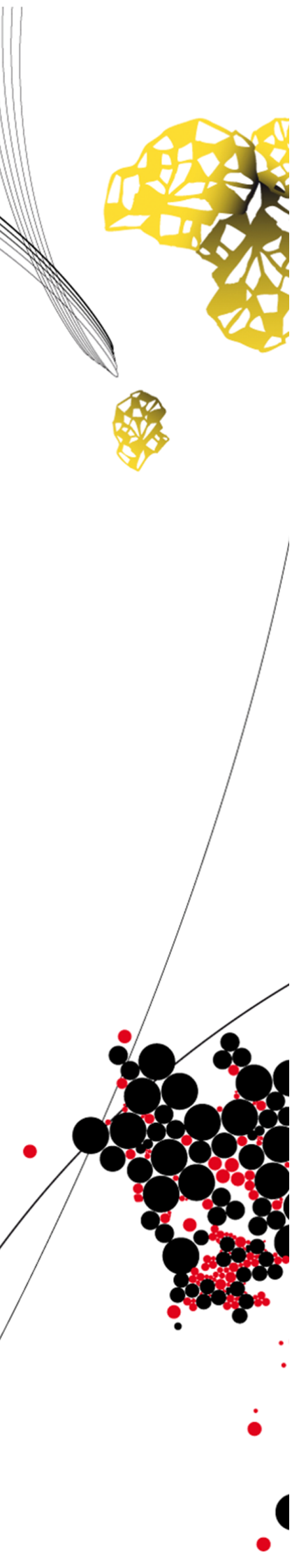

UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

Balancing Security and Usability in Web Single Sign-On

Ahmed Bakry Helmy Ahmed

M.Sc. Thesis
June 29, 2020

Supervisors:
Dr. A. Peter (UT)
Dr. A. Sperotto (UT)
Prof. dr. S. J. Mullender (UT & Cisco)

UT Research Chair:
Services and CyberSecurity (SCS)

Company:
Chief Technology and Architecture Office (CTAO)
Cisco Systems Netherlands,
Haarlerbergweg 13-19, 1101 CH
Amsterdam-Zuidoost

Acknowledgement

Foremost, I would like to start by expressing my sincere gratitude to my supervisor Prof. Sape Mullender for his continuous support during my thesis. He has always been a great professor, supervisor, manager and college. I would like to thank him for all the valuable arguments we had throughout my thesis. During the first two months of my thesis, I was extremely stressed and worried as I didn't have a clear path to follow. He used to say, while smiling, "Ahmed, don't worry I have my complete confidence in you. You will be more than fine. I am also sure you will finish your thesis in a very short time.". He was right, I shouldn't have been very worried. However, being worried turned out to be very positive when I received the green light after just 4 months from starting my thesis. However, this wouldn't have happened without his guidance.

I would also like to thank my supervisor Dr. Andreas Peter for his continuous feedback which was extremely helpful. I also thank him for his guidance and all the questions he was asking which helped me in making a good plan for my thesis. I would also like to thank Dr. Anna Sperotto for participating in my thesis committee.

Besides my supervisors, I thank my colleges: Peter Bosch, Jeffrey Napper, Alessandro Duminuco and Julien Barbot for all the good times I spent at Cisco. They have always been supportive on both the personal side and the professional side.

Last but not least, I would like to thank my family and my friends who have always kept me motivated during my quarantine time at home.

Abstract

Web Single Sign-On (SSO) systems enable users to access multiple enterprise services by authenticating once to the enterprise Identity Provider (IdP). Although SSO improves usability, the current SSO implementations work on a per-user-application basis. This means that users still need to authenticate for every application they use on the same device. Moreover, Web SSO systems rely on Multi-Factor Authentication (MFA) to achieve a high-level of security, while requiring MFA for each used application leads to frequent user annoyance. The thesis addresses this problem by proposing a novel method for synchronizing authentication information across local applications. This is achieved by using a local agent that acts as a local IdP for local applications and as a local application to the main IdP. We propose three design choices and evaluate each of them in terms of security, usability and deployability. Moreover, a prototype for the first two designs has been implemented. The results show that our solution improves the usability of Web SSO without compromising security.

Furthermore, current SSO implementations do not provide a mechanism for the IdP to synchronously push security updates (e.g., a change of authorization) to Service Providers (SPs) without terminating the session. We address this problem by proposing a novel SSO authentication mechanism that allows the IdP to synchronously update the current user authorization with no need for session termination (if not necessary). The proposed solution works such that the SP signals a Call-Back URL (CBU) to the IdP when exchanging the first SSO authentication messages. This CBU is used by the IdP to synchronously push updates towards the SP. Examples are revoking an old authorization, renewing an about-to-expire one, triggering MFA, limiting or extending user's access rights. Finally, we evaluate our solution and show that the integrity of the CBU is ensured.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Despite the enormous efforts being made to address security issues, the number of companies suffering from security breaches is still increasing [1]. Developing up-to-date security solutions does not automatically result in increased security. Reaching the desired security goals depends on whether the solutions developed are accepted and thus used by users. This problem is commonly known as the security-usability tradeoff [2]. For example, strong passwords need to be long, random and updated regularly, making them hard to remember. Moreover, people often need to remember multiple passwords for different services, devices or applications. As a result, people tend to choose easy passwords or write their passwords on a note, which puts security at risk. A recent study of 5 million exposed passwords showed that the top two used passwords were “123456” and “123456789” [3]. Furthermore, strong passwords can still be stolen through Shoulder Surfing attacks (i.e., observing someone typing their password), Trojan Horse attacks, or Phishing attacks, which are number one in data breaches [4]. Attackers may use fake emails and websites that are close to being legitimate to deceive people and steal their passwords. Therefore, additional authentication factors, such as the use of hardware tokens or phones, are necessarily required to overcome these shortcomings. While security can be improved, MFA has a notable effect on usability. It involves explicit additional user interaction, which may cause occasional user annoyance, especially when it is needed for each service.

Enterprises address this problem by providing their users with a Web SSO experience. Web SSO is an SSO system that involves a web browser or an application that exchanges HTTP-based identity-related information between the system’s participating entities. Web SSO allows the user to remember only one set of credentials to access different unrelated services managed by the same authentication authority (i.e., the SSO server). MFA can therefore be deployed with less friction. However,

the current SSO mechanism works on a per-user-application basis. This means that each time a different application is used, the user needs to authenticate to the SSO server. Enterprise users use multiple user agents to access the enterprise services such as mailer apps, VPN clients, video conferencing apps, company phones, or web browsers. Thereby, SSO still involves unnecessary extra manual user efforts. We address this issue by proposing an improved SSO experience that reduces the number of explicit authentications when accessing enterprise services. Our solution improves the usability of Web SSO without compromising security. Furthermore, SSO sessions may last for hours or even several days, especially on mobile devices. Authorizing users is based on dynamic data such as device location, device health, IP address, and user posture. Today's SSO systems determine access authorization only at the start of the session (i.e., at the time of authentication) and therefore do not enable dynamic access authorization. In addition, when a security incident of some sort is detected (e.g., an app vulnerability has been discovered), it is signaled asynchronously towards the service. Administrators must then manually examine the situation, which delays the mitigation process. In this thesis, we address this shortcoming by proposing a novel SSO mechanism for dynamic access authorization.

1.2 Thesis Goal

The aim of this thesis is to solve the two problems mentioned in Section 1.1; to improve the usability of SSO without compromising security; and to enable SSO frameworks to cope with changes in authorization and thus to improve the security of SSO systems.

1.3 Research Questions

Based on the problem statement and the aim of this thesis, the following main research question is derived:

How can we establish a balance between security and usability in Web SSO systems?

In order to answer this question, we divide it into the following sub-questions:

Q1 How well do the state-of-the-art web authentication systems maintain a balance between security and usability? And which of the proposed solutions can be deployed for Web SSO without compatibility issues?

Q2 How can the number of explicit user authentications in Web SSO be reduced without compromising security?

Q3 How can the change of authorization be adapted during an active session in Web SSO?

1.4 Thesis Contribution

This thesis contributes to the state-of-the-art by:

(i) Reviewing the state-of-the-art user-to-web authentication systems in terms of security, usability and deployability.

(ii) Proposing a novel SSO authentication mechanism that reduces the number of explicit user authentications when using multiple applications on the same device. We call this novel mechanism a true per-device SSO. Moreover, We introduce three design alternatives and evaluate each of them in terms of security, usability and deployability.

(iii) Proposing a novel SSO authentication mechanism that allows SSO systems to react dynamically to changes in user authorization without the need for terminating the session. Our solution also enables MFA to be triggered during an active session, which paves the way for continuous authentication in SSO systems.

Two US patents have been submitted by Cisco Systems, Inc., for the two solutions described in chapters 4 and 5. Note that at the time of writing this thesis, the first submitted patent, which implements the change of authorisation procedures in SSO, was accepted for filing. While the second patent, which provides a true per-device SSO experience, is currently under review by the Patent Committee at Cisco. The abstracts of the two patents are attached at the end of this thesis.

1.5 Research Method

The research methods used in this thesis to answer each of our research questions are summarized in Table 1.1. The first sub-question is answered through a literature analysis and an evaluation of the different authentication techniques used in the reviewed papers. While, the second sub-question comprises a design of three possible solution alternatives with an evaluation in terms of security, usability and deployability with a prototype of the first two designs. Finally, the third sub-question is answered by designing a new SSO authentication mechanism and an evaluation of the proposed design.

Table 1.1: Research method used to answer the defined sub-questions

Research Question	Research Method	Chapter
Q1	Literature analysis and evaluation	Chapter 3
Q2	Design, evaluation and prototype	Chapter 4
Q3	Design and evaluation	Chapter 5

1.6 Thesis Outline

The remainder of this thesis is structured as follows: in Chapter 2, we start with a background information explaining the terms used in the thesis. In Chapter 3, a comprehensive literature review on related web authentication systems is conducted to discuss how well each system maintains a balance between security and usability. Chapter 4 describes our novel authentication mechanism to improve the usability of Web SSO without compromising on security. This chapter also includes a comparison of our three different solution designs. Chapter 5 describes how the change of user's authorization can dynamically be adjusted in Web SSO frameworks without the need for terminating the session. Finally, Chapter 6 presents the conclusion, thesis limitations and future work.

Chapter 2

Background

In this chapter, we provide a general background on the different terms used throughout the remainder of this thesis. Moreover, we give an overview of today's commonly used Web SSO protocols. More specific background information and related work are presented throughout the thesis.

2.1 Authentication and Authorization

2.1.1 User Authentication

Authentication is the first line of defense against unauthorized access attempts. It was early defined by Lamport et al. [5] as the process where "a user identifies himself by sending x to the system; the system verifies his identity by computing $F(x)$ and checking that it equals the stored value y". This early definition is still valid today, although different authentication factors can be used. In this section, an overview of the methods used for user authentication is provided and the terms included in this study are further explained.

Modes of Authentication

There are different authentication modes for web authentication, depending on the authenticating entity and the location of the verifier. These modes are shown in Figure 2.1 and explained as follows:

(i) User-to-device: In this mode, the user authenticates to a local device (e.g., smartphone) by typing a password or via physical biometrics (e.g., fingerprints). The device then verifies the received credentials and grants the user access accordingly.

(ii) Device-to-web: In this mode, the device authenticates itself to the remote server by submitting a signed assertion using a locally stored cryptographic

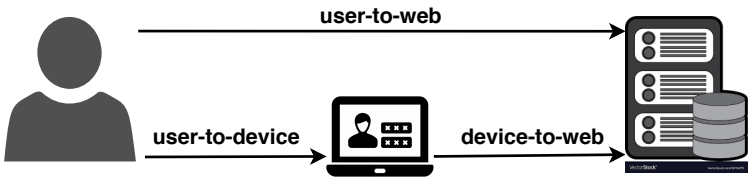

Figure 2.1: User-to-device, device-to-web and user-to-web authentications

key (e.g., a device certificate). This operation is usually invisible to the user and involves no explicit user interaction.

(iii) User-to-web: The standard method for authenticating users on the web is done by typing a password in a web form, which is submitted to the web server over the internet for verification. Although the password is entered on a local device, this device is a passive object that does not perform any verification on behalf of the server. Some user-to-web authentication schemes consist of two stages; user-to-device then device-to-web. For example, a banking app may authenticate the user via biometrics, which unlocks a locally stored key used by the application to authenticate itself to the server.

Types of Authentication Factors

There are different types of authentication factors that people can use to prove their identity. These factors are shown in Figure 2.2, and explained as follows:

(i) Knowledge-based factor (something the user knows): This factor relies on the knowledge of a secret such as a PIN, a password or security questions.

(ii) Ownership-based factor (something the user owns): The possession of physical objects such as smartphones, smart cards, or physical tokens.

(iii) Physical biometrics (something the user is): Examples are fingerprints, facial recognition and Iris.

(iv) Behavioural biometrics (something the user does): Examples include mouse and keystroke dynamics, gait analysis, applications usage and mobility patterns.

Single-Factor vs Multi-Factor Authentication

Knowledge-based factor was the first to be used due to its simplicity and ease of use. However, it has been shown to be vulnerable to multiple attack vectors and

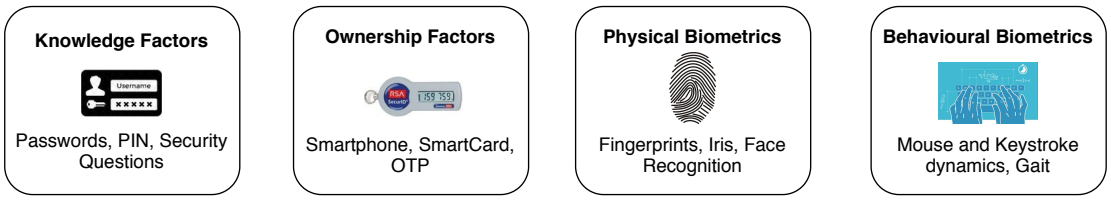

Figure 2.2: Types of authentication factors

insufficient in proving the user's identity [6]. Other authentication factors are usually used for stronger authentication to overcome this shortcoming. Depending on the number of (different) factors, authentication can be categorized as:

(i) Single-Factor Authentication (SFA): The use of one factor such as a username/password pair.

(ii) Two-Factor Authentication (2FA): The use of two different factors such as passwords and biometrics.

(iii) Multi-Factor Authentication (MFA): The use of two or more different authentication factors.

Implicit vs Explicit Authentication

User authentication can further be categorized in terms of the amount of effort made by users to prove their identity as:

(i) Implicit authentication: The authentication process is transparent to the end user. For example, the system may authenticate users when they approach their devices (i.e., proximity) or by evaluating their typing behaviour.

(ii) Explicit authentication: The authentication process involves an explicit user interaction such as typing a password or answering a security question.

Start-of-session vs Continuous Authentication

User authentication can be categorized based on the time when the system verifies the identity of the user as:

(i) Start-of-session authentication: The user is authenticated once at the beginning of each session. After a successful authentication, the system provides the user with an identifier in the form of a session cookie. As long as the cookie is valid, the user is not asked for authentication.

(ii) Continuous authentication: An authentication mechanism that periodically verifies the user's authenticity throughout the session. Continuous authentication systems use behavioural dynamics for implicit authentication and trigger MFA when there is sufficient doubt about the identity of the user.

