email_2, password_1, \\
 & fwdomain_3, tag_3), = idpdoc_1, = IDPname_3)); \\
p_{22} & \mathbf{if}\ password_1 = \mathsf{getpss}(email_2)\ \mathbf{then} \\
p_{23} & \mathbf{let}\ ia_3 = \mathsf{sign}((tag_3, email_2, fwdomain_3), \mathtt{skidp})\ \mathbf{in} \\
p_{24} & \mathbf{out}(\mathtt{priv}, (\mathtt{httpsSend}, ia_3, idpdoc_1, IDPname_3)).
\end{array}
$$

Figure 3.7: SPRESSO Server-side Processes

related to the HTTP(S) messaging and richer server functions. This is due to the reason that most browser based API calls are abstracted away for web servers and more customized functions are needed to describe the functionality of the servers.

- HTTP(S) Messaging. After the browser has initiated the HTTP(S) connection, the server accepts this request and establish the connection. For example, the RP server establishes the HTTPS connection with the RPdoc ((1), line p_1). Similarly, RP and IDPdoc also establishes the HTTPS connection with the IDP server ((4) line p_3 and p_{18}; (10), line p_{20}). With the established connection, HTTP(S) sending/receiving makes up the significant part of the server processes ((3-4), line p_2-p_4 and p_{19}; (6), line p_9; (8), line p_{11}-p_{12}; (19), line p_{13}; (22), line p_{16}; (10), line p_{21}; and (12), p_{24}).

- Server functions. Various server functions can be modeled using the customized functions we described in client-side process modeling part. The ciphertext can be generated using $\mathsf{enc}(\mathsf{message}, \mathsf{key})$ ((5), line p_8); the ciphertext can be thereafter decrypted with the corresponding key using $\mathsf{dec}(\mathsf{ciphertext}, \mathsf{key})$ ((20), line p_{14}). The digital signature can be generated with the secret key of the server using $\mathsf{sign}(\mathsf{message}, \mathsf{secret\_key})$ ((11), line p_{23}) and verified using the corresponding public key by $\mathsf{getmsg}(\mathsf{signature}, \mathsf{public\_key})$ ((20), line p_{15}).

3.5.2 BrowserID

The general process of the BrowserID protocol is shown in Fig. 3.8. Following this process, the BrowserID protocol is modeled, as shown in Fig. 3.9, Fig. 3.10 and Fig. 3.11.

Due to the similarity between BrowserID and SPRESSO, we hereby only highlight the difference occurs in the protocol modeling. Fig. 3.9 shows the overall model of the BrowserID protocol.

The model of the client-side process of the BrowserID protocol is shown in Fig. 3.10. There are again three windows in the client-side process:

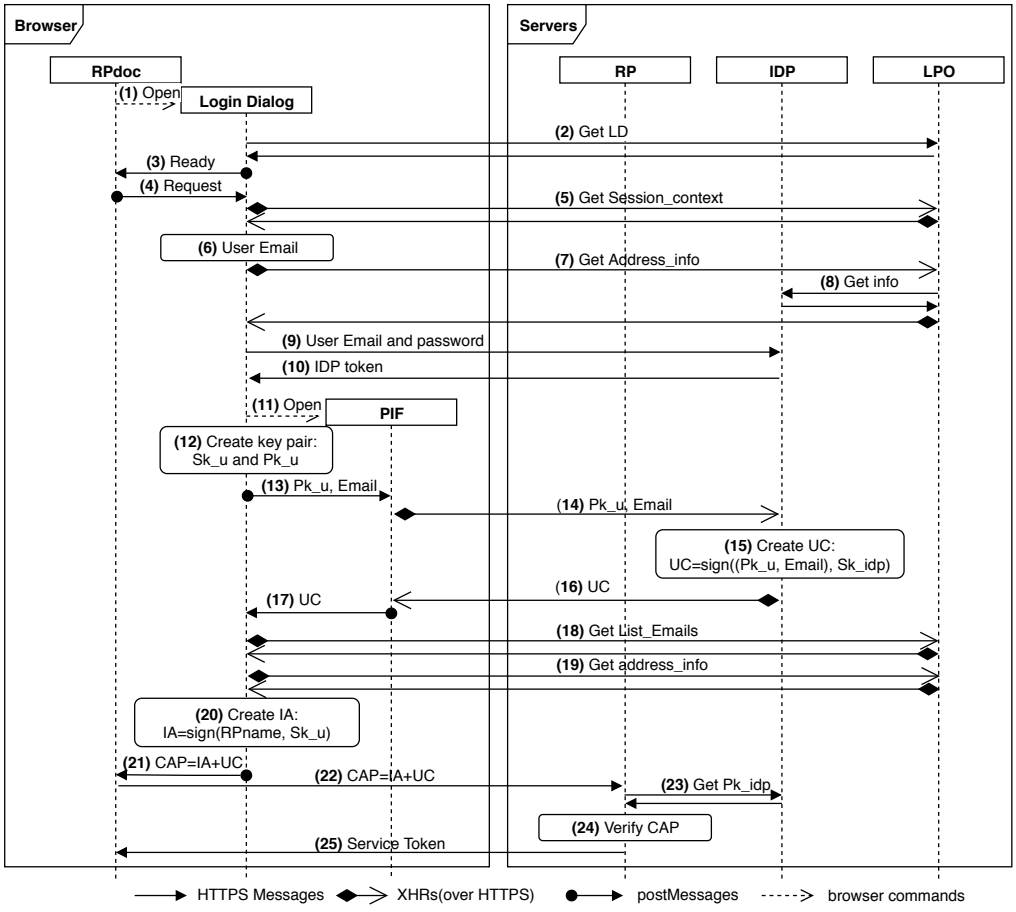


Figure 3.8: BrowserID Protocol Flow Chart [66]

$$
\begin{array}{ll}
BrowserID\_proc & := \\
b_1 & \textbf{new}\ \texttt{skidp}; \textbf{let}\ pkidp = \texttt{pk}(\texttt{skidp})\ \textbf{in} \\
b_2 & \textbf{new}\ \texttt{IDPname}; \textbf{new}\ \texttt{RPname}; \textbf{new}\ \texttt{Account}; \\
b_3 & (!\textbf{out}(\texttt{c}, \texttt{IDPname})|!\textbf{out}(\texttt{c}, \texttt{RPname})|C(\texttt{Account}, \texttt{RPname})| \\
b_4 & !IDP\_proc(\texttt{IDPname})|!RP\_proc(\texttt{RPname})|!WebInfra)
\end{array}
$$

Figure 3.9: BrowserID Protocol

C(account, rp) :=
a_1 $\mathbf{in}(\mathtt{c}, IDPname_1)$;
a_2 **let** $email = (account, IDPname_1)$ **in**
a_3 **let** $RPname_1 = rp$ **in**
a_4 **new** $\mathtt{root}$; $\mathbf{out}(\mathtt{priv}, (\mathtt{OW}, \mathtt{root}))$;
a_5 $\mathbf{in}(\mathtt{priv}, (= \mathtt{root}, rpdoc))$; **new** $\mathtt{ready}$; **new** $\mathtt{request}$;
a_6 $RPdoc\_proc|LD\_proc|PIF\_proc$

RPdoc_proc :=
a_7 $\mathbf{out}(\mathtt{priv}, (\mathtt{OW}, rpdoc))$; $\mathbf{in}(\mathtt{priv}, (= rpdoc, ld))$; $\mathbf{out}(\mathtt{priv}_{\mathtt{ld}}, ld)$;
a_8 $\mathbf{in}(\mathtt{priv}, (= \mathtt{PMR}, = ld, = rpdoc, = \mathtt{ready}))$;
a_9 $\mathbf{in}(\mathtt{priv}, (\mathtt{PMS}, ld, rpdoc, \mathtt{request}))$;
a_{10} $\mathbf{in}(\mathtt{priv}, (= \mathtt{PMR}, = ld, = rpdoc, (IA, UC, email_1)))$;
a_{11} $\mathbf{out}(\mathtt{priv}, (\mathtt{httpsConnect}, rpdoc, RPname_1))$;
a_{12} $\mathbf{out}(\mathtt{priv}, (\mathtt{httpsSend}, (IA, UC, email_1), rpdoc, RPname_1))$;
a_{13} $\mathbf{in}(\mathtt{priv}, (= \mathtt{httpsReceive}, servicetoken, = rpdoc, = RPname_1))$;

LD_proc :=
a_{14} $\mathbf{in}(\mathtt{priv}_{\mathtt{ld}}, ld_1)$; $\mathbf{out}(\mathtt{priv}, (\mathtt{PMS}, ld_1, rpdoc, \mathtt{ready}))$;
a_{15} $\mathbf{in}(\mathtt{priv}, (= \mathtt{PMR}, = ld_1, = rpdoc, = \mathtt{request}))$;
a_{16} **let** $password = \mathsf{getpss}(email)$ **in**
a_{17} $\mathbf{out}(\mathtt{priv}, (\mathtt{httpsConnect}, ld_1, IDPname_1))$;
a_{18} $\mathbf{out}(\mathtt{priv}, (\mathtt{httpsSend}, (email, password), ld_1, IDPname_1))$;
a_{19} $\mathbf{in}(\mathtt{priv}, (= \mathtt{httpsReceive}, (IDPtoken, = email),$
$= ld_1, = IDPname_1))$;
a_{20} $\mathbf{out}(\mathtt{priv}, (\mathtt{OW}, ld_1))$;
a_{21} $\mathbf{in}(\mathtt{priv}, (= ld_1, pif))$;
a_{22} $\mathbf{out}(\mathtt{priv}_{\mathtt{pif}}, (IDPtoken, pif))$;
a_{23} **new** $\mathtt{sk_u}$; **let** $pk_u = \mathsf{pk}(\mathtt{sk_u})$ **in**
a_{24} $\mathbf{out}(\mathtt{priv}, (\mathtt{PMS}, ld_1, pif, (pk_u, email)))$;
a_{25} $\mathbf{in}(\mathtt{priv}, (= \mathtt{PMR}, = ld_1, = pif, UC_1)$;
a_{26} **let** $IA_1 = \mathsf{sign}(RPname_1, \mathtt{sk_u})$ **in**
a_{27} **let** $CAP = (IA_1, UC_1)$**in**
a_{28} $\mathbf{out}(\mathtt{priv}, (\mathtt{PMS}, ld_1, rpdoc, (CAP, email)))$;

PIF_proc :=
a_{29} $\mathbf{in}(\mathtt{priv}_{\mathtt{pif}}, (IDPtoken_1, pif_1))$;
a_{30} $\mathbf{in}(\mathtt{priv}, (= \mathtt{PMR}, ld_2, = pif_1, (pk_{u1}, email_2)))$;
a_{31} $\mathbf{out}(\mathtt{priv}, (\mathtt{httpsConnect}, pif_1, IDPname_1))$;
a_{32} $\mathbf{out}(\mathtt{priv}, (\mathtt{httpsSend}, (pk_{u1}, email_2), pif_1, IDPname_1))$;
a_{33} $\mathbf{in}(\mathtt{priv}, (= \mathtt{httpsReceive}, UC_2, = pif_1, = IDPname_1))$;
a_{34} $\mathbf{out}(\mathtt{priv}, (\mathtt{PMS}, ld_2, pif_1, UC_2)$.

Figure 3.10: BrowserID Client-side Process

RPdoc which is the RP client-side window, LDdoc which is the login dialog loaded from the Mozilla's login.persona.org (LPO) and PIF (provinsioning iframe) loaded from the IDP. They are modeled as three separate sub-processes running in parallel (line a_6). Since all the three windows need to communicate with their corresponding servers, the HTTPS channel establishment is included in each window model: the RPdoc connects the RP server ((22), line a_{11}), the LDdoc and the PIF connect the IDP server ((9) and (14), line a_{17} and a_{31}). The HTTPS message delivery is modeled the same as SPRESSO. The invoking of the browser APIs are also modeled the same way as SPRESSO, including `OW`, `PMS` and `PMR`. The customized functions modeled in the client-side process of BrowserID, however, is a little bit different compared to SPPRESSO. For example, a user generated digital signature named IA_1 is created in LDdoc by signing the RP domain name $RPname_1$ with the user's private key $\mathtt{sk_u}$ ((20), line a_{26}). Therefore, a newly generated private-public key pair for the user must be created before the digital signature. We model this public key pair generation by ((12), line a_{23}). Note that the LDdoc is loaded from the server LPO (as shown in Fig. 3.8) which serves the purpose of loading basic protocol information. Therefore we do not specifically model the interaction between the LDdoc and the LPO server for simplicity on the assumption that the information from LPO server is correctly delivered automatically.

The model of the server processes of the BrowserID protocol is shown in Fig. 3.11. Similar to the client-side process discussed earlier, the standard browser APIs are modeled the same way as SPRESSO protocol. The customized functions are applied in slightly different ways. One example is the signature verification process in BrowserID. Since one user is authenticated by the certificate assertion pair(CAP) which includes identity assertion (IA) and user certificate (UC), there are two digital signatures created: IA generated by the LDdoc and UC generated by IDP. Therefore, the RP needs to verify both signatures by first extracts the user public key from the UC using IDP's public key; then further extracts the RP domain name from IA using this extracted user's public key. This 'locked' verification process is modeled as ((24), c_6-c_7) in the RP server model.

RP_proc(RPname$_2$) :=
c_1 **in**(c, ($rpdoc_1$, = $RPname_2$));
c_2 **in**(priv, (= httpsReceive, (IA_3, UC_3, $email_3$), = $rpdoc_1$, = $RPname_2$));
c_3 **let** ($account_1$, $IDPname_2$) = $email_3$ **in**
c_4 **out**(priv, (httpsConnect, $RPname_2$, $IDPname_2$));
c_5 **in**(priv, (= httpsReceive, $pkidp_1$, = $RPname_2$, = $IDPname_2$));
c_6 **let** (pk_{u2}, $email_4$) = getmsg(UC_3, $pkidp_1$) **in**
c_7 **let** $rpname$ = getmsg(IA_3, pk_{u2}) **in**
c_8 **if** $rpname$ = $RPname_2$ **then**
c_9 **new** servicetoken_1;
c_{10} **out**(priv, (httpsSend, servicetoken_1, $rpdoc_1$, $RPname_2$)).

IDP_proc(IDPname$_3$) :=
c_{11} **in**(c, (ld_3, = $IDPname_3$));
c_{12} **in**(priv, (= httpsReceive, ($email_5$, $password_1$), = ld_3, = $IDPname_3$));
c_{13} **if** $password_1$ = getpss($email_5$) **then**
c_{14} **new** IDPtoken_1; **out**(priv, (httpsSend, (IDPtoken_1, $email_5$), ld_3, $IDPname_3$));
c_{15} **in**(c, (pif_2, = $IDPname_3$));
c_{16} **in**(priv, (= httpsReceive, (pk_{u3}, = $email_5$), = pif_2, = $IDPname_3$));
c_{17} **let** UC_4 = sign((pk_{u3}, = $email_5$), skidp) **in**
c_{18} **out**(priv, (httpsSend, UC_4, pif_2, $IDPname_3$));
c_{19} **in**(c, ($RPname_3$, = $IDPname_3$));
c_{20} **out**(priv, (httpsSend, $pkidp$, $RPname_3$, $IDPname_3$)).

Figure 3.11: BrowserID Server-side Processes

3.5.3 OAuth Protocols

OAuth is one of the most widely deployed open standards for user authorization/authentication and it provides the basis for many other SSO specifications such as OpenID and OpenID Connect. It has been adopted by major identity providers(IDPs) including Twitter, Facebook, Google, Microsoft, etc. There are two versions of OAuth protocols currently in service namely OAuth 2.0 and OAuth 1.0a. They are specified in detail in[207] and [135]. We hereby detail the modeling of OAuth1.0a and OAuth 2.0 protocol flows using our framework in this section.

OAuth1.0a. The general process of the OAuth1.0a protocol is shown in Fig. 3.12. Following this process, the OAuth1.0a protocol is modeled, as shown in Fig. 3.13, Fig. 3.14 and Fig. 3.15.

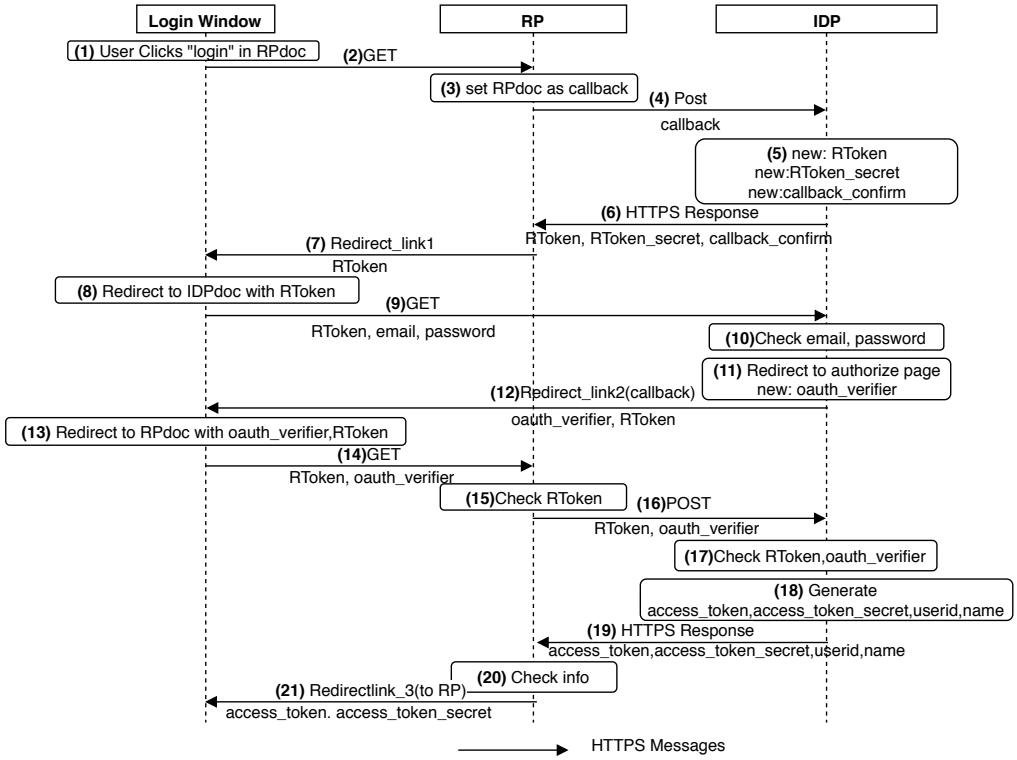


Figure 3.12: OAuth1.0a Authentication Process

$P_{OAuth} :=$
o_1 **new** IDPname; **new** RPname; **new** Account;
o_2 (!**out**(c, IDPname)|!**out**(c, RPname)|!$WebInfra$|
o_3 C(Account, RPname)|!IDP_proc(IDPname)|!RP_proc(RPname))

Figure 3.13: The OAuth1.0 protocol.

The biggest difference of the OAuth protocols compared to the SPRESSO or BrowserID is that there is only one login window in the browser instead of three. Therefore, the client-side process is modeled as a single process without sub-processes. Instead of using different windows to handle the communication with different servers, OAuth protocols redirect the current page to the specific URL to serve as the client-side page of the specific server. In terms of modeling, we assign two identities to the only window to handle the communication with different servers separately rather than directly modeling the page redirection, shown in lines t_5 and t_9 as example.

C(account, rp) :=
t_1 **in**(c, $IDPname_1$); **let** $email = (account, IDPname_1)$ **in**
t_2 **new** root; **out**(priv, (OW, root));
t_3 **in**(priv, (= root, $rpdoc$));
t_4 **let** $RPname_1 = rp$ **in**
t_5 **out**(priv, (httpsConnect, $rpdoc$, $RPname_1$));
t_6 **let** $callback = rpdoc$ **in**
t_7 **out**(priv, (httpsSend, ($callback$, $IDPname_1$), $rpdoc$, $RPname_1$));
t_8 **in**(priv, (= httpsReceive, ($requesttoken$, = $IDPname_1$),
= $rpdoc$, = $RPname_1$));
t_9 **new** idpdoc; **out**(priv, (httpsConnect, idpdoc, $IDPname_1$));
t_{10} **let** $password$ = getpss($email$) **in**
t_{11} **out**(priv, (httpsSend, ($requesttoken$, $email$, $password$),
idpdoc, $IDPname_1$));
t_{12} **in**(priv, (= httpsReceive, ($verifier$, = $requesttoken$),
= idpdoc, = $IDPname_1$));
t_{13} **out**(priv, (httpsSend, ($verifier$, $requesttoken$), $rpdoc$, $RPname_1$));
t_{14} **in**(priv, (= httpsReceive, ($accesstoken$, $tokensecret$),
= $rpdoc$, = $RPname_1$)).

Figure 3.14: OAuth1.0a Client-side Process

The server processes are modeled in the similar way compared to SPRESSO protocol as we detailed earlier.

OAuth2.0. The general process of the OAuth1.0a protocol is shown in Fig. 3.17. Following this process, the OAuth1.0a protocol is modeled, as shown in Fig. 3.16, Fig. 3.18 and Fig. 3.19.

The biggest difference between OAuth1.0a and OAuth2.0 is that there are two different modes of applying the protocol: authentication and authorization. Since the protocol flow in different mode will vary slightly, we reflect this variation in our model by introducing the parameter named *mode* (line o_7).

The modeling of the client-side process is similar to that of the OAuth1.0a protocol. The modeling of server side processes considers the mode of the protocol by introducing the sub-processes and distinguishing the protocol flow based on the value of the parameter *mode*. If the *mode*=authorization, then the RP server would just use the access token to retrieve the user permitted information ((12-14), line e_{11} and line e_{26}-e_{28}). If the *mode*=authentication, then the RP server would first use the access token to retrieve the client ID

$\boldsymbol{RP\_}$ $\boldsymbol{proc(RPname_2)}$:=
r_1 **in**(c, ($rpdoc_1$, = $RPname_2$));
r_2 **in**(priv, (= httpsReceive, ($callback_1$, $IDPname_2$), = $rpdoc_1$, = $RPname_2$));
r_3 **out**(priv, (httpsConnect, $RPname_2$, $IDPname_2$));
r_4 **out**(priv, (httpsSend, $callback_1$, $RPname_2$, $IDPname_2$));
r_5 **in**(priv, (= httpsReceive, ($requesttoken_1$, $secret_1$),
= $RPname_2$, = $IDPname_2$));
r_6 **out**(priv, (httpsSend, ($requesttoken_1$, $IDPname_2$), $rpdoc_1$, $RPname_2$));
r_7 **in**(priv, (= httpsReceive, ($verifier_1$, = $requesttoken_1$),
= $rpdoc_1$, = $RPname_2$))
r_8 **out**(priv, (httpsSend, ($verifier_1$, $requesttoken_1$), $RPname_2$, $IDPname_2$));
r_9 **in**(priv, (= httpsReceive, ($accesstoken_1$, $tokensecret_1$, $userinfo$),
= $RPname_2$, = $IDPname_2$));
r_{10} **out**(priv, (httpsReceive, ($accesstoken_1$, $tokensecret_1$), $rpdoc_1$, $RPname_2$)).
$\boldsymbol{IDP\_}$ $\boldsymbol{proc(IDPname_3)}$:=
r_{11} **in**(c, ($RPname_3$, = $IDPname_3$));
r_{12} **in**(priv, (= httpsReceive, ($callback_2$), = $RPname_3$, = $IDPname_3$));
r_{13} **new** $\texttt{requesttoken}_2$; **new** secret;
r_{14} **out**(priv, (httpsSend, ($\texttt{requesttoken}_2$, secret), $RPname_3$, $IDPname_3$));
r_{15} **in**(c, ($idpdoc_1$, = $IDPname_3$));
r_{16} **in**(priv, (= httpsReceive, (= $\texttt{requesttoken}_2$, $email_1$, $password_1$),
= $idpdoc_1$, = $IDPname_3$));
r_{17} **if** $password_1$ = $\texttt{getpss}(account_1)$ **then**
r_{18} **new** $\texttt{verifier}_2$;
r_{19} **out**(priv, (httpsSend, ($\texttt{verifier}_2$, $\texttt{requesttoken}_2$), $idpdoc_1$, $IDPname_3$));
r_{20} **in**(priv, (= httpsReceive, (= $\texttt{verifier}_2$, = $\texttt{requesttoken}_2$),
= $RPname_3$, = $IDPname_3$));
r_{21} **new** $\texttt{accesstoken}_2$; **new** $\texttt{tokensecret}_2$; **new** $\texttt{userinfo}_1$;
r_{22} **out**(priv, (httpsSend, ($\texttt{accesstoken}_2$, $\texttt{tokensecret}_2$, $\texttt{userinfo}_1$),
$RPname_3$, $IDPname_3$)).

