IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 20, 2020, accepted October 14, 2020, date of publication October 26, 2020, date of current version November 9, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3033570

Single Sign-On: A Solution Approach to Address Inefficiencies During Sign-Out Process

LOKESH SARAVANAN RAMAMOORTHI[1], (Member, IEEE), AND DILIP SARKAR[2], (Senior Member, IEEE)

[1]Department of Electrical and Computer Engineering, University of Miami, Coral Gables, FL 33146, USA
[2]Department of Computer Science, University of Miami, Coral Gables, FL 33146, USA

Corresponding author: Lokesh Saravanan Ramamoorthi (lokeshr@miami.edu)

ABSTRACT In a Single Sign-on (SSO) environment, an Identity Provider (IDP) authenticates a user for the first Service Provider (SP). The IDP creates an active IDP session and stores its information in the user's web browser. Each SP also creates and maintains one active service session. Using state-transition diagrams, we illustrate sign-in and sign-out processes. An information security vulnerability situation is created because users are unaware of an active IDP session in the user's browser and signs-out only from SP sessions. One solution to this problem is educating users. Another solution is to implement the SSO that ensures the termination of the IDP session as soon as user signs-out from all services that the IDP authenticated. The first solution appears to be simple, but practically an impossible task to educate millions of web based SSO users worldwide. The second solution is better because one good implementation solves the problem for all users. In this article, we propose several solutions for terminating the hidden active IDP session. Also, we review the data storage-methods commonly used for storing information of SP and IDP sessions in the browsers. Moreover, we propose a browser extension for conveniently and efficiently managing active SP and IDP sessions. In our proposed browser extension, we have recommended IndexedDB browser storage for storing active session information. We believe our proposed browser extension is simple, but efficient solution for eliminating hidden active IDP session.

INDEX TERMS Authentication, authorization, identity provider, information security, service provider, single sign-on, web browser security.

I. INTRODUCTION

Information security requires access control. *Authentication* is commonly used for controlling access to information and services. When a user attempts to access information or service, the user is challenged, i.e., in computer security, challenge–response authentication is a family of protocols in which one party presents a question (''challenge'') and another party must provide a valid answer (''response'') to be authenticated, to provide his or her credentials, such as username and passwords, etc., before access is authorized. For ease of access to multiple services, Federated Identity management or Single Sign-on (SSO) [5] process is widely used. In SSO, a user may access multiple services after a single authentication. Moreover, the SSO process simplifies maintaining the directory of legitimate users.

SSO process is convenient because a user needs only one set of credentials to access all services, instead of multiple sets of credentials [4], [9], [30]. In SSO, authorization for access to services or information is separated from authentication of users. The authentication is a **service** and provided by IDP. After a user is authenticated, the IDP creates an active session and stores its information in user's browser for provisioning access to other authorized services.

Most of the users do not know about the existence of the active IDP sessions and they think their session is with the SP. Thus, after the user signs-out of the service, he or she assumes that there is no active session [26], [31]. This lack of knowledge may lead to unintended and unauthorized access to services. For example, user *A* works on a computer in a library, then *A* signs-out of the service and leaves. Next, if another user *B* comes and accesses (clicks on) the same service in the computer, user *B* will not need any authorization and will get access to the service. Unfortunately, user *B* is accessing service that was authorized for the user *A*. This issue happens because *A*'s IDP session was still active in the browser. Using *state transition diagrams*, we illustrate the steps of SSO process, and we illustrate in *Section IV-C* how an active IDP session remains after the user signs-out of the services.

The associate editor coordinating the review of this manuscript and approving it for publication was Kuo-Hui Yeh.

One solution to this problem is educating users, and the other solution is to implement the SSO in a way that when a user signs-out of a service the active IDP session is also terminated instantly. The first solution appears to be simple, but practically an impossible task [39], as millions of SSO users have to be educated. The second solution is better because a single good implementation solves the problem for all users.

In this article, we demonstrate the sign-in and sign-out processes using finite-state machines (FSM). We illustrate how and when the active sessions for IDP and the SP are created. Also, we show how an active IDP session may remain after a user signs-out from all SP sessions. Furthermore, we propose several solutions for terminating the residual active IDP session, if there is any. In summary, the key contributions of this work are:

1) We demystify the single sign-on process by modeling the interactions of the SPs, the IDP, and the browser with FSM.
2) We demonstrate, using FSMs, how a naive user may leave an active IDP session after the user signs-out from all SPs.
3) We propose four possible solutions for terminating the residual active IDP session.
4) We review data-storage methods available to the browsers for tracking the active SPs and the IDP sessions.
5) We propose an implementation of a browser extension module. An agent in the proposed module monitors the active sessions of the SPs and the IDP, while another agent terminates the residual active IDP session, if any. We believe that the IndexedDB data-storage method is the best for monitoring active sessions of the SPs and the IDP.

Rest of the paper is organized as follows: In *Section II*, related work is reviewed and scope of our work is discussed; especially, the experimental platform that we have used to identify the problem and browsers for which our proposed solutions will work. *Section III* presents how browser-centric SSO is implemented. In *Section IV*, using state transition diagrams we illustrate steps for sign-in, sign-out, and time-out processes. In the same section, we also demonstrate how users may leave active IDP sessions unknowingly. In *Section V*, we propose solutions for eliminating active IDP sessions that users may leave unintentionally. In *Section VI*, we review the data storage-methods commonly used for storing information of SP and IDP sessions in the browsers. In *Section VII* section, we propose a browser extension for conveniently and efficiently managing active SP and IDP sessions. We propose to use IndexedDB storage. We present our concluding remarks and future work in *Section VIII*.