2.1.2 User Authorization

Authorization is the process of granting the user access to the requested resource based on the system's predefined policies and the user's role. It happens as a result of valid authentication by consulting a database containing user access rights. It specifies which resources and services users may have access to after proving that they are, in fact, who they claim to be. For example, bank customers can set up an online account to access the online service of the bank; however, the system must ensure that each customer can only access his or her account and not other customers' accounts.

2.2 Single Sign-On

Single Sign-On (SSO) [7] is an umbrella term for managing the identity and access rights of users by a single trusted entity (i.e., the SSO server). It allows users to access multiple services or applications by authenticating to this trusted entity only once. It was first introduced as in an enterprise solution that gives users access to multiple enterprise applications with one time login. However, over time, its aim has changed from enterprise to the web. With SSO, users do not need to remember more than one set of credential (i.e. username and password) stored in a unified identity management database. Without SSO, each website needs to maintain its own database of user credentials, and users need to log in whenever they need access to another service. SSO therefore improves usability by reducing the number of authentications. While it improves security as users need to remember only one password to access multiple services that they can choose to be strong.

2.2.1 SSO Terminology

This section explains the common terms and definitions used in SSO authentication protocols. The following are the entities participating in the SSO authentication flow:

(i) Identity Provider (IdP): An entity that is trusted by service providers and is responsible for authenticating users and storing their identity information. After successful authentication, the IdP issues an assertion that contains the identity

attributes of the user, which are used by the service provider to identify the user.

(ii) Authorization Server (AS): An entity responsible for managing user access rights (i.e., authorization) and creating access tokens. Note that the IdP can also function as an authorization server when it issues the access tokens.

(iii) Service Provider (SP): An entity that provides a certain service to users, which is protected by a security guard. When the user sends an access request to the SP, the security guard requests an authorization from the user.

(iv) User Agent (UA): Software running on the user's device. It can be a browser (external or embedded in a local app) or a local native application. It is used by the user to communicate with the web servers.

Note that other terms for the same entities may also be used by different SSO protocols such as OAuth2.0. However, all Web SSO authentication protocols share the same general terminology as defined in this section.

2.2.2 Cookies vs Tokens

Cookies

The Hypertext Transfer Protocol (HTTP) [8] is a stateless protocol, in which every request is served independently by the web server. In other words, there is no link between requests sent over the same connection. Although HTTP is stateless, it is not sessionless. Netscape [9] extended HTTP with a mechanism that allows web servers to keep track of the status of the user's connection. This mechanism works by communicating session identifiers with each request sent to the server. When the user authenticates to the server, it creates a unique identifier (i.e., a cookie) and sends it to the browser in the HTTP set-cookie parameter. The browser then saves this cookie and includes it in every request it sends to the same web server. The server uses this identifier, which is stored in the cookie, to check the identity information and authorization associated with the user. Moreover, an HTTP cookie may be sent to only the web server that created it (i.e., a host-only cookie) or to a number of sub-domains by specifying the domain attribute of the cookie. It is also important to set the secure attribute to force the browser to send the cookie securely over HTTPS. HTTP Cookies are used in SSO systems to continuously check the authenticity of the user and thus avoid the need for re-authentication when the user moves from one service provider to another.

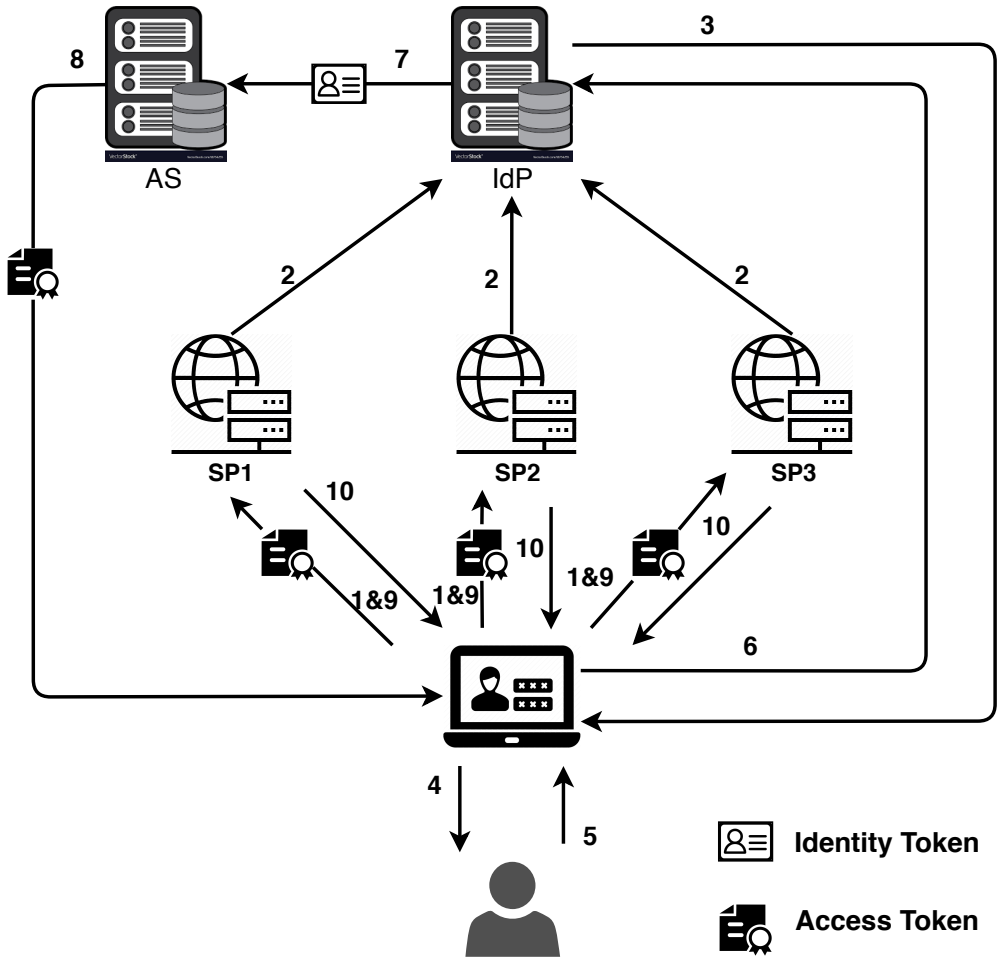

Figure 2.3: Single Sign-On Authentication

Tokens

SSO tokens are objects communicated between the IdP and the SP (via the UA) and contain the user's identity and authorization attributes. The information contained in the token is either stored in the server's local storage and accessed by a reference stored in a cookie, or in the cookie itself. Moreover, tokens differ in the way they are secured, and in their notations (e.g., JSON, XML, etc.) according to the used SSO solution. The access token may directly carry the attributes, and must therefore be signed by the IdP (e.g., SAML tokens) or may contain a reference to an internal database containing the attributes (e.g., OAuth 2.0).

2.2.3 SSO Authentication Flow

The general SSO authentication flow, as shown in Figure 2.3, works as follows:

1. The user (via the UA) tries to access a resource at the SP. If a valid access

token exists, go to step 9.

2. The SP redirects the UA to the IdP for authentication.
3. The IdP asks the UA for login credentials.
4. The UA presents the login page to the user.
5. The user enters the login credentials.
6. The UA sends the login credentials to the IdP.
7. If the IdP is satisfied about the user's identity, it sends an identity token to the AS. Otherwise, repeat step 3.
8. The AS checks the user's access rights, creates an access token, and sends it to the UA.
9. The UA tries to access the resource using the access token.
10. the SP grants access to the requested resource.

2.2.4 Web SSO Frameworks

This thesis is targeted for user-to-web authentication; therefore, we focus on the SSO solutions used for the web. Moreover, we focus on the most commonly used Web SSO frameworks such as SAML, OpenID Connect and OAuth2.0 [10]. There are still other SSO solutions, such as Kerberos [11]. However, Kerberos is mostly used in network authentication systems and is barely used for Web SSO. Therefore, it is discarded from the thesis.

Security Assertion Markup Language (SAML)

SAML [12] is an XML-based SSO framework developed by the OASIS Security Services Technical Committee (SSTC). It provides authentication and authorization by exchanging security information across multiple business partners in the form of SAML assertions. These assertions are signed, and if the signing entity is trusted, the SAML assertions can be trusted too, also across security-domain boundaries. SAML is supported by a number of service providers such as Amazon, Google, Salesforce, Box, and many others. The SAML standard defines exact rules and syntax for handling these assertions. The roles (i.e., entities) involved in the SAML authentication process are defined in Section 2.2.1. SAML supports two types of binding for the communication of assertions between the IdP and the SP; the HTTP

redirect binding and the HTTP POST binding. The redirect binding is used for communicating short SAML messages (i.e., SAML requests), while long SAML messages (i.e., SAML responses) are communicated using the HTTP POST binding. Security in SAML depends on the underlying trust model, which depends on the key management infrastructure. SAML messages are secured by means of an XML signature, and this is considered true only if the keys used in the exchange are trusted. The signature is intended to provide message integrity, while confidentiality can be achieved through end-to-end encryption (e.g., SSL and/or TLS).

Open Authorization 2.0 (OAuth2.0)

OAuth2.0 is an SSO protocol used for authorization purposes and not for authentication. However, a valid authorization implies that the user is indeed authentic. Authorization in OAuth 2.0 is represented by an access token that contains access rights and not identity information. On the other hand, authentication is handled using the OpenID Connect framework [13], which is an identity layer on top of OAuth 2.0. It introduces another token called an identity token, which contains the user's identity attributes. Similar to the SSO terminology defined in Section 2.2.1, the OAuth 2.0 authorization flows involve the following entities:

(i) Resource Owner: An entity that can grant access to the owned resource. This is typically the end user who owns the requested resource.

(ii) Client: The application code requesting access to a certain resource on behalf of the end user. The client may exist on the server (i.e., a web application), on the device (i.e., desktop/mobile application), or on the browser itself.

(iii) Resource Server: The service provider (i.e., similar to SAML SP) which hosts the protected resources.

(iv) Authorization Server: The identity provider which authenticates the end user and issues authorization tokens.

(v) User Agent: A browser or a native application used by the end user to interact with the client.

The OAuth 2.0 standard defines various authorization flows according to the location of the client. In this thesis, we consider the code authorization flow in which the client is a web or a desktop application interacting with the website. The Implicit Flow, Resource Owner Flow and Client Credential Flow are of less interest to this research and are therefore discarded. In the Code Authorization Flow, the AS sends an authorization code to the UA. The UA then forwards this code to the client that

uses it to request the access token from the AS. This is different from the authorization steps described in Section 2.2.3 as the access token is not issued directly. In OAuth 2.0, bearer tokens (e.g., the authorization code), which are used to obtain access tokens, are sent as unsigned URL parameters. In order to ensure that the access token can only be obtained by a legitimate client, the identity token is also sent to the AS with the authorization code (via HTTP POST). Security in OAuth 2.0 therefore depends on the protocol implementation (e.g., the used authentication protocol) and relies on the security of TLS.

Chapter 3

Overview of Less-Effort MFA Systems

3.1 Motivation

The main thesis goal is to explore ways to establish a good balance between security and usability in Web SSO systems. These systems rely on MFA for a higher level of security, usually requiring manual user interaction (e.g., using an OTP generator). Multiple authentication solutions have been proposed to achieve this balance by requiring explicit user interaction only when there are sufficient doubts about the identity of the user. Examples of such solutions are proximity-based or risk-based authentication. In this chapter, we discuss how this balance is addressed in the state-of-the-art user-to-web authentication systems, and hence answer the first research sub-question. Moreover, based on the literature review, we conclude by making recommendations on how this balance can be established in Section 3.4.

3.2 Proximity-based Authentication

Proximity-based web authentication systems use the proximity of two different devices (e.g., a smartphone and a laptop) as a second transparent authentication factor. The system determines proximity by cross-checking the information it receives from both devices. In this section, we evaluate the security and deployability of existing proximity-based authentication schemes (i.e., Bluetooth, WiFi, and ambient sound). Other techniques, such as comparing the GPS coordinates of the two devices [14], require the use of special hardware (i.e., GPS sensors) and are therefore not considered.

3.2.1 Bluetooth

PhoneAuth [15] is a proximity-based transparent 2FA solution that leverages Bluetooth for short range communication between the user's computer and their phone. The system implements a challenge-response protocol between the remote server and the application running on the user's phone. First, the user logs in to the web server using a username and password pair. The server then issues a login ticket and sends it to the application via an API exposed by the computer's browser. When the ticket is received, the application signs it using the stored private key and sends the signed ticket back to the browser over an authenticated Bluetooth channel. Finally, the browser uses the ticket to complete the login process. PhoneAuth improves the usability of web authentication by introducing a transparent second authentication factor. However, it requires exposing a Bluetooth API by the browser, which is not yet supported. Furthermore, the security of the proposed solution relies on the authenticity of the Bluetooth channel. In the case of an unauthenticated channel, a co-located attacker (e.g., using a powerful antenna) will manage to log in using only the username and password despite the proximity check. However, the system is resilient to remote attacks, regardless of the authenticity of the channel.

3.2.2 Virtual WiFi

Shrivanian et al. [16] propose an alternative approach that uses WiFi for proximity-based authentication. Similar to PhoneAuth, a challenge-response protocol is implemented between the remote server and the phone (via., the computer's browser). Their solution requires both devices to be connected to the same wireless channel. The authors use a special software that creates a virtual, therefore secure, WiFi connection between the two devices. The system is resilient to both remote and co-located attacks and adds a significant usability improvement without compromising security. However, it still requires the use of special software (i.e., to create a virtual WiFi), which needs to be installed on every newly used device.

3.2.3 Ambient Sound

Sound-Proof [17] is a proximity-based 2FA solution targeted for web authentication. The proximity is verified by comparing the ambient sound recorded by both devices. The user's computer, first, transmits the recorded sound to an application running on the user's smartphone over the Internet. The application then verifies that the received sound matches the one it has recorded and sends a signed assertion to the web server accordingly. It can also provide continuous authentication by periodically comparing the two ambient sounds to ensure the two devices remain

Table 3.1: Overview of the reviewed proximity-based user-to-web authentication schemes

Work	Approach	Security	Usability	Deployability
[15]	Bluetooth	Resilient to both remote and co-located attacks when using an authenticated channel	Transparent 2FA	Requires a Bluetooth API to be supported by the browser
[16]	Virtual WiFi	Resilient to both remote and co-located attacks	Transparent 2FA	Requires a computer software to create a virtual WiFi connection
[17]	Ambient sound	Resilient to remote attacks only	Transparent 2FA	Requires HTML5 and WebRTC API to be supported

in close proximity. However, this may affect the phone's battery life. The authors show that their proposed solution is resilient to remote attacks, even if the attacker has sufficient knowledge of the user's environment. For example, by embedding a malicious application on the user's phone to record the ambient noise around the user. Moreover, replay attacks are prevented by time stamping the recorded sound. Although achieving significant usability improvement with an almost zero false rejection rate, Sound-Proof is not resilient to co-located attacks. Finally, the system requires HTML5 and the WebRTC API to be supported by browsers.