Figure 3.15: OAuth1.0a Server-side Process

P_{OAuth} :=
o_4 **new** IDPname; **new** RPname; **new** Account;
o_5 **new** authentication; **new** authorization; **new** auth_request;
o_6 (!**out**(c, IDPname)|!**out**(c, RPname)|!$WebInfra$|
o_7 **let** $mode$ = authentication **in**
($mode$ *can be* authentication *or* authorization)
o_8 C(Account, RPname)|!IDP_proc(IDPname)|!RP_proc(RPname))

Figure 3.16: The OAuth2.0 protocol.

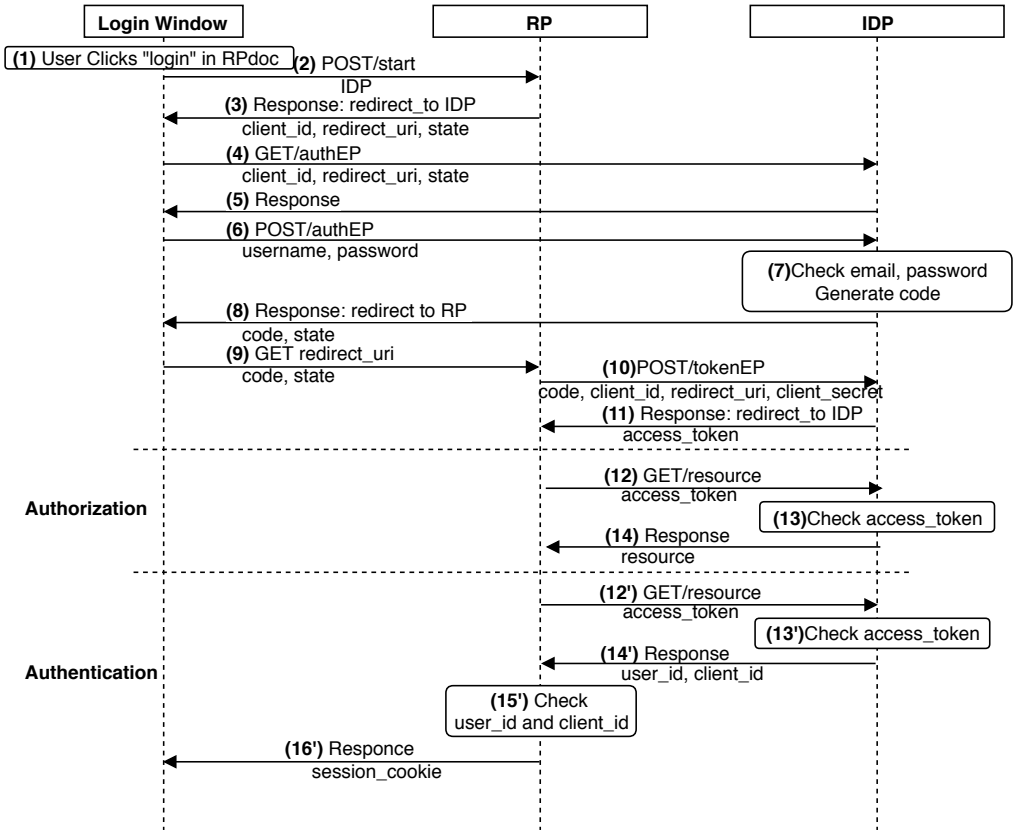


Figure 3.17: OAuth2.0 Authentication Process

and user ID from the IDP ((12'-14'), line e_{12} and line e_{29}-e_{31}). Then the RP will generate a session cookie and send to the user if the client ID and user ID are valid ((15'-16'), line e_{13}).

3.5.4 Verification

We first query the security (i.e., authentication) property of the SSO protocol by querying the correspondence between two login events against the built-in Dolev-Yao network attacker model. Then we analyze the privacy (i.e., unlinkability) property. We transform the IDP_proc into the honest-but-curious attacker according to the Definition 1 and query the privacy property in ProVerif. Next, we transform the IDP_proc into the malicious IDP server according to the Definition 2 and query privacy. Finally, we add the malicious IDP client defined by Definition 4 to each protocol's main process (for example, the main process $SPRESSO\_proc$ of SPRESSO

C(account, rp) :=
d_1 **in**(c, $IDPname_1$);
d_2 **let** $RPname_1 = rp$ **in**
d_3 **new** root; **out**(priv, (OW, root));
d_4 **in**(priv, (= root, $rpdoc$));
d_5 **out**(priv, (httpsConnect, $rpdoc$, $RPname_1$));
d_6 **out**(priv, (httpsSend, $IDPname_1$, $rpdoc$, $RPname_1$));
d_7 **in**(priv, (= httpsReceive, ($clientid$, $redirecturi$, $state$),
= $rpdoc$, = $RPname_1$));
d_8 **out**(priv, (httpsConnect, $rpdoc$, $IDPname_1$));
d_9 **out**(priv, (httpsSend, ($clientid$, $redirecturi$, $state$),
$rpdoc$, $IDPname_1$));
d_{10} **in**(priv, (= httpsReceive, = auth_request,
= $rpdoc$, = $IDPname_1$));
d_{11} **let** $password$ = getpss($account$) **in**
d_{12} **out**(priv, (httpsSend, ($account$, $password$), $rpdoc$, $IDPname_1$));
d_{13} **in**(priv, (= httpsReceive, ($code$, = $state$),
= $rpdoc$, = $IDPname_1$));
d_{14} **out**(priv, (httpsSend, ($code$, $state$), $rpdoc$, $RPname_1$));
d_{15} **in**(priv, (= httpsReceive, ($sessioncookie$, = authentication),
= $rpdoc$, = $RPname_1$)).

Figure 3.18: OAuth2.0 Client-side Process

protocol) and query the privacy property.

Through the analysis, we have found a novel privacy attack where the malicious IDP server is able to distinguish which particular RP the victim user is logging into using two sets of public-private key pairs. The detailed trace is discussed below.

A logic flaw in SPRESSO and BrowserID.. We summarize the attack trace generated by ProVerif and normal trace as shown in Fig. 3.20. Note that each block represent a summarized action in this trace. The first row of the block presents the summary of this action; second row shows the involved participants of this action; the last row shows the corresponding steps (shown in Fig. 3.4) involved in this action. The normal trace is presented using black blocks while the attack trace is marked out using the red blocks. When a victim user uses his/her account `Account` which is registered from a malicious IDP to log in to an RP, the RP server requests a public key from the malicious IDP server. At this step, for a particular

RP_proc(RPname$_2$) :=

e_1 **in**(c, ($rpdoc_1$, = $RPname_2$));
e_2 **in**(priv, (= httpsReceive, $IDPname_2$, = $rpdoc_1$, = $RPname_2$));
e_3 **new** clientid$_1$; **new** clientsecret;
e_4 **new** redirecturi$_1$; **new** state$_1$;
e_5 **out**(priv, (httpsSend, (clientid$_1$, redirecturi$_1$, **new** state$_1$), $rpdoc_1$, $RPname_2$));
e_6 **in**(priv, (= httpsReceive, ($code_1$, = state$_1$), = $rpdoc_1$, = $RPname_2$));
e_7 **out**(priv, (httpsConnect, $RPname_2$, $IDPname_2$));
e_8 **out**(priv, (httpsSend, ($code_1$, clientid$_1$, redirecturi$_1$, clientsecret), $RPname_2$, $IDPname_2$));
e_9 **in**(priv, (= httpsReceive, $accesstoken$, = $RPname_2$, = $IDPname_2$));
e_{10} **out**(priv, (httpsSend, ($mode$, $accesstoken$), $RPname_2$, $IDPname_2$));
e_{11} (**in**(priv, (= httpsReceive, (= authorization, $resource$), = $RPname_2$, = $IDPname_2$));)|
e_{12} (**in**(priv, (= httpsReceive, (= authentication, $userid$, = clientid$_1$), = $RPname_2$, = $IDPname_2$));
e_{13} **new** sessioncookie$_1$;
e_{14} **out**(priv, (httpsSend, sessioncookie$_1$, $rpdoc_1$, $RPname_2$))).

IDP_proc(IDPname$_3$) :=

e_{15} **in**(c, ($rpdoc_2$, = $IDPname_3$));
e_{16} **in**(priv, (= httpsReceive, ($clientid_2$, $redirecturi_2$, $state_2$), = $rpdoc_3$, = $IDPname_3$));
e_{17} **new** auth_request$_1$;
e_{18} **out**(priv, (= httpsSend, auth_request$_1$, $rpdoc_3$, $IDPname_3$));
e_{19} **in**(priv, (= httpsReceive, ($account_1$, $password_1$), = $rpdoc_3$, = $IDPname_3$));
e_{20} **if** $password_1$ = getpss($account_1$) **then**
e_{21} **new** code$_2$; **out**(priv, (= httpsSend, (code$_2$, $state_2$), $rpdoc_3$, $IDPname_3$));
e_{22} **in**(c, ($RPname_3$, = $IDPname_3$));
e_{23} **in**(priv, (= httpsReceive, (= code$_2$, = $clientid_2$, = $redirecturi_2$, $clientsecret_1$), = $RPname_3$, = $IDPname_3$));
e_{24} **new** accesstoken$_1$;
e_{25} **out**(priv, (= httpsSend, accesstoken$_1$, $RPname_3$, $IDPname_3$));
e_{26} (**in**(priv, (= httpsReceive, (= authorization, = accesstoken$_1$), = $RPname_3$, = $IDPname_3$));
e_{27} **new** resource$_1$;
e_{28} **out**(priv, (= httpsSend, (authorization, resource$_1$), $RPname_3$, $IDPname_3$));)|
e_{29} (**in**(priv, (= httpsReceive, (= authentication, = accesstoken$_1$), = $RPname_3$, = $IDPname_3$));
e_{30} **new** userid$_1$;
e_{31} **out**(priv, (= httpsSend, (authentication, userid$_1$, $clientid_2$), $RPname_3$, $IDPname_3$))).

Figure 3.19: OAuth2.0 Server-side Processes

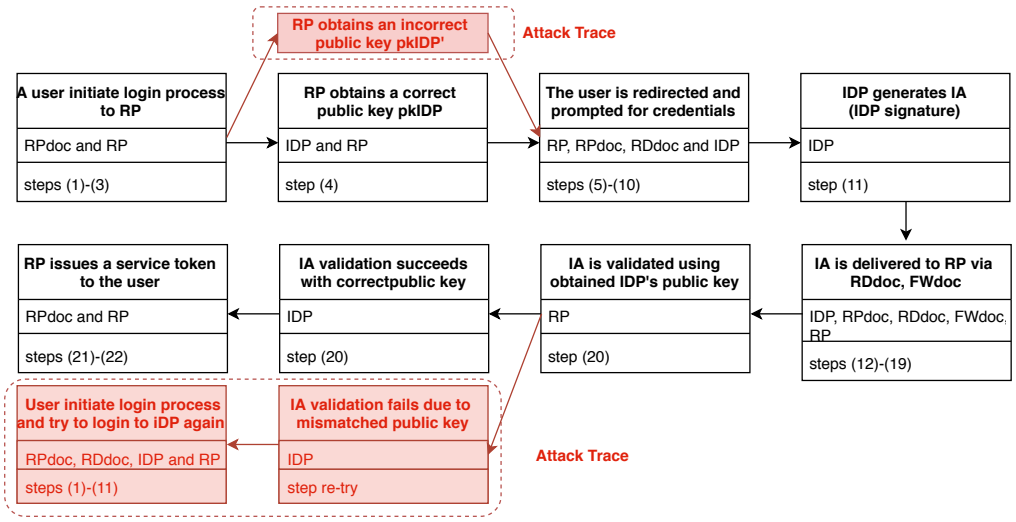


Figure 3.20: SPRESSO attack trace and normal trace

RP $\texttt{RP}_\texttt{i}$, if the malicious IDP wants to learn its login users, the IDP can issue an incorrect public key $\texttt{pkIDP}'$ to it ((4) in Fig. 3.4); for other RPs, the IDP issues the normal public key `pkIDP`. Later in identity assertion (`IA`) generation, the IDP always uses the private key corresponding to `pkIDP` ((11) in Fig. 3.4). As a result, a failure is caused when $\texttt{RP}_\texttt{i}$ verifies the `IA` using the public key $\texttt{pkIDP}'$ it fetched previously ((20) in Fig. 3.4). This implies that the user is successfully logged in to the IDP, but actually fails to log in to the RP. We assume that the user will log in again upon receiving a login failure notification, which is common in reality. Upon receiving the second log in request ((10) in Fig. 3.4), the malicious IDP knows the identity of the user who wants to log in to $\texttt{RP}_\texttt{i}$. This sabotages the declared unlinkability property of SPRESSO. Note that BrowserID also suffers from this type of attack.

3.6 Related Work

Verifying Security Property. Armando et al. [8] built formal models of the SAML-based Web Browser SSO protocol as well as the SSO protocol implementation for Google Apps. They analyzed the security property of the formal models using a state-of-the-art model checker– SATMC [9]

and revealed a security flaw in the protocol used by Google such that a malicious service provider is able to impersonate the victim user at another service provider. Bansal et al. [18] analyzed the OAuth2.0 protocol using the applied pi-calculus and the WebSpi library which facilitates the modeling of web applications and web-based attackers. They modeled several settings of the OAuth2.0 protocol and automatically analyzed them using ProVerif [31]. Several web implementation flaws including CSRF attack and unauthorized access were identified on Facebook, Yahoo, etc. Wang et al. [199] proposed a systematic approach to find implicit assumptions in the SDKs used by major SSO providers including Facebook, Microsoft. They formally analyzed if the currently used SDK can imply the desired authentication/authorization properties. They applied the approach on three popular SDKs and identified several implicit assumptions made in the SDKs. They further verified that applications built with the SDKs are vulnerable due to the implicit assumptions made in the SDKs. Sun et al. [170] modeled OAuth2.0 protocol using the High Level Protocol Specification Language (HLPSL) and verified the model using the automated tool AVISPA. They found three root weaknesses and further identified the concrete CSRF attacks by evaluating the weaknesses in the 123 real world relying party websites. Bai et al. [16] proposed an automatic approach to extract the SSO protocol specification from the real world implementations and perform formal analysis on the extracted protocol specification. They analyzed widely deployed SSO protocols such as BrowserID, Facebook Connect and Window Live ID and identified and confirmed 7 attacks including replay attack, CSRF attack, secret token leak, etc. Fett et al. [65] formally analyzed BrowserID with a comprehensive web model and identified several vulnerabilities including flaws due to the web infrastructure settings. They further confirmed the attack on the real world implementations.

Verifying Privacy Property. It is observed that a lot of prior work has done on the security property analysis and the privacy property has drawn little attention until recently. Not much work has been done on the SSO privacy checking and verification. BrowserID developed by Mozilla is claimed to preserve the SSO privacy that prevents identity provider

(IDP) from learning which third-party website a user is trying to log in to. Fett et al. [65, 66] have analyzed the privacy property of BrowserID manually by trace indistinguishability with a comprehensive protocol model. They have found that the promised privacy property in BrowserID can be compromised.

3.7 Summary

In this chapter, we present a formal framework consisting of a web infrastructure formal model, three attacker models, and the formalization of the authentication and unlinkability properties. We have analyzed SPRESSO using our framework and have detected a previously-unknown flaw which allows a malicious IDP to use an incorrect public key to differentiate the users which log in to a particular RP.

Chapter 4

Assessing Certificate Validation User Interfaces of WPA Supplicants

4.1 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. WPA Enterprise empowers WPA with authentication capability through the EAP (Extensible Authentication Protocol) authentication framework [140], which is carried by the Remote Authentication Dial In User Service (RADIUS) protocol [144]. EAP supports a wide variety of authentication mechanisms without any pre-negotiation or involvement of the pass-through agents (e.g., an access point), such that it is compatible with traditional password-based and token-based authentication methods.

EAP could directly utilize TLS as an authentication method per se by reusing its mutual authentication feature based on client and server certificates, leading to the EAP-TLS [177]. 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 [117] (Microsoft Challenge-Handshake Authentication Protocol).

This leads to other widely deployed protocols such as EAP-TTLS (Tunneled TLS) [61] and PEAP (Protected EAP) [60]. They eliminate the use of the 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 the server certificate is crucial for identifying a legitimate server[1]. Otherwise, failures could cause the access point impersonation attack in WPA Enterprise, or the *evil twin attack* [44, 36, 22]. 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. Considering 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 even trade secrets. This threat is further exacerbated by the increasingly popular BYOD (bring your own device)—the great variety in employees' devices and installed OSes would significantly enlarge the attack surface.

To mitigate such risk, WPA supplicants have implemented defense strategies to allow users to configure for mandatary validation, and to warn users once an evil twin is detected. Nonetheless, they may not be as effective as their counterparts in web browsers' Extended Validation (EV) certificate user interface (UI) [179, 64], which displays the padlock icon and actively blocks suspicious sites. For example, some supplicants leave insecure "*do not validate* (certificate)" as the by-default option [22].

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,

[1]The server authenticating the client is supported by EAP-TLS, but this is not included in the analysis of our work.

MacOS and Windows) that affect at least 95% of global mobile device users according to their market share [161]. We enumerate 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 [53] and many studies have been conducted on other non-browser user agents [63, 71].

By examining the related Android source code, we link the root cause of these vulnerabilities to the immature WPA supplicant designs and implementations, and insecure APIs exposed by them. In AOSP's `android.net.wifi` package, we find the `wifiManager.addNetwork` API allows an option of "*do not validate*", and the `wifiManager.addNetworkSuggestions` API confuses the identifiers of the suggested configuration files, allowing the attacker to manipulate the certificates accepted by `wpa_supplicant`. In the scenarios where a phase-2 authentication method is not specified, `wpa_supplicant` accepts any method proposed by the authenticator, allowing the attacker to manipulate the options of phase-2 authentication and further obtain login credentials.

We conduct three studies to better understand the users' susceptibility to the evil twin attack on all OSes. First, we survey 129 WPA Enterprise users to investigate their behavior during the configuration process. The majority (68%) of the participants choose to accept the prompted certificate without doubting its validity, even though most (113/129) of them have an engineering or IT background. Then, we perform a review of the Wi-Fi configuration guidelines of the global top 200 universities, revealing that only 38% (76/200) of them manage to provide secure instructions to their users. Last, we deploy an evil twin in an Internet technology company in a confined scenario. Within a 40-minute period, the simulated attack obtains the login credentials of 166 individuals, indicating the severity of the weakness in the existing certificate validation UIs.

The security of WPA has been continually and actively studied in the literature [36, 22, 96, 150, 185, 186, 187]. Among them, Brenza et al. [36]

leverage valid server certificates to exploit the `eap_workaround` compatibility setting in the phase-2 MSCHAPv2 protocol and hijack established connections in Eduroam. Bartoli et al. [22] investigate credential theft in Eduroam arising from incorrect network profile and insecure supplicant behaviour. As a comparison, our work targets the UIs of the certificate validation. In summary, this work makes the following contributions.

- **A large-scale study on the effectiveness of certificate validation UIs in WPA supplicants.** We have analyzed 13 mobile device brands/models and 16 OSes/versions, and revealed that the majority 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.
- **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 independently in parallel with Google).
- **Characterization of user susceptibility.** Our three studies have revealed users' susceptibility to the weaknesses in certificate validation UIs. This suggests the necessity of enhancing certificate validation UIs to defend against the evil twin attack.

4.2 Preliminaries

Our study focuses on client-side issues during WPA Enterprise's authentication process, 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 recall the WPA Enterprise configuration process in the mainstream OSes/devices.

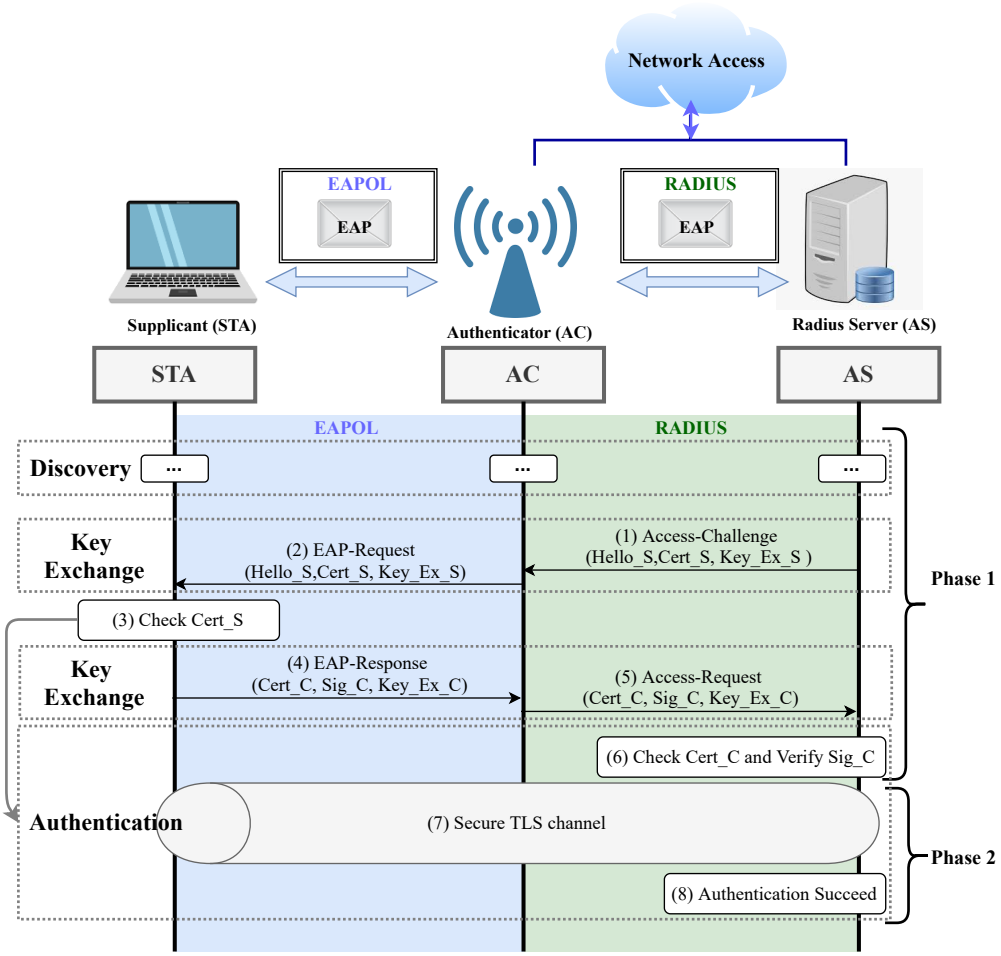


Figure 4.1: Overview of WPA Enterprise Structure

4.2.1 WPA Enterprise and its Authentication

As shown in Figure 4.1, there are three principal participants in the WPA Enterprise authentication model: a *supplicant* (STA) which corresponds to a user's 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. To be admitted to the network, the supplicant and the RADIUS server have to be mutually authenticated, through EAP (extensive authentication protocol) using X.509 client/server certificates or/and inner authentication methods. Considering 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.