II. RELATED WORK AND SCOPE OF THE PROPOSED WORK

A. RELATED WORK

Most of the articles we researched, focus on technological design and implementation vulnerabilities of SSO. The articles also discuss the attacks on the identified vulnerabilities and some of them provide solutions to avoid or mitigate the risks that arise due to the attacks. In this section, we discuss these articles and compare them in Table 1.

The RFC-6819 has documented comprehensive threat models for OAuth 2.0 protocol [37]. These threat models of the protocol include clients, authorization mechanisms, and refreshing an access token. Also, countermeasures for all these threat models have been proposed.

Thomas Groß performed a detailed protocol-level analysis of Security Assertion Markup Language (SAML) based Single-Sign on [20]. In a survey, Simpson and Groß categorized malicious and accidental security issues in Federated Identity Management (FIM) frameworks and presented solutions to those security incidents that have been recommended by others [35].

Armando *et al.* [1], [2], Yang and Manoharan [42], and Fett *et al.* [11] have performed a comprehensive analysis of the protocol to demonstrate XSS, CSRF, replay attacks, and impersonation attacks. Also, Armando *et al.* described solutions to mitigate and even solve these problems [1]. These studies have shown that Cross-site Scripting (XSS) and Cross-Site Request Forgery Attacks (CSRF) are two common vulnerabilities to the security of OAuth authorization protocols.

D. Fett *et al.* created an expressive web model of OAuth 2.0 standard and performed a comprehensive analysis of the model. They discovered that 307 redirect attack, IdP mix-up attack, state leak attack, and naïve RP session integrity attack to break the security of OAuth [11]. Also, they proposed changes to OAuth standard and analytically proved that these changes eliminate all four identified vulnerabilities. Using protocol analyzing tools, Nair *et al.* identified security vulnerabilities in SSO implementations of Facebook®, Google®, and JanRain® [28]. As part of their research, Mainka *et al.* developed a fully-automated *OpenId Attacker* tool and used it for ID Spoofing (IDS), Key Confusion (KC), and Token Recipient Confusion (TRC) attacks on several OpenID implementations, including Sourceforge®, Drupal®, ownCloud® and JIRA® [25]. Shi *et al.* analyzed the vulnerabilities of mobile apps implementation of SSO. They designed and implemented MoSSOT (Mobile SSO Tester), an automated blackbox security testing tool for Android applications utilizing the SSO services from three mainstream service providers — Facebook®, WeChat® and Sina® [33].

As reviewed, security vulnerabilities of OAuth standards and SSO implementations have been studied extensively and also, fixes to eliminate exposed vulnerabilities have been proposed. In our summary table, *Table 1*, we listed out the above-mentioned papers to provide a comparative snapshot of the related work.

In the table,

- *Column 1:* Identifies the paper,

TABLE 1. Comparison summary of related work.

Author & Year	Focus of the Research	Threats / Attacks discussed by the research	Are special tools used for analysis?	Does the research propose any solutions or countermeasures?
E. T. Lodderstedt et. al (2013) [37]	Oauth 2.0 Protocol	Authorization Code, Implicit Grants, User Credentials	No	Yes
Simpson Sean et al. (2016) [35]	Federated Identity Management (FIM) Frameworks	Cross-Site Scripting (XSS), CSRF (Cross Site Request Forgery)	No	Yes
Thomas Groß (2003) [20]	SAML Protocol Vulnerability	Connection Hijacking, Man in the Middle, HTTP Referrer Attack	No	No
Armando et al.(2011) [1]	Authentication Flaw in SSO Protocols	Cross-site Scripting (XSS) and Cross-Site Request Forgery(CSRF) vulnerabilities	No	Yes
Armando et al.(2013) [2]	SAML SSO and OpenID protocols	Cross-Site Scripting, Cross Site Request Forgery, User Impersonation attacks	No	Yes
Yang et al.(2013) [42]	OAuth 2.0 authorization protocol	Intermediary authorization abuse, replay attacks, impersonation attacks and forced-login CSRF attacks	No	No
D. Fett et. al (2016) [11]	OAuth 2.0 standards web model	307 redirect attack, IdP mix-up attack, state leak attack, and naïve RP session integrity attacks	No	Yes
C. Mainka et. al (2016) [25]	OpenID, OpenID Connect and SAML SSO Systems	ID Spoofing (IDS), Key Confusion (KC), and Token Recipient Confusion (TRC) attacks	Yes (OpenID attacker tool)	No
A. Nair et. al (2015) [28]	SSO Implementations in major websites (Google ID, Facebook, JanRain)	Logic Flaws and weak authorization	Yes (BRM analyzer)	No
Shi et. al (2019) [33]	Android applications utilizing the SSO service	Augmented Token replacement attack, Authentication code maintenance failures, Disclosure of app secrets	Yes (MoSSOT)	No

- *Column 2:* Mentions the focus of the research — which area of SSO implementation or framework is being analyzed by the paper,
- *Column 3:* Lists the threats / attacks discussed by the research — list of the information security attacks being focused by the research,
- *Column 4:* Lists if any special tools used for analysis — Did the researchers create or use any specialized tools for their research methods,
- *Column 5:* Mentions the solutions or countermeasures Proposed — whether the research focus is mainly on the attacks or does it also provides any solutions or countermeasures to deal with the attacks.

Our work studies a process based vulnerability of OAuth based SSO framework. In this article, we use state diagrams to demonstrate the security vulnerability during a sign-out scenario, and present several possible solutions to mitigate the vulnerability.

B. SCOPE OF THE PROPOSED WORK

In this section, we describe the scope of our experimental settings for the reported observations. The three key components for web-based SSO are (a) Web browser, (b) IDP, and (c) web-based SP. The roles of these three components are described in detail in *Section III*.