3.3 Context-aware Authentication

Context-aware authentication systems aim to improve the usability of MFA by relying on context checking as an implicit authentication factor. They are also known as risk-based or adaptive authentication systems, in which the number and type of explicit factors are determined based on an estimated risk score and preconfigured policies. In this section, we discuss the security of these systems with respect to the attacks listed in Section 3.3.2.

3.3.1 Contextual Features

The following contextual features are commonly used by risk-based authentication systems to build a behavioural profile for each user or group of users.

(i) Geolocation: The geolocation can be determined by several means (e.g., GPS,

WiFi, cellular triangulation, or IP2Geolocation databases [18]). Modern web browsers usually expose APIs through which the website can request users' consent to obtain their coordinates. However, geolocation accuracy depends on the used technique and whether the device is located indoors or outdoors.

(ii) Time features: Time plays an important role in authentication decisions. Examples of used time features are login time, time of day and day of week. Geolocation and time of day are usually combined to determine the geovelocity of the user. For example, it is considered suspicious when a user authenticates from Amsterdam, while the last login attempt was from San Jose two hours ago.

(iii) Browser fingerprinting: Web browsers reveal information about the device on which they run. Although this is mostly used for device-to-web authentication, risk-based authentication systems use this information to build a user-related device profile. However, this method of device identification is not very accurate, as most of the identified features can be spoofed and are not identical per device. Some of the revealed attributes are browser type and version, operating system, user-agent string, cookies, supported languages, installed fonts, screen resolution, and active plugins.

(iv) Behavioural features: Behavioural-based features (e.g., mouse and keystroke dynamics) are mostly used for continuous authentication. However, some risk-based authentication systems combine them with contextual features to improve security. The effect of behavioural-based features on usability is discussed in Section 3.4.

3.3.2 Attacker Model

In our security analysis, we assume that the attacker has managed to obtain the user's password. We exclude the ability of the attacker to record and imitate user behaviour (e.g., mouse and keystroke dynamics). Moreover, we exclude the case where the attacker steals the MFA device (e.g., YubiKey or phone). These are extremely powerful attacks and not considered in the analysis. The following attacks are considered to assess the security of the reviewed systems:

A1 Context guessing and spoofing: In this attack, the attacker attempts to impersonate the user by guessing and spoofing the contextual features (e.g., time of day, geolocation, etc.).

A2 Phishing and spoofing: In this attack, the victim visits a phishing website that captures both the password and multiple contextual features. The attacker

then attempts to spoof the captured attributes to impersonate the legitimate user.

A3 Session hijacking and context spoofing: In this attack, the attacker steals the session cookie by exploiting Cross-Site Scripting (XSS) vulnerabilities, or via an improperly configured HTTPS connection [19], and attempts to spoof the contextual features. Alternatively, the attacker can physically access the authenticated device.

3.3.3 Risk-based Authentication Systems

Rocha et al. [20] introduce a risk-based authentication system in which the user behaviour is represented by a set of events related to an executed activity. The system builds three profiles for each user; an explicit, implicit and session profile. The explicit profile is constructed during the user's first interaction with the system by explicitly collecting information from the user's device (e.g., calls and schedules). The implicit profile is created by processing users' actions and their spatio-temporal characteristics using a Vector Space Model (VSM) as in [21]. The session profile consists of the current status of the user (i.e., context) and the last performed actions. Both the session and the implicit profiles consist of the used device, location, timestamp, the requested application and its sensitivity. The system calculates the degree of similarity between the two profiles using a VSM permutation model to classify a login event as normal, abnormal or suspicious according to predefined thresholds. When an abnormal or suspicious event is detected, a security question is presented to the user as an additional authentication factor. The success of A1 and A2 depends on the used contextual features, which are not clearly indicated in the paper. For example, the paper does not indicate how the device is being identified. Moreover, it depends on the system's predefined risk thresholds (i.e., high threshold leads to high success probability). We therefore consider the systems to be partially resilient to A1 and A2. Since the authenticity of the user is checked only at the start of the session, the system is not resilient to A3. Furthermore, the system relies on challenge questions, which are not a strong form of MFA.

Bakar et al. [22] propose an adaptive context-aware authentication system built on top of a unified authentication platform (i.e., an SSO system). The system follows a rule-based approach to classify login activities as normal or abnormal based on predefined trust-level thresholds. An event is considered to be normal if it has occurred more than 10 times in the last 14 days and if its frequency of occurrence is more than 30%. Each login event is represented by the following attributes: time of day, geolocation (i.e., IP to geolocation), the requested application, browser type and operating system. Each attribute is assigned a preconfigured weight that con-

Table 3.2: Overview of the reviewed risk-based authentication systems

Work	Features	Algorithm	Security
[20]	Device, location, timestamp, application, application constraints	Vector Space Model (VSM)	Partially resilient to A1 & A2, non-resilient to A3
[22]	Time of day, geolocation, application, browser, operating system	Static rules	Partially resilient to A1 & A2, non-resilient to A3
[23]	IP address, user agent	Logistic Regression	Less resilient to A1 & A2, non-resilient to A3
[24]	Browser, useragent, device model, software version, colour depth, language, screen resolution, plugins (client-side); IP address and range, geolocation, time of access and request headers (server-side)	Hoeffding trees	Resilient to A1 & A2, non-resilient to A3
[25]	login time, IP address, country and useragent string; mouse and keystroke dynamics	Random Forest classifier	Resilient to A1, A2 & A3

tributes to the user's attribute penalty. The system deploys four different authentication methods; password, OTP token, smsPIN, and digital certificate. Each method is assigned a trust score according to the strength of the presented authentication factor. The final trust score is calculated by subtracting the user's attribute penalty from the trust score of the used authentication factor. The system presents an additional authentication factor when the final score is less than that is required by the requested application. For low-risk applications where passwords are sufficient, A1 and A2 are likely to succeed, as the used contextual features can be spoofed. However, for high-risk applications where a second authentication factor is presented, both A1 and A2 are not sufficient to gain access. Moreover, the system does not continuously verify the identity of the user and therefore it is not resilient to A3.

Freeman et al [23] conducted the first public assessment of risk-based authentication systems by designing Reinforced Authentication; a context-aware adaptive authentication framework. The system follows a statistical machine learning approach using two attributes; IP address and user agent string. Multiple features are extracted from these attributes, such as symbolic location (i.e., country and city),

IP address reputation, OS type and version and browser. The algorithm compensates for unseen events by using multiple smoothing techniques such as Backoff smoothing and Linear Interpolation. The algorithm is tested using a real life dataset from LinkedIn resulting in a recall of 89% at a false positive rate of 10%. The used attributes can easily be spoofed; however, spoofing the same IP address for two-way communication is not useful. The system is not resilient to an internal attacker sharing the same public IP address of the user's device. Moreover, a remote attacker may obtain an IP address from the same ISP to gain access. The system is therefore not resilient to both A1 and A2. Although the system aims to strengthen password-based login and not to completely get rid of MFA, A3 is likely to succeed due to the absence of continuous user authentication.

Preuveneers et al. [24] designed SmartAuth; a context-aware continuous authentication platform built on top of OpenAM. The system provides periodic context validation to continuously verify the user's authenticity by using Hoeffding trees [26]. The system collects two classes of features; client-side and server-side. The client-side features include browser, user agent, device model, software version, colour depth, language, screen resolution and plugins. While the server-side features are IP address, IP range, geolocation, time of access and request headers. For each attribute, the user must provide consent before it can be collected and used. Each attribute is assigned a dynamic score based on the obtained consent. The system preserves user's privacy by hashing the client-side fingerprints using a similarity-preserving hash function before they are sent over the network. To prevent phishing and replay attacks, the system adds counters and timestamps to the fingerprints before they are hashed. These fingerprints are compared to the last received fingerprints to check their validity. For an invalid fingerprint, an OTP (e.g., sent via SMS or email) is required as an additional authentication factor. Although hashing the device fingerprint with a counter makes the system resilient to both A1 and A2, this is inefficient for an attacker employing A3 to steal the session cookie. An attacker employing A3 is able to steal both the counter and the cookie.

Solano et al. [25] propose a risk-based authentication mechanism that combines browser fingerprinting and behavioural dynamics into one model. The former consists of login time, IP address, country and user agent string. While the latter is a combination of mouse and keystroke dynamics. Login features are converted into a vector of normalized probabilities computed by estimating the likelihood of each feature value and assigning 1 to the most likely one. Moreover, mouse and keystroke dynamics are combined into a single-feature vector. The system uses one separate Random Forest classifier for each vector. The results of the two classifiers are then combined into a single model using a linear convex combination of equal weight. The proposed model is evaluated using two real datasets; a browser fingerprinting

dataset from the banking domain, and The Wolf Of SSUTD (TWOS) dataset [27], which contains mouse and keystroke dynamics. The results show a reduction in the false negative rate (i.e., undetected attacks) at the expense of a high false positive rate. For a threshold between 0.2 and 0.25, the model achieves a false negative rate of 1.9%-3.5% at the cost of 35%-24.6% false positive rate respectively. In other words, one out of four legitimate users will need to preform 2FA. The authors argue that these detection rates are due to the used dataset, where 75% of the entries are attacks. The system is clearly resilient to both A1 and A2, as the attacker needs to capture and replay the user behavioural dynamics, which is considered extremely difficult. Furthermore, the user's authenticity is periodically verified, making the system resilient against A3.

3.4 Discussion

In this chapter, we have reviewed the state-of-the-art web authentication systems addressing the balance between security and usability to answer the first sub-question. Proximity-based authentication systems can achieve a good balance by establishing a secure and transparent second authentication factor. However, some deployability issues arises; both [15] and [16] require the use of special plugins that are not compatible with current browsers. On the other hand, [17] does not require the use of a special software, however it is not resilient to co-located attacks. A comparison between the three papers is shown in Table 3.1. Moreover, risk-based authentication systems address this balance by triggering MFA only when there are sufficient doubts about the identity of the user. These doubts are determined through a risk assessment by checking the current context against the reference profile of the user. However, the used contextual features are susceptible to spoofing. Preuveneers et al. [24] adds additional spoofing resistance by dynamically varying the used client-side contextual attributes and hashing them with a counter. However, an attacker stealing the user's device may succeed to authenticate only by using the stolen password. Furthermore, combining behavioural biometrics with contextual features creates resilience against the three defined attack vectors, such as in [25]. However, relying on behavioural biometrics may affect the desired balance, since they are very complex to deploy, sometimes turn intrusive, and may change over time. Therefore, the user of behavioural biometrics for context-aware web authentication systems is still an open issue. A summary of the reviewed risk-based authentication systems is given in Table 3.2.

To achieve a good balance between security and active user involvement, we recommend the use of contextual features alongside MFA. Although, the contextual features can be spoofed, they can still improve the security of SSO systems by

choosing an appropriate second authentication factor such as in [22]. In the desired system, guessing attacks can be mitigated by setting a maximum number of failed logins and using a stronger form of MFA accordingly. Phishing attacks can also be mitigated by using two (or more when necessary) different authentication factors (e.g., a password and a smartphone). The risk of session hijacking and context spoofing can be reduced by ensuring the use of proper TLS connections and educating users about safe browsing (i.e., to reduce the risk of XSS attacks). However, detecting device theft during an authenticated session without affecting the system's usability is still a challenge. People should be instructed not to leave their devices unattended without taking proper measures (e.g., logging out, turning the screen off, etc.). To improve usability (i.e., reduce active user involvement), we propose a novel SSO authentication mechanism to synchronize the authentication information (i.e., session cookies) between multiple applications on the same device in Chapter 4. Finally, in Chapter 5, we propose a novel mechanism to dynamically adapt the user's authorization according to the user's posture. The proposed mechanism also allows MFA to be triggered during an authenticated session to reduce security risks when anomalies are detected.

Chapter 4

True Per-device SSO

4.1 Motivation

Enterprise SSO infrastructures enable users to authenticate with a single (trusted) entity (i.e., an IdP or SSO server). Once the user has authenticated for one service, via passwords and/or additional authentication factors, this authentication can serve for another service as long as the user has a valid session with the IdP. Users do not need to manually re-authenticate as they move from one service to another. However, the current SSO authentication mechanism only works on a per-user-application basis. This means that users can authenticate only once per session to access any of the services that they are authorized to use, provided that they use the same application. When using another application, the user must manually re-authenticate from the same device. In this chapter, we address this problem by proposing a novel, true per-device SSO experience for all applications on the user's device. True per-device SSO enables users to access the enterprise services from multiple applications with manual authentication only needed once per session. Our design goals are to improve usability (i.e., reduce the manual efforts involved when using multiple applications during a session) without compromising security (i.e., without introducing a new attack vector). Moreover, the proposed solution must be deployable on current SSO systems without requiring any modifications at service providers or client applications.

4.2 Related Work

Logan et al. [28] propose a method for synchronizing authentication information (i.e., session cookies) between multiple browsers. In their solution, a remote proxy server is used to manage cookies on behalf of browsers. However, this only works for one type of application (i.e., a browser) and therefore does not provide the user with a true per-device single-sign-on experience. Moreover, authentication cookies are

stored remotely and provided to the browsers when a request is received from the same IP address. This is clearly problematic as devices behind the same NAT box share the same public IP address.

Cox et al. [29] have improved usability of the Plan 9 operating system used in Bell Labs by introducing a central component called Factotum. It is a trusted agent that takes control of all security interactions on behalf of all local applications or servers on the machine on which it operates. Moreover, it protects itself from being paged out to disk, against being debugged, and provides a central location for storing secrets and running security protocols. Although Factotum meets our usability and security design goals, it is never used for SSO purposes. To the best of our knowledge, there is no study that provides the user with a true per-device SSO authentication experience.

4.3 Proposed Solution

Similar to Factotum, our solution includes a trusted agent running locally on the user's device that manages the authentication for the local applications. However, it stores only the SSO session cookies and not user secrets. These cookies are only valid for the duration of a session or when the user logs out. Depending on the design choice, the local agent may only interact with the local applications on behalf of the IdP. Alternatively, it may act as a proxy that interacts with the IdP on behalf of the applications. It therefore behaves as an IdP for local applications and as a trusted application for the IdP, so we call it a Local Identity Provider Agent (LIPA). For this to happen, we propose three design alternatives for our novel authentication mechanism. In designs I and II, the IdP is aware of the LIPA existence and neither the applications nor the service providers need to be modified. Moreover, in case the IdP code cannot be modified, we propose a third design that does not require any modifications at any of the three participating entities. The three solutions are presented and discussed in Sections 4.3.2, 4.3.3 and 4.3.4. In this chapter, We introduce the following new entity participating in the SSO authentication flow:

- LIPA: a local application that manages authentication and authorization information across multiple applications on the same device.

Since the thesis focuses on web (i.e., cookie-based) SSO protocols such as SAML and OpenID, the following two session cookies need to be distinguished:

(i) Identity Provider Cookie (IdPCookie): A session cookie that is installed on the UA by the IdP using the HTTP set-cookie header. It identifies the user's session with the IdP and is sent by the UA to the IdP within each subsequent request.

(ii) Service Provider Cookie (SPCookie): A session cookie that is installed on the UA by the SP using the HTTP set-cookie header and acts as a reference to the user's session information (e.g., authorization and identity attributes) stored at the SP.