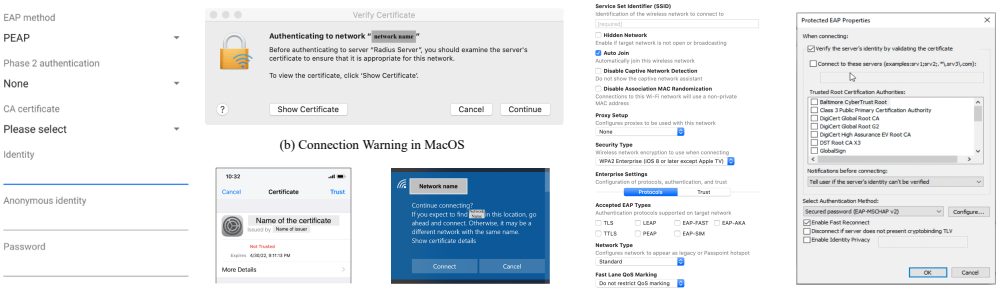
In this work, we focus on the three most widely-deployed EAP authentication protocols in WPA Enterprise [70], i.e., EAP-TLS, EAP-TTLS and PEAP, since others are either proven insecure previously (EAP-PWD [187]) or rarely used in practice (EAP-SIM and EAP-AKA). A generic EAP protocol flow is shown in Figure 4.1. The authentication process incorporates three components, i.e., *discovery*, *key exchange* and *authentication*. Phase 1 includes the discovery and the key exchange, for which all three protocols follow the same process. During the discovery, a supplicant connects with the AC and initiates the authentication process. During the key exchange, an encryption key is established between the supplicant and the AS via the TLS handshake (steps (1)-(2) and (4)-(5)). The authentication is slightly different among the 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), based on the secure TLS tunnel, after completing the AS authentication in phase 1. The supported phase-2 authentication protocols (i.e., the inner authentication protocols) include PAP which uses a pair of username and password for authentication, MSCHAPv2 which extends MSCHAP, and GTC (Generic Token Card) which uses the password and a one-time token for authentication.

4.2.2 Certificate Validation UIs

During the discovery stage, it is a common setting that the supplicant scans a list of available networks and then automatically connects to one it recognizes, i.e., one that has been previously connected to, or is listed in the preferred network list. Thus, securely configuring the WPA Enterprise



(a) Configuration Page in Android (b) Connection Warning in MacOS (c) Connection Warning in iOS (d) Connection Warning in Windows (e) Configuration Page in MacOS/iOS (f) Configuration Page in Windows

Figure 4.2: Configuring WPA Enterprise in Mainstream OSes

connection during the initial 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 recall 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 4.2(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 (cf. Section 4.4.2).

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 4.2 (b) and (c). Generally, the detailed configuration is "hidden" inside the profile and the user can manually generate this profile via the Apple Configurator This configuration turns out to be complex and non-intuitive such that users, even with network security knowledge, are prone to trust malicious access points (cf. Section 4.6.1).

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 4.2

(d). Alternatively, the user can manually set their configuration if the default settings are not supported. To this end, the user needs to open the Network and Sharing Center and select "*Set up a new connection or network*", followed by "*Manually connect to a wireless network*". The options for the EAP properties are available as shown in Figure 4.2 (f). With such a UI design, the users also tend to blindly trust any certificate and proceed with the connection (cf. Section 4.6.1).

4.3 Characterizing Weaknesses in Certificate Validation UIs

The complexity of configuring the certificate validation when connecting the supplicant with a WPA Enterprise network may lead to security weaknesses. In this section, we present the evil twin attack, and characterize the weaknesses in certificate validation UIs.

4.3.1 The Evil Twin Attack

4.3.1.1 Attacker Model

Our work considers the *active, on-path* evil twin attacker model [44, 36, 63], where the attacker deploys an access point in the vicinity of the victim user. 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 tunnel, it obtains the authentication credentials in phase 2.

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 trivial 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.

4.3.1.2 Attacker Capabilities

Through manually examining the WPA Enterprise authentication process, we summarize required attacker capabilities to launch the evil twin attack. Below we enumerate 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 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[2] or FreeRADIUS[3]. Section 4.4.1 discusses the setup in our experiments, demonstrating the setup is feasible and inexpensive.

Wi-Fi Channel. The attacker is able to deploy the evil twin in a different radio frequency 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 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 [24]. 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 [5] could be applied.

[2]https://w1.fi/hostapd/
[3]https://freeradius.org/

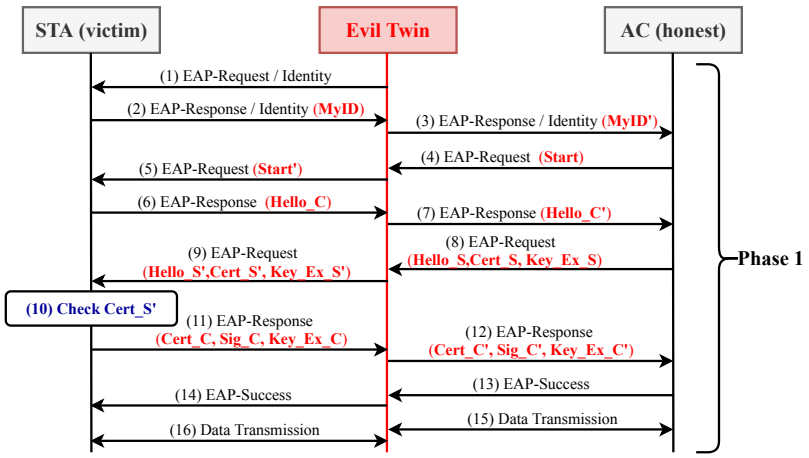


Figure 4.3: Attack Steps against Flawed Phase-1 Server Certificate Validation

4.3.1.3 System Prerequisites

We consider the following assumptions regarding the system configurations. We first assume that the infrastructure of WPA Enterprise, including the authenticator and the RADIUS server, are securely implemented. Otherwise the attacker is able to trivially impersonate as a legitimate user to access the network. Then, we assume that the enterprise wireless network supports the popular WPA Enterprise authentication protocols, including EAP-TLS, EAP-TTLS and PEAP. For EAP-TTLS and PEAP, the phase-2 authentication protocols such as MSCHAPv2, GTC and PAP are supported. This configuration is commonly adopted in the existing networks.

4.3.2 Weaknesses in Certificate Validation UIs

To the best of our knowledge, there are no available guidelines on client-side vulnerability assessment to spot weaknesses from certificate validation UIs. We thus resort to a manual inspection. In this section, we present the identified weaknesses that may be subject to the evil twin attack.

4.3.2.1 Weaknesses in Phase 1

Figure 4.3 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 misconfigured. As of EAP-TLS, the supplicant would mistake the attacker 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 [140]. 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 4.3.2.2). We have classified the following weaknesses in phase 1 that may lead to these attack steps.

V1-1: ***GUI flaws 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 normal users without security expertise are able to provide secure responses. Nevertheless, our inspection finds that several flaws may exist. For example, 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.

V1-2: ***Lack of conspicuous visual cues when any errors occur.*** The user would have a chance of rectifying the misconfigurations or terminating insure connections, if conspicuous warnings against insecure options or untrusted certificates are given. Our inspection finds that there may exist several flaws of lacking conspicuous warnings. For example, 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 option of "*unspecified*" without any warning.

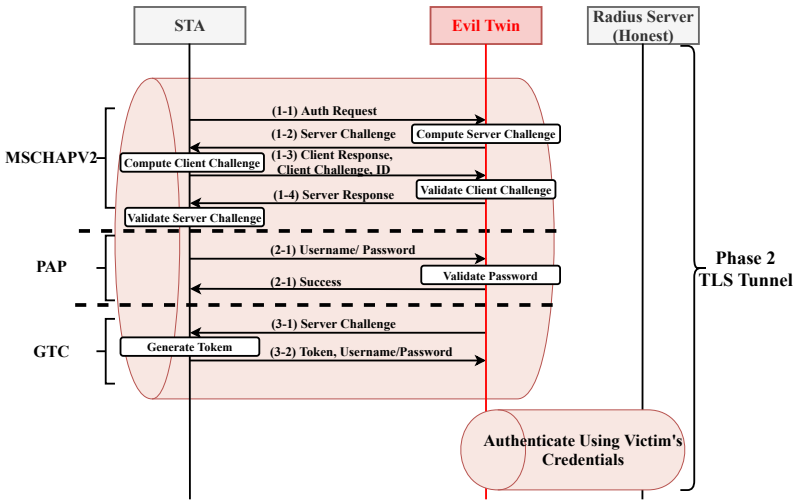


Figure 4.4: Flawed Client Authentication in Phase 2

4.3.2.2 Weaknesses in Phase 2

The legacy user authentication protocols, such as MSCHAPv2, PAP and GTC, may assume a securely established TLS tunnel. Therefore, there may be security breach in them once the TLS connection in phase 1 is compromised, as shown in Figure 4.4. Among the existing phase-2 authentication protocols, MSCHAPv2 provides relatively stronger user authentication, as it does not transmit username/password in plaintext. However, there have been well-documented approaches [151, 90, 25] and tools (e.g., Openwall[4]) 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.

Besides the threat resulted from the compromised phase 1, the following vulnerability in phase 2 is possible.

V2-1: ***GUI flaws in configuration of phase 2.*** The supplicant may set the phase-2 authentication protocol as "*Unspecified*" or "*None*" by default. In this scenario, the phase 2 authentication protocol may be set by the server. The evil twin thus is able to manipulate it to PAP or GTC.

[4]https://www.openwall.com/john

4.4 Assessing Certificate Validation UIs of Android 10

With the knowledge on the weaknesses in certificate validation UIs, we propose a testing approach to examining the connection processes and identifying these weaknesses. Considering that the open-source nature of Android allows us to conduct an in-depth scrutiny, we first apply our approach to Android 10. In this section, we present this study.

4.4.1 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 4.5). 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`[5], and purchased a certificate trusted by all devices for `radius.anonymous.domain`. Following the setting commonly seen in practice, we use 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 different set of fake server and CA certificates which are generated and self-signed by `test.domain`. 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 the 2.4GHz channel.

[5]The domain name is anatomized for double blind review.

Figure 4.5: A Photo of Our Experimental Setup

Third, the RADIUS server is modified to allow connections from any devices. To this end, we disable client validation by modifying the function `eap_server_tls_ssl_init` in `src/eap_server/eap_server_tls.c`.

Testing Procedure. We enumerate through all combinations for the configuration items listed in Table 4.1 which includes the UI items shown in Figure 4.2(a). Each combination is regarded as a test case. We also generate a client certificate (`client.crt`) as the input to the "*User Certificate*" item, and a username/password pair *A/P(A)* for authentication. A test case can be represented as a tuple, for example, ⟨PEAP, PAP, "*Use system certificate*", "*Please select*", *A*, *P(A)*⟩. We notice that besides the configuration GUI, Android also provides the programmatic API `WifiManager.addNetworSuggestions` to implement Wi-Fi configuration in applications. It is extensively used by enterprises to implement their BYOD schemes. Therefore, we build an application which incorporates our test cases to test this API. Our testing application is released at [146].

We execute the test cases in two steps using a Pixel 4 phone 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 set the connection with a new test case. In the

Table 4.1: Fields and Options on Android WPA Enterprise Configuration UI

Field	Available Options
EAP Method	**PEAP, TLS, TTLS**
Phase 2 authentication	**None, PAP, MSCHAP, MSCHAPV2, GTC**
CA Certificate	**Please select, Use system certificate, Do not validate**
User Certificate	**Please select, client.crt, Do not provide**

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 disclosed 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 (the user susceptibility is studied in Section 4.6).

4.4.2 Testing Results on Android 10

After the experiments with each test case, we have found the following weaknesses from the Wi-Fi configuration of Android 10.

- **Finding #1:** When CA certificate is not validated (i.e., test cases ⟨ *, *, "*Do not validate*", *, *A*, *P(A)* ⟩), the supplicant is connected to the evil twin in EAP-TLS, and in EAP-TTLS and PEAP, *A/P(A)* for PAP and GTC and challenge-response hash value for MSCHAPV2 are sent to the evil twin.

- **Finding #2:** When adding the network configuration automatically through our application using a seemingly secure test case ⟨PEAP/TTLS, MSCHAPV2, "*Use system certificate*", `client.crt`, *A*, *P(A)*⟩, the supplicant is still connected to the evil twin.

- **Finding #3:** When both CA certificate and phase-2 protocol are not specified for EAP-TTLS/EAP-PEAP (i.e., test cases ⟨PEAP/TTLS, *None*, "*Do not validate*", *, *A*, *P(A)*⟩), the supplicant uses the phase-2

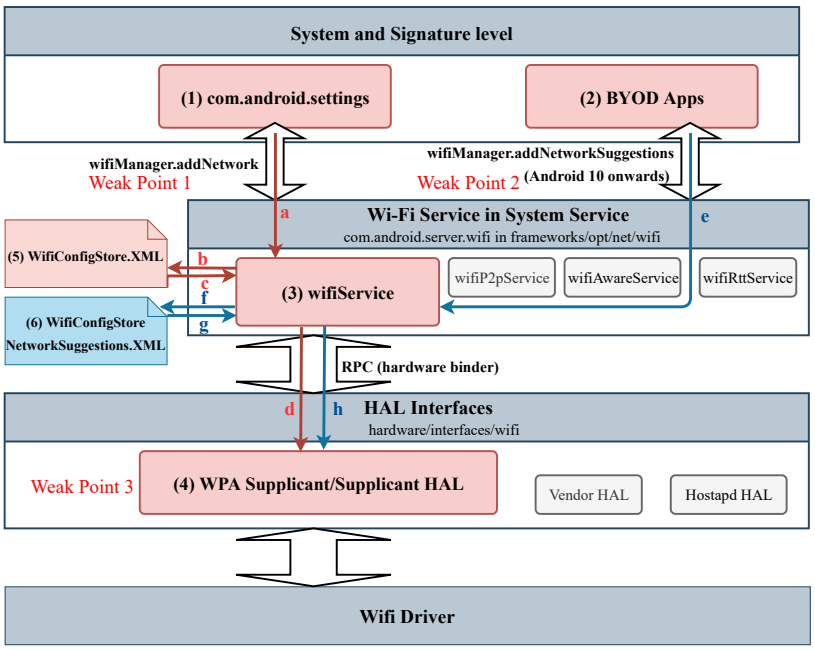


Figure 4.6: Android Wi-Fi Architecture for WPA Supplicants

protocol specified by the evil twin (MSCHAPV2 in our setting), such that the challenge-response hash is obtained by the evil twin.

4.4.3 Investigating Root Causes of Findings

We review the source code of the `android.net.wifi` package to confirm our findings and investigate their root causes. Through this, we have located three weak points that lead to our findings. To facilitate the understanding of our investigation, we first briefly introduce the Android Wi-Fi management in Section 4.4.3.1.

4.4.3.1 Internals of Android Wi-Fi Manager

Figure 4.6 shows the components that are distributed in each layer of Android OS and involved in Wi-Fi management. As mentioned in Section 4.4.1, the enterprise network for Android devices can be configured manually through the Settings UI, or automatically through third-party applications incorporated with Wi-Fi suggestion API, e.g., applications developed by BYOD-supporting enterprises (referred to as BYOD applications

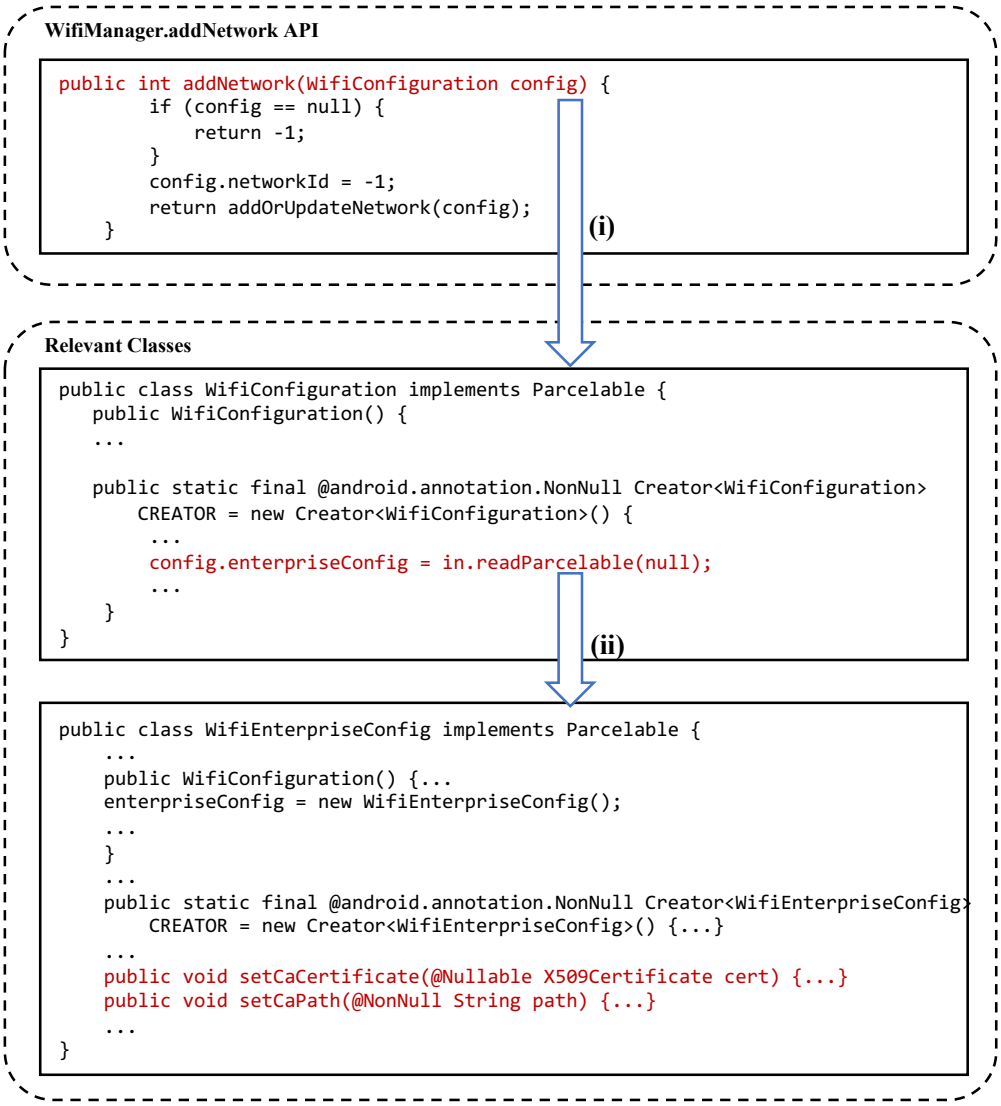


Figure 4.7: Android Source Code Associated with the `addNetwork` API Weaknesses

hereafter).

Configuration through Settings UI. Android Settings application invokes the `WifiManager.addNetwork` API to interact with the system service `WifiServices` through AIDL (Android interface description language). `WifiService` further configures the WPA supplicant for Wi-Fi connection through hardware binder (the configuration path is represented as (1)-a-(3)-d-(4) in Figure 4.6). For the initial connection to a wireless network, or connection to a network with new SSIDs, a configuration file

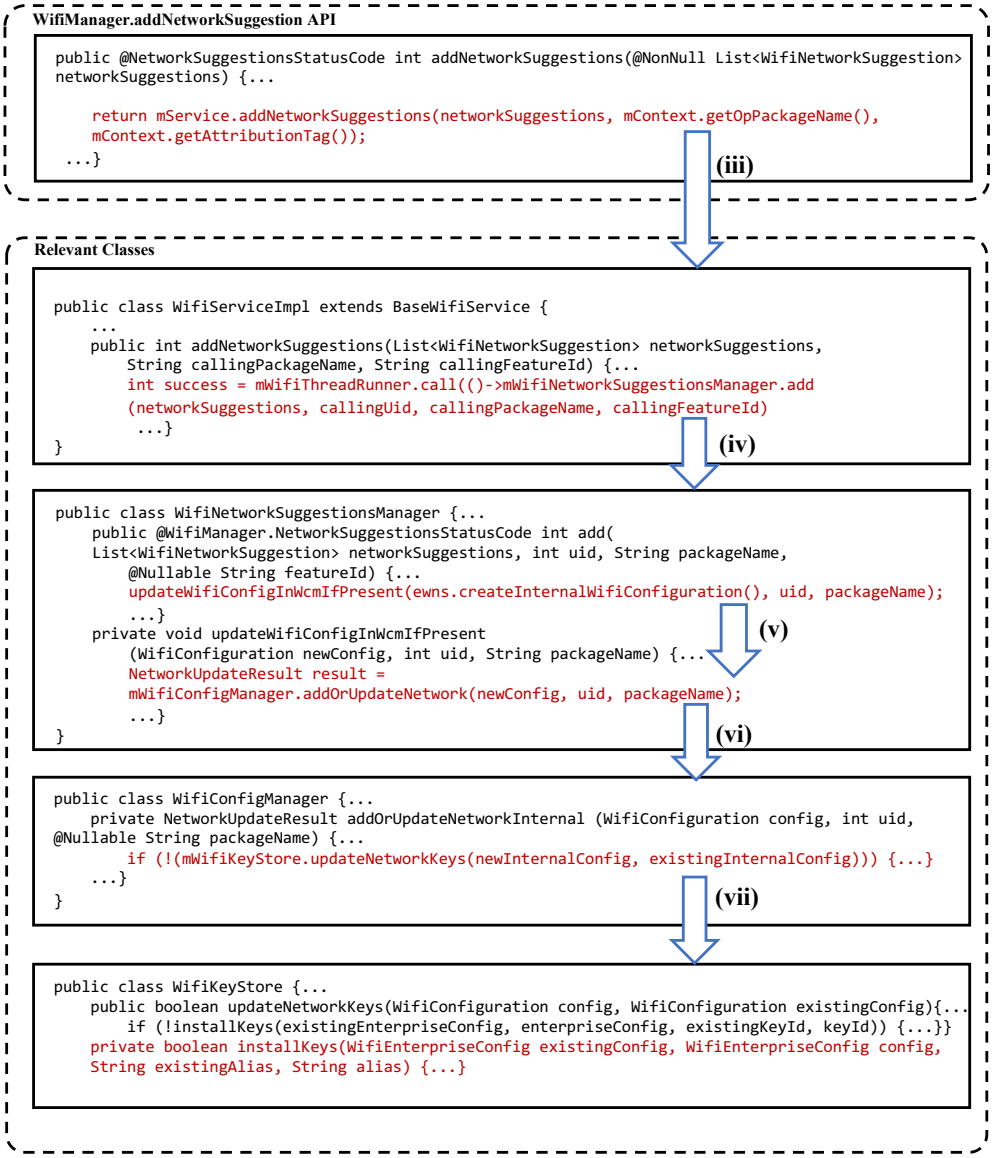


Figure 4.8: Android Source Code Associated with the `addNetworkSuggestion` API Weaknesses

`WifiConfigureStore.XML` is generated by `WifiServices` (the generation path is (1)-a-(3)-b-(5) in Figure 4.6). It is saved to restore the Wi-Fi configuration, enabling the automatic Wi-Fi connection after the device reboots (the configuration path is (5)-b-(3)-d-(4) in Figure 4.6).