We used Google Chrome® (version 77.0.3865) web browser for our observations, because Chrome® is used by 60% of users in the world [29]. Google Chrome® is based on the open-source Chromium browser — a part of the Chromium project [8]. Also, Chromium browser is the foundation of Microsoft Edge®, Samsung Internet Browser®, Opera®, and Torch®; since the foundation of these browsers are Chromium, we expect that, if our experiments are repeated on any of these Chromium-based browsers, the outcome will be same or very similar.

Many software applications embed browsers within themselves for providing seamless experience to users. The underlying browsers are called embedded browsers. An embedded browser may retain partial or full functionality of the original browser. We believe, if any of the Chromium-based browser is embedded within an application, results will be very similar.

The IDPs used for our observations are based on Shibboleth® (version 3.4.0), a Federated Identity Management (FIM) Framework. Shibboleth is an open-source software that implements widely used federated identity standards, principally the OASIS SAML, to provide a federated single sign-on and attribute exchange framework [34]. Other notable SSO frameworks available are:

1) Lightweight Directory Access Protocol (LDAP) based OpenLDAP and Microsoft Active Directory (AD) Schema and

2) OpenIDConnect, an identity layer implementation on the top of OAuth protocol.

Because, Shibboleth is the most widely deployed web-based federated identity management system, we employed it for our observations.

For establishing state transition patterns during Sign-in and Sign-out, we used several SPs, including the cloud based file storage service Box®, the web-based HR management application WorkDay®, and the web-based office productivity software Office 365®. Irrespective of the service providers, state transitions were same. This is because, during the SSO process the SPs only request for and receive the authentication information, whereas the Web browser and the IDP play a more active role in orchestrating the authentication redirects, and authenticating the user respectively.

The next section describes in detail, the role of browsers, IDPs and SPs during the Single Sign-on process.

III. KEY COMPONENTS OF WEB-BASED SSO IMPLEMENTATION

The OAuth 2.0 standard [7], [21] is widely used in web-based SSO implementations. It uses *SAML* as a means for requesting an OAuth 2.0 access token as well as for client authentication [3]. SAML is an eXtensible Markup Lanugage (XML) based framework [6] for communicating user authentication, entitlement, and attribute information between IDP and SP via a web browser. Shibboleth leverages the features of users' web-browser to handle the coordination between users, IDPs, and the service providers. There are three main components in SAML based SSO implementation [24]. Fig. 1 is based on Shibboleth's conceptual framework [21] and shows a view of the three main components of web-based SSO implementation.

1) Web Browser of the user,
2) Identity Provider - (often) authentication server of the user's organization, and
3) Service Provider - the application software that provides services to the user.

A. WEB BROWSER

As described next, web browser — a software application used to access the Internet via. Hypertext Transfer protocol (HTTP) — is a key component of the SAML based SSO implementation.

Initially, commercial Web browsers were created to interpret and render HyperText Markup Language (HTML) encoded contents, and display rendered pages to the user, because most contents used to be text based static HTML pages. However, in the past decade, with the increase in the use of the Internet, many services provided to the users via the web where static HTML pages were not sufficient to deliver the services from the web-applications. Thus, technologies such as Java Server Pages, Active Server Pages, and others were introduced for the delivery of dynamic web contents. Moreover, with the advancement of graphics processing and high resolution displays, the multimedia contents that the websites display have advanced significantly.

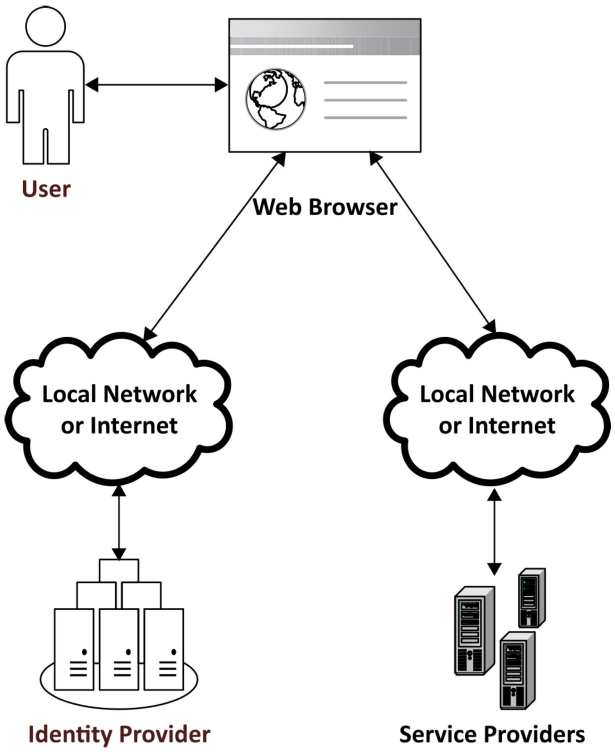

FIGURE 1. Conceptual view of a browser-centric single sign-on authentication process [24]. Interaction sequence is shown in Fig. 2.

To keep up with the advances, web browsers increased their capabilities. Besides HTML page interpretation and faster rendering, modern web browsers perform such complex functions as processing Cascading Stylesheets (CSS) instructions and execution of JavaScript functions [27]. Now, browsers have evolved into very powerful applications. They are used for faster rendering, handling multi-format content, managing sessions, processing advanced graphics, and caching to provide an enhanced browsing experience for the users [27].

1) TABBED BROWSING

Modern web browsers provide a feature to the users called browser tabs. The browser tabs enable users to view several websites in a single browser window simultaneously; websites are separated by tabs of the same browser window. Tabbed browsing enhances the browsing experience, and is easier to manage multiple web pages. When a user opens several pages of a same website in different tabs, the tabs usually share the session information [23]. When a browser signs-in to a website on one tab, and opens another link from the same website on a new tab, both the tabs share the information, so that the user is not required to sign-in again. The user adaptation for tabbed browsing was also positive, as they might keep all their browsing activities organized in a single window [10].