In the proposed authentication mechanism, only the IdPCookie is synchronized between the applications by the LIPA. This avoids the need for additional manual authentication when using another application for the first time in a session.

4.3.1 Standard SSO Authentication

In this section, we present how the standard SSO authentication flow operates before introducing the LIPA. The flow is shown in Figure 4.1 and works as follows:

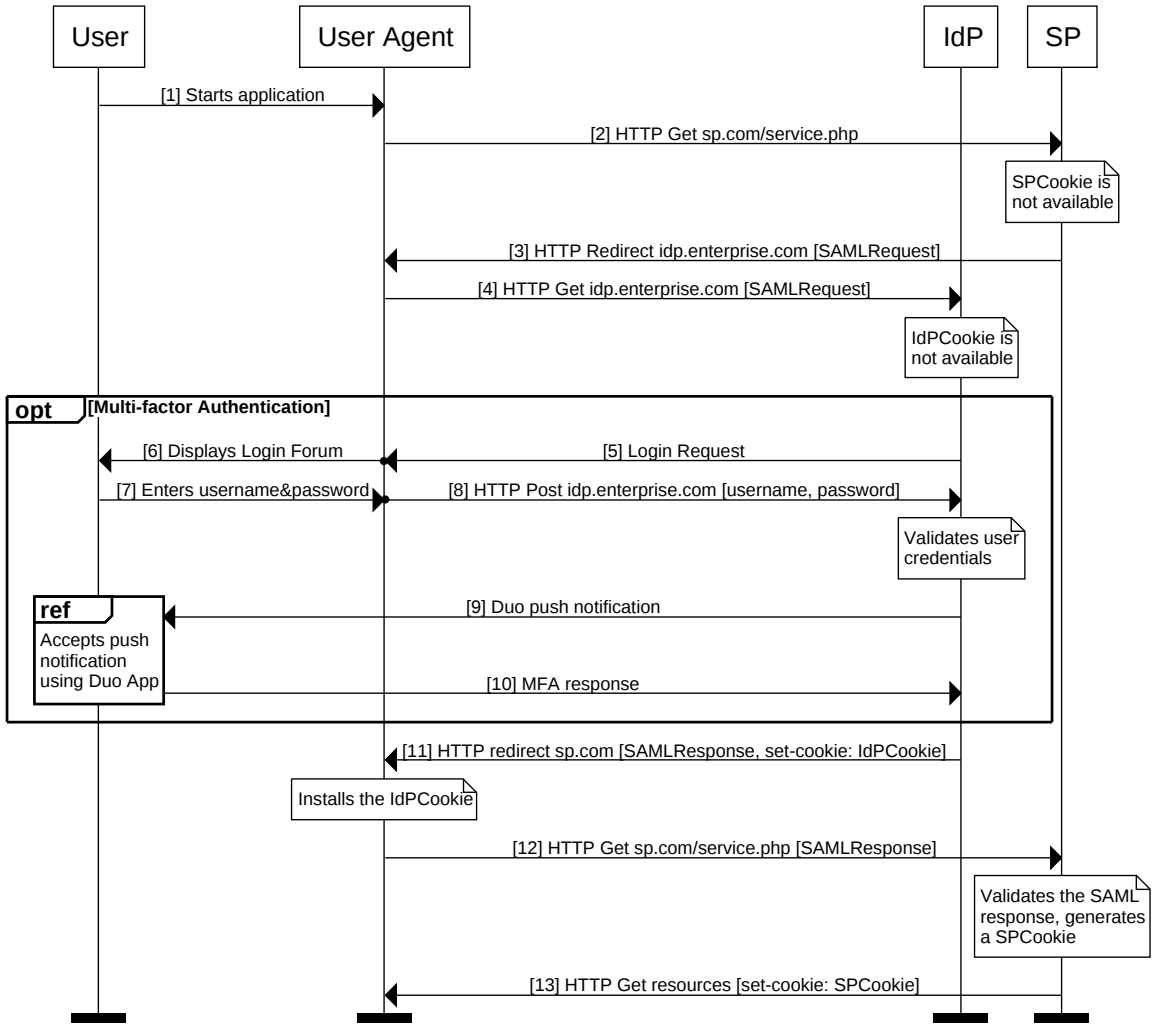

Figure 4.1: The standard SSO user authentication mechanism

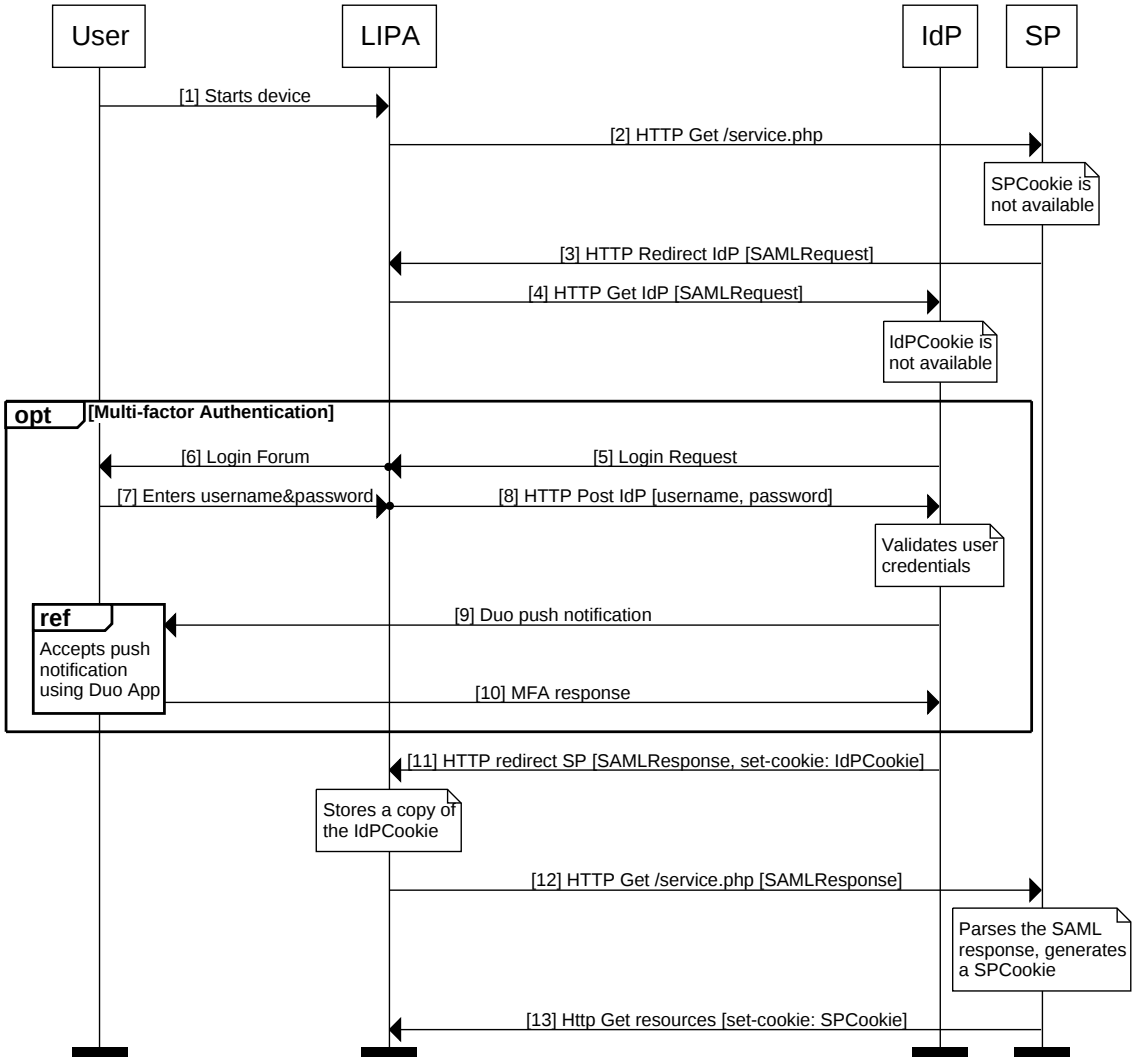

Figure 4.2: LIPA registration with the IdP (IdPCookie not available)

1. The user requests a resource by using a UA for the first time. This request is sent to the back-end server of the SP without an authorization grant (i.e., the SPCookie).

2. The SP observes the absence of the SPCookie and redirects the UA to the IdP with an encoded authentication request (e.g., in the form of a SAML request).

3. The UA forwards the request to the IdP.

4. First, The IdP checks the user's authentication status by validating the IdP-Cookie. In this scenario, the user is not yet authenticated. The IdP therefore initiates an authentication dialogue with the UA asking the user to authenticate (e.g., via password and/or MFA).

5. After a valid authentication, the IdP installs the IdPCookie on the UA that allows the UA to re-authenticate without resubmitting the password. Moreover, the IdP provides the UA with an authorization token (e.g., in the form of a SAML response) that allows the UA to authenticate to the SP, and finally redirects the UA back to the SP.

6. Before visiting the SP with the authorization token, the UA installs the IdP-Cookie, and then forwards the response to the SP.

7. Finally, the SP validates the authorization token, installs the SPCookie on the UA, which acts as a reference to the validated token, and replies with the requested resource.

4.3.2 Solution Design I

In the first design, the IdP is aware of the existence of the LIPA, which is started before any other application can be used (e.g., when the device is started). In order to obtain an IdPCookie from the IdP, the LIPA sends an access request to a service provided by the IdP. When the IdP is satisfied with the LIPA's identity (via, e.g., password and MFA), it generates an authorization token for the LIPA and provides an IdPCookie to the LIPA. The LIPA stores the cookie and uses it for its future interactions with the IdP. This results in the registration of the LIPA with the IdP as shown in Figure 4.2. When the IdP receives a request from another UA, it redirects the UA with the request to the LIPA to obtain the IdPCookie as shown in Figure 4.3. The modified SSO authentication flow becomes:

1. After the LIPA registration is complete, the user requests a service using a UA for the first time. The UA sends an access request to the back-end server of the SP. This time without the SPCookie.

2. The SP redirects the UA to the IdP with an encoded authentication request (e.g., a SAML request) as a parameter in the redirect URL.

3. The UA visits the IdP with the authentication request and without an IdPCookie. Then the IdP redirects the request to the LIPA.

4. The UA now sends the request to the LIPA, which is running on localhost.

5. The LIPA ensures that the request is sent by a local application and includes the authentication cookie it has in the HTTP set-cookie header. Finally, the UA is redirected back to the IdP.

6. The UA stores the cookie for the IdP domain (i.e., *.idp.enterprise.com) and includes it in the requests it sends to the IdP.

7. The IdP validates the cookie, provides the UA with an authorization token ,and redirects the UA back to the SP.

8. The SP validates the authorization token it receives from the UA, and replies with the requested resource as well as a reference to the authorization token (i.e., SPCookie).

This solution requires the LIPA to authenticate to the IdP before any other application can be used. The LIPA must therefore be started when the device is started. Moreover, the IdP needs a reliable way to identify the device from which it receives the request to verify the presence of the LIPA. For managed devices, the device can be identified using digital certificates. However, for unmanaged devices, there is no accurate method for checking the presence of the LIPA. A possible solution to overcome this problem is discussed in Section 4.3.3.

4.3.3 Solution Design II

In this solution, the LIPA does not need to register itself with the main IdP when the device is started. Instead, the IdPCookie is stored when the user requests a service using the first UA in a session. In this solution, the LIPA's URL is a subdomain of the main IdP (e.g., local.idp.enterprise.com). This URL is registered in DNS with the same IP address as of the IdP. In other words, there is a remote host running on the IdP server with the same URL as the LIPA's to avoid receiving an HTTP 404 response when there is no LIPA running on the user's device. This remote host only forwards the requests without storing or adding any cookies. There is therefore no need to assume the existence of the LIPA as mentioned earlier in solution I. When the LIPA is present on the device, it can be contacted by adding an entry in the /etc/hosts file that maps the LIPA's URL to localhost address (i.e., overriding DNS).

Figure 4.4 shows how the IdPCookie is made available to the LIPA when the first UA is started. In this scenario, the IdPCookie is not previously available neither at the UA nor the LIPA. The call flow in Figure 4.2 changes into:

1. When the user starts the UA, an access request is sent to the back-end server of the SP.

2. The SP generates an authentication request (e.g., a SAML request) and redirects the UA to the IdP.

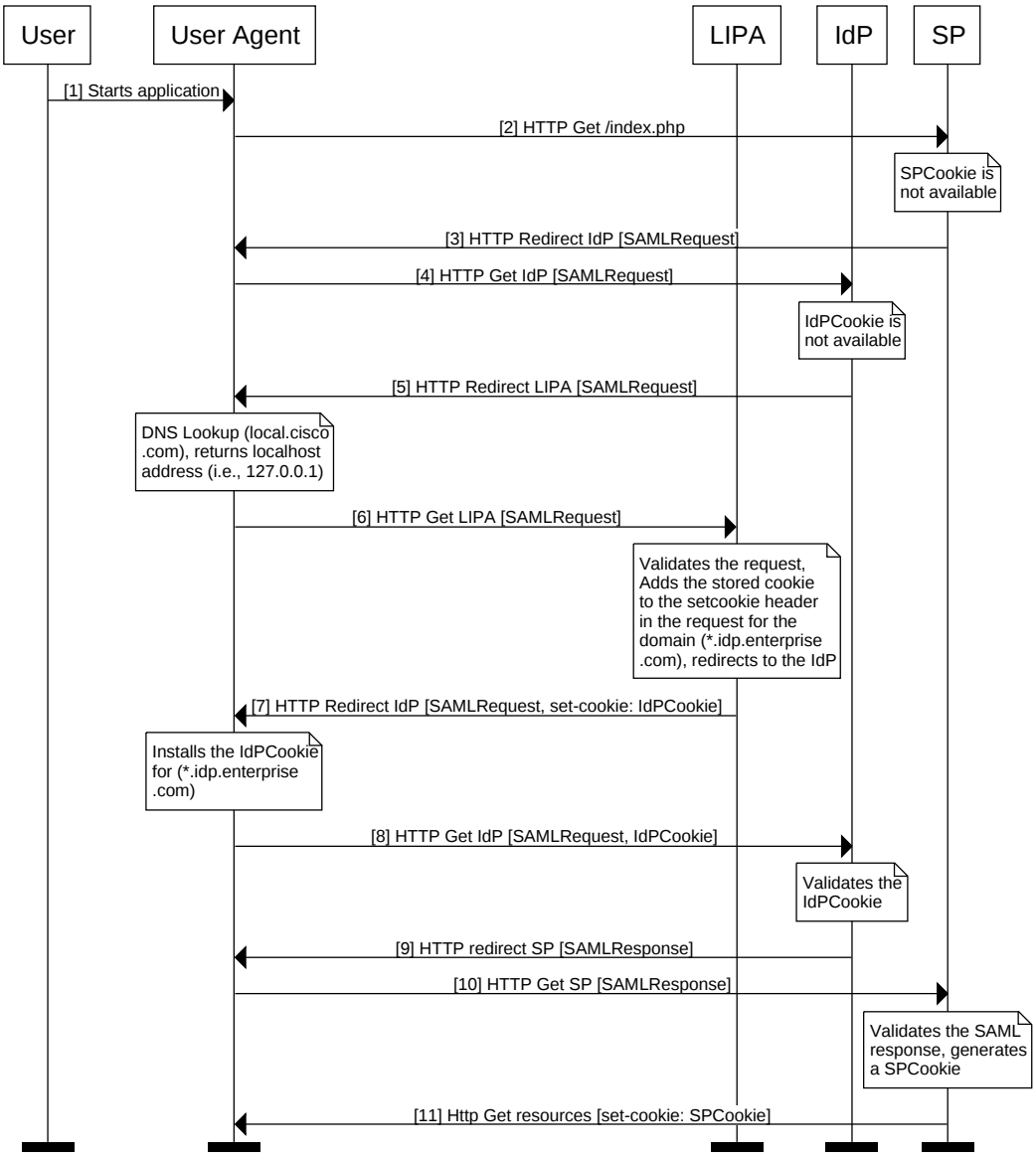

Figure 4.3: The LIPA provides authentication for a local application (Solution I)

3. The IdP observes the absence of the IdPCookie at the UA and redirects the UA to the LIPA with the received authentication request.