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 ser-

vices available to the third-party applications through the Wi-Fi suggestion API, a distinct configuration path is adopted as follows. A configuration file `WifiConfigStoreNetworkSuggestions.XML` is generated by a BYOD application through `WifiNetworkSuggestion` class in `WifiService` (the generation path is (2)-e-f-(6) in Figure 4.6). 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 4.6).

4.4.3.2 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 our three findings. Their locations are labelled in Figure 4.6.

Weak point #1: ***incorrect settings of the*** `WifiManager.addNetwork` ***API***. This weak point leads to the finding #1. As shown in Figure 4.7, the existence of insecure settings specified in class `WifiEnterpriseConfig` allows vulnerable options in the configuration UI. In particular, it allows setting the `setCaCertificate` or `setCaPath` to `null` (default setting until Android 7, as shown by arrows **(i)** and **(ii)** in Figure 4.7). This results in 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 of the*** `WifiManager.addNetworSuggestions` ***API***. This weak point leads to the finding #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, the `addOrUpdateNetworkInternal` method from class `WifiConfigManager` is invoked (as described by the path from arrow **(iii)** through arrow **(vi)** in Figure 4.8). In this case, the method will update the Wi-Fi configuration (shown by arrow arrow **(vii)**) 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 choice on the phase-2 protocol***. This

weak point leads to the finding #3. We find this weakness when we run the test case ⟨ PEAP, None, "*Do not validate*", *, *, * ⟩, where the phase-2 protocol is not specified. However, the evil twin is still able to derive the challenge-response hash value for MSCHAPV2. From the function `eap_peer_select_phase2_methods` in `eap_tls_common.c`, we find that when the phase-2 protocol is set as "*None*" 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, then this protocol will be chosen. 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.

4.4.4 Responsible Disclosure

For each of the found vulnerabilities, we have promptly notified and warned Google. 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. We ensure that the reported vulnerabilities have been well acknowledged by the manufacturers and securely fixed in their updates, before disclosing details of the vulnerabilities. For the weakness in Wi-Fi Settings UI (weak point #1), we have been acknowledged by various major smartphone manufacturers (listed on CVE[6] with IDs *CVE-2020-12484*, *CVE-2020-1836* and *CVE-2020-9206*). For the weakness in `wpa_supplicant` (weak point #3), we have been acknowledged by Google (listed on CVE with ID *CVE-2020-2021*). Its severity level is rated by NVD[7] as critical with 9.8 base score. For the weakness in the `WifiManager.addNetworSuggestions` API (weak point #2), Google has confirmed the issue upon our warning (vulnerability ID 150500247).

[6]https://cve.mitre.org
[7]https://nvd.nist.gov

Table 4.2: Vulnerability Evaluation in Various OSes/Versions

Vulnerabilities	OS/Versions				
	Windows 7, 8 and 10	**MacOS 10.13-10.15**	**iOS 9-13**	**Android (AOSP) 6-10**	**Android (OEM) 7-10**
V1-1 (phase 1)	✓	✓	✓	✓	✓
V1-2 (phase 1)	✓	✓	✓	✓	✓
V2-1 (phase 2)	✗	✗	✗	✓	✓

✓ denotes the presence of vulnerabilities, and ✗ otherwise.

4.5 Assessing Certificate Validation UIs of other Devices and OSes

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 4.3.2. 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 4.4.1. Our evaluation aims to investigate the prevalence of the weaknesses and their security implications.

4.5.1 Weakness Prevalence

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 [161], as listed in Table 4.2. We find that the weaknesses in the certificate validation UIs are ubiquitous among these OSes and devices.

Windows. All versions of Windows (i.e., Windows 7, 8 and 10) require the users' permission before continuing with the authentication in Phase 1 (V1-1), as demonstrated in Figure 4.2 (d). As part of a secure procedure, the user should check the fingerprints of the server certificate by clicking on the

button "*Show certificate detail*". Otherwise, the user is at risk of credential disclosure. Take 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 clicking "*Connect*". We have reported this vulnerability to Microsoft (MSRC Case number 62537).

Windows 7, 8 and 10 suffer from the lack of conspicuous warnings when the certificate validation setting is insecure (V1-2). 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 4.2 (f). In addition, there is also no warning when the device is connected to the evil twin (V1-2).

iOS and MacOS. Similar to Windows, all latest versions of iOS (v9-13) and MacOS (v10.13-10.15) seek the user's permission to accept the certificate from the network being connected in phase 1 (V1-1). 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 an insecure certificate and connect the device to the evil twin (V1-2). 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 (V1-1), which remains a notable threat to the wireless connection security. More specifically, Android 6 or earlier has CA certificate option "*Unspecified*" by default, while Android 7 and versions onwards remove this choice but add the "*Do not validate*" option. There are warnings on insecure connections during the initial configuration if CA certificate is not validated. However, there is no warning during the subsequent connections, and there is also no warning when the user is connected to the evil twin (V1-2). Versions before Android 10 allow the phase-2 authentication protocol to be left blank

(V2-1), 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 [211]. 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 similar vulnerabilities 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.

4.5.2 Security Impacts

In this section, we brief the security impacts caused by the hijacked connections and leakage of the login credentials.

Client-side Security Impacts.

For EAP-TLS, the server and client identities are verified through the mutual authentication in phase 1. The supplicant is prone to be connected to an evil twin if the server and CA certificates are not correctly validated (i.e., CA certificate is set as "*Unspecified*" or "*Do not validate*"), such that the legitimate connection can be easily hijacked, and the evil twin is able to obtain all messages sent in plaintext. Even the upper layers use encryption to protect data transmission, the attacker is still able to obtain information regarding the user's online activities with a combination of traffic analysis techniques such as web fingerprinting [158].

For EAP-TTLS and PEAP with MSCHAPv2 as the phase-2 authentication method, the weakly encrypted passwords can be obtained [160] if the

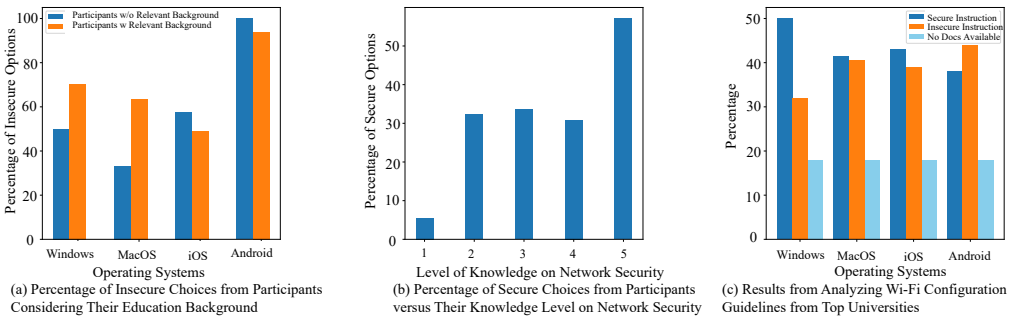
TLS connection established in phase 1 is compromised. For EAP-TTLS and PEAP without specifying phase-2 authentication protocol, which is a common practice in the enterprise network setting in Android OSes, the attacker is able to downgrade the preferred MSCHAPv2 protocol to GTC and PAP to retrieve the user's credentials in plaintext.

Enterprise Network-side Security Impacts. For EAP-TTLS and PEAP, the obtained password in plaintext enables the attacker to access the enterprise network as a legitimate user. Given that a significant proportion of the enterprises adopt the SSO scheme which uses the same username/-password set for not only the network access but also for corporate accesses such as working emails and desktop logins [175], this attack greatly exposes the risk of abusing the privileges of the victim employees. This attack also breaches through the presumably secure and sophisticated defences of the enterprises.

4.6 Assessing User Susceptibility

In addition to our analysis, a study by CISCO in 2020 [1] 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. Furthermore, these weaknesses raise more complications compared with those involving automatic configuration, owing to the wider discrepancies in security awareness levels, version and types of devices involved. In this section, we aim to better understand the user susceptibility to weaknesses in certificate validation UIs through three user-related studies.

The first study is from network users' perspective. We conducted a survey among frequent enterprise network users to learn their perception when connecting and authenticating to an enterprise networks. The second study is from the organization's perspective. We examine the available wireless network configuration guidelines from top universities to investigate whether their instructions and documentations give their Wi-Fi users secure configurations. The third study is a practical study. We deploy an evil



(a) Percentage of Insecure Choices from Participants Considering Their Education Background

(b) Percentage of Secure Choices from Participants versus Their Knowledge Level on Network Security

(c) Results from Analyzing Wi-Fi Configuration Guidelines from Top Universities

Figure 4.9: Results of Susceptibility Study

twin in a realistic confined environment to check how the identified UI weaknesses can eventually lead to real-world attacks.

4.6.1 User Study

Our user study aims to explore whether users' perception on the configuration options and response to the warnings could make them susceptible to the evil twin attack. In this section, we brief our study. Details including the questionnaire are available in the full report [146].

Our questionnaire has the main section of *WPA Enterprise configuration steps* where we include screenshots to simulate the configuration process for different types of devices and OSes. Once the participant selects his/her device type, he/she would be lead to the questions specific to the device type. The questions cover authentication configurations (such as certificate validation methods, choice of authentication protocols, etc.) and handling of warnings on potential vulnerable configurations. In addition, to study the correlation between the participants' choices and their security awareness and education level, we also include sections for self-evaluation on network/security knowledge and awareness for insecurity in Wi-Fi connection.

We have collected 129 valid responses from the participants who are frequent users of enterprise wireless networks. The majority of the participants have a computing/engineering-related background (over 87.6%) and have some knowledge regarding network security (over 77.8% chose 3 and above out of a range of 0-5, with 5 being the expert level of knowledge

and vice versa). As is shown in Figure 4.9 (a), the participants incline to continue with the insecure connection option, i.e., to choose '*Connect* (to the evil twin)' or '*Accept* (a fake certificate)', even when warnings are presented. Despite this risky preference, around 70% of the participants still perceive their credentials to be secure. As observed in Figure 4.9 (b), a relevant education background (i.e., engineering and computing) does not guarantee the option for secure configurations, as only fewer than 60% among those perceiving themselves to be security experts have made the correct choices. In addition, given the crucial role WPA Enterprise serves, we are surprised to find that only 24% of the participants are confident that they have received proper training and instructions on securely connecting to enterprise wireless networks.

4.6.2 Examining Universities' Wi-Fi Guidelines

The complexity of the configuration may also cause misinterpretation of enterprise network administrators. The Wi-Fi connection guidelines may be a suitable source to study how they interpret the security of WPA Enterprise. Therefore, we also conduct a study regarding this. 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 [143], 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 then review them to identify the weaknesses discussed in this paper.

Among the 200 universities, 176 of them have made their guidelines publicly available. From these guidelines, our review finds that nearly half of the universities provide insecure configurations, as is shown in Figure 4.9 (c). Similar errors or weaknesses have been found among the insecure guidelines. For Windows, iOS and MacOS, the user is instructed to skip the server and CA certificate validation when they are prompted with "*Continue* (to connect)". The insecure guidelines miss reminding the user to click "*Show the details*" and check 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*".

4.6.3 A Practical Study on Attack Feasibility

To study the feasibility of committing the evil twin attack in real-world enterprise networks, we have conducted a confined experiment in a well-known Internet technology company[8] with over 50k employees. This company has been proactively utilizing advanced security technologies to maintain a high-level enterprise network security.

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. The setup of the evil twin follows the system configuration detailed in Section 4.4.1. Considering that PEAP-MSCHAPv2 is implemented in this company, the evil twin has been configured to support only GTC in phase 2 to trigger the protocol downgrade, as specified in Section 4.4.3.2, to obtain the user's credentials in plaintext. The evil twin is configured to launch the attack once the employees have moved into the effective range of the evil twin. To confine the impact from this study, we limit our active experiment time to a total of continuous 40 minutes.

Our experiment manages to collect passwords from the mobile devices of 166 individual employees without being noticed by any of them. For "victims" using Android devices, their leakage is because they have chosen not to validate the CA certificate in their initial connection to the target network, prior to our experiment. Therefore, the credentials are automatically sent to the evil twin as elaborated in our Finding #1 (cf. Section 4.4.2). For "victims" using iOS, they are prompted with a request to trust the fake certificate when their devices are connected with the evil twin. Most of them choose to trust the certificate (and this confirms the survey in Section 4.6.1).

[8]Name omitted for double blind review.

After that, their credentials are automatically sent to the evil twin. The alarming number of passwords collected during the short period raises an alert on the severity of the vulnerabilities discussed in this paper.

Ethical Considerations. Prior to the experiment, we have obtained permission and clearance from the law department of the company. We harvest the employees' passwords only to serve the purpose of demonstrating the feasibility of the evil twin attack in WPA Enterprise, without further intruding the enterprise internal system or abusing the employees' privileges. We have responsibly erased all the obtained passwords and associated records after the completion of this study.

4.7 Mitigation

In this section, we discuss possible countermeasures to mitigate the security threats from evil twin attacks.

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) from both of their certificate validation UIs (e.g., Android Settings) and APIs (e.g., the `android.net.wifi.WifiManager`). Taking Google's patch for our reported CVE-2020-0201 as an example (Figure 4.10), the user should not be presented with the option `NONE` when choosing the phase-2 authentication protocol.

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 in `WifiConfiguration.java` of Android 10 and 11 (Figure 4.11) fixes our reported vulnerability A-150500247. It creates different alias for suggested and the previously saved Wi-Fi configurations to avoid the unauthorized manipulation on saved configurations.

WPA supplicants should also be installed with strategies for mandatory validation and displaying conspicuous warnings to users once an evil twin is detected. Their certificate validation UIs are recommended to follow the design in web browsers' Extended Validation (EV) certificate user

```
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 4.10: Patch for `WifiConfigController.java` against the Vulnerability of CVE-2020-0201

interface (UI) [179, 64], which displays a conspicuous padlock icon and actively blocks suspicious access points.

Developing Wi-Fi Configuration Application for Android and Windows 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 correct server certificate validation and secure choice of the EAP methods and phase-2 authentication protocols. In particular, When EAP-TTLS and PEAP protocols are used, MSCHAPv2 should always be chosen as the phase-2 authentication protocol.

Securing Wi-Fi Configuration Profiles for iOS and MacOS Devices. Since the Wi-Fi access settings in iOS/MacOS are configured through configuration profiles, the enterprise must ensure that server certificate validation is correctly specified and the EAP methods and phase-2 authentication protocols are securely chosen when generating such profiles. Similarly, if EAP-TTLS and PEAP protocols are implemented, the phase-2 authentication protocol should be set to MSCHAPv2 to provide better

```
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 4.11: Patch for `WifiConfiguration.java` against Vulnerabilities Listed in Android Vulnerability A-150500247

password secrecy.

4.8 Related Work

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 [185, 150]. Cassola et al. [44] presented a novel and practical attack against WPA Enterprise, leveraging the combination of active jamming, inadequacy of wireless user interface mechanisms and insecure trust model. Vanhoef et al. [186] 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. [187] conducted a systematic evaluation on the Dragonfly handshake protocol,

and found vulnerabilities that result in downgrade, denial-of-service 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. [22] investigated the prevalence of WPA2 Enterprise vulnerabilities arises from incorrect client devices. Brenza et al. [36] 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 downgrade 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 [184], to avoid this downgrade attack [185]. They also proposed countermeasures against key reinstallation attack [186], including disabling the reinstallation of already-in-use-keys and implementing single-use keys during handshakes. Several works [148] have proposed secure device pairing to prevent the evil twin attacks, using properties in or around the communicating devices to establish their identity.

4.9 Summary

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 three user-related studies. We have also recommended countermeasures as mitigation. 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.

Chapter 5

Intra-domain Fingerprinting Social Media Websites Through CDN Bursts

5.1 Introduction

Deploying encrypted channels has become a security principle for Internet communication. As reported by WatchGuard [204], more than 80% of the 100,000 top-visited websites have been under the protection of HTTPS. Despite providing sufficient confidentiality, the encrypted channels still inadvertently expose the metadata, such as endpoints' IP addresses, packet sizes and traffic direction, to the network attacker. These metadata have been fully or partially abused by fingerprinting attacks for identifying websites that a user has just visited [80, 41, 202, 78, 88, 91, 105, 133, 147, 203, 209]. The attacks aim to distinguish the web domains, and thus we refer to them as *inter-domain website fingerprinting* (or *inter-domain WSF*). The state-of-the-art inter-domain WSF has achieved a significantly high accuracy ($> 95\%$). Their capacity, however, may be restricted when fingerprinting pages out of the same social media website which contains millions of theme-wise similar (e.g., layout, style and HTML elements) but *content-wise* distinct pages.

Even though the contents shared on the social media pages are generally considered less privacy-sensitive, the accumulative behaviors of browsing

them encompass indicative information regarding a user, such as his/her political orientations and religious beliefs [69]. Concerns on such risks have been on the rise due to the increasing emphasis on censorship and surveillance worldwide nowadays. As shown by [101], the global Internet censorship is undergoing a shift from the website filtering to the content-wise surveillance, particularly amid the increasingly stringent scrutiny on social media websites after their involvement in misinformation spreading related to a series of terrorist attacks in New Zealand [192], Sri Lanka [181] and Myanmar [55].

In this work, we conduct a systematic study on fingerprinting the web pages within social media websites. We refer to it as *intra-domain web page fingerprinting* (or *intra-domain WPF*). Our study assumes an on-path passive attacker that eavesdrops on the network, e.g., a malicious ISP or an honest-but-curious gateway. The attacker merely utilizes the metadata of the traffic[1] without breaking the encryption, and strives to find features to differentiate web pages that are from the same web domain. To make our study realistic, we consider a slightly different threat model from those in the inter-domain WSF literature—the intra-domain WPF is not designated to be against the anonymity networks such as VPN or Onion Router (Tor) that incorporate a proxy or a collection of relays to hide IP addresses and routes (and also part of metadata such as the packet sizes in Tor). As is revealed by the latest statistics, only 7% of Internet users use VPNs frequently [74]; the Tor traffic (400 Gbit/s bandwidth [142]) accounts for merely 0.06% of the total global Internet traffic (77,800 GB/s actual traffic [85]). Hence, our applied threat model may retain sufficient fidelity for studies on the scale of nation-wide network or even Internet.

The intra-domain WPF is more challenging than it appears, although it may be considered as a trivial task due to the success of the inter-domain WSF and the availability of advanced learning techniques. In a typical social media website, almost all its web pages (such as Facebook user profile pages) are generated from the same template, such that similar traffic patterns

[1]The *traffic* in this paper refers to encrypted traffic unless stated otherwise.

are exhibited. Consequently, those previously distinguishable features that classification algorithms rely on are blurred out. Some "higher-definition" fingerprints capable of differentiating among such similar web pages are demanded. The used features should be highly individualized and robust in characterizing the subtle differences among pages.

To identify such fingerprints, we first examine the traffic of social media websites (detailed in Section 5.2.3). Characterized by their so-called *social-content-centric* feature, social media websites experience enormous traffic from the large-size contents such as photos and videos their users upload/-download. For example, in Facebook, 350 million photos are uploaded and 8 billion videos are viewed each day [12]. To facilitate efficient delivery of these large-size contents, social media websites tend to host them separately, typically with CDN, from the web servers which host only small objects like HTML and CSS files. We find that when a browser renders a social web page, the traffic between the browser and the CDN may exhibit patterns that can uniquely characterize the page.

Based on this finding, we propose a fingerprint named *CDN bursts*, by adapting the notion of the *traffic bursts* that are extensively used in the inter-domain WSF literature [118, 201]. A CDN burst is defined as an aggregation of several temporally adjacent network packets originated from a certain CDN server (identified by its unique IP address); a sequence of bursts further characterize a web page. From the perspective of the application layer, the personalized social media contents diversify the Document Object Model (DOM) trees among pages. Since the browser's rendering process in essence is to segment network stream into objects (e.g., images and videos), the DOM structure would then "shape" this semantics-aware segmentation, and implicitly influence how the browser fetches contents from CDN servers. As a result, the information conveyed by CDN bursts, such as the size of a retrieved content and the intervals of retrievals, may (partially) expose the DOM structure. This may be further strengthened by the dynamic or asynchronous loading features (e.g., Ajax) extensively used by social media sites.

To further demystify the CDN bursts, we conduct a series of in-depth

studies. We design an algorithm to extract CDN bursts incorporating both temporal and volumetric factors from the network traces, and to derive a feature set for classification. Being aware of the importance to estimate information leaked by the features, thanks to a recent work by Li et al. [105], we also quantify the mutual information between the web pages and each CDN burst, which measures the amount of information contained in and (possibly) leaked by the CDN burst. Our experimental evaluation demonstrates that this amount of information is sufficient for the intra-domain WPF to fingerprint social media pages. Finally, we propose countermeasures by obscuring the temporal and volumetric characteristics that the attacker utilizes to construct CDN bursts.

Contributions. Our study of fingerprinting the social media pages is the first systematic research of its kind. We show that intra-domain WPF, although previously considered as unachievable as finding a needle in a haystack [132], can achieve a performance at least as accurate as that of the state-of-the-art inter-domain WSF. This sheds light on an underresearched domain and may inspire more future research on the *general* intra-domain WPF. As a summary, the contributions of this paper are as follows.

- **A new type of web fingerprinting attack.** We propose the novel intra-domain WPF that aims to differentiate the web pages within the same social media website, assuming a realistic and representative attacker model.