Private browsing or incognito mode is an option available in the browser. In this mode of browsing, the browsing history

and the information entered in forms is erased as soon as the browser window is closed [15]. However, during an active browsing session in private mode, the browser tabs within the private window share the information among one another.

In a typical SSO implementation, the browser is the user-agent and assists by performing following tasks,

1) URL redirection, between the SP and the IDP,
2) Session information storage, and
3) Forwarding the assertion information of the user to the SP through SAML.

B. IDENTITY PROVIDER

Identity Provider (IDP) is the entity that authenticates the user and provides authorization information to the SP. Usually, every organization has a centralized authentication server which maintains the list of the users in its realm.

In a SSO implementation, the IDP performs following functions,

1) Receives user-authentication requests from SPs via user's web browser,
2) Challenges the user via. his/her web browser for sign-on credentials,
3) Creates and forwards the SAML assertion response to SP, and
4) Records the information of authenticated session in the user's web browser.

C. SERVICE PROVIDER

Service Provider (SP) in the context of SSO is the resource provider, the service that the user would like to access for a resource. Usually, a SP is the same portal or website where the user looks for a resource. In a SSO environment, a SP initiates the sign-in process by requesting the IDP of the user, via browser, to authenticate the user for service.

In a SSO implementation, the SP performs following functions,

1) Forwards requests for authentication to the IDP via browser,
2) Validates the authentication response received from the IDP, and
3) Provides or denies access to the requested resource based on privileges authorized by the IDP of the user.

Among these three entities described in this section, the web browser of the user plays a central role, because it coordinates interactions among SP(s), IDP, and the user. Moreover, it maintains the status of active-authenticated sessions.

Next, using state transition diagrams we show the steps in sign-in, sign-out, and time-out events.

IV. STATE TRANSITIONS DURING SIGN-IN, SIGN-OUT, AND TIME-OUTS

Here, we assume that the SPs, and the IDP belong to different platforms (physically or logically). Whenever a user attempts to sign-on to any of the SPs, the user is redirected to the IDP for authentication. The state transitions diagrams shown in this section were drawn by observing domain URLs and corresponding delay-times. We used *Network* tab in the *Developer Tools* option of the browser *menu*. For ease of description, the discussion is divided into two cases:

1) the user has no active session with any of the SPs, and
2) the user has one or more active session with some SPs.

As discussed in *Section VI*, even though there are a variety of storage mechanisms available for the applications, majority of applications use cookies for session management. For convenience, Users opt to *Remember Me* option during the sign-in process to avoid authenticating every time. If this option is selected, the IDPs store the authenticated information of the user in cookies.

A. USER SIGN-IN ACTIVITY: NO ACTIVE SIGNED-IN SESSIONS

Since the user has no active session with any SP, the browser has no record of recent sign-on sessions. Fig. 2 shows the state transition diagram that the browser goes through for the sign-on process. First, the browser is in state S_0. Once a user clicks on a SP's icon, the browser moves to state S_1 and the SP checks for a possible active service session with it; because it is an initial request, the SP requires that the user be authenticated. For this, the SP finds the source of request (Where are you from? - WAYF) and the browser moves to the

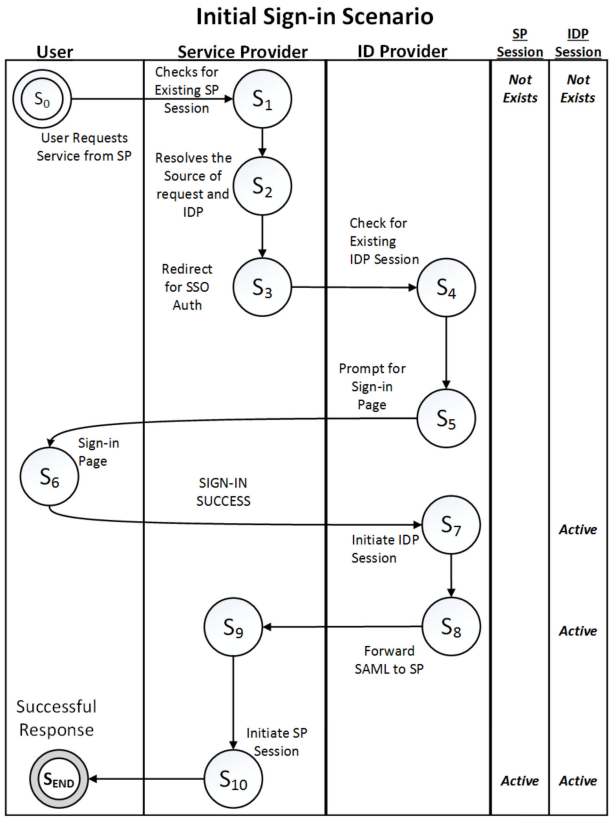

FIGURE 2. State transition diagram of web browser during SSO Authentication process - user initial sign-in.

state S_2. The SP redirects the browser to the IDP of the user, and the browser enters in the state S_3.

Once the browser is redirected from the SP to the IDP for authentication, the IDP checks the browser for an existing active IDP session while the browser is in the state S_4. Since no IDP session exists the browser moves to the state S_5; the IDP challenges the user for valid credentials, the browser moves to state S_6 and displays sign-on page; then the user provides sign-in credentials for possible sign-on. Once the user provides valid credentials (which might be a simple user id and password or a complicated authentication involving a captcha or a multifactor SMS verification, IDP validates user's credentials, forwarded by the browser), IDP creates a new active-session and asserts it on the browser, transitioning the browser's state to S_7. IDP stores the information in the browser cookies. Usually, it also sets a *timer* to T_{IDP} for monitoring idle time of the IDP session. If the session is idle longer than T_{IDP}, the IDP signs-out the user. More about timers and timeouts are discussed in *Section IV-E*.