4. The LIPA verifies that the request is received from a local trusted application. Then redirects the UA back to the IdP with a URL parameter (e.g., from_local_-agent=true) to indicate that the request is from the LIPA. This is necessary to avoid redirecting the UA to the LIPA again to obtain the IdPCookie.

5. The IdP observes the absence of the IdPCookie at the LIPA. Therefore, it starts an authentication dialog with the UA.

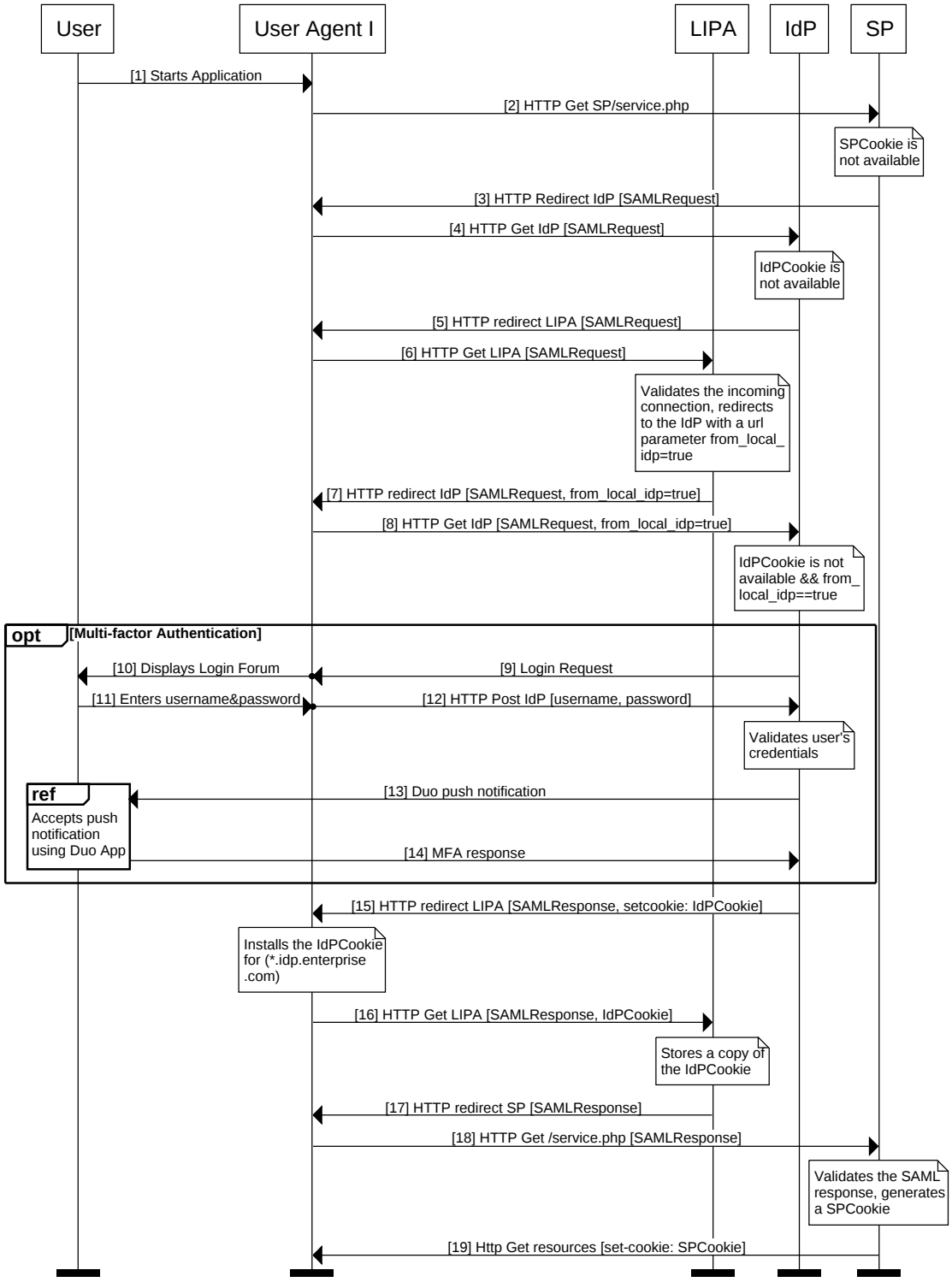

Figure 4.4: Solution Design II (IdPCookie is not available)

6. When the IdP is satisfied with the user's identity, it generates an authorization grant (e.g., a SAML response), installs an IdPCookie on the UA, and redirects the UA to the LIPA instead of the SP.

7. The LIPA stores a copy of the IdPCookie before redirecting the UA to the SP's Assertion Consumer Service (ACS) on behalf of the IdP. This is done by interpreting the relay state parameter in the response, and for saving an additional redirect from the LIPA to the IdP.

8. This time, the UA visits the SP with the authorization grant. The SP validates the response, creates a SPCookie, and provides the UA with the requested resource.

When the user starts another UA, also for the first time, the previously obtained IdPCookie is used to avoid manual authentication as shown in Figure 4.5. The modified call flow becomes:

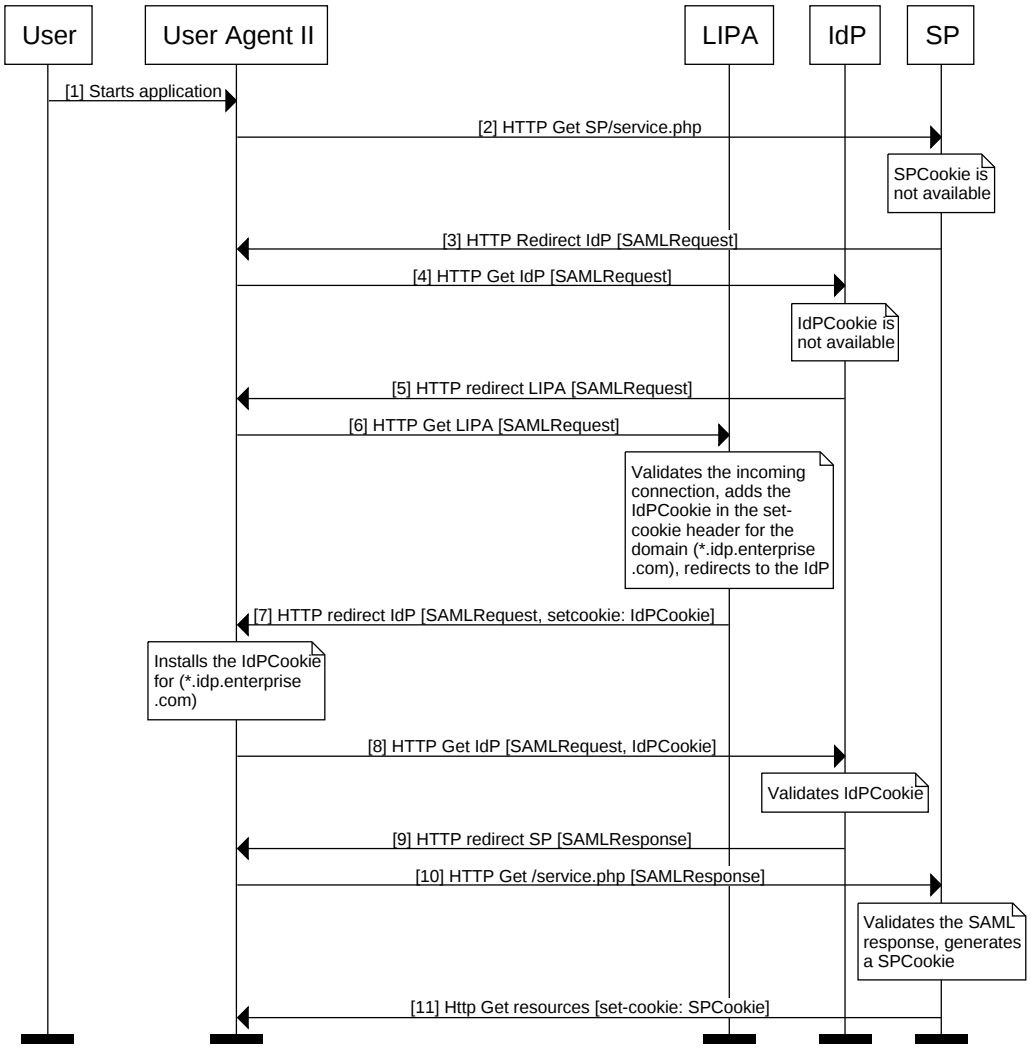

Figure 4.5: Solution Design II (The IdPCookie is available at the LIPA)

1. An access request is sent to the SP without the SPCookie.

2. The SP redirects the UA to the IdP for authorization.

3. The IdP validates the request and redirects the UA to the LIPA.

4. The LIPA verifies that the request is sent by a local trusted application, installs the IdPCookie on the UA for the IdP domain (e.g., *.idp.enterprise.com) and redirects the UA back to the IdP.

5. The UA stores the IdPCookie and sends it this time to the IdP.

6. The IdP validates the IdPCookie, issues an authorization token, and redirects the UA back to the SP with an authorization grant.

7. The UA visits the SP with a valid authorization grant, and receives the requested resources.

Note that in solutions I and II, there is no direct interaction between the IdP and the LIPA. Instead, the authentication cookie is communicated through the UA by specifying the cookie's domain as (*.idp.enterprise.com). The LIPA's URL must therefore be a sub-domain of the IdP to make the LIPA receive the IdPCookie from the UA. Moreover, the LIPA requires a signed SSL certificate to be trusted by the UA and to receive the cookie.

4.3.4 Solution Design III

In this section, we present a third design alternative that does not require any modifications at any of the participating entities (i.e., the IdP, the SP and the UA). In this design, the LIPA acts as an HTTP reverse proxy with two separate TLS sessions, one with the IdP and the other with the UA. Both the UA and the IdP are not aware of the existence of the LIPA, while acting as a man in the middle. Unlike [28], the LIPA intercepts only the traffic from and to the IdP. The LIPA impersonates the IdP while talking to the local applications. It can direct the IdP traffic to itself by adding an entry to the /etc/hosts file that maps the IdP's URL to localhost, or by modifying the routing table on the machine it runs. Moreover, the LIPA should later be able to send traffic to the actual IdP by ignoring the added DNS entry or route.

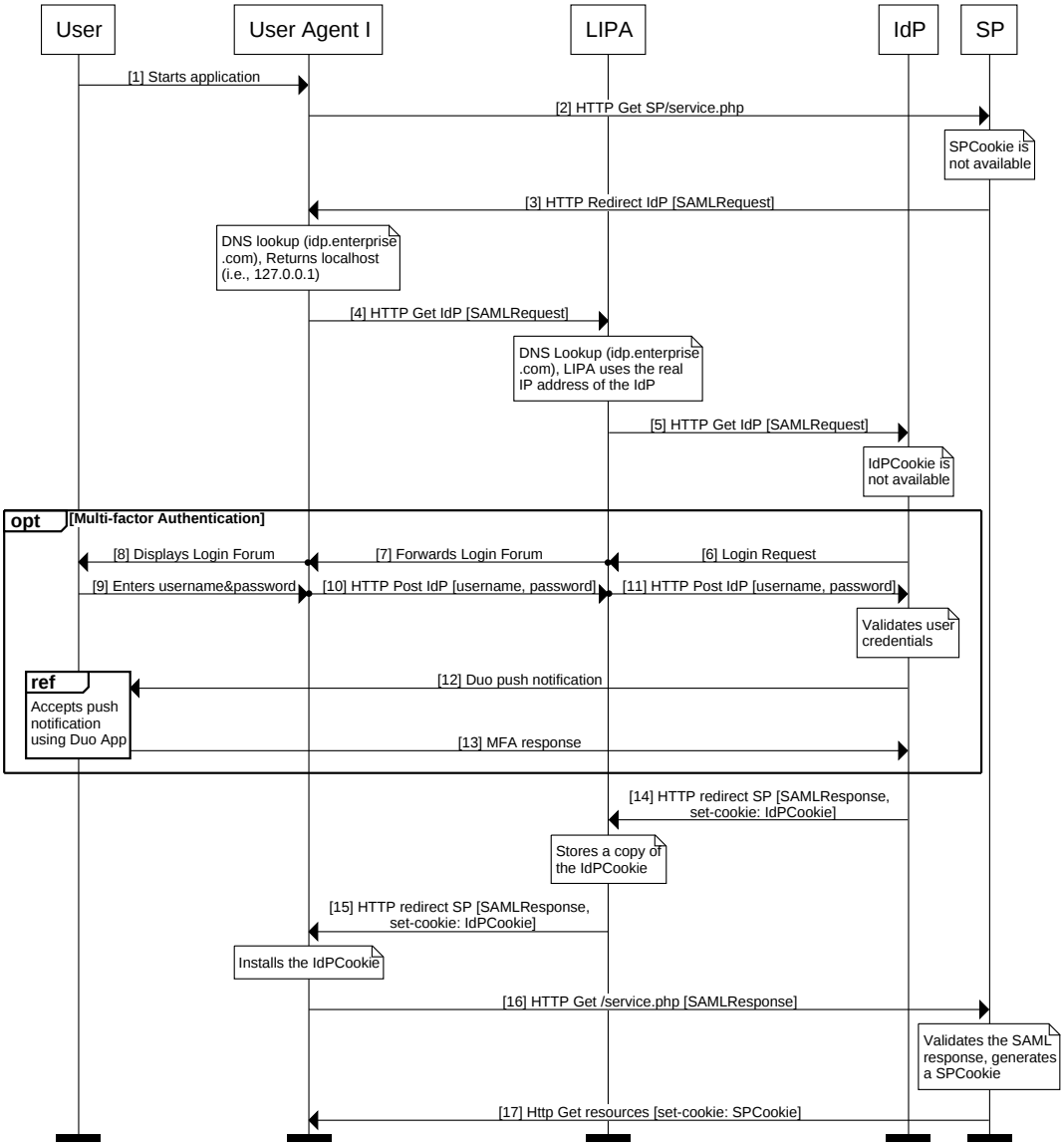

Figure 4.6: Solution Design III (The IdPCookie is not available)

Figure 4.6 shows how the LIPA obtains a copy of the IdPCookie for itself, when the first UA is used. The modified call flow is as follows:

1. The UA sends an access request to the SP.
2. The SP replies with a redirect to the main IdP due to the absence of the SP-Cookie.
3. The UA forwards the request to the LIPA by interpreting the added DNS entry or route.
4. The LIPA checks whether the request contains an IdPCookie or, otherwise, if it has a copy of the cookie. In this scenario, the LIPA has not yet authenticated

with the main IdP, and therefore does not have a valid IdPCookie. It then forwards the request to the IdP using its real IP address.