- **A novel fingerprint and its countermeasures.** We propose CDN bursts as a new fingerprint to characterize social media pages. Through the formalization of CDN bursts and quantification of information leakage, we demonstrate their feasibility and effectiveness in intra-domain fingerprinting. We also propose and evaluate the possible countermeasures.

- **A comprehensive evaluation and practical results.** We analyze the performance of the proposed intra-domain WPF attack on four top-visited social media websites, including Facebook, YouTube, Twitter

Figure 5.1: The intra-domain WPF attacker model

and Instagram. We follow the data collection approaches used by website fingerprinting literature [201, 132] to acquire a representative dataset, which consists of over 424,000 traces from over 12,000 unique web pages. Our dataset is released to facilitate future studies [86].

5.2 The Intra-domain WPF Attack

In this section, we study the CDN traffic and demonstrate that CDN bursts contain distinguishable features for classifying the traffic of social media pages.

5.2.1 Problem Statement and Attacker Model

The intra-domain WPF is a type of traffic analysis attack. Considering such a scenario: a social media website hosts a set of publicly accessible web pages $\mathbb{W}$, and the victim user may visit any page in a set $\mathbb{V}$ ($\mathbb{V} \subset \mathbb{W}$). The attacker owns a classifier $\mathcal{C}$ trained from a set of pages $\mathbb{A}$ ($\mathbb{A} \subset \mathbb{W}$). Given the traffic τ generated during loading a page $\omega \in \mathbb{V}$, the goal of the attack is to maximize the probability $\Pr(\mathcal{C}(\tau) = \iota_\omega)$, where ι_ω is the label[2] of ω.

Figure 5.1 shows an overview of the intra-domain WPF attacker model. Similar to prior fingerprinting studies, we assume a *passive* on-path attacker on the network or transport layer. The attacker can passively collect but not intercept, modify, delay or decrypt network packets. In reality, the possible attackers could range from compromised network access points, proxies and routers, local network administrators, Internet Service Providers (ISP), to Autonomous Systems (AS) between the victim user and the web servers.

[2]We treat each page as a class, so the label means its class label. For simplicity, we abuse ω as the label of itself in the remaining of this paper. An $\omega \notin \mathbb{A}$ is labeled as $\perp$.

The attacker model in the intra-domain WPF is slightly different from that in the inter-domain WSF. In the latter, the web domains that a victim has visited are considered sensitive, while in the intra-domain WPF, the attacker is more concerned with the content-centric privacy. In particular, the attacker's general interest is whether the target victim has visited a particular *page* (which includes a politician's tweets, for example), while the visited web domain (e.g., twitter.com) is out of interest. Therefore, the intra-domain WPF is not aimed at VPN and Onion Router (Tor) which are commonly used for hiding the specific services (i.e., domains). In addition, our attacker model also excludes the web attacker who can remotely execute code on the client side, e.g., JavaScript code injected into a benign web page.

In website fingerprinting studies, it is common to evaluate the attacks against both *closed-world* and *open-world* models [201, 40]. We define them using the relation between $\mathbb{A}$ and $\mathbb{V}$. The case of $\mathbb{A} = \mathbb{V}$ corresponds to the closed-world model, and the case of $(\mathbb{A} \neq \mathbb{V}) \wedge (\mathbb{A} \cap \mathbb{V} \neq \emptyset)$ to the open-world model. These two models in essence represent the extent to which the attacker is able to train his classifier on the pages that are likely to be visited by the victim. The ideal scenario for the attacker is to know all pages the victim may visit, i.e., the closed-world model. This has been regarded unrealistic [133, 147] though. Nevertheless, the closed-world model is useful to benchmark classification and fingerprinting approaches. We thus keep both models in our evaluation.

5.2.2 CDN Bursts as Classification Features

Extracting a set of consistently informative features from the network traffic is the utmost important factor for a successful classification. In general, the size, temporal distribution and source/destination are the major characteristics of the packets. Therefore, we generalize a traffic generated while loading a social media page w as a sequence $T(w)$: $T(w) = \langle (p_1, t_1, ip_1), (p_2, t_2, ip_2), \ldots, (p_N, t_N, ip_N) \rangle$, where (p_i, t_i, ip_i) is a packet originated from ip_i with timestamp t_i, and size p_i ($p_i > 0$). We call

$T(w)$ a *loading trace* of web page w.

In the CDN traffic, the incoming packets are more informative than the outgoing ones (in terms of size variation, source, etc.). We therefore further reorganize the loading trace $T(w)$ to group the incoming CDN packets from ip into $T_{ip}^{in}(w)$:

$$T_{ip}^{in}(w) = \langle (p_1^{in}, t_1^{in}, ip), \ldots, (p_M^{in}, t_M^{in}, ip) \rangle,$$

where $M \leq N$. Hereafter, we use *trace*s to indicate incoming traces.

Given a CDN trace $T_{ip}^{in}(w)$, a CDN burst is a segment of the trace containing a sequence of consecutive and temporally adjacent packets. The k-th CDN burst is defined as B_k^{in}:

$$B_k^{in} = \langle (p_j^{in}, t_j^{in}, ip), (p_{j+1}^{in}, t_{j+1}^{in}, ip), \ldots , (p_q^{in}, t_q^{in}, ip) \rangle,$$

where $0 < j \leq q \leq M$; $p_j^{in}, p_{j+1}^{in}, \ldots, p_q^{in}$ are consecutive incoming packets; $t_{l+1} - t_l < \delta$ for all $l \in [j, q)$. The threshold δ which we name as *burst threshold* is a parameter whose optimal value can be fine-tuned (discussed in Section 5.5.3).

Based on the definitions above, we summarize the CDN burst feature engineering as Algorithm 1. It takes as input a CDN trace $T_{ip}^{in}(w)$ originated from IP address ip when loading page w, and outputs the CDN burst feature set. Intuitively, it first aggregates packets into bursts and then calculates the burst sizes which are the final features written into the feature set.

The rationale for proposing CDN bursts is that the social media pages may be embedded with large-size objects (e.g., images and videos) which are typically hosted by CDN servers. Objects with various sizes may appear in different places among pages (based on the order that page owners upload them to the "timelines", for example), such that the DOM structures of these pages may vary. This results in significant variation in the browser's page downloading behaviors. When an object is being downloaded, it is split into multiple packets for transmission, and these packets are likely to appear in a batch. We observe that the time interval between such two consecutive packets is much shorter (0.1%) than that between the HTTPS requests to fetch two adjacent objects (~$0.01ms$ v.s. ~$10ms$), because it takes time

Algorithm 1: CDN Burst Based Feature Engineering

Input: $T_{ip}^{in}(w)$
Output: Feature Set $\mathbb{F}$
Function ***FeatureEngineering*** $(T_{ip}^{in}(w))$
    $\mathbb{F} = \emptyset$, $k = 1$, $B_1^{in} = \langle\rangle$
    for i *in* $range(1, M)$ **do**
        $B_k^{in}.append((p_i, t_i, ip))$
        if $t_{i+1}^{in} - t_i^{in} < \delta$ **then**
            $B_k^{in}.append((p_{i+1}, t_{i+1}, ip))$
        else
            $\mathbb{F}.append(total\ size\ of\ all\ packets\ in\ B_k^{in})$
            $k \leftarrow k + 1$
    return $\mathbb{F}$

for the browser to parse and render the page before sending out more HTTPS requests. In this situation, the CDN bursts become distinguishable classification features since a burst is essentially a "reconstruction" of an application-level object.

The burst threshold δ is a subtle factor. In an ideal scenario where the browser sends sequential requests for the objects, any δ between $0.01ms$ and $10ms$ would enable the "object reconstruction". However, considering that modern browsers send requests in parallel (e.g., the web worker in HTML5), it may be more robust to reconstruct a batch of objects into a burst. We take this strategy in our construction, and detail the selection of δ in Section 5.5.3.

5.2.3 CDN Bursts in Real-world Scenarios

To explore the feasibility of the intra-domain WPF using CDN bursts, we conduct an empirical study on two of the most followed Instagram profile pages, National Geographic (ranked 14^{th} most followed, denoted as `Ins_page_1`) and Katy Perry (ranked 20^{th} most followed, denoted as `Ins_page_2`). We simulate a scenario shown in Figure 5.2 where the victim user is browsing two pages `Ins_page_1` and `Ins_page_2`. The theme and

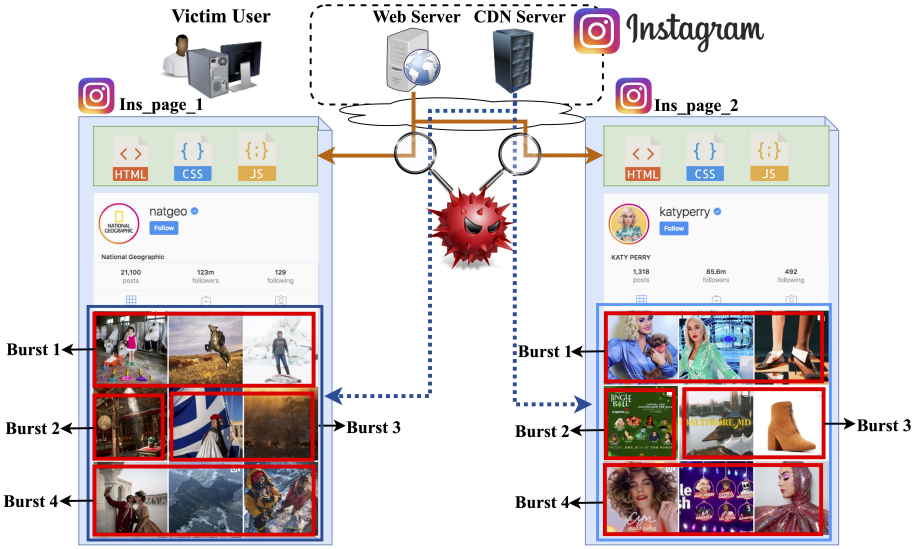


Figure 5.2: A running example

script files which include the HTML, CSS and JavaScripts are fetched from the web server, and the large-size image/video contents are stored by the CDN sever (with IP address 157.240.25.63). We visit each Instagram profile for 50 times and record the generated network traffic.

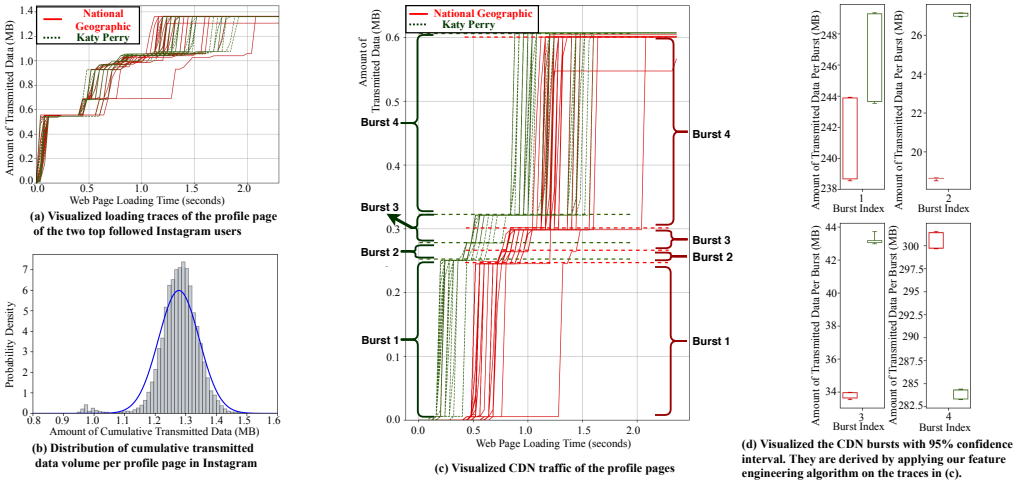


Figure 5.3: Visualization of network traffic, CDN traffic and CDN bursts of two example Instagram profile pages

First, we visualize the traces derived from the traffic in Figure 5.3 (a), without applying our feature engineering algorithm. We plot the cumulative incoming data size versus time to reflect the *temporal and volumetric features*

commonly used by website fingerprinting techniques[41, 202, 201]. As is shown, the traces are almost indistinguishable, and significant parts are even intertwined. This situation, in practice, would be further worsened by the network fluctuations.

We also assess the effectiveness of the *cumulative volumetric features*, which are another extensively used type of features [132, 78]. We investigate the distribution of the cumulative transmitted data volume of 30,000 collected Instagram profile pages included in one of our datasets (further discussed in Section 5.3.2), and find that 95% of those pages have the cumulative transmitted data volume in range between 1.1 MB to 1.4 MB, as shown in Figure 5.3 (b). This suggests that the classification algorithms which heavily rely on the cumulative volumetric features might also be ineffective. Besides, considering the traces are from the same web server, other traditional statistical features [78, 201] which are stemmed from the variances among inter-domain web servers, such as maximum packet size and packet count per second, are likely to fail.

We further break down the traffic based on its origins. We extract the CDN traffic out of the overall network traffic (by filtering packets from IP address 157.240.25.63) and plot it in Figure 5.3 (c). As is shown, the CDN traffic exhibits consistent and distinctive differences in burst sizes between the two pages, despite indistinguishable cumulative CDN traffic sizes ($\sim$0.6 MB for both pages). We then apply our feature engineering on all CDN traces of these two pages. For the purpose of intuitive visualization and comparison, we plot the value ranges of the derived feature set in Figure 5.3 (d). As can be seen, each burst size varies within a relatively narrow band, suggesting the features are robust and consistent. There is almost no overlapping region between bursts (except a marginal one for burst 1), indicating that the CDN bursts are distinguishable representations, or *fingerprints*, of their traces.

5.3 Data Collection

The prerequisite for machine learning algorithms is a dataset with abundant representative data. However, since our work is the first one conducting the intra-domain WPF, there is not dataset that could be directly used. Therefore, we resort to build a new setup for data collection. During the process of data collection (presented soon in Section 5.3.1), we take account of attacker's capabilities (defined in Section 5.2.1) to retain representativeness. We assume the attacker can use a network sniffing tool to record metadata of the network-layer packets, including the size of the packets, the timestamp of sending/receving a packet, the addresses (port and IP), etc. Our collected datasets have been made available online [86] after anonymization to facilitate future research.

5.3.1 Setup

Without losing generality, the machine configurations and locations are intentionally diversified in our data collection process. We distribute our setup along the campuses of National University of Singapore and the University of Queensland (denoted by A and B respectively for simplicity), located in two continents. On Campus A, we utilize 4 *OpenStack* [130] Ubuntu 16.04 virtual machines hosted by a server with 20 CPUs and 50GB of RAM in our department, and 2 Windows 10 desktops with 8 CPU Cores and 16GB of RAM located in a research lab. On Campus B, we utilize 3 Ubuntu 16.04 virtual machines with 1 CPU and 2 GB of RAM located in an office. For loading the web pages, we use Firefox version 66.0.3 on Linux machines and Chrome version 70.0.3538.77 on Windows machines.

5.3.2 Data Collection Methodology

Considering that social media pages contain many dynamically-loaded contents, our data collection process has to simulate the actual browsing activities of the users, rather than straightforwardly sending HTTP requests without rendering pages. To this end, we use a web testing automation tool *Selenium* [153] to automatically generate page visits. Similar to other

fingerprinting studies ([202, 78, 157]), the browser caches are disabled so that the captured traces are complete; the page visits are in the sequential manner (i.e., *single-tab*) so that the collected traces are not interweaved. We acknowledge these two settings may give the attacker advantage, but in reality, the caches of the main-stream browsers are usually too small ($<$ 80 MB) to host many large-size objects, and the dwell time on a social media page is relatively long that interweaved traces could be rarely generated.

We use the *tcpdump* [174] to record the network traffic. We capture the HAR (HTTP Archive) files to record the browser activities during web page loading. To automatically log the HAR files, we invoke browsers' built-in developer modes by setting devtools.netmonitor.har.enableAutoExportToFile in Firefox and using the *automated chrome profiling* [136] library in Chrome. Note that the HAR files are only required in the training process to identify the IP addresses of the CDN servers (detailed in Section 5.3.4), but not required when conducting the actual attack.

We select the top four social media websites from the Alexa Top Sites for our evaluation, including Facebook, YouTube, Twitter and Instagram. For each site, up to 30,000 (to be detailed soon) most liked or followed user profiles listed on Trackalytics [182] are collected. On each profile page, the collection process has to let the browsers stay sufficiently long for the page to be fully loaded. This time is set to be 14 seconds since the average web page loading time in most countries is around 8 to 10 seconds [32]. Between two pages, a break of at least 8 seconds is set to comply with the privacy policy of the websites and to avoid being blocked.

Closed-world Datasets. Datasets $\mathbb{D}_1$ and $\mathbb{D}_2$ are collected from Campus A and Campus B respectively. Dataset $\mathbb{D}_1$ is collected within two weeks. It contains the traces of the top 1,000 most liked/followed user profiles pages from each website. Due to its diversity (in terms of machines, software stacks and time span), we mainly use $\mathbb{D}_1$ to evaluate the performance of the intra-domain WPF in the closed-world model. Dataset $\mathbb{D}_2$ contains the traces of the top 300 pages from each site. It is collected within a relatively stable environment and within a relatively short time (mostly 2 days, but 1 week for Facebook due to its visit frequency constraint) so that it is "clean"

for benchmarking our work. In both datasets, each page is recorded for 55 traces.

Open-world Datasets. For evaluation against open-world model, we have collected the dataset $\mathbb{D}_3$ from Campus A. $\mathbb{D}_3$ includes the traces of 300 *monitored* pages (denoted by $\mathbb{M}$) and up to 30,000 *unmonitored* pages (denoted by $\mathbb{U}$) for each site. Each page in $\mathbb{M}$ is recorded for 55 traces. For each page in $\mathbb{U}$, three traces are recorded for redundancy. We cross validate these three traces by their sizes to remove the faulty ones, and keep only one trace for training and testing (detailed in Section 5.3.4).

5.3.3 Ethical Considerations

Our data collection generates overall 424K visits to the four social media websites. These visits are distributed over four weeks, and much fewer than their billions of daily visits. The impact to their servers is thus negligible. We only store the IP packets, in which all application-layer data are encrypted. Before releasing the datasets to facilitate future research, we anonymize any field that may reveal identities, including IP addresses, port numbers, labels, etc.

5.3.4 Data Pre-processing

Outlier Removal. Given the collected datasets, we first remove the faulty traces where no packets or only partial packets are received. To further eliminate the possible noisy training data, we use a *interquartile range* (IQR) based approach [102] to remove the outliers, a standard way to detect noise used by literature [132]. Intuitively, the traces whose total transmitted data sizes deviate significantly from median value are considered as outliers and therefore are removed. For pages in $\mathbb{D}_1$, $\mathbb{D}_2$ and the 300 monitored pages in $\mathbb{D}_3$, we calculate the total transmitted data size for each trace in the training dataset, and eliminate those whose total size is out of the interquartile range $[Q_1 - 1.5IQR, Q_3 + 1.5IQR]$, where Q_1 and Q_3 represent first and third quartiles, and $IQR = Q_3 - Q_1$. In this way, an average of 5% traces in the training dataset are removed. Note that outlier removal is

not applied on the testing traces. This ensures an unbiased evaluation of the classifier, and simulates a realistic scenario where the attacker has little control over the testing data.

In the remaining of this paper, when we refer to the three datasets, we actually mean the datasets after pre-processing.

CDN Traffic Identification. In reality, a CDN URL may be mapped to a list of IP addresses for load balancing. Therefore, we conservatively assume the attacker has to construct such a map before starting the attack. Because the number of IP addresses is quite small (at most 14 according to our study), it is feasible for the attacker to obtain a full list by gathering the URLs from profile pages and querying DNS servers multiple times prior to the attack. To simulate this, we query through parameters such as *requestId* and *ResourceReceiveResponse* in the collected HAR files, and find the URLs corresponding to image and video files. Then we resolve them into IP addresses by querying public DNS servers including Google DNS, OpenDNS Home, Alternate DNS and CleanBrowsing.

5.4 Learning Algorithm and Feature Informativeness

In this section, we discuss the web fingerprinting as a classification problem in the context of intra-domain WPF. We also examine the informativeness of our classification features derived from CDN bursts, inspired by [105] which urges for the quantification of the information leakage from the classification features.

5.4.1 Technique and Evaluation Metrics

Classification Technique. In our datasets, each web page in the monitored set (i.e., the $\mathbb{A}$ in closed world and the $\mathbb{M}$ in open world) is a class, and thus $|\mathbb{A}|$ or $|\mathbb{M}|+1$ classes are included (one extra indicates the unmonitored class). To suit the multi-class classification, we select the random forests in our work. Random forests are a classification technique [34] based on the votes made by the majority of decision tree ensemble included in

the forest. Random forests have been employed to classify the encrypted network traffic by recent studies on website fingerprinting [209, 78]. They require shorter training time and fewer training instances than traditional machine learning (such as SVM and KNN) [132, 202, 201, 110] and deep learning algorithms (such as autoencoder or CNN) [105, 126, 157, 104]. For example, fewer than 100 labeled training data for each class is sufficient for random forests, while CNN may require at least 500 each [208] such that it demands much longer collection period than the former. The efficiency of random forests is crucial for fingerprinting social media pages whose contents change so frequently that timely updating of the datasets and retraining of the classifiers are necessary. In addition, although random forests rely on manually crafted feature sets, the analysts would be able to interpret the features better (e.g., to analyze the feature-based information leakage and their relative importance) compared to those derived from deep neural networks.

Each tree in the random forests is trained from the fixed-sized labeled feature sets. Despite the bootstrap sampling process used for tree construction, cross validation is still necessary in an unbiased evaluation for our classifier. Cross validation is also important to determine a robust burst threshold δ. Therefore, we apply a 10-fold stratified cross validation. The training part (9 folds) is used to generate a classifier from the "fingerprint" of each monitored page. The testing part (1 fold) is used to validate whether a trace matches any of the fingerprints. We pad/truncate the feature vector to a fixed length of 50 which is larger than most vectors' dimensions.

Classification Performance Measurement. To determine the classification performance, we adopt the following four metrics[3] commonly used in the web fingerprinting literature [78, 152].

- **Accuracy** ($accuracy$, $\frac{TP+TN}{total}$) is the ratio of the correctly classified web pages to the total number of input web pages.

[3]To facilitate the metrics definitions, we use the following denotations: TP (true positive), FP (false positive), TN (true negative) and FN (false negative).

- **True positive rate** (TPR, $\frac{TP}{TP+FN}$) is the probability of correctly classifying the web pages.

- **False positive rate** (FPR, $\frac{FP}{FP+TN}$) is the probability of incorrectly classifying the web pages.

- **Bayesian detection rate (BDR)** is the probability that a web page is correctly identified by the classifier as a labelled monitored web page. For simplicity, we assume the visiting of web pages by users follow the uniform distribution. Using Axelsson's definition [15], we formulate BDR as

$$BDR = \frac{TPR \cdot f_m}{TPR \cdot f_m + FPR \cdot f_u},$$

  where f_m is the fraction of traces from monitored pages in the total web traces, and f_u is that for unmonitored pages.