Also, the IDP creates a SAML response that contains assertion information of the authenticated user for the browser, that moves it to the state S_8. The assertion info of SAML contains information about the user's organization and his or her authorization level. The browser forwards the IDP's SAML response to the SP and enters to the state S_9. After receiving the SAML assertion, the SP creates a session for the user and the browser state changes to S_{10}; now, a service session is initiated. The SP stores the information either in Local Storage or in the browser cookies. The SP sets a timer to T_{SP} for monitoring idle time of the SP session. Further discussions on timers and timeout are presented in *Section IV-E*.

Note that an active IDP session starts from state S_7, that for the SP start from S_{10}. The two timers may run in parallel. We will see how values in T_{IDP} and T_{SP} affect each other.

As presented, in the SSO approach for an initial sign-in, the browser coordinates communication between the SP and IDP. Moreover, browser maintains session information for both the SP and IDP. Browsers may use one more data-storage methods for managing session information. The most common session data storage methods for browsers presented in *Section VI*. Fig. 3 shows the instances of IDP and SP session information organized in the browser. Next, let us see how the states of a browser change when user opens a new browser tab to access another resource of SP_1.

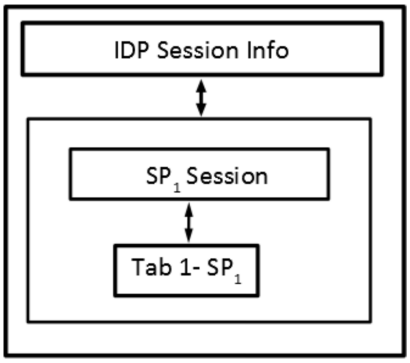

FIGURE 3. IDP and SP sessions during initial sign-in.

B. USER ACCESSING THE SERVICE BY OPENING A NEW BROWSER TAB

Let us consider two scenarios where a user accesses services in a tabbed-browsing fashion, 1) *Scenario 1:* user tries to access to a web page of a SP that was already signed-in, and 2) *Scenario 2:* user tries to access a web page of different SP that leverages the same IDP

1) SCENARIO 1

Once a user signs-in to a SP's website, he or she may access another page of the same website opening a new browser tab. Let us consider a scenario, where an employee is working on several assignments for an organization. All the assignment information is available via *Workday*® system. The employee can sign-in to the system and browse the information for a work assignment. If needed, the employee can open the information of other work assignment from a new browser tab.

Fig. 4 illustrates the states of the browser when a user tries to open another tab of the already signed-in SP. Once the user requests for the resource, the browser tab moves from its initial state S_0 to state S_1. In this state, the SP looks for an existing SP session. As the user was signed on to this SP, the session is active. Hence, the browser takes information from the existing SP session and displays the page requested by the user. The state of the browser moves to S_3. As the session information of a SP is shared by multiple tabs, the user is not prompted to sign-in again.

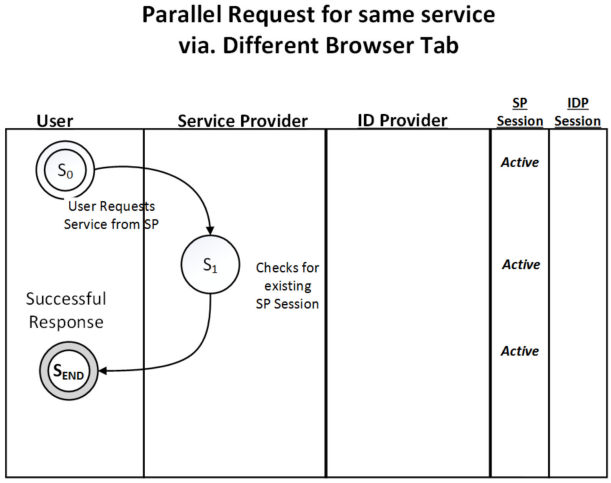

FIGURE 4. Tabbed browsing - parallel request for same service.

Fig. 5 shows how the browser maintains IDP Session and SP Session, also how different browser tabs share the session information of SP_1.

2) SCENARIO 2

In this tabbed browsing scenario, the user signs-in for a service and opens a new tab for another service. Let us consider services SP_1 and SP_2, both leverage authentication

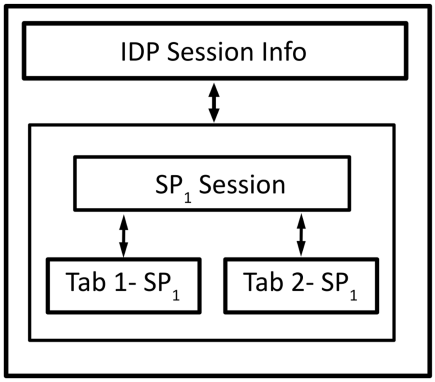

FIGURE 5. Accessing SP_1 from multiple tabs.

of the organization's IDP. Initially, user signs on to service SP_1, following the same steps mentioned in *Section IV-A*. Then, the user wishes to open another browser tab and access another service, SP_2. The service SP_2 also leverages the authentication provided by the same IDP of the organization.