5. The IdP starts an authentication dialogue with the LIPA, which it forwards to the UA.
6. After a successful authentication, the IdP provides the LIPA with an authorization grant and the IdPCookie. Then it redirects the LIPA to the SP.
7. The LIPA stores a copy of the IdPCookie and forwards the response to the UA.
8. The UA can now contact the SP using the authorization grant.

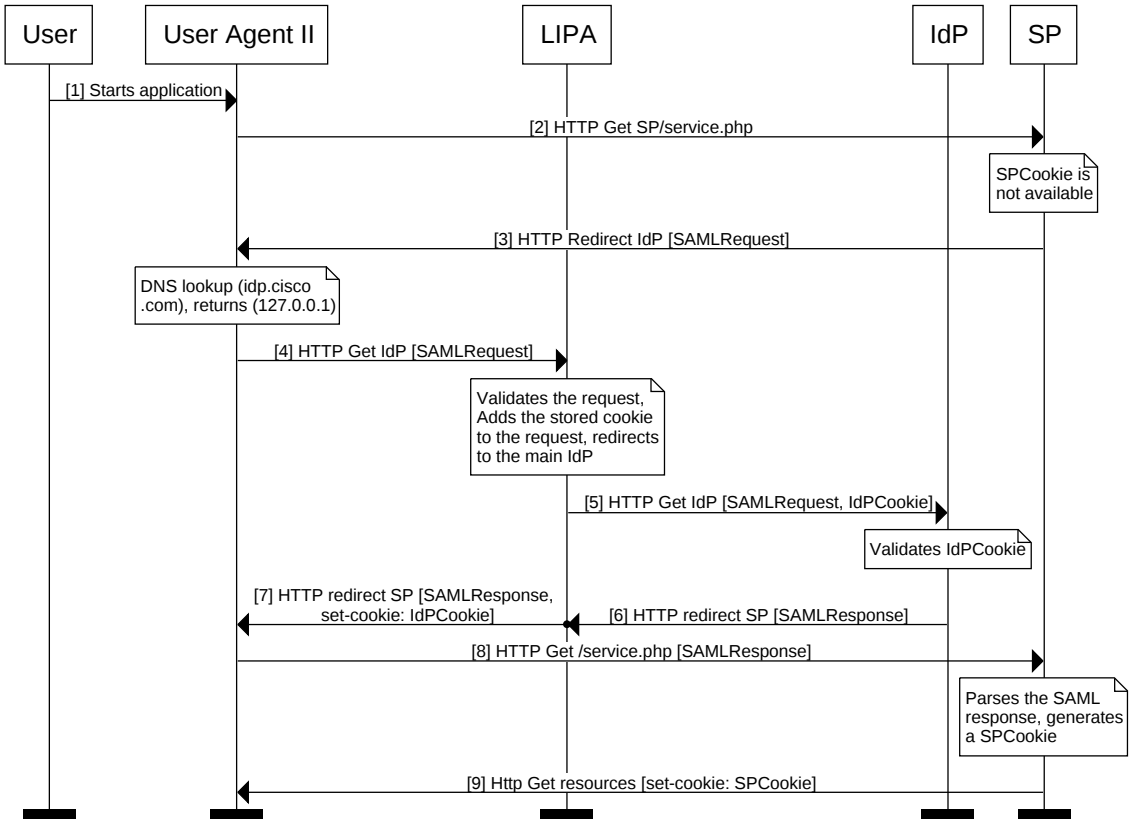

Figure 4.7: Solution Design III (The IdPCookie is available at the LIPA)

Figure 4.7 shows how the previously obtained IdPCookie is made available to the second UA without manual authentication. The corresponding call flow is as follows:

1. The second UA sends an access request to the SP.
2. The SP encodes an authentication request in the redirect URL and redirects the UA to the IdP.

3. The UA sends the request to the LIPA, which has a copy of the IdPCookie.
4. The LIPA verifies that the request arrives from a local trusted application, retrieves the stored IdPCookie, and forwards the request to the main IdP with the cookie.
5. The IdP validates the cookie, generates an authorization token, and redirects the LIPA to the SP.
6. The LIPA forwards the response to the UA.
7. This time the UA visits the SP with a valid authorization.
8. Finally, the SP validates the authorization token, generates a SPCookie and provides the UA with the requested resource.

4.4 Evaluation

In this section, the three proposed solution designs are evaluated with respect to the three design goals; security (i.e., ensuring at least the same security level), usability (i.e., reducing the number of manual authentications) and deployability (i.e., compatibility and ease of deployment).

4.4.1 Security

In our proposed solutions, users manually prove their identity only once per session, instead of every time a new application is started. This is done by managing and synchronizing the IdPCookie between all local trusted applications. A copy of the IdPCookie is stored by the LIPA and may remain available even when all applications are closed. It is therefore necessary to ensure at least the same security level as when the LIPA is not present. In this section, we discuss how the proposed designs meet the defined security goals.

Cookie Storage

To protect the cookie from being stolen by a malicious application, the LIPA must store the cookie securely. The cookie must at least be stored in the same way as done by the local applications or the browsers. The LIPA can do that by storing the cookie in an isolated storage in memory [30]. Encrypting the cookie may not add any security improvements, as there might be at least one unencrypted copy stored by another application.

Cookie Deletion

In the absence of the LIPA, the application ultimately deletes the session cookie when it is closed, the user logs out, or when the device is rebooted. It is necessary to ensure that there is no new security compromise by keeping a copy of the session cookie at the LIPA. The cookie must therefore be removed from the LIPA when the user logs out (e.g., single log out), the user deliberately stops the LIPA, or when the device is rebooted. As long as the LIPA is running, a previously closed application can still reuse the same cookie when it is restarted. In this case, an additional authentication is saved without any added security risk.

Co-located and Remote Attacks

The main functionality of the LIPA is to provide the IdPCookie to local applications when it receives an authentication request. We must ensure that this cookie is made available to only local applications and not to an application running on a different device. The LIPA runs a local web server, which means remote attackers cannot access it. However, an internal attacker, behind the same NAT box, may redirect the request to the victim's LIPA by specifying the private IP address of the victim. Therefore, the LIPA must validate the incoming request (i.e., by checking its source IP address) to ensure that only local applications are served. Spoofing the source IP address is not useful because the response will not reach the attacker's device. Furthermore, as long as users install the real trusted LIPA application on their machine, the system's overall security relies on the strength of the adopted authentication method (e.g., passwords only or MFA). As discussed in Section 3.4, MFA is necessary to achieve a high level of security. Hence, the system's security depends on the strength of the used MFA technique. While evaluating the different MFA methods is not part of this master's thesis. In the proposed solution, the Duo Mobile App [31] is used for MFA due to its improved usability. Since stealing the MFA device is considered difficult, knowing the password is not enough to impersonate the user. However, the attacker may succeed (with the presence or absence of the LIPA) by:

1. Stealing the device during an active session.

2. Stealing the session cookie due to an improper TLS connection [32] or via malicious browser extensions.

3. A powerful phishing attack (e.g., by using Modlishka phishing tool [33]). This can only work, though, if the attacker is watching at the right time.

In order to mitigate these attacks, behavioural biometrics (e.g. application use, mouse and keystroke dynamics, gait, etc.) may be used to continuously check the authenticity of the user and, if necessary, to trigger MFA. However, this may affect the usability of the system (i.e., nonzero false rejection rate) as discussed in Section 3.4.

4.4.2 Usability

The second design goal of the system is to improve usability with a convenient end user experience. In the first solution, the LIPA must authenticate with the IdP before any application can be used (e.g., when starting the device). This requires explicit user interaction, which may not result in a good user experience. On the other hand, in solutions II and III, the LIPA functions transparently to the end user. In solutions I and II, the IdP redirects the UA to the LIPA whether or not it is running on the device. The requests must therefore be handled even when the LIPA is not present to avoid any service interruption. This is achieved by running a remote host on the IdP server with the same URL of the LIPA. When the LIPA is present on a device, the localhost address is used instead by adding an entry in the /etc/hosts file that overrides DNS. Finally, in all the three solutions, MFA is triggered only once per session improving the usability of MFA (i.e., less active user involvement). The only added cost is the need for additional redirects in case of the first two solutions. However, they are performed transparently to the end user.

4.4.3 Deployability

In this section, we discuss how our proposed solutions can be deployed without any compatibility issue. Only in the first solution, the LIPA requires a user interface to authenticate with the IdP as any other UA (e.g., using the default browser). In both solutions I and II, the LIPA hosts a local web server with a subdomain of the IdP. While in solution III, it acts as a reverse proxy that opens two TLS sessions; one with the UA and another with the IdP. Therefore, an SSL certificate (e.g., signed by the IdP) is needed for the LIPA. Moreover, in solution III, the LIPA intercepts only traffic to/from the IdP. It needs to manipulate DNS or modify the routing table to direct the IdP traffic to itself. Therefore, it must run with a ”superuser” capability on the host in which it runs.

4.5 Discussion

In this chapter, the second research sub-question is answered to improve our system’s usability without compromising on security. This is achieved by using a lo-

Table 4.1: Overview of the proposed LIPA design alternatives

Design	Security	Usability	Deployability
I	Meets the security goals	Involves explicit user interaction	Requires a user interface and hosting a web server as a subdomain of the IdP
II	Meets the security goals	Transparent to the end user	A web server as a subdomain of the IdP
III	Meets the security goals	Transparent to the end user	Must run with superuser capabilities

cal identity provider agent that manages the IdPCookie on behalf of local applications. We have introduced three design alternatives and discussed how each of them meets the system design goals. As shown in Table 4.1, design II outperforms the other two designs. Since, it runs transparently to the end user and can easily be deployed compared to solutions I and III. Moreover, solution III requires a superuser capability, therefore, it may not be desirable. However, it can be deployed without any modifications to either the SP, the IdP or the UAs. Finally, a proof-of-concept implementation of the first two solutions is described in Appendix A.

Chapter 5

Dynamic Access Authorization

5.1 Motivation

Traditional service providers often used Authentication, Authorization and Accounting (AAA) services such as RADIUS to manage the authentication and authorization of their users. The RADIUS protocol allows the authorization server (i.e., IdP or RADIUS server) to dynamically change the authentication, authorization and accounting session attributes after the user is authenticated [34]. However, this change is signaled asynchronously towards the SP, which updates the session asynchronously according to the new authorization changes. Nowadays, enterprises are moving towards the use of SSO services using SAML and OAuth, which allow the authorized user to simultaneously log in to multiple enterprise and cloud services after authenticating with the central enterprise IdP. These authorization decisions are based on dynamic attributes and predefined policies that dynamically change according to user, device and network postures. The current user's authorization must therefore be dynamically and immediately adapted when the user's posture changes (e.g., change in location, IP address, device health, etc.).

Unlike traditional AAA services, current SSO procedures do not provide a mechanism for the IdP to signal Change of Authorization (CoA) events towards the SP (i.e., similar to RADIUS CoA). In a more dynamic environment where the user's authorization can change rapidly, the SSO service can at best only use short lived authorizations (e.g., SAML or OAuth tokens) and periodically redirect the user to the IdP to evaluate the current authorization and obtain new access tokens. However, this creates a load on both the IdP and the SP. Moreover, it may cause service disruptions leading to frequent end user annoyance, and that would thus be undesirable. In order to address this problem, we propose a novel mechanism to address the change of authorization in current SSO frameworks such as SAML and OAuth. This mechanism can also handle the changing confidence in user identity by allowing triggering MFA during an ongoing session. Our proposed solution must enable

the CoA synchronously without introducing a security vulnerability.

5.2 Related Work

The SAML standard [35] defines a mechanism for the IdP to signal session termination towards the SP, which is referred to as IdP-initiated logout. Similar functionality is supported in OAuth 2.0 [36] which allows the client to send a revocation request to the authorization server to invalidate a token and to signal the results towards the SP. However, both approaches do not handle dynamic changes in the user authorization, instead they only asynchronously terminate the session (i.e., when the user logs out).

Symeonidis et al. [37] describe a revocation mechanism for SAML long-lived authorizations. The introduced mechanism allows the SP to discover revocation of authorization by consulting a revocation list located at the IdP. There are three components introduced in the article: a revocation list manager located at the IdP that marks a session as invalid, a revocation database that holds identifiers for the revoked sessions, and a revocation controller at the SP that discovers a revoked session. The article does not provide details on when or how the SP polls for updates. However, it is an example of a communication from the IdP to the SP to terminate an ongoing session.

A Continuous Access Evaluation Protocol (CAEP) has been introduced in [38] to allow both the IdP, the SP and the applications to control the authorization of an ongoing session. CAEP is a standard-based proposal to dynamically signal a change of user authorization during an active session. It follows a publish-and-subscribe model that allows an IdP to publish updated information about the session to all subscribed entities such as SPs and mobile applications. In this model, any entity may act as a publisher or a subscriber to certain type of information. However, all the dynamic authorization updates are communicated asynchronously between the participating entities. To the best of our knowledge, there is no prior work that enables user's CoA synchronously in today's SSO systems.

5.3 Proposed Solution

In this section, we introduce a novel CoA procedures for enabling today's SSO systems to dynamically and synchronously update the current authorization of the user without the need for session termination. In the proposed solution, SPs can react immediately to different security events such as the change in user, device or application posture in an enterprise network, or when the user's access rules for enterprise services change. Based on the event, a current authorization may be

revoked, changed, or renewed before expiration. We aim to implement the CoA with the minimal possible changes to SSO frameworks such as SAML and OAuth 2.0. To achieve this goal, the SP relays a call-back URL (CBU) to the IdP during the process of exchanging authentication messages. The CBU includes an identifier to the user, and is used by the IdP as a destination for submitting HTTPS-based POST messages to trigger the CoA at the SP. In case of using RADIUS for AAA services, where the IdP acts as a RADIUS client, the IdP relays this CBU to the RADIUS server. Then, the RADIUS uses it to submit CoA messages. For simplicity and a more general case, both the identity management and authorization functionalities are embedded in one single entity (i.e., the IdP).