Among the four metrics, *accuracy* is used for evaluating the classification performance in the closed-world setting, since the class sizes (a class size refers to the number of traces for each page) are balanced in $\mathbb{D}_1$ and $\mathbb{D}_2$ (i.e., each data class takes a fixed proportion of the total sampled data). In contrast, TPR, FPR and BDR are used in the open-world setting since the sizes of the monitored and unmonitored classes are significantly imbalanced. TPR and FPR check whether the correct classification rate and false alarm rate for monitored pages are balanced, while BDR measures the feasibility of the attack in reality by taking into consideration the fraction of the monitored pages in the sampled pages. The reason we use BDR as the evaluation metric in open-world setting is because a low FPR does not necessarily imply a high probability of successful attacks when the fraction of the monitored web pages are extremely low, also known as the *base rate fallacy paradox* [21].

5.4.2 Feature Information Leakage Measurement

Fundamentally, classifiers can distinguish among classes since the classification features contain some amount of information. These features, on the

other hand, can leak information to the attacker. To quantify such leakage and also to figure out the feature importance, we formalize the information contained in each feature as the *mutual information* in a similar way as presented by Li et al. [105]. It quantifies the amount of information about a random variable (i.e., the identity of a web page) derived from another variable (i.e., the features extracted from the network traffic).

Definition 8. The amount of information about a web page contained in the fingerprint of the trace is $I(F; W)$:

$$I(F; W) = H(W) - H(W|F), \tag{5.1}$$

where F is a random variable corresponding to a single or a set of features in the fingerprint of a trace; W is a random variable about the web page identity; $I(F; W)$ denotes the mutual information between variables F and W; $H(W)$ denotes the information entropy of variable W; $H(W|F)$ denotes the conditional entropy of the variable W given the variable F.

Below we calculate $H(W)$ and $H(W|F)$. Assuming that the page the user is browsing is $w \in \mathbb{V}$, $H(W)$ can be calculated as follows.

$$\begin{aligned} H(W) &= -\sum_{w \in \mathbb{V}} Pr(w) \log_2 Pr(w) \\ &= -\frac{C|\mathbb{V}|}{C|\mathbb{A}| + |\mathbb{V} - \mathbb{A}|} \log_2 \frac{C}{C|\mathbb{A}| + |\mathbb{V} - \mathbb{A}|} \end{aligned} \tag{5.2}$$

where C is the number of traces for each monitored page; $Pr(w)$ is the probability that the identified page is w.

Given a variable F_j representing the j-th feature whose value range is denoted by $\mathbb{F}_j$, the conditional entropy of the variable W, i.e., $H(W|F_j)$ is, as follows.

$$H(W|F_j) = -\sum_{f_j \in \mathbb{F}} Pr(f_j) \sum_{w \in \mathbb{V}} Pr(w|f_j) \log_2 Pr(w|f_j), \tag{5.3}$$

where $Pr(f_j)$ is the probability that feature F_j has a value f_j, and $Pr(w|f_j)$ is the probability that the identified page is w given that the value of feature F_j is f_j.

According to Equation 5.3, the key for calculating the final information leakage is to derive the probability density function (PDF) of feature F_j and the conditional probability $Pr(w|f_j)$. We estimate the former using a histogram-based approach commonly used for entropy estimation [134]. In particular, we partition the range $\mathbb{F}_j$ of feature F_j into n equal intervals (l_j^i, u_j^i) where $i \in \{1, 2, ..., n\}$. $Pr(f_j)$ is represented by the fraction of traces whose feature F_j evaluates to $f_j \in (l_j^i, u_j^i)$, against the total number of traces in the whole dataset. $Pr(w|f_j)$ is represented by the fraction of traces whose labels are w and feature F_j evaluates to $f_j \in (l_j^i, u_j^i)$, against the total number of traces whose feature F_j evaluates to $f_j \in (l_j^i, u_j^i)$ in the whole dataset. Therefore,we have

$$Pr(f_j) = \frac{K_j^w}{C|\mathbb{V}|} \text{ and } \Pr(w|f_j) = \frac{N_j^w}{K_j^w}, \tag{5.4}$$

where K_j^w is the number of traces in $\mathbb{D}_3$ whose feature F_j evaluates to $f_j \in (l_j^i, u_j^i)$; N_j^w is the number of traces that are from web page w and feature F_j evaluates to $f_j \in (l_j^i, u_j^i)$.

All in all, the information leakage of the j-th feature can be quantified as follows.

$$\begin{aligned} I(F_j; W) =& H(W) - H(W|F_j) \\ =& \sum_{f_i \in \mathbb{F}} \frac{K_j^w}{C|\mathbb{V}|} \sum_{w_i \in \mathbb{V}} \frac{N_j^w}{K_j^w} \log_2 \frac{N_j^w}{K_j^w} \\ & - \frac{C|\mathbb{V}|}{C|\mathbb{A}| + |\mathbb{V} - \mathbb{A}|} \log_2 \frac{C}{C|\mathbb{A}| + |\mathbb{V} - \mathbb{A}|} \end{aligned} \tag{5.5}$$

This is used to quantify information leakage by CDN bursts in Section 5.5.1.

5.5 Evaluation

Our evaluation focuses on the performance of the intra-domain WPF and the effectiveness of CDN bursts as classification features. We explore the three main research questions.

- **RQ1.** Can CDN bursts be used as a distinguishable "fingerprint" for the intra-domain WPF?

- **RQ2.** Is intra-domain WPF feasible in reality (i.e., the open-world scenario) and to what extend can it scale?
- **RQ3.** What factors may affect the performance of CDN bursts in the intra-domain WPF?

5.5.1 RQ1: Intra-domain WPF in Closed-world

Experiment Setup. To answer RQ1, we choose accuracy (cf. Section 5.4.1) as the evaluation metric of the intra-domain WPF in the closed-world model. In the literature, the closed-world model is extensively used as a baseline to benchmark the fingerprinting algorithm [147, 78, 40, 132, 201]. To avoid the influence on accuracy due to the unbalanced class distribution in the datasets, we use our datasets $\mathbb{D}_1$ and $\mathbb{D}_2$ in the experiments. We apply 10-fold stratified cross validation on the datasets. According to our study on the burst threshold δ, which is detailed soon in Section 5.5.3, we set δ to be 0.05s when we apply our feature engineering algorithm (Algorithm 1) in all experiments.

Performance of Intra-domain WPF with Varying Number of Pages. We use varying numbers of web pages (i.e., $|\mathbb{A}|$ or $|\mathbb{V}|$ as in the closed-world model, $\mathbb{A} = \mathbb{V}$) ranging from 200 to 1,000 from $\mathbb{D}_1$ for the evaluation. As shown in Figure 5.4, we achieve accuracy in the range of 0.92 to 0.99. It is also noticeable that the accuracy has a minor drop for Twitter. This may be because in Twitter, the CSS and JavaScript files are also located in the same CDN servers as images and videos, which may slightly disturb the CDN bursts.

Performance of Inter-domain Features for Intra-domain WPF. As the inter-domain WSF has been well researched, we also explore the applicability of its techniques to the domain of intra-domain WPF. We apply two state-of-the-art inter-domain WSF approaches, k-fingerprinting (denoted by k-fp) proposed by Hayes et al. [78] and CUMUL proposed by Panchenko et al. [132], to our datasets. K-fp is based on random forests and CUMUL is on SVM. For k-fp, we adapt our data format for it and use its released code in our experiment; for CUMUL, due to incompatibility

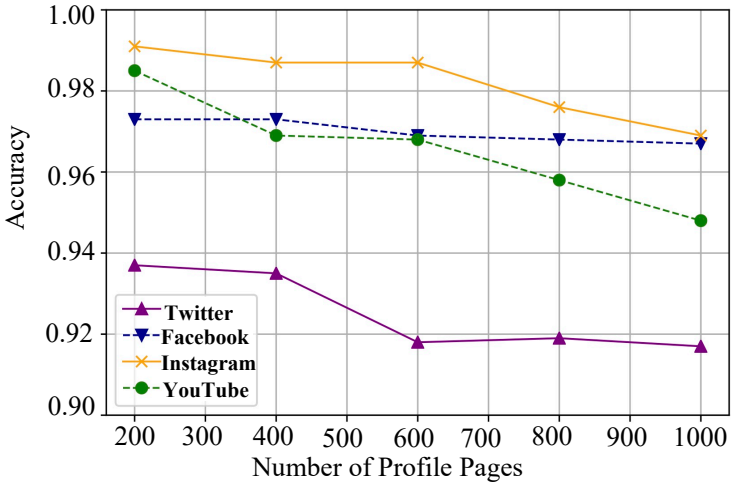


Figure 5.4: Intra-domain WPF with varying number of pages

Table 5.1: Comparison of fingerprinting approaches on $\mathbb{D}_2$

Attacks	**Accuracy**			
	Instagram	Facebook	Twitter	YouTube
k-fp	0.181	0.833	0.340	0.367
CUMUL	0.829	0.964	0.909	0.894
WPF	0.953	0.932	0.903	0.948

issues, we re-implement it by closely following the same feature extraction and classification techniques detailed in its paper. To avoid bias, we use $\mathbb{D}_2$ in our experiments due to its cleanness (cf. Section 5.3.2), and the results are listed in Table 5.1. We also measured the computation time (i.e., model training time, including cross validation and excluding feature engineering) and memory consumption, as listed in Table 5.2 where the data is collected on a Surface Pro 6 with Intel i5 4-Core CPU@1.6GHz and 8G RAM.

Although it may be unfair to directly compare approaches targeting different attacker models, our approach generally achieves higher accuracy and lower overhead (shorter training time and less memory consumption) than the other two techniques. Our approach slightly outperforms CUMUL in general, but requires much shorter training time and less resource. There are modest improvements in accuracy, ranging from 5 to 10 percentage points,

Table 5.2: Time and memory consumption

Attacks	**Training Time (min) / Memory (GByte)**			
	Instagram	Facebook	Twitter	YouTube
k-fp	6.2 / 5.5	5.0 / 5.4	5.0 / 5.6	4.5 / 4.7
CUMUL	> 3.5 hours / ~5.0 in each site			
WPF	1.5 / 3.7	3.0 / 4.9	2.5 / 4.7	2.0 / 4.1

compared with CUMUL in Instagram and YouTube. This improvement is likely because CDN bursts capture the subtle differences of the objects better than the cumulative features that CUMUL relies on. Our approach marginally underperforms CUMUL in accuracy by 0.1 to 3 percentage points in Facebook and Twitter, plausibly due to the fact that content personalization results in greater variations in the cumulative sizes of pages, which could be better captured by inter-domain fingerprinting techniques like CUMUL. The accuracy of k-fp is relatively low since the features used by it mostly focus on the packet counts rather than the packet lengths. For intra-domain web pages, the packet count features have lower distinguishing capacity since web pages generated from the same template and sourced from the same web server would exhibit similar packet-wise patterns.

Apart from the existing inter-domain WSF approaches, we also explore the capacity of deep learning in the intra-domain WPF using a larger dataset we have collected. For each website, we select the most liked or followed 100 pages and each page is recorded with 500 traces. We conduct our experiments with a recent deep learning based WSF approach named DF, which is proposed by Sirinam et al [157]. We have observed that the size of the training samples has a significant impact on the classification performance. In particular, when DF is evaluated using the same setting as [157], its accuracy (ranging from 0.5 to 0.85) is significantly lower than our approach (> 0.9); when evaluated using a larger dataset (>300 traces per page), its accuracy becomes comparable with ours. This implies that our approach may be more realistic in practice than the deep learning based

Table 5.3: Information leakage by the features of CDN bursts (bit) measured in the closed-world model

Burst Index	1	2	3	4	5	6	7	8
Info Leak	3.87	3.22	2.64	2.08	1.83	1.71	1.51	0.52

counterparts which require extended data collection period and higher computation capacity.

Information Leakage by CDN Bursts. We use dataset $\mathbb{D}_2$ to quantitatively evaluate the information included in CDN bursts which is utilized in classification (cf. Equation 5.5). We measure the average information leakage for both individual CDN bursts and the cumulative CDN burst. The results of the most informative 8 bursts are shown in Table 5.3. The CDN bursts are indexed according to their sizes. Larger bursts are observed to leak more information (up to 3.87 bits) than the smaller ones (down to 0.5 bit).

We note that the information leakage is calculated on the basis of each feature as an indication of its effectiveness. There may be correlation but not causation between the information leakage of individual features and the final classification accuracy since the classifier also utilizes additional information pertaining to groups of features. This type of information leakage will be investigated as our future work.

Intra-domain WPF Using Web-server Bursts. We also apply our feature engineering algorithm to web server traffic to explore whether the web-server bursts can be used as fingerprints. We measure the classification accuracy for Instagram traces in our dataset $\mathbb{D}_1$ as an illustration, as shown in Figure 5.5. It is observed that the accuracy using web-server bursts is significantly lower than that using CDN bursts. On the other hand, the accuracy is not extremely low due to the marginal differences among web pages. However, this difference becomes trivial as the number of pages increases, and thus the accuracy declines at a faster pace (23.3% from 200 pages to 1,000 pages) than that based on CDN bursts (2.8% from 200 pages to 1,000 pages). This indicates the web-server bursts may not be robust features.

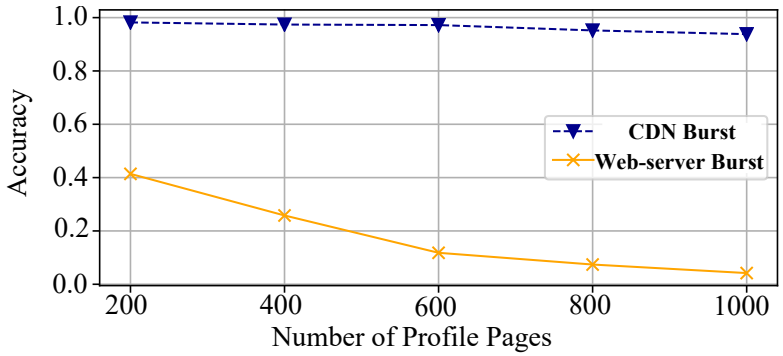


Figure 5.5: Performance comparison of web-server bursts and CDN bursts for intra-domain WPF in Instagram pages.

5.5.2 RQ2: Intra-domain WPF in Open-world

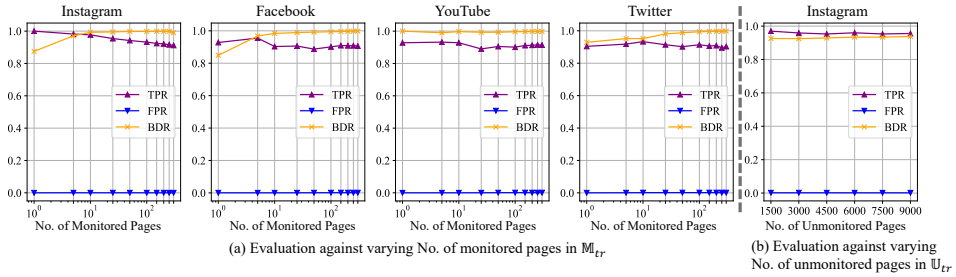


Figure 5.6: Performance of intra-domain WPF in the open-world model

Experiment Setup. To answer RQ2, we conduct the intra-domain WPF in the more realistic open-world model. Due to significant size imbalance in the monitored (i.e., $\mathbb{M}$) and unmonitored set (i.e., $\mathbb{U}$), we use TPR, FPR and BDR as our evaluation metrics. TPR and FPR determine whether the detection rate and false alarm rate are balanced, i.e., the classification *correctness*. For *feasibility* evaluation, we use the metric BDR, which takes into consideration not only TPR and FPR, but also the ratio of monitored and unmonitored set size, such that the rates on the unmonitored set (much larger than monitored set) would not dominate the metrics.

We use the open-world dataset $\mathbb{D}_3$ in our experiments. Recall that $\mathbb{D}_3$ includes 55 traces/page $\times$ 300 monitored pages (i.e., $\mathbb{M}$), and one trace/page $\times$ up to 30,000 unmonitored pages (i.e., $\mathbb{U}$) for each of the four sites. Two variables are investigated for their influence on the performance of the

Table 5.4: Training and Testing Datasets

Training Set	9 folds of the traces in $\mathbb{M}_{tr}$ and a set of randomly selected traces $\mathbb{U}_{tr} \subset \mathbb{U}$
Testing Set	The remaining 1 fold the traces in $\mathbb{M}_{tr}$ and the remaining traces in the set $(\mathbb{U} - \mathbb{U}_{tr})$

intra-domain WPF using 10-fold stratified cross validation. For each round, we construct the training and testing datasets as is shown in Table 5.4. We first select a subset $\mathbb{M}_{tr}$ from $\mathbb{M}$, and use nine folds of the traces from each page for training and the remaining one fold for testing. Then, we select a subset $\mathbb{U}_{tr}$ from $\mathbb{U}$ for training and the remaining traces for testing. In our setting, $\mathbb{M}_{tr}$ is to simulate attacker's target set; $\mathbb{U}_{tr}$ is to simulate that the attacker includes extra web pages outside $\mathbb{M}$ into his training dataset, indicating the attacker's capability of profiling some web pages outside his targeted page set; $\mathbb{U}_{te}$ is to simulate the background noise when the user is browsing the intra-domain web pages but outside the attacker's training dataset. We analyze the feasibility of intra-domain WPF by varying the size of $\mathbb{M}_{tr}$ and $\mathbb{U}_{tr}$ separately in this section.

Performance of Intra-domain WPF with Varying Number of Monitored Pages in Training Set. In this experiment, the size of $\mathbb{M}_{tr}$ varies while we maintain 1,000 pages in $\mathbb{U}_{tr}$ and 6,000 pages in $\mathbb{U}_{te}$. We let the size of $\mathbb{M}_{tr}$ vary from 1 to 300, and determine the TPR, FPR and BDR. Our results are in Figure 5.6 (a). It is observed that intra-domain WPF achieves a stable performance against an increasing number of monitored pages in the training set. TPR exhibits a slight downward trend which stabilizes at around 0.9, while FPR shows a slight upward trend which stabilizes at around 0.001. The BDR quickly converges to nearly 1.0 at 50 monitored pages from around 0.9 at 1 monitored page.

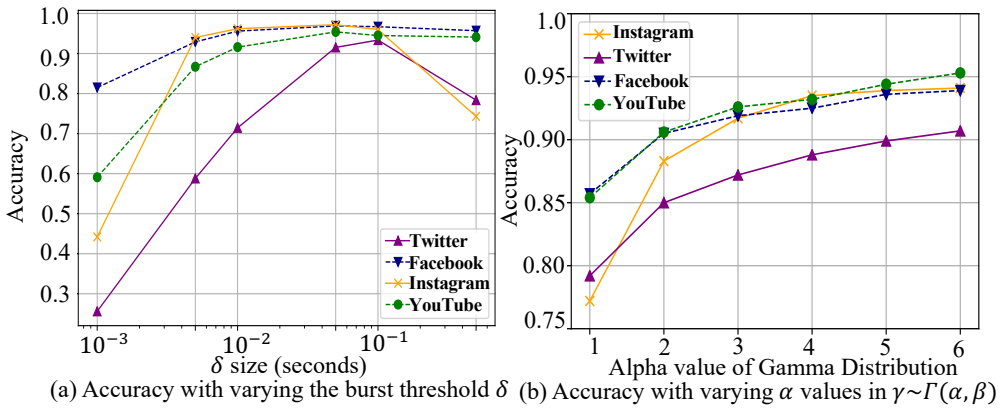
Performance of Intra-domain WPF with Varying Number of Unmonitored Pages in Training Set. In this experiment, the size of $\mathbb{U}_{tr}$ varies while we maintain unchanged 30 pages in $\mathbb{M}_{tr}$ and 20,000 pages in $\mathbb{U}_{te}$. We use Instagram, with 29,000 collected unmonitored pages, as a case study

to evaluate the influence of the size of $\mathbb{U}_{tr}$. Intuitively, the more unmonitored pages the attacker includes in the training set, the higher precision and fewer mistakes he would make in picking out the monitored pages. The reason we choose a monitored set of 30 pages against an unmonitored set of 20,000 pages is to simulate a realistic scenario in which the monitored pages take up merely a tiny portion (~2%) of the total pages possibly visited by the victim. We include varying sizes of $\mathbb{U}_{tr}$ from 1,500 to 9,000, and determine the TPR, FPR and BDR, as shown in Figure 5.6 (b).

It is observed that our approach achieves a consistent performance against an increasing number of unmonitored pages in the training set, with marginally decreasing TPRs at around 0.96 (from 0.97 down to 0.96), decreasing FPRs at around 0.0015 (from 0.0017 down to 0.0014) and increasing BDR at around 0.93 (from 0.92 up to 0.94). The consistently high BDR indicates the feasibility and high successful rate of intra-domain WPF to identify a small amount of pages from a significantly larger set. This result shows that intra-domain WPF is feasible without requiring much knowledge of pages outside attacker's target set.

5.5.3 RQ3: Influencing Factors of CDN Bursts

In this section, we study the influence of the burst threshold δ and network latency on fingerprinting performance.



(a) Accuracy with varying the burst threshold δ (b) Accuracy with varying α values in $\gamma \sim \Gamma(\alpha, \beta)$

Figure 5.7: Evaluation on Influencing Factors

Burst Threshold δ. According to the feature engineering method intro-

duced in Section 5.2.2, we group consecutive packets into a CDN burst when the inter-packet time gap is less than a value referred to as the burst threshold δ. The choice of δ would impact the total number of CDN bursts and the size of each burst. To determine a suitable δ for the real-world application of the intra-domain WPF, we measure the classification accuracy against various δ values ranging across magnitudes (from 1ms to 500ms) using dataset $\mathbb{D}_1$. When calculating the accuracy, we also apply 10-fold stratified cross validation to ensure the robustness and generalization in the results and to eliminate impacts of random noise and errors. The result is shown in Figure 5.7 (a). It is observed that the accuracy peak when δ is around 50ms to 100ms. The accuracy is significantly lower when δ value is very small (around 1ms) since many small and varying-sized CDN bursts are derived, resulting in inconsistent feature set. This confirms our estimation made in Section 5.2.2 that due to the concurrent requests and network latency, it is more robust to reconstruct a batch of objects, rather than an individual object, into a burst. Nonetheless, the accuracy starts dropping when δ increases beyond the optimal value. This is because multiple original bursts are grouped into fewer CDN bursts, reducing the amount of information.

Variation in Network Latency. The variation in the network latency, possibly due to unstable network condition, could lead to the fluctuation on the Packet Interval Time (PIT). This may "break" the CDN bursts since the burst threshold δ is a temporal parameter. To investigate this influence, we construct a simulated dataset based on $\mathbb{D}_2$ by introducing extra latency to the timestamps of the original traces. The added latency γ follows a Gamma distribution ($\gamma \sim \Gamma(\alpha, \beta)$ with a shape parameter α and a rate parameter β) which is commonly used for modeling the network latency [92]. We adopt 6 Gamma distributions with different spread levels (i.e., α. The higher the α, the less spread out the gamma distribution will be.) and a fixed mean value (i.e., α/β) of 48ms which is the average latency in the major CDN service providers [50]. We construct the CDN burst feature set from the simulated dataset and measure the classification performance using the same approach in Section 5.5.1. Our results are shown in Figure 5.7 (b).