As depicted in Fig. 6, the first section denotes that there is an active browsing session for SP_1, followed by the next section, SP_2 Activity, details us through the steps the browser goes through. When the user opens a new browser tab, the tab is in state S_0. When the user requests a resource from SP SP_2, the browser moves to S_1. In this state, the SP looks for an active SP_2 session. As there is no active SP_2 session exists, SP_2 tries to resolve the source of the request, depicted in state S_2. Based on the source of the request, the SP resolves the IDP and redirects the browser for SSO authentication. Here, the browser moves to state S_3. In state, S_4, the browser

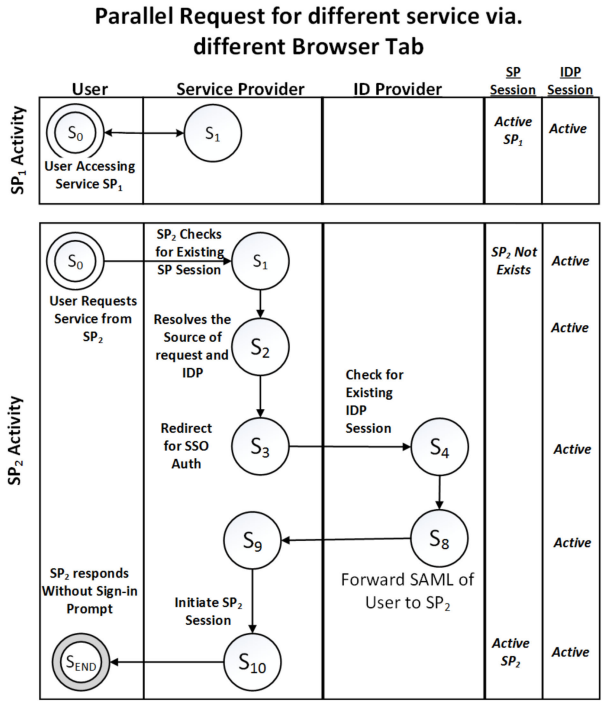

FIGURE 6. Tabbed browsing - parallel request for a different service.

looks for existing IDP session. As the user is already logged in to SP_1 via the previous tab, IDP session is active. So, instead of challenging the user for authentication credentials, the browser retrieves the information from an existing IDP session. The state transitions from S_4 to S_8. The IDP provider creates a SAML response to the requesting SP, i.e., SP_2. The browser transitions to state S_9, where the SP receives SAML from IDP, and parses the user information. In state S_{10}, SP_2 creates a new SP_2 session and as a final state S_{End}, displays the requested resource for the user in the newly opened tab.

Fig. 7 illustrates how the browser maintains IDP Session and SP Sessions for multiple SPs, and how different browser tabs of a SP shares the SP session. However, there is a single IDP session stored in the browser that is accessible by all the SPs.

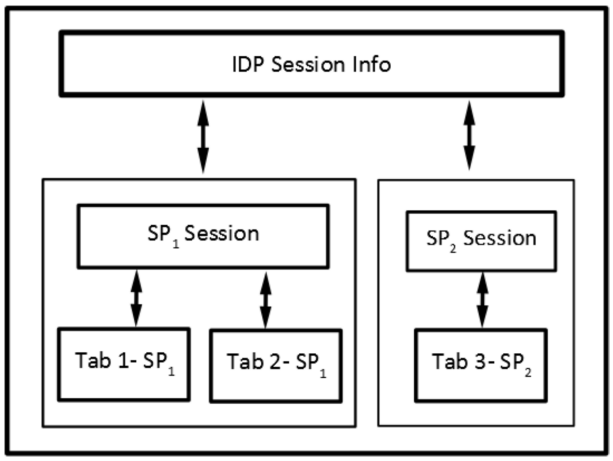

FIGURE 7. Accessing SP_1 and SP_2 via multiple tabs.

Even though the browser allows access to SP_2 without requiring the user to provide credentials, there is a risk now, where the user might sign-out of one SP and assume that he or she will be signed out from all other SPs. We will discuss this scenario in detail in the forthcoming *Section IV-C*.

There are two issues that are critical for users:

1) status of the IDP session after the user signs-out from the SP and
2) possible time-out scenarios because of user's inactivity.

We present them in the following two subsections. First, we present potential (physical) security risks, and then propose possible solutions.

Now, we consider possible situations that arise, because of sign-out by the user.

C. SIGN-OUT ACTIVITY - NO PARALLEL TABBED SESSIONS

As illustrated in Fig. 2, from state S_{10} the browser has two active sessions: One for the IDP and another for the SP, which leads to four possible cases. 1) *Case 1:* user signs-out from the SP session and the IDP session signs-out as a side effect, 2) *Case 2:* user signs-out from the SP session but the IDP session remains active, 3) *Case 3:* user signs-out from the IDP session and the SP session signs-out as a side effect, and

4) *Case 4:* user signs-out from the IDP session but the SP session remains active

As outlined next, each scenario arises from different reasons. Let us consider an employee X, in an organization who wants to check his or her payroll deductions. The user X signs-out from the IDP or *Workday*® (payroll management system), the SP.

1) CASE 1

In this case, as X signs-out from the *Workday*® session, the IDP session signs-out as a side effect.

Recall that X signed-on from the organization's website using the browser in his or her computer. The user visited the organization's website and clicked on service icon of *Workday*®. The browser coordinated with the organization's IDP server for authentication. After successful authentication, the user's authorization level is determined and asserted.

Like most common users, X had no knowledge of IDP's involvement and authentication service. From X's perspective, he or she signed-on to *Workday*® for service and hence, when signing-out from the *Workday*®, X expects that the browser not to have an active (IDP) session.

This would happen only if the SPs and IDP coordinate active sessions. The IDP should know the number of SP sessions that are currently active. As soon as the user signs-out from the last SP, the IDP terminates the IDP session of the user as well.

2) CASE 2

In this case, X signs-out from the *Workday*® session, but IDP session remains active.