5.3.1 SAML CoA

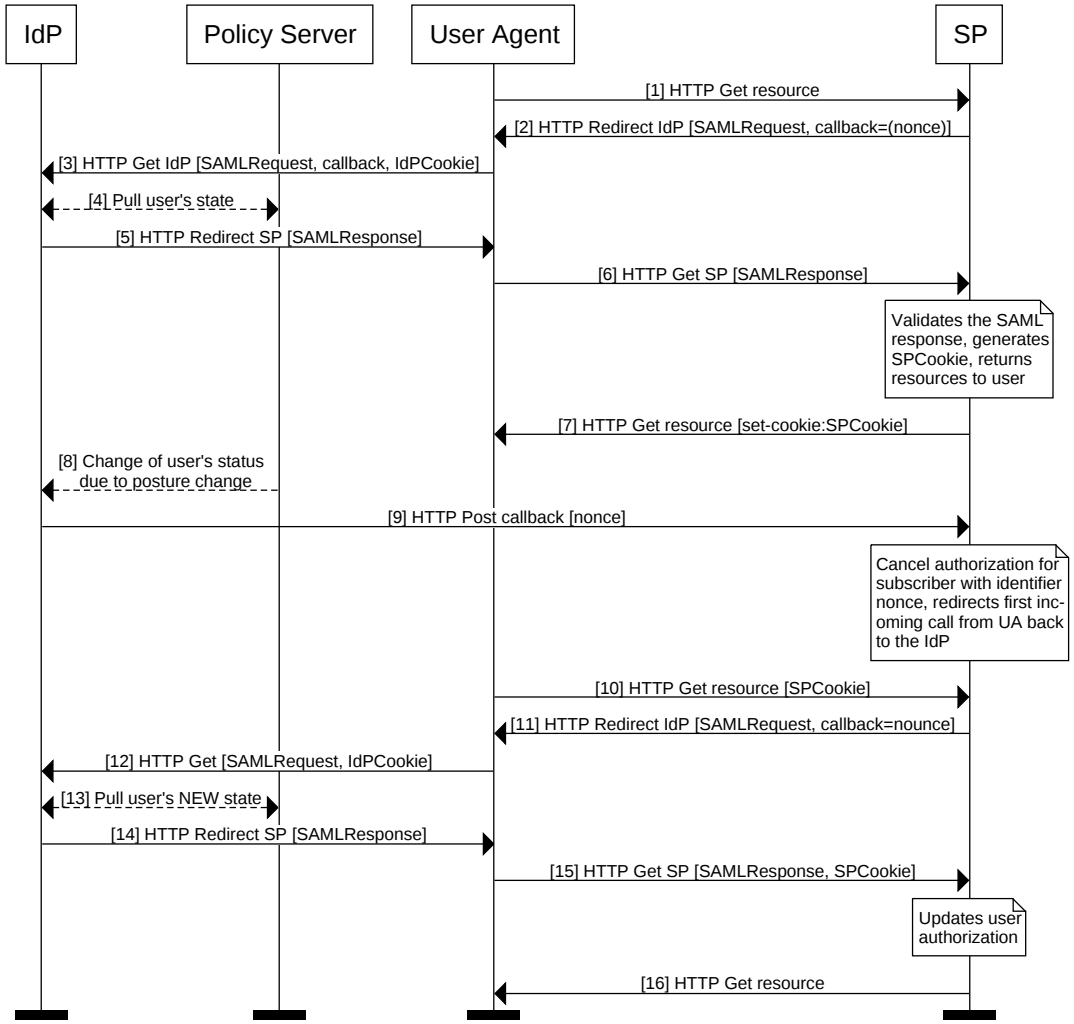

Figure 5.1: Change of user authorization (SAML SSO)

Figure 5.1 describes how the CoA procedures are implemented in SAML-based SSO systems. The CBU may be added as an attribute to the Extensions element in the SAML request, or as a URL parameter in the redirect URL (i.e., from the SP to the IdP). We shall discuss how our solution ensures the integrity of the CBU in Section 5.5. The following steps describe a scenario in which a CoA event is triggered when using SAML:

1. When the SP receives a request from the UA, it creates a unique identifier for the user (i.e., a nonce), adds the CBU as an attribute to the SAML request, and redirects the UA to the IdP (i.e., to obtain an authorization for the requested resource).

2. First, the IdP authenticates the user (if not previously authenticated), consults the policy server to check the user's predefined policies, and stores the CBU for the user. Finally, it creates and signs a SAML assertion, which acts as an authorization grant.

3. The user now visits the SP with a valid authorization, and hence receives the requested resource.

4. When a change in authorization is detected, the IdP sends a SAML assertion using the CBU to the SP to inform it with the detected event.

5. The SP immediately cancels the old SAML authorization for the user with identifier nonce. It then redirects the user back to the IdP when it receives the first incoming request to obtain the new authorization.

6. The IdP consults the policy server to retrieve the new user's authorization and redirects the UA to the SP with the new authorization.

5.3.2 OAuth CoA

A similar approach can be taken for OAuth 2.0 authorization framework. The current OAuth 2.0 standard defines a token revocation service endpoint in the authorization server (i.e., the IdP) that can be used by the resource server (i.e., the SP) to invalidate an existing token when the user logs out. However, there is no signaling mechanism from the authorization server to the resource server for token invalidation, and hence it does not define CoA procedures. Similar to SAML CoA, we propose to define a CoA endpoint for the resource server that can be called by the authorization server (or another authorized entity such as RADIUS). This CoA endpoint is accessed through a callback URL (CBU) sent to the authorization server

with the authorization code request (i.e., the first redirect message from the SP to the IdP).

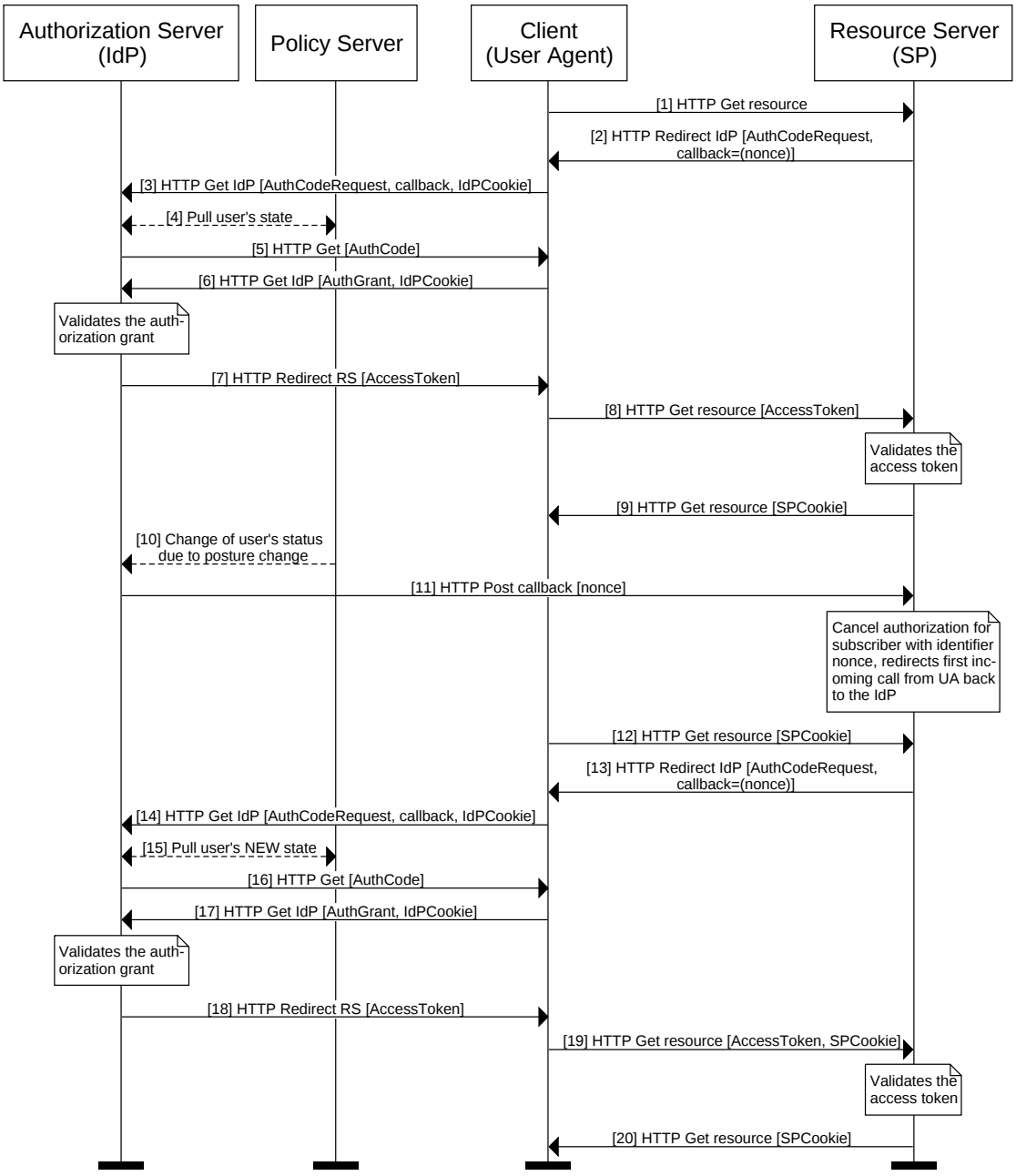

Figure 5.2: Change of user authorization (OAuth SSO)

The OAuth 2.0 standard defines multiple variants for its authorization call flow. In this section, we use the authorization code flow, which is the most common one. On a high-level view, the OAuth 2.0 flow is similar to the SAML authentication call flow. For simplicity, both the client and the UA are merged into one entity as shown in Figure 5.2. The following steps describe how the OAuth 2.0 authorization protocol

works when a CoA event is detected:

1. The client requests a resource on behalf of the user from the resource server.
2. The resource server redirects the client to the authorization server with an authorization code request and the CBU.
3. The client redirects the user (via the UA) to the authorization server for authentication. In this scenario, the user has a valid session with the authorization server (i.e., a valid IdPCookie), and hence not asked for credentials.
4. First, The authorization server consults the policy server to check the user's predefined policies. Then, it sends an authorization grant, which in turn contains an authorization code, to the UA with a redirect message to the client.
5. The client uses the authorization grant to request an access token from the authorization server.
6. The authorization server validates the authorization code, generates an access token, and redirects the client back to the resource server.
7. The resource server returns the requested resource to the client after validating the received access token.
8. When the current authorization changes, the authorization server sends a CoA assertion to the resource server.
9. The resource server immediately cancels the old authorization for the client with identifier nonce, and redirects the client back to the authorization server once the next request is received.

Note that OAuth2 supports variants of such procedure where on step 5, in Figure 5.2, the authorization server provides an authorization grant that includes the access token directly, thus avoiding steps 6 and 7.

5.4 Evaluation

In this section, we evaluate our proposed solution with respect to the predefined design goals. The first goal is to enable CoA procedures synchronously on today's SSO systems. This goal is achieved by relaying a CBU to the IdP, which is used as a destination for signed SAML assertions. The second goal is to ensure that there is no introduced security vulnerability. Therefore, we discuss the security of the proposed mechanism by considering two attack scenarios:

S1 A cheating user manipulating the CBU to prevent CoA from being triggered by the IdP at the service provider.

S2 An attacker (e.g., a man-in-the-middle) knowing the CBU and using it to interrupt the service for the end user (i.e., denial of service).

To ensure that our solution is resilient to S1, we must ensure the integrity of the CBU. In Section 5.3.1, we assumed that the SAML request, which conveys the CBU to the IdP, is always signed by the SP. However, not all SPs sign the SAML request that is sent as an HTTP GET parameter in the redirect URL. Therefore, a cheating client can easily manipulate the CBU. Similarly, there is no provision in the OAUth 2.0 for the resource owner to sign the authentication code request. To overcome this problem, the integrity of the CBU can be ensured as follows:

(i) In SAML: Assuming the CBU is not signed by the SP, we propose adding the CBU as an attribute to the SAML response, which is always signed by the IdP and validated by the SP. After receiving the SAML response, the SP compares the received CBU with the one it knows before the user is granted access. This does not prevent the attacker from changing the CBU, however this change can later be detected by the SP.

(ii) In OAuth 2.0: A similar mitigation approach, as in SAML, can be deployed. According to OAuth 2.0 standard [39], the information content associated with the token is made available to the resource server in two ways by using:

   (a) Handles: A handle is a reference to an internal data structure located at the authorization server and contains the attributes of a specific token. When using this method, the tokens do not have to be signed or encrypted. The resource server can securely retrieve the CBU from the authorization server and verify its integrity before the user is granted access.

   (b) Assertions: It is also possible to use security assertions such as, but not limited to, SAML statements as a mechanism to send the attributes to the resource server. These assertions are signed by the authorization server for integrity. They may also be encrypted for confidentiality. Using this method, the resource server can ensure the integrity of the CBU by validating the SAML assertion.

Finally, to mitigate S2, in which the CBU is used for denial of service, the CoA messages may be conveyed to the SP in the form of SAML assertions. These assertions are signed by the IdP for integrity and authenticity. Therefore, the SP

needs to provide an API for receiving and validating the CoA assertions, which can be accessed through the CBU.

5.5 Discussion

In this chapter, we have answered our third sub-question by proposing a novel mechanism to support the change of user authorization in today's SSO frameworks. In this solution, the SP relays a CBU to the IdP when the first authorization messages are exchanged. The CBU is used by the IdP to send assertions to the SPs informing them with possible policy changes. This novel mechanism can be used in, but not limited to, the following scenarios:

1. Authorization must be revoked due to a security breach, a detected anomaly or a discovered vulnerability. In this scenario, the user is denied access immediately.

2. When there is sufficient confidence about the user's identity (e.g., through continuous monitoring). The IdP can issue a new authentication token to replace the old one that was about to expire. Therefore, allowing the use of short lived authentication tokens without high load on the SP and the end user.

3. Re-authenticating the user (e.g., triggering MFA), when the user behaviour is abnormal, or when there are indications that the user is not legitimate. For example, if the user is not in the immediate proximity of the device.

4. The access may be limited to fewer resources (or instead extending it to more resources) depending on the user's context. For example, a SP may limit access to less critical resources when the user moves to a foreign country with weak IP protection.

By adopting the proposed CoA procedures, SP's can react immediately to various security events. This improves the security of existing SSO systems and paves the way for continuous user authentication.

Chapter 6

Conclusion and Future Work

In this thesis, the main research goal was to establish a balance between security and usability in Web SSO systems while ensuring deployability (i.e., compatibility with current web systems). Our main research question was, therefore, how we can establish a balance between security and usability in Web SSO systems. In order to answer this question, we started by discussing why relying on secrets (e.g., passwords) is not sufficient for proving user's identity. Passwords are vulnerable to, but not limited to, phishing and spoofing attacks. We therefore showed that MFA (i.e., using two different authentication factors) is necessary to meet our security requirements. However, in scenarios such as an enterprise where users access enterprise-related services using multiple applications, MFA has a significant effect on usability. We addressed this problem by reducing the number of explicit authentications that the user needs to perform during a session.

The main research question was divided into three subquestions. To answer the first subquestion, we conducted an extensive literature study to review different related web authentication systems and discussed how each system addressed the balance between security and usability. We showed that proximity-based authentication can achieve a good balance, however, the existing proximity-based solutions face deployability challenges. On the other hand, context-aware authentication systems that rely on context checking as a second authentication factor may still be vulnerable to context spoofing. However, we argue that using context as a method for selecting an appropriate second authentication factor, while not replacing it, can improve security in Web SSO systems.