As could be expected, the accuracy of intra-domain WPF increases as the latency becomes more concentrated. When the network latency distribution is concentrated within a small range (indicating that the network condition is relatively stable), there are only tiny shifts induced in the PITs. These shifts can be tolerated by a sufficiently large burst threshold δ, causing less loss to the accuracy. In contrast, when the latency spreads out, a larger latency can break through the burst threshold, generating "noisy" bursts.

5.6 Limitations and Countermeasures

In this section, we discuss the limitations of our intra-domain WPF. We also propose and evaluate countermeasures against such attacks.

5.6.1 Limitations of Intra-domain WPF

Dependency on IP addresses. Our intra-domain WPF relies on the server IP addresses to extract the CDN packets out of the network traffic. This becomes infeasible when the network communication is protected by anonymizers such as VPN and Tor, in which only the proxy or the entry node is trackable. The anonymizers mix up CDN packets and web server packets, such that the boundary to group packets into bursts is blurred out. As a result, the distinguishability of the bursts may drop, as shown by our experiments on Instagram (Figure 5.4), web server bursts (Figure 5.5) and Tor browser (Section 5.6.2). However, the intra-domain WPF remains a prominent threat since only a small proportion (~7%) of Internet users frequently use VPNs and merely one third of them use it for social websites [74], and the Tor population is much smaller.

Content Drift in Social Media Websites. Website fingerprinting has been criticized for being impractical due to the changes in the web page contents over time, also known as the content drift [91]. Web pages in social media websites get updated particularly frequently, with millions of people posting on their social media daily [212]. The updated content in a page would result in the shift in the distribution of the objects, potentially causing changes in the patterns of CDN bursts and undermining the trained

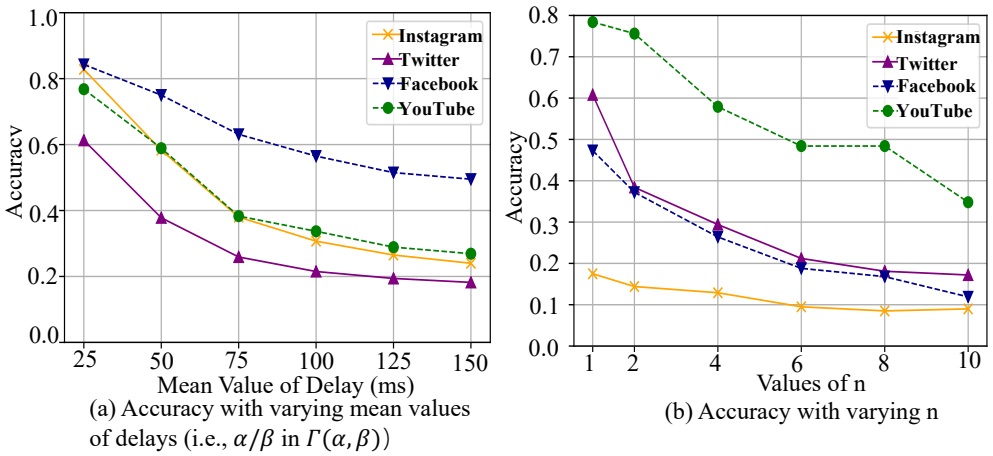
classifier. This requires updating the dataset and retraining the model timely to maintain a high accuracy. In this regard, our approach endures less impact from content drift than those requiring large numbers of training instances (i.e., deep learning based approaches).

5.6.2 Possible Countermeasures

Given that the intra-domain WPF utilizes the temporal and volumetric features of the CDN traffic, we propose defenses to mitigate these aspects in the traffic. To investigate their effectiveness, we use a small (for efficiency of experiments) dataset (denoted by $\mathbb{D}_4$) that includes 50 traces of 50 randomly selected Instagram profile pages from $\mathbb{D}_2$. The experiments conducted in this section are all on $\mathbb{D}_4$, unless stated otherwise.

Deviating Packet Interval Time in CDN Servers. One countermeasure is to break the expected packet arrival periodicity by deviating the so-called Inter Packet Arrival Time (IPAT). This obscures the boundary among the bursts such that the feature engineering algorithm may fail to produce robust features. To do this, the CDN server should add a marginal and randomized delay before it sends out a packet. This countermeasure is inspired by our finding from the experiment on the network latency in Section 5.5.3 that a spread-out network latency can significantly lower the accuracy of the intra-domain WPF. To explore its effectiveness, we simulate it by adding a random value that follows a Gamma distribution ($\Gamma(\alpha, \beta)$) to the arrival timestamp of each packets in our dataset. We adopt 6 Gamma distributions with a fixed spread level at α=0.25 and varying average delays α/β ranging from 25ms to 150ms.

As shown in Figure 5.8 (a), the intra-domain WPF accuracy drops significantly with increasing mean values of random delays, given a fixed spread level. Intuitively, the patterns of CDN burst would be disrupted to a greater extent due to a larger random delay, indicating a better defense performance. In general, this defense would not incur an intolerable delay, compared with the average network latency (50ms, cf., Section 5.5.3), and it does not increase bandwidth overhead. Nonetheless, this countermeasure



(a) Accuracy with varying mean values of delays (i.e., α/β in $\Gamma(\alpha, \beta)$)

(b) Accuracy with varying n

Figure 5.8: Simulated Evaluation on Defenses

requires involvement of the CDN service providers due to the server-side changes.

Loading Dummy Packets from Web Browsers. This countermeasure forces the browser to send dummy requests to the same CDN server in parallel with the genuine requests, creating dummy CDN traffic to reshape the original CDN burst patterns. To simulate this countermeasure, we insert dummy packets into the traces in $\mathbb{D}_4$. The inserted packets are extracted from Instagram pages which are not included in $\mathbb{D}_4$. We parameterize the portion of packets to be inserted as $\frac{n}{10}$ $(0 \leq n \leq 10)$ of the original traces. Intuitively, n dummy packets are inserted into every 10 packets.

Our results are shown in Figure 5.8 (b). It suggests that this countermeasure can perform well even with a small proportion of dummy packets. For example, the attack accuracy for Instagram drops to 0.175, down from 0.97, with only 10% of dummy packets. For Twitter and Facebook, the attack accuracy drops to below 40% with only 20% of the inserted dummy packets. The defense performs better with even larger proportion of dummy packets. This countermeasure introduces limited bandwidth overhead and does not incur much latency. It requires no changes to the CDN server since it is deployed at browser side.

Anonymity Networks. Another possible defense is to browse the web pages through Tor network which conceals the IP information and incurs

random latency due to the different routes selected. Since only entry node IP address is available, the bursts can be extracted only from the overall network trace. To explore the effectiveness, we constructed a small test dataset $\mathbb{D}_5$ including 50 traces for each of the 50 Instagram profiles loaded through Tor browser. We compare the classification accuracy on $\mathbb{D}_5$ with that on $\mathbb{D}_4$, and find the accuracy drops to 0.292, compared to the that of 0.97 on $\mathbb{D}_4$.

5.7 Related Work

Fingerprinting Websites. Cai et al. [41] proposed an attack on Tor based on the optimal string alignment distance (OSAD) feature, achieving 80% accuracy in 100 websites. Wang et al. [202] improved the OSAD based approach and achieved accuracy over 90% in both closed-world and open-world settings. Wang et al. [201] further improved the performance of inter-domain WSF attack on Tor with large open-world datasets. Wang et al. [203] explored the feasibility of practically deploying the inter-domain WSF system. Recently, Wang [200] has proposed a new metric for evaluating the feasibility of detecting the sensitive web pages in reality, and used three optimizers to improve the classifier performance. Gu et al. [75] evaluated the performance of inter-domain WSF attacks under multi-tab browsing setting. Kwon et al. [100] launched a inter-domain WSF attack on the Tor hidden services based on the long-lived features of Tor circuits. Hayes et al [78] proposed a novel inter-domain WSF technique based on a variant of random forest, which achieved a TPR of 85% and an FPR of 0.02% when monitoring 30 web pages out of over 100,000 unique web pages. Panchenko et al. [132] presented an approach based on an SVM classifier, utilizing the cumulative behavior representation of the web page loading trace.

In the most recent contributions, Deep Neural Networks (DNN) have been more adopted besides traditional machine learning algorithms. Rimmer et al. [147] proposed a CNN-based inter-domain WSF that automates feature engineering. The attack trained on 2,500 traces per site and achieved 96.3% accuracy. Oh et al. [126] utilized unsupervised DNN to extract low

dimension informative features and achieved an accuracy of 94%. Sirinam et al. [157] presented CNN-based attack which achieved high accuracy on both undefended and defended Tor traffic. Sirinam et al. [158] further proposed an attack based on N-shot learning which enables feature extraction using a pre-trained classifier. Their approach improves the attack robustness and shorten the classifier preparation time.

Fingerprinting Web Pages. Another direction in web fingerprinting is to further identify the similar web pages or the web contents within the same web domain, or intra-domain web page fingerprinting (WPF). To the best of our knowledge, intra-domain WPF is an under-researched area with limited number of published works. Miller et al. [118] utilized similar packet burst features extracted from the traces of a sequence of linked web pages within a website.This work requires constructing an interconnected structure of the targeted web pages accessible within a website. This restricts its applicability in scenarios where the web pages are independent of each other, such as social media profile pages. Shen et al. [154] proposed an attack to distinguish similar web pages using the cumulative packet length feature. This work assumes an active attack model that is able to intercept network traffic between victim user and web servers.

5.8 Summary

In this paper, we propose the novel intra-domain WPF, which aims to determine which web page a user has just browsed. The proposed fingerprinting is based on a specially engineered feature set named CDN bursts. We show that the intra-domain WPF is feasible in social media websites, by investigating the informativeness of the classification features derived from CDN bursts. We comprehensively evaluate the intra-domain WPF with datasets collected from four top social media websites, which contain traces of over 10k unique pages and 400k page. Our evaluation demonstrate its high fingerprinting accuracy (an average accuracy of up to 96% and a false positive rate as low as 0.02%) and computational efficiency ($<$ 5mins training time). The datasets have been released to

facilitate research in this area. To the best of our knowledge, this work is the first intra-domain WPF that is based on the patterns in CDN traffic. Our work should raise an alert on this previously neglected threat. Furthermore, we hope it will inspire more future research on the general intra-domain WPF.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Network Security protocols are critical for protecting the network security and privacy. However, they have been continually rendered error prone due to the complex protocol development life cycle from the design to implementation. Hence, analyzing the security and user privacy properties become imperative before the network security protocol deployment in practice. In this thesis, we propose three approaches, targeting the security and privacy analysis on both perspectives of protocol design and implementation.

In Chapter 3, we propose a formal framework for analyzing the network security protocol designs. Our framework includes a formal model of the network and web infrastructure, the attacker models, as well as formal definitions of the security and privacy properties. We construct the protocol models in the applied pi calculus and automatically verify them using the tool Proverif. We apply our framework to analyze four mainstream SSO protocols and find that they can correctly preserve the security property yet they fail to provide user privacy. We have identified a novel privacy attack that allows an honest-but-curious identity provider to track the services the target user logged into.

In Chapter 4, we propose a systematic approach based on software testing to assess the effectiveness of network security protocol implementations against the MITM attacks. We apply our approach to scan the vulnerabilities in the certificate validation user interfaces in the WPA en-

terprise authentication scheme, covering laptops and phones installed with four mainstream OSes. During the scan for each device, we enumerate through all the configuration options in the WPA supplicant while connecting to our evil twin testbed. For each test case, we validate the security of each Wi-Fi configuration and the effectiveness of visual-based security warnings upon insecure configurations during the initial and subsequent connections to the wireless networks. Through the study, we have identified five server vulnerabilities in the existing user interface of WPA supplicant implementations.

In Chapter 5, we present a traffic analysis technique to evaluate the user privacy leakage from the network security protocol implementations. Although protected by securely implemented encrypted communication channels, the network traffic patterns still inadvertently reveal information regarding user browsing activities. Our approach utilizes the network traffic characteristics between the client and the CDN servers to fingerprint the similar web pages within the same domain. We design a CDN traffic burst-based feature engineering algorithm and further quantify the information leakage from each feature. According to our evaluation, the feature information is sufficient to accurately identify social media pages. Such attack further enables probing the detailed contents browsed by the user, leading to significant compromise in user privacy.

6.2 Future Research Directions

6.2.1 Model Checking Network Security Protocol Implementations

In this thesis, the formal analysis on network security protocols is limited to the design level due to the notorious scalability limitation known as state-explosion [47] when verifying large programs. Even though software testing and traffic analysis is effective in detecting security and privacy vulnerabilities in the protocol implementations, they are insufficient to prove the satisfaction of the properties.

In view of the challenges, we plan to resort to a hybrid technique incorporating both software model checking and program reduction techniques, such that we can directly check the security and privacy properties against the proposed attack models on the protocol implementation execution. On the one hand, software model checking performs exhaustive search on the protocol implementation state space for the property violations, which is similar to the traditional model checking. Some existing software model checkers can be used to serve this purpose, such as CMC [122] and Java PathFiner [189]. On the other hand, reduction (e.g., partial-order reduction [72]) and program abstraction techniques (e.g., property preserving abstraction [108]) can be utilized to further mitigate the state-explosion problem for model checking large-scale and complicated programs.

6.2.2 Automatically Fingerprinting Intra-domain Web Pages

In addition to model checking security and privacy properties at the implementation level, web fingerprinting remains a powerful method for the user privacy analysis. As we have acknowledged in Chapter 4, there are several limitations to be addressed and improved in the proposed intra-domain WPF technique despite its effectiveness.

As future work, we plan to adopt the deep neural networks as promising techniques to enable the automated feature extraction, eliminating the need to manually extract and construct features from the CDN traffic. Through a carefully designed model architecture, we aim to enable the learning of subtle features such as those contained in the CDN bursts.

The web pages are continuously evolving, degrading the effectiveness of the pre-trained models over time. We plan to mitigate this so-called content drift problem through a combination of deep-learning and traditional machine learning techniques, similar to the approach proposed by Sirinam et al. [158].

Bibliography

[1] "2020 Global Networking Trends Report, 2020", https://www.cisco.com/c/dam/m/en_us/solutions/enterprise-networks/networking-report/files/GLBL-ENG_NB-06_0_NA_RPT_PDF_MOFU-no-NetworkingTrendsReport-NB_rpten018612_5.pdf, visited in April 2021.

[2] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication", in *POPL*, 2001, pp. 104–115.

[3] M. E. Acer, E. Stark, A. P. Felt, S. Fahl, R. Bhargava, B. Dev, M. Braithwaite, R. Sleevi, and P. Tabriz, "Where the wild warnings are: Root causes of chrome https certificate errors", in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1407–1420.

[4] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques", *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016.

[5] "Aircrack-ng, 2008", https://www.aircrack-ng.org, visited in April 2021.

[6] D. Akhawe, A. Barth, P.E.Lam, J. Mitchell, and D. Song, "Towards a formal foundation of web security", in *CSF*, 2010, pp. 290–304.

[7] D. Akhawe, B. Amann, M. Vallentin, and R. Sommer, "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*, 2013, pp. 59–70.

[8] A. Armando, R. Carbone, L. Compagna, J. Cuellar, and L. Tobarra, "Formal analysis of saml 2.0 web browser single sign-on: Breaking the saml-based single sign-on for google apps", in *Workshop on Formal Methods in Security Engineering*, 2008.

[9] A. Armando and L. Compagna, "Sat-based model-checking for security protocols analysis", *International Journal of Information Security*, 2008.

[10] D. J. Arndt and A. N. Zincir-Heywood, "A comparison of three machine learning techniques for encrypted network traffic analysis", in *2011 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, 2011, pp. 107–114.

[11] A. Ashokkumar, S. Talaifar, W. Fraser, R. Landabur, M. Burhmester, A. Gómez, B. Paredes, and W. Swann, "Censoring political opposition online: Who does it and why", Jul. 2020.

[12] S. Aslam, "Facebook by the numbers: Stats, demographics and fun facts", `https://www.omnicoreagency.com/facebook-statistics`, visited in April 2021.

[13] "Assigned Number", `https://tools.ietf.org/html/rfc1340`, visited in April 2021.

[14] T. Avgerinos, A. Rebert, S. K. Cha, and D. Brumley, "Enhancing symbolic execution with veritesting", in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 1083–1094.

[15] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection", *ACM Transactions on Information and System Security*, pp. 186–205, 2000.

[16] G. Bai, J. Lei, G. Meng, S. S. Venkatraman, P. Saxena, J. Sun, Y. Liu, and J. S. Dong, "AuthScan: Automatic Extraction of Web Authentication Protocols from Implementations", in *NDSS*, 2013.

[17] G. Bai, J. Hao, J. Wu, Y. Liu, Z. Liang, and A. Martin, "Trustfound: Towards a formal foundation for model checking trusted computing platforms", in *FM*, 2014, pp. 110–126.

[18] C. Bansal, K. Bhargavan, A. Delignat-Lavaud, and S. Maffeis, "Discovering concrete attacks on website authorization by formal analysis", in *CSF*, 2012, pp. 247–262.

[19] C. Bansal, K. Bhargavan, A. Delignat-Lavaud, and S. Maffeis, "Post", in. 2013, ch. Keys to the Cloud: Formal Analysis and Concrete Attacks on Encrypted Web Storage, pp. 126–146.

[20] C. Bansal, K. Bhargavan, A. Delignat-Lavaud, and S. Maffeis, "Discovering concrete attacks on website authorization by formal analysis", *J. Comput. Secur.*, vol. 22, no. 4, pp. 601–657, 2014.

[21] M. Bar-Hillel, "The base-rate fallacy in probability judgments", *Acta Psychologica*, pp. 211–233, 1980.

[22] A. Bartoli, E. Medvet, A. De Lorenzo, and F. Tarlao, "(in)secure configuration practices of wpa2 enterprise supplicants", in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, 2018.

[23] S. Bekrar, C. Bekrar, R. Groz, and L. Mounier, "Finding software vulnerabilities by smart fuzzing", in *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, 2011, pp. 427–430.

[24] J. Bellardo and S. Savage, "802.11 denial-of-service attacks: Real vulnerabilities and practical solutions",, ser. SSYM'03, 2003, p. 2.

[25] C. Benjamin, "Mschapv2acc", `https://github.com/polkaned/mschapv2acc`, visited in April 2021.

[26] R. Bhatia, F. Hao, M. Kodialam, and T. Lakshman, "Optimized network traffic engineering using segment routing", in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 657–665.

[27] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools", *Ieee communications surveys & tutorials*, vol. 16, no. 1, pp. 303–336, 2013.

[28] S. Bishop, M. Fairbairn, M. Norrish, P. Sewell, M. Smith, and K. Wansbrough, "Rigorous specification and conformance testing techniques for network protocols, as applied to tcp, udp, and sockets", in *ACM SIGCOMM Computer Communication Review*, 2005.

[29] B. Blanchet, "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules", in *CSFW*, 2001, pp. 82–96.

[30] B. Blanchet and D. Cadé, "Cryptoverif: Cryptographic protocol verifier in the computational model", `http://prosecco.gforge.inria.fr/personal/bblanche/cryptoverif/`,visited in April 2021.

[31] B. Blanchet, C. Stentzel, X. Allamigeon, V. Cheval, and B. Smyth, "ProVerif: Cryptographic protocol verifier in the formal model", `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`, visited in April 2021.

[32] M. S. Blog, "Average page load times for 2018-how does yours compare?" `https://www.machmetrics.com/speed-blog/average-page-load-times-websites-2018/`, visited in April 2021.

[33] T. Bolognesi and E. Brinksma, "Introduction to the iso specification language lotos", *Computer Networks and ISDN Systems*, vol. 14, no. 1, pp. 25–59, 1987.

[34] L. Breiman, "Random forests", *Machine Learning*, pp. 5–32, 2001.

[35] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral, "Deep packet inspection as a service", in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014, pp. 271–282.

[36] S. Brenza, A. Pawlowski, and C. Pöpper, "A practical investigation of identity theft vulnerabilities in eduroam", in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2015.

[37] BrowserID, "B. adida and t. kuijsten and a. nennker and a. narayanan", `https://github.com/mozilla/id-specs/blob/prod/browserid/index.md`, visited in April 2021.

[38] C. Brubaker, S. Jana, B. Ray, S. Khurshid, and V. Shmatikov, "Using frankencerts for automated adversarial testing of certificate validation in ssl/tls implementations", in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, 2014, pp. 114–129.

[39] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication", *ACM Transactions on Computer Systems*, pp. 18–36, 1990.

[40] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses", in *CCS*, 2014, pp. 227–238.

[41] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses", in *CCS*, 2012, pp. 605–616.

[42] A. Callado, C. Kamienski, G. Szabó, B. P. Gero, J. Kelner, S. Fernandes, and D. Sadok, "A survey on internet traffic identification", *IEEE communications surveys & tutorials*, vol. 11, no. 3, pp. 37–52, 2009.

[43] A. Callado, J. Kelner, D. Sadok, C. A. Kamienski, and S. Fernandes, "Better network traffic identification through the independent combination of techniques", *Journal of Network and Computer Applications*, vol. 33, no. 4, pp. 433–446, 2010.

[44] A. Cassola, W. K. Robertson, E. Kirda, and G. Noubir, "A practical, targeted, and stealthy attack against WPA enterprise authentication", in *20th Annual Network and Distributed System Security Symposium, NDSS*, The Internet Society, 2013.

[45] H. Cheng and R. Avnur, "Traffic analysis of ssl encrypted web browsing", 1998.

[46] L. Chuat, A. Abdou, R. Sasse, C. Sprenger, D. Basin, and A. Perrig, "Sok: Delegation and revocation, the missing links in the web's chain of trust", 2019.

[47] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, "Model checking and the state explosion problem", in *LASER Summer School on Software Engineering*, Springer, 2011, pp. 1–30.

[48] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching-time temporal logic", in *Logic of Programs, Workshop*, 1981, pp. 52–71.

[49] J. Clause, W. Li, and A. Orso, "Dytan: A generic dynamic taint analysis framework", in *Proceedings of the 2007 international symposium on Software testing and analysis*, 2007, pp. 196–206.

[50] CloudMatch, "Cdn network test", `https://cloudharmony.com/speedtest-downlink-uplink-latency-dns-for-cdn`, visited in April 2021.

[51] J. Cobbe, "Algorithmic censorship by social platforms: Power and resistance", *Philosophy & Technology*, Oct. 2020.

[52] "Curl, command line tool and library for transferring data with URLs", `https://curl.se`, visited in April 2021.

[53] D. A. Dai Zovi and S. Macaulay, "Attacking automatic wireless network selection", in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, 2005, pp. 365–372.

[54] A. Dainotti, A. Pescape, and K. C. Claffy, "Issues and future directions in traffic classification", *IEEE network*, vol. 26, no. 1, pp. 35–40, 2012.

[55] A. Davis, "How social media spurred myanmar's latest violence", `https://iwpr.net/global-voices/how-social-media-spurred-myanmars-latest`, visited in April 2021.