Fig. 8 illustrates this situation. It arises when *Workday*®, and the IDP do not interact with each other after the sign-on process. Both *Workday*®, and the IDP are managing their own sessions. Thus, when X signs-out from *Workday*®, it (the *Workday*®) does not inform the IDP that he or she has signed-out and the IDP does not take any action for terminating X's IDP session. Above situation creates a potentially high security-risk condition [38] in any public facility, such as a conference room computer. Because the IDP session is active after X signs-out, anyone can sign-on from the browser

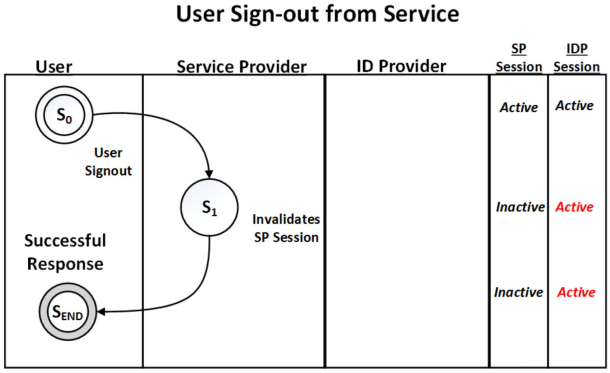

FIGURE 8. User sign-out from service.

that X used to all services that X has permission to use. The level of privileges, of course, are the same that X has.

Let us see the details with an example. After the user X signs-out, another user Y comes to the computer and clicks on a SP's icon. This scenario is represented in Fig. 9. The top section denotes the user X signing out from a service, but the IDP session stays active. In the bottom section, the user Y using the same computer requests a service. Because the IDP has an active session, the user Y does not need any authentication from the IDP. The browser states denoted from S_0 to S_4, is the same as an intial sign-on scenario. At state S_4, the IDP discovers an active session and jumps to S_8 by passing states S_5 to S_7. Then, the IDP creates the SAML response based on X's active session. Consequently, SP also displays the web page or resource specific to the user X and not the user Y.

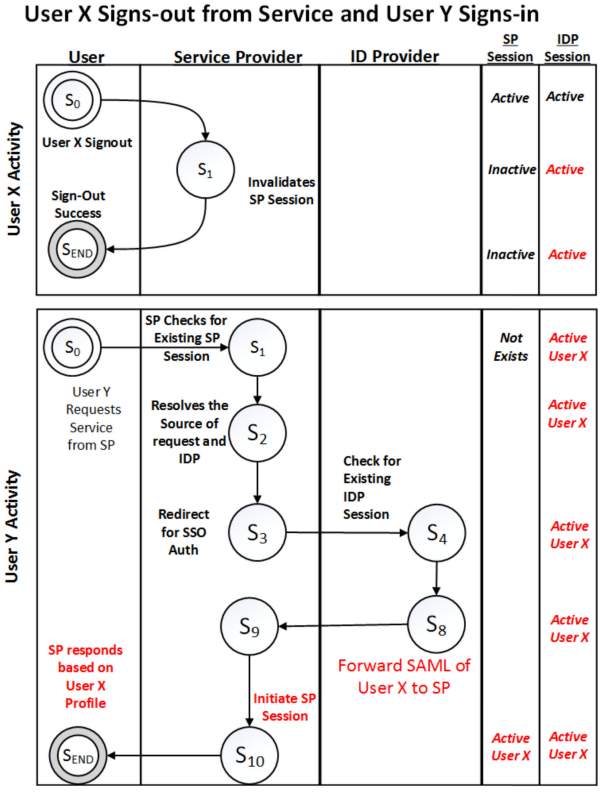

FIGURE 9. User X signs-out from service and user Y signs-in.

3) CASE 3

In this case, user signs-out from the IDP session and SP session signs-out as a side effect.

For most common users does not know about the existence of a central IDP and hence unlikely to attempt to sign-out from IDP. But it is a possible scenario, which is very undesirable. Although the IDP session was initiated by the *Workday*® when X signed-on to the *Workday*®, the IDP is acting as a parent or master process. Hence, the IDP terminates the service session of the *Workday*®, once user X signs-out from the IDP.

This situation is an over-protective security action, which may cause many harms to users. Suppose, X has edited his or her profile in the *Workday*®, and have not saved the changes yet. With the termination of the *Workday*® session, X's new profile is not saved – the old profile is kept. Suppose the user edited his or her email address. Since this is not saved, all the future emails will be delivered to X's old email address.

4) CASE 4

In this case user signs-out from the IDP session but SP session remains active.

This scenario is comparatively less risky than scenario 2. However, this scenario is not in the desired spirit of SSO, because an active session of the *Workday*® should enable X to access other services from her organization. Since there is no active IDP session, the user has to sign-on again for accessing another SP.

Problems will happen in a scenario, where after X signs-out from IDP, but the SP session remains active. So, when another user Y tries to access the same service, the service will direct Y to the resource that belongs to user X. However, if user Y tries to access a different service, then user Y will be redirected to the IDP for authentication.

Most likely, in this case, the IDP and the *Workday*® are keeping independent sessions as we discussed in Case 2.

D. SIGN-OUT ACTIVITY - TABBED BROWSING WITH PARALLEL SESSIONS

In this section, we will discuss the sign-out activity by the user in a tabbed browsing scenario. The key reason we discuss the sign-out activity for parallel tabbed sessions is that, the user does not explicitly signs-in when opening a request in a new tab. It is possible that the user can assume sign-out activity also does not require explicitly signing-out from all the related tabs. Here, there are two cases of sign-out, the user might encounter in the tabbed browsing session. 1) *Case 1:* User signs-out from a SP Session tab where other tabs of the same SP are active, and 2) *Case 2:* User signs-out from a SP Session tab where other tabs of different SP are active.