To answer the second subquestion, we have introduced a local IdP agent, which needs to be installed on the local device. The LIPA manages the authentication information across all local trusted applications. The results show that usability can be further improved without compromising security in Web SSO. The proposed solution allows users to access the enterprise-related services by authenticating only once during a session, as long as their posture remains normal. We have introduced

three design alternatives, and showed that the second design outperforms the other two. It allows the LIPA to obtain a copy of the SSO cookie when the user authenticates using the first application in a session. This cookie is then communicated to other applications and prevents unnecessary authentications. we evaluated our proposed solution in terms of the three design requirements; security, usability and deployability.

Furthermore, as mentioned earlier, the current SSO implementations do not allow for adapting the user authorization during an ongoing session. In the event when the user's authorization changes or when re-authentication is required, this is currently handled asynchronously by terminating the session and establishing a new one. In this thesis, we have addressed this problem by providing change of authorization procedures for current SSO frameworks such as SAML and OAuth 2.0, and we have thus answered the third subquestion. The solution allows SPs to react immediately to security events by communicating a CBU to the IdP while exchanging the first authentication messages. This CBU is used by the IdP to synchronously push security updates towards SPs. Moreover, we have discussed how the integrity of the CBU can be ensured.

To sum up, in this thesis, we have improved both the usability and security of Web SSO systems while ensuring deployability. However, achieving a perfect balance between security and usability is still a challenge. Although, the proposed solution meets our requirements, the overall system is still not perfectly secure. An attacker may hijack the session by physically stealing the user's device during an authenticated session. When the LIPA is installed, the attacker will then automatically access all the applications that use the same SSO service. With no LIPA installed, the attacker is still able to steal the cookie and may use it for other applications, however with extra manual efforts. Behavioural biometrics may be a possible solution to this problem, however they may affect usability, which is not desirable. It is therefore recommended for future work to investigate the use of behavioural biometrics in Web SSO with the presence of the LIPA. Moreover, we did not discuss how the change in user authorization could be detected. In other words, how and where the user's behavioural dynamics and/or contextual features are to be recorded and verified.

Finally, it is good to investigate how the LIPA functionalities may be deployed on the cloud. The main challenge will then be cookie management and how to correctly identify the end device when a request is received by the LIPA. The cookie must then be deleted when the session expires (e.g., when the device is shut down). While, the device may be identified via an installed certificate (e.g., by using the WebAuth API [40]).

Bibliography

[1] M. A. Sasse and I. Flechais, “Usable security: Why do we need it? how do we get it?” O’Reilly, 2005.

[2] M. A. Sasse, M. Smith, C. Herley, H. Lipford, and K. Vaniea, “Debunking security-usability tradeoff myths,” *IEEE Security & Privacy*, vol. 14, no. 5, pp. 33–39, 2016.

[3] “Teamsid. worst passwords of 2019.” [Online]. Available: https://www.teamsid.com/1-50-worst-passwords-2019/

[4] “Verizon. “2019 data breach investigations report”.” [Online]. Available: https://enterprise.verizon.com/resources/reports/dbir/

[5] L. Lamport, “Password authentication with insecure communication,” *Communications of the ACM*, vol. 24, no. 11, pp. 770–772, 1981.

[6] C. Herley, P. C. Van Oorschot, and A. S. Patrick, “Passwords: If we’re so smart, why are we still using them?” in *International Conference on Financial Cryptography and Data Security*. Springer, 2009, pp. 230–237.

[7] 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 *Proceedings of the 6th ACM workshop on Formal methods in security engineering*, 2008, pp. 1–10.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext transfer protocol–http/1.1,” 1999.

[9] D. Kristol and L. Montulli, “Http state management mechanism,” RFC 2965, October, Tech. Rep., 2000.

[10] “Sso protocol adoption predictions.” [Online]. Available: https://www.gluu.org/blog/gluu-web-authentication-sso-protocol-adoption-predictions

[11] J. Kohl, C. Neuman *et al.*, “The kerberos network authentication service (v5),” RFC 1510, september, Tech. Rep., 1993.

[12] H. Lockhart and B. Campbell, “Security assertion markup language (saml) v2. 0 technical overview,” *OASIS Committee Draft*, vol. 2, pp. 94–106, 2008.

[13] N. Sakimura, J. Bradley, M. Jones, B. De Medeiros, and C. Mortimore, “Openid connect core 1.0 incorporating errata set 1,” *The OpenID Foundation, specification*, vol. 335, 2014.

[14] C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Capkun, “Smartphones as practical and secure location verification tokens for payments.” in *NDSS*, vol. 14, 2014, pp. 23–26.

[15] A. Czeskis, M. Dietz, T. Kohno, D. Wallach, and D. Balfanz, “Strengthening user authentication through opportunistic cryptographic identity assertions,” in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 404–414.

[16] M. Shirvanian, S. Jarecki, N. Saxena, and N. Nathan, “Two-factor authentication resilient to server compromise using mix-bandwidth devices.” in *NDSS*, 2014.

[17] N. Karapanos, C. Marforio, C. Soriente, and S. Capkun, “Sound-proof: usable two-factor authentication based on ambient sound,” in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 483–498.

[18] “Ip to geo-location.” [Online]. Available: https://pypi.org/project/ip2geotools/

[19] M. Kranch and J. Bonneau, “Upgrading https in mid-air,” in *Proceedings of thz 2015 Network and Distributed System Security Symposium. NDSS*, 2015.

[20] C. C. Rocha, J. C. D. Lima, M. Dantas, and I. Augustin, “A2best: An adaptive authentication service based on mobile user’s behavior and spatio-temporal context,” in *2011 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2011, pp. 771–774.

[21] G. Salton, A. Wong, and C.-S. Yang, “A vector space model for automatic indexing,” *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[22] K. A. A. Bakar and G. R. Haron, “Adaptive authentication based on analysis of user behavior,” in *2014 Science and Information Conference*. IEEE, 2014, pp. 601–606.

[23] D. Freeman, S. Jain, M. Dürmuth, B. Biggio, and G. Giacinto, “Who are you? a statistical approach to measuring user authenticity.” in *NDSS*, 2016, pp. 1–15.

[24] D. Preuveneers and W. Joosen, “Smartauth: dynamic context fingerprinting for continuous user authentication,” in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 2185–2191.

[25] J. Solano, L. Camacho, A. Correa, C. Deiro, J. Vargas, and M. Ochoa, “Risk-based static authentication in web applications with behavioral biometrics and session context analytics,” in *International Conference on Applied Cryptography and Network Security*. Springer, 2019, pp. 3–23.

[26] P. Domingos and G. Hulten, “Mining high-speed data streams,” in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 71–80.

[27] A. Harilal, F. Toffalini, J. Castellanos, J. Guarnizo, I. Homoliak, and M. Ochoa, “Twos: A dataset of malicious insider threat behavior based on a gamified competition,” in *Proceedings of the 2017 International Workshop on Managing Insider Security Threats*, 2017, pp. 45–56.

[28] B. Logan and T. Mossing, “Method, apparatus and computer program product for automatic cookie synchronization between distinct web browsers,” Jul. 5 2007, uS Patent App. 11/326,570.

[29] R. Cox, E. Grosse, R. Pike, D. L. Presotto, and S. Quinlan, “Security in plan 9.” in *USENIX Security Symposium*, vol. 2, 2002.

[30] “System isolated storage.” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/io/isolated-storage

[31] “Duo mobile app.” [Online]. Available: https://duo.com/product/multi-factor-authentication-mfa/duo-mobile-app

[32] S. Sivakorn, I. Polakis, and A. D. Keromytis, “The cracked cookie jar: Http cookie hijacking and the exposure of private information,” in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 724–742.

[33] “Modlishka duszyski phishing tool,” 2002. [Online]. Available: https://github.com/drk1wi/Modlishka

[34] M. Chiba, G. Dommety, M. Eklund, D. Mitton, and B. Aboba, “Dynamic authorization extensions to remote authentication dial in user service (radius),” *Internet Engineering Task Force, Request for Comment*, vol. 3576, pp. 1–25, 2003.

[35] J. Hughes and E. Maler, “Security assertion markup language (saml) v2. 0 technical overview,” *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08*, pp. 29–38, 2005.

[36] T. Lodderstedt and M. Scurtescu, “Oauth 2.0 token revocation,” *IETF RFC 7009*, 2013.

[37] I. Symeonidis and B. Preneel, “Sarl: A revocation mechanism for long lived assertions on shibboleth,” 2014.

[38] “Continuous access evaluation protocol (caep),” 2019. [Online]. Available: https://cloud.google.com/blog/products/identity-security/re-thinking-federated-identity-with-the-continuous-access-evaluation-protocol

[39] T. Lodderstedt, M. McGloin, and P. Hunt, “Oauth 2.0 threat model and security considerations,” *IETF2013*, 2013.

[40] “Webauth guide.” [Online]. Available: https://webauthn.guide

[41] “Simplesamlphp.” [Online]. Available: https://simplesamlphp.org

[42] “Apache2 with virtual hosts enabled.” [Online]. Available: https://httpd.apache.org/docs/2.4/vhosts/examples.html

[43] “Demo.” [Online]. Available: https://drive.google.com/drive/folders/1ldg-h3LtBuWoxwH3GGC6728Lh7gUn6ZU?usp=sharing

Appendix A

Prototype

A prototype for the two solutions described in Sections 4.3.2 and 4.3.3 was implemented using SimpleSAMLphp [41]. It is a lightweight open source platform written in PHP that implements Web SSO with several federation protocols (such as SAML, OpenID and OAuth). In this prototype, SimpleSAMLphp was configured as both an SP and as an IdP with a custom authentication source as of Cisco internal login. Moreover, few modifications were added to the SimpleSAMLphp source code, such that the IdP redirects authentication requests to the LIPA as described in the two designs. The prototype includes two SPs (*service.cisco.exp* and *service2.cisco.exp*) and an IdP (*idp.cisco.exp*) which were hosted on a Cisco internal server (Ubuntu 18.04.4 LTS). While the LIPA code was hosted on a local laptop (macOS Catalina v10.15.4). A DNS entry is added to the /etc/hosts file on the laptop to map the LIPA's URL (*micro.idp.cisco.exp*) to localhost address (i.e., *127.0.0.1*). All the code used in the prototype was written in PHP and the servers were running on Apache2 [42] as virtual hosts.

Testing was performed by contacting any of the two SPs using three different UAs (i.e., Google Chrome, Safari and Firefox). The first solution was tested by running a short Python script that contacts an IdP service (*idpservice.cisco.exp*) to register the LIPA. In this scenario, the script uses one of the UAs for login. While, the second solution is tested by contacting any of the SPs using two different UAs. In both scenarios, authentication was required only once when contacting the SPs using the three UAs. Note that the code produced in this thesis is developed at Cisco and therefore not made public. However, a demo is made available at [43] for illustration. To request access for the full code with installation details, please contact *ahmedbakryhelmy@gmail.com* and *samullen@cisco.com*.

Patent No. 1

A method for implementing RADIUS change-of-authorization in an Identity Provider

Peter Bosch, Sape Mullender, Alessandro Duminuco, Ahmed Ahmed, Aaron Woland CISCO

January 17, 2020

Abstract:

Traditional AAA services authenticate, authorize and account for (enterprise) users using a (remote) access service; RADIUS is such a protocol. While RADIUS supports functions for establishing and tearing down sessions, RADIUS also supports functionality for a RADIUS service to change the authorization of a user session. This can happen for instance when a device's posture changes, or another system triggers the action due to malicious activity. Such authorization changes are asynchronously signaled towards the service, which then needs to update the session asynchronously according to the changed conditions.

Change of Authorization (CoA) can be employed to immediately end a user's session, disconnecting them from the network or resource. CoA can also be used to force a re-authentication and re-authorization of the user with the new conditions being examined, resulting in a new level of access. CoA can also be used to push an immediate change, such as ACL changes, moving VLANs or more.

Single-sign-on procedures are used to provide AAA services towards a service provider (e.g., a web-service). Yet today, no mechanism exists to signal to the service provider that a change of authorization is required, akin to the equivalent RADIUS functionality. Single-sign-on is most used to get access to a cloud application and is also used for cloud-based remote-access services, cf. Cisco's VPNaaS product, zScaler private access, and for many other vendors. Thus, in the case where the authorization of a user using a (web or other) service changes, as per the earlier example, at best the service provider can only use short-lived SAML statements and require periodic polling of the service provider to obtain a changed authorization. This is undesirable as it creates load on the service provider, the identity provider, and may create disruptions to the service.

To address the absence of a change of authorization procedure in today's single-sign-on system, we are introducing a novel method to emulate a change of authorization with single-sign-on procedures. In a nutshell, the service provider signals a call-back URL in a single-sign-on session. When change of authorization needs to be enforced, the call-back can be triggered to revoke an earlier SAML authorization, which causes the service provider to require a re-authentication and

re-authorization thus emulating a RADIUS change-of-authorization. Similar approaches can be used for other authorization frameworks such us OAuth 2.0.

The introduction of the IdP-based change of authorization is (a) needed to provide for integration with legacy RADIUS services in existing network for remote access functionality, and (b) needed to enable new cloud-based security product features with integrated and automatic posture operations (cf. MDM, AnyConnect HostScan, DUO posture and more).

Restatement of invention

Using a Call-Back URL, security events can be processed dynamically for ongoing sessions using service providers relying on SAML-based authentication and authorization. A full integration between RADIUS and SAML can be achieved that includes important security events that involve ongoing client-server sessions.

Advantages

This invention is important for Cisco. A great deal of security infrastructure and procedures are based on RADIUS and it will take a long time before RADIUS is phased out. Meanwhile, service providers and browsers are massively using SAML-based cookies and SAML statements for authentication and authorization.

An example of this is Cisco's new VPNaaS (= Umbrella Access) service product. A mechanism to relay security events into ongoing sessions is important. And the integration between RADIUS protocols and SAML protocols is equally important. This CPOL shows how security events can dynamically be delivered between RADIUS servers and SAML-based service providers.

The use of CoA is not limited to only RADIUS or DIAMETER sources. Other applications and services could leverage the same mechanisms, not only the bridge from the AAA protocol.

This CPOL is clearly unique in the industry, in the way it dynamically changes the authorization without terminating the session.

Patent No. 2

True per-device Single Sign-On

Ahmed Ahmed, Sape Mullender, Peter Bosch, Alessandro Duminuco

CISCO

April 03, 2020

Abstract

Enterprise single-sign-on infrastructures achieve two things: users need to provide their password only to a trusted enterprise authentication service, and, once the user has authenticated for one service, that authentication can serve for another service as well. In other words, users do not have to type their password over and over as they move from one enterprise service to another. At the moment, this tends to work on a per-user-application basis: a browser can be used to move from one service to another without re-authenticating, but that user's authentication is not valid for the mailer application, or for a different browser. We address this problem by proposing a true single-sign-on experience for all applications on a user's device.

Restatement of invention

A local Identity provider can be run on any host to allow all applications to share the single-sign-on state created by the first authentication operation triggered by the first application to contact a service provider. The local Identity Provider can be installed in a way that neither application code, nor service provider code needs to be modified in any way.

Advantages

Users are no longer forced to type their passwords as they open different applications connecting to corporate services in the morning. It creates a single application that deals with corporate-security interactions; with this approach, less trust needs to be placed in individual applications installed on a host. Moreover, in the future, the local Identity Provider could become a platform for enterprise posture verification — i.e., help determine whether or not users are behaving badly.