[56] J. De Ruiter and E. Poll, "Protocol state fuzzing of tls implementations", in *Proceedings of the 24th USENIX Conference on Security Symposium*, 2015, pp. 193–206.

[57] S. Delaune, S. Kremer, and M. Ryan, "Verifying privacy-type properties of electronic voting protocols", *Journal of Computer Security*, vol. 17, no. 4, pp. 435–487, 2009.

[58] D. Dolev and A. C.-C. Yao, "On the security of public key protocols", *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–207, 1983.

[59] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the https certificate ecosystem", in *Proceedings of the 2013 Conference on Internet Measurement Conference*, 2013, pp. 291–304.

[60] "Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs, 2005", `https://tools.ietf.org/html/rfc4017`, visited in April 2021.

[61] "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0), 2008", `https://tools.ietf.org/html/rfc5281`, visited in April 2021.

[62] S. Fahl, Y. Acar, H. Perl, and M. Smith, "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*, 2014, pp. 507–512.

[63] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "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*, 2012, pp. 50–61.

[64] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz, "Measuring HTTPS adoption on the web", in *USENIX Security*, 2017, pp. 1323–1338.

[65] D. Fett, R. Küsters, and G. Schmitz, "An expressive model for the web infrastructure: Definition and application to the BrowserID SSO system", in *IEEE S&P*, 2014.

[66] D. Fett, R. Küsters, and G. Schmitz, "Analyzing the BrowserID SSO System with Primary Identity Providers Using an Expressive Model of the Web", in *ESORICS*, 2015, pp. 43–65.

[67] D. Fett, R. Küsters, and G. Schmitz, "SPRESSO: A secure, privacy-respecting single sign-on system for the web", in *CCS*, 2015, pp. 1358–1369.

[68] P. Fiterau-Brostean, B. Jonsson, R. Merget, J. de Ruiter, K. Sagonas, and J. Somorovsky, "Analysis of dtls implementations using protocol state fuzzing", in *29th USENIX Security Symposium*, 2020, pp. 2523–2540.

[69] D. Garcia, "Leaking privacy and shadow profiles in online social networks", *Science Advances*, 2017.

[70] E. Geier, "Wi-fi hotspots: Setting up public wireless internet access", *Networking Technology*, 2006.

[71] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "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*, 2012, pp. 38–49.

[72] P. Godefroid, "Partial-order methods for the verification of concurrent systems - an approach to the state-explosion problem", 1995.

[73] X. Gong, N. Borisov, N. Kiyavash, and N. Schear, "Website detection using remote traffic analysis", in *PETS*, 2012, pp. 58–78.

[74] R. Greenberg, "Vpn use and data privacy stats for 2021", `https://www.vpnmentor.com/blog/vpn-use-data-privacy-stats/`, visited in April 2021.

[75] X. Gu, M. Yang, and J. Luo, "A novel website fingerprinting attack against multi-tab browsing behavior", in *IEEE Computer Supported Cooperative Work in Design*, 2015, pp. 234–239.

[76] S. Hanna, E. C. R. Shinz, D. Akhawe, A. Boehmz, P. Saxena, and D. Song, "The emperor's new api: On the (in)secure usage of new client side primitives", in *W2SP*, 2010.

[77] J. Hao, Y. Liu, W. Cai, G. Bai, and J. Sun, "Vtrust: A formal modeling and verification framework for virtualization systems", in *ICFEM*, 2013, pp. 329–346.

[78] J. Hayes and G. Danezis, "K-fingerprinting: A robust scalable website fingerprinting technique", in *USENIX Security*, 2016, pp. 1187–1203.

[79] D. Herrmann, R. Wendolsky, and H. Federrath, in *CCS Workshop on Cloud Computing Security*, 2009, pp. 31–42.

[80] A. Hintz, "Fingerprinting websites using traffic analysis", in *PETS*, 2003, pp. 171–178.

[81] C. A. R. Hoare, "Communicating sequential processes",, vol. 21, no. 8, pp. 666–677, 1978.

[82] R. Holz, L. Braun, N. Kammenhuber, and G. Carle, "The ssl landscape: A thorough analysis of the x.509 pki using active and passive measurements", in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, 2011, pp. 427–444.

[83] G. J. Holzmann, "The model checker spin", *IEEE Trans. Softw. Eng.*, vol. 23, no. 5, pp. 279–295, 1997.

[84] L. S. Huang, A. Rice, E. Ellingsen, and C. Jackson, "Analyzing forged ssl certificates in the wild", in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, 2014, pp. 83–97.

[85] "Internet live stats", `https://www.internetlivestats.com`, visited in April 2021.

[86] "Intra-domain webpage fingerprinting dataset", `https://github.com/ResearchWPF/Intra-domain-WPF.git`, visited in April 2021.

[87] D. Jackson, "Tools and algorithms for the construction and analysis of systems: 8th international conference, tacas", in. 2002, p. 20.

[88] R. Jansen, M. Traudt, and N. Hopper, "Privacy-preserving dynamic learning of tor network traffic", in *CCS*, 2018, pp. 1944–1961.

[89] C. B. Jones, "Systematic Software Development Using VDM (2nd Ed.)" 1990.

[90] W. Joshua, "Asleep - recovers weak leap password", `https://github.com/joswr1ght/asleap`, visited in April 2021.

[91] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks", in *CCS*, 2014, pp. 263–274.

[92] M. Karakaş, "Determination of network delay distribution over the internet", Ph.D. dissertation, Citeseer, 2003.

[93] F. Kerschbaum, "Simple cross-site attack prevention", in *Workshop on Security and Privacy in Communications Networks*, 2007, pp. 464–472.

[94] H. Kim, K. C. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: Myths, caveats, and the best practices", in *Proceedings of the 2008 ACM CoNEXT conference*, 2008, pp. 1–12.

[95] J. C. King, "Symbolic execution and program testing", *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, 1976.

[96] C. Kohlios and T. Hayajneh, "A comprehensive attack flow model and security analysis for wi-fi and wpa3", *Electronics*, vol. 7, p. 284, Oct. 2018.

[97] D. Kumar, Z. Wang, M. Hyder, J. Dickinson, G. Beck, D. Adrian, J. Mason, Z. Durumeric, J. A. Halderman, and M. Bailey, "Tracking certificate misissuance in the wild", in *IEEE Symposium on Security and Privacy*, 2018, pp. 785–798.

[98] S. Kumar, J. Turner, and J. Williams, "Advanced algorithms for fast and scalable deep packet inspection", in *2006 Symposium on Architecture For Networking And Communications Systems*, 2006, pp. 81–92.

[99] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems", in *Computer Aided Verification*, vol. 6806, Jul. 2011, pp. 585–591.

[100] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas, "Circuit fingerprinting attacks: Passive deanonymization of tor hidden services", in *USENIX Security*, 2015.

[101] J. L. Zittrain, R. Faris, H. Noman, J. Clark, C. Tilton, and R. Morrison-Westphal, "The shifting landscape of global internet censorship", *SSRN Electronic Journal*, Jan. 2017.

[102] E. Langford, "Quartiles in elementary statistics", *Journal of Statistics Education*, vol. 14, no. 3, 2006.

[103] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell", *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1–2, pp. 134–152, 1997.

[104] Q. Li, Y. Qi, Q. Hu, S. Qi, Y. Lin, and J. S. Dong, "Adversarial adaptive neighborhood with feature importance-aware convex interpolation", *IEEE TIFS*, 2020.

[105] S. Li, H. Guo, and N. Hopper, "Measuring information leakage in website fingerprinting attacks and defenses", in *CCS*, 2018, pp. 1977–1992.

[106] Y. Li, X. Yin, Z. Wang, J. Yao, X. Shi, J. Wu, H. Zhang, and Q. Wang, "A survey on network verification and testing with formal methods: Approaches and challenges", *IEEE Communications Surveys & Tutorials*, Aug. 2018.

[107] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, and C. Wilson, "An end-to-end measurement of certificate revocation in the web's pki", in *Proceedings of the 2015 Internet Measurement Conference*, 2015, pp. 183–196.

[108] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem, "Property preserving abstractions for the verification of concurrent systems", *Form. Methods Syst. Des.*, vol. 6, no. 1, pp. 11–44, 1995.

[109] L. Lu, E.-C. Chang, and M. Choon Chan, "Website fingerprinting and identification using ordered feature sequences",, Sep. 2010, pp. 199–214.

[110] S. Ma, F. Thung, D. Lo, C. Sun, and R. H. Deng, "Vurle: Automatic vulnerability detection and repair by learning from examples", in *European Symposium on Research in Computer Security*, Springer, 2017, pp. 229–246.

[111] K. Mahadewa, K. Wang, G. Bai, L. Shi, J. S. Dong, and Z. Liang, "HOMESCAN: Scrutinizing Implementations of Smart Home Integrations", in *23rd International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2018, pp. 21–30.

[112] K. Mahadewa, K. Wang, G. Bai, L. Shi, Y. Liu, J. S. Dong, and Z. Liang, "Scrutinizing Implementations of Smart Home Integrations", *IEEE Transactions on Software Engineering (TSE)*, 2019.

[113] C. Meadows, "A system for the specification and analysis of key management protocols", in *Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, 1991, pp. 182–195.

[114] C. Meadows, "Formal verification of cryptographic protocols: A survey", in *Proceedings of the 4th International Conference on the Theory and Applications of Cryptology: Advances in Cryptology*, ser. ASIACRYPT '94, 1994, pp. 135–150.

[115] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, "Traffic matrix estimation: Existing techniques and new directions", *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 161–174, 2002.

[116] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols", in *Proceedings of the 25th International Conference on Computer Aided Verification*, 2013, pp. 696–701.

[117] "Microsoft PPP CHAP Extensions, 1998", `https://tools.ietf.org/html/rfc2433`, visited in April 2021.

[118] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar, "I know why you went to the clinic: Risks and realization of https traffic analysis", in *PETS*, 2014, pp. 143–163.

[119] R. Milner, "Communication and Concurrency", 1989.

[120] E. F. Moore, "Gedanken-experiments on sequential machines", in *Automata Studies. (AM-34), Volume 34*. 1956, pp. 129–154.

[121] G. Munz and G. Carle, "Real-time analysis of flow data for network attack detection", in *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007, pp. 100–108.

[122] M. Musuvathi, A. Chou, D. L. Dill, and D. Engler, "Model checking system software with cmc", in *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, 2002, pp. 219–222.

[123] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, "The art of software testing", Wiley Online Library, 2004, vol. 2.

[124] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers", *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, Dec. 1978.

[125] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning", *IEEE communications surveys & tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[126] S. E. Oh, S. Sunkam, and N. Hopper, "P1-fp: Extraction, classification, and prediction of website fingerprints with deep learning", *PoPETs*, pp. 191–209, 2019.

[127] T. Omrani, A. Dallali, B. C. Rhaimi, and J. Fattahi, "Fusion of ann and svm classifiers for network attack detection", in *2017 18th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, 2017, pp. 374–377.

[128] L. Onwuzurike and E. De Cristofaro, "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*, 2015.

[129] "OpenSSL", https://www.openssl.org, visited in April 2021.

[130] OpenStack, https://www.openstack.org, visited in April 2021.

[131] "OpenVPN", https://openvpn.net, visited in April 2021.

[132] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel, "Website fingerprinting at internet scale", in *NDSS*, 2016.

[133] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks", in *WPES*, 2011, pp. 103–114.

[134] L. Paninski, "Estimation of entropy and mutual information", *Neural Computation*, pp. 1191–1253, 2003.

[135] A. Parecki and C. messina, "Oauth 2.0", https://oauth.net/2/.

[136] Paulirish, "Automated chrome profiling", https://github.com/paulirish/automated-chrome-profiling, visited in April 2021.

[137] T. Petsios, A. Tang, S. Stolfo, A. D. Keromytis, and S. Jana, "Nezha: Efficient domain-independent differential testing", in *2017 IEEE Symposium on security and privacy (SP)*, 2017, pp. 615–632.

[138] A. Pfitzmann and M. Köhntopp, "Anonymity, unobservability, and pseudeonymity — a proposal for terminology", in *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, 2001, pp. 1–9.

[139] M. Pióro and D. Medhi, "Routing, flow, and capacity design in communication and computer networks", 2004.

[140] "PPP Extensible Authentication Protocol (EAP), 1998", https://tools.ietf.org/rfc/rfc2284.txt, visited in April 2021.

[141] T. T. Project, "Browse privately. explore freely." https://www.torproject.org/.

[142] T. T. Project, "Tor metrics", https://metrics.torproject.org/bandwidth.html, visited in April 2021.

[143] "QS World University Rankings 2020", https://www.topuniversities.com/university-rankings/world-university-rankings/2020, visited in April 2021.

[144] "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP), 2003", https://www.ietf.org/rfc/rfc3579.txt, visited in April 2021.

[145] A. Razaghpanah, A. A. Niaki, N. Vallina-Rodriguez, S. Sundaresan, J. Amann, and P. Gill, "Studying tls usage in android apps", in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, 2017, pp. 350–362.

[146] "Repository for supplementary material", https://github.com/anonymous-research-security/USENIX-Submission-2021.git, visited in April 2021.

[147] V. Rimmer, D. Preuveneers, M. Juárez, T. van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning", in *NDSS*, 2018.

[148] V. Roth, W. Polak, E. Rieffel, and T. Turner, "Simple and effective defense against evil twin access points", in *Proceedings of the First ACM Conference on Wireless Network Security*, 2008, pp. 220–235.

[149] M. D. Ryan and B. Smyth, "Applied pi calculus", in *Formal Models and Techniques for Analyzing Security Protocols*, V. Cortier and S. Kremer, Eds., 2011, ch. 6.

[150] D. Schepers, A. Ranganathan, and M. Vanhoef, "Practical side-channel attacks against wpa-tkip", in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019, pp. 415–426.

[151] B. Schneier, Mudge, and D. Wagner, "Cryptanalysis of microsoft's pptp authentication extensions (ms-chapv2)", in *Secure Networking — CQRE [Secure]*, 1999, pp. 192–203.

[152] R. Schuster, V. Shmatikov, and E. Tromer, "Beauty and burst: Remote identification of encrypted video streams", in *USENIX Security*, 2017, pp. 1357–1374.

[153] Selenium, `http://www.seleniumhq.org`, visited in April 2021.

[154] M. Shen, Y. Liu, S. Chen, L. Zhu, and Y. Zhang, "Webpage fingerprinting using only packet length information", in *ICC*, 2019, pp. 1–6.

[155] Z. Shu, J. Wan, J. Lin, S. Wang, D. Li, S. Rho, and C. Yang, "Traffic engineering in software-defined networking: Measurement and management", *IEEE access*, vol. 4, pp. 3246–3256, 2016.

[156] SimilarTech, `https://www.similartech.com/categories/widget`, 2016.

[157] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning", in *CCS*, 2018, pp. 1928–1943.

[158] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright, "Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning", in *CCS*, 2019, pp. 1131–1148.

[159] J. M. Spivey, "Understanding Z: A Specification Language and Its Formal Semantics", 1988.

[160] U. D. O. C. I. of Standards and Technology, "DATA ENCRYPTION STANDARD (DES)", `https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf`, visited in April 2021.

[161] StatCounter, "Operating System Version Market Share", `https://gs.statcounter.com/os-market-share`, visited in April 2021.

[162] Statista, "Worldwide internet user penetration from 2014 to 2021", `https://www.statista.com/statistics/325706/global-internet-user-penetration/`.

[163] S. Steinbrecher and S. Köpsell, "Modelling unlinkability", in *Privacy Enhancing Technologies*, 2003, pp. 32–47.

[164] J. G. Steiner, C. Neuman, and J. I. Schiller, "Kerberos: An authentication service for open network systems", in *IN USENIX CONFERENCE PROCEEDINGS*, 1988, pp. 191–202.

[165] F. Stjernfelt and A. M. Lauritzen, "Facebook and google as offices of censorship", in *Your Post has been Removed: Tech Giants and Freedom of Speech*. 2020, pp. 139–172.

[166] C. M. Stone, T. Chothia, and F. D. Garcia, "Spinner: Semi-automatic detection of pinning without hostname verification", in *Proceedings of the 33rd Annual Computer Security Applications Conference*, 2017, pp. 176–188.

[167] G.-L. Sun, Y. Xue, Y. Dong, D. Wang, and C. Li, "An novel hybrid method for effectively classifying encrypted traffic", in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, 2010, pp. 1–5.

[168] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "PAT: towards flexible verification under fairness", in *Computer Aided Verification*, 2009, pp. 709–714.

[169] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic", in *IEEE S&P*, 2002.

[170] S.-T. Sun, K. Hawkey, and K. Beznosov, "Systematically breaking and fixing openid security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures", *Comput. Secur.*, vol. 31, no. 4, pp. 465–483, 2012.

[171] M. I. Svanks, "Integrity analysis: Methods for automating data quality assurance", *Information and Software Technology*, vol. 30, no. 10, pp. 595–605, 1988.

[172] A. Takanen, J. D. Demott, C. Miller, and A. Kettunen, "Fuzzing for software security testing and quality assurance", 2018.

[173] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic", in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016, pp. 439–454.

[174] TCPdump, `https://www.tcpdump.org`, visited in April 2021.

[175] P. Technologies, "Penetration testing of corporate information systems: Statistics and findings, 2019", `https://www.ptsecurity.com/ww-en/analytics/corp-vulnerabilities-2019/#id2`, visited in April 2021.

[176] "The avispa home page", `http://www.avispa-project.org/`,visited in April 2021.

[177] "The EAP-TLS Authentication Protocol, 2008", `https://tools.ietf.org/html/rfc5216`, visited in April 2021.

[178] "The GnuTLS Transport Layer Security Library", `https://www.gnutls.org`, visited in April 2021.

[179] C. Thompson, M. Shelton, E. Stark, M. Walker, E. Schechter, and A. P. Felt, "The web's identity crisis: Understanding the effectiveness of website identity indicators", in *USENIX Security*, 2019, pp. 1715–1732.

[180] C. Tian, C. Chen, Z. Duan, and L. Zhao, "Differential testing of certificate validation in ssl/tls implementations: An rfc-guided approach", *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 4, Oct. 2019.

[181] N. Y. Times, "Sri lanka blocks social media, fearing more violence", `https://www.nytimes.com/2019/04/21/world/asia/sri-lanka-social-media.html`, visited in April 2021.

[182] Trackalytics, "Free social media and website statistics/analytics", `https://www.trackalytics.com/`, visited in April 2021.

[183] O. Tripp, M. Pistoia, S. J. Fink, M. Sridharan, and O. Weisman, "Taj: Effective taint analysis of web applications", *ACM Sigplan Notices*, vol. 44, no. 6, pp. 87–97, 2009.

[184] M. Vanhoef and F. Piessens, "Practical verification of wpa-tkip vulnerabilities", in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, 2013, pp. 427–436.

[185] M. Vanhoef and F. Piessens, "Predicting, decrypting, and abusing wpa2/802.11 group keys", in *Proceedings of the 25th USENIX Conference on Security Symposium*, 2016, pp. 673–688.

[186] M. Vanhoef and F. Piessens, "Key reinstallation attacks: Forcing nonce reuse in wpa2", in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1313–1328.

[187] M. Vanhoef and E. Ronen, "Dragonblood: Analyzing the Dragonfly handshake of WPA3 and EAP-pwd", in *IEEE Symposium on Security & Privacy (SP)*, 2020.

[188] V. Varadharajan, "Use of a formal description technique in the specification of authentication protocols", *Computer Standards & Interfaces*, vol. 9, no. 3, pp. 203–215, 1990.

[189] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda, "Model checking programs", *Automated Software Engg.*, vol. 10, no. 2, pp. 203–232, 2003.

[190] V. L. Voydock and S. T. Kent, "Security mechanisms in high-level network protocols", *ACM Computing Surveys*, vol. 15, no. 2, pp. 135–171, 1983.

[191] D. Wagner and B. Schneier, "Analysis of the ssl 3.0 protocol", in *USENIX Security*, 1996.

[192] J. Wakefield, "Christchurch shootings: Social media races to stop attack footage", `https://www.bbc.com/news/technology-47583393`, visited in April 2021.

[193] G. Wang, J. Yu, and Q. Xie, "Security analysis of a single sign-on mechanism for distributed computer networks", *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 294–302, 2013.

[194] K. Wang, "Full version of sso protocol models", `https://github.com/Chriswangkl321/TIFS_SSO_models.git`, visited in April 2021.

[195] K. Wang, G. Bai, N. Dong, and J. S. Dong, "A Framework for Formal Analysis of Privacy on SSO Protocols", in *Security and Privacy in Communication Networks (SecureComm)*, 2018, pp. 763–777.

[196] K. Wang, J. Zhang, G. Bai, R. Ko, and J. S. Dong, "It's Not Just the Site, It's the Contents: Intra-domain Fingerprinting Social Media Websites Through CDN Bursts", in *30th The Web Conference (WWW)*, 2021.

[197] K. Wang, Y. Zheng, Q. Zhang, G. Bai, M. Qin, D. Zhang, and J. S. Dong, "Assessing Certificate Validation User Interfaces of WPA Supplicants", in *Under Review*, 2021.

[198] R. Wang, S. Chen, and X. Wang, "Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services", in *IEEE S&P*, 2012.

[199] R. Wang, Y. Zhou, S. Chen, S. Qadeer, D. Evans, and Y. Gurevich, "Explicating sdks: Uncovering assumptions underlying secure authentication and authorization", in *Proceedings of the 22Nd USENIX Conference on Security*, 2013, pp. 399–414.

[200] T. Wang, "High precision open-world website fingerprinting", in *IEEE S&P*, 2020, pp. 231–246.

[201] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting", in *USENIX Security*, 2014, pp. 143–157.

[202] T. Wang and I. Goldberg, "Improved website fingerprinting on tor", in *WEPS*, 2013, pp. 201–212.

[203] T. Wang and I. Goldberg, "On realistically attacking tor with website fingerprinting", *PoPETs*, pp. 21–36, 2016.

[204] WatchGuard, "Internet security report", https://www.watchguard.com/wgrd-resource-center/security-report-q3-2018, visited in April 2021.

[205] "WireGuard", https://www.wireguard.com, visited in April 2021.

[206] "WolfSSL", https://www.wolfssl.com, visited in April 2021.

[207] O. C. Workgroup, "OAuth Core 1.0 Revision A", http://oauth.net/core/1.0a/, visited in April 2021.

[208] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: An overview and application in radiology", *Insights into Imaging*, pp. 611–629, 2018.

[209] J. Yan and J. Kaur, "Feature selection for website fingerprinting", *PoPETs*, pp. 200–219, 2018.

[210] Y. Zhang, M. Roughan, C. Lund, and D. Donoho, "An information-theoretic approach to traffic matrix estimation", in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003, pp. 301–312.

[211] X.-y. Zhou, Y. Lee, N. Zhang, M. Naveed, and X. Wang, "The peril of fragmentation: Security hazards in android device driver customizations", in *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 409–423.

[212] J. Zote, "Sprout social", https://sproutsocial.com/insights/social-media-statistics/, visited in April 2021.