1) CASE 1

Fig. 5 illustrates us how multiple tabs of a SP share the SP session information. Let us consider an example, User X signs-in into *Workday*® application system via Organization's IDP credentials. The user accesses the application simultaneously using two browser tabs. As we mentioned in *Section IV-B1*, when opening a second tab for *Workday*®, the user did not provide user credentials, as the second tab retrieved the active session information from the browser. Hence, the user assumes that signing out from one of the tabs will sign-out all the active *Workday*® sessions.

In reference to Fig. 8, the sign-out of a service invalidates SP Session. However, if the same service is accessed via a different tab, the service does not prompt the user.

The diagram assumes that the IDP is not signed out or timed out. As per the top section of the Fig. 10, when the

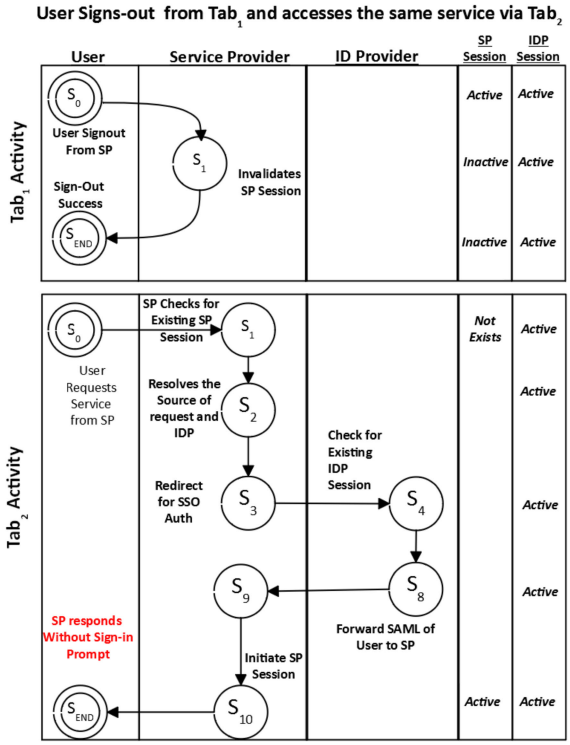

FIGURE 10. User Signs-out from a service in a Tab and accesses the same service via another Tab.

user signs-out the SP session is invalidated. When the user accesses the service via another tab, the browser does not requires user to sign-in again. If we infer the section of Tab_2 *Activity*, the browser transitions from state S_4 to S_8, because the IDP session is still active. So, during the second time request of a service via a different tab, the service retrieves SAML from the IDP session and does not explicitly challenges the user for sign-in credentials.

Here, the scenario is bit confusing for the user because, the user may assume that the sign-out from one tab of a service signs-out all the active sessions, whereas in the actual scenario of SSO implementation as long as IDP session is active, the service is still accessible without user credentials.

2) CASE 2

Let us consider the case where the user signs-out from a service SP_1 and requests a different service SP_2 from another browser tab. Assuming both the SP_1 and SP_2 leverage the service of the same IDP, when the user requests for SP_2, the browser does not prompt for user sign-in credentials.

As in Fig. 7, despite how many services being accessed, they all share the existing active IDP session.

Fig. 11 is identical when compared with the previous case. The top section illustrates the state transition of the browser when the user performs a sign-out activity. The next section shows how the browser transitions when the user requests for another service SP_2 via another tab. After the state S_4, SP_2 looks for an existing IDP session. As the IDP session is still

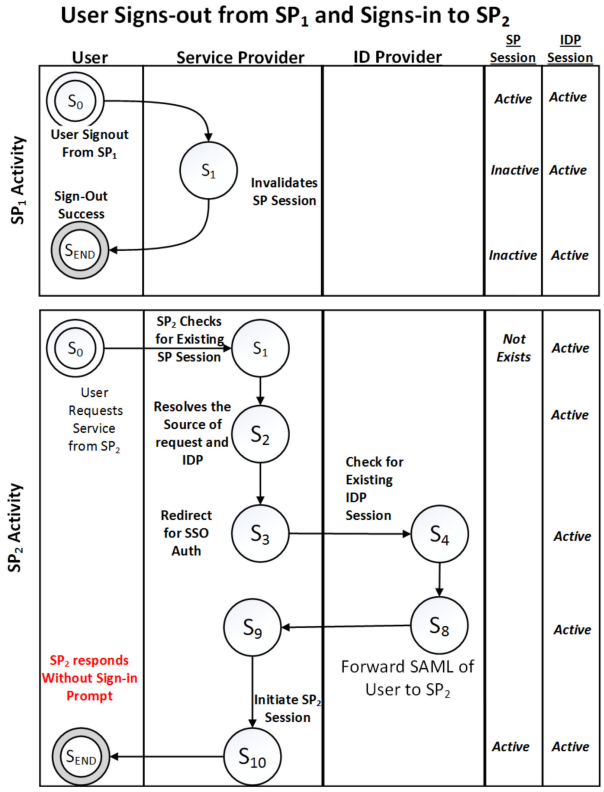

FIGURE 11. User Signs-out from a service in a Tab and accesses a different service via another Tab.

valid, the IDP provider generates a SAML response and sends it to SP_2.

Hence, nowhere in the sign-out process, neither the user is notified nor the IDP is invalidated explicitly.

E. INACTIVITY AND SESSION TIMEOUTS

In *Section IV-A*, while we presented state transitions during sign-on process, we introduced two timers, T_{SP} for the SP and T_{IDP} for the IDP. They are set to a predetermined value at sign-on, and reset to the same predetermined value after every interaction with the user. They are used for terminating sessions that are idle longer than a predetermined period [12]. With two timers, there are four possible combinations of statuses that the two timers can have. Table 2 shows the four combinations.

TABLE 2. Four timeout conditions for inactivity.

Current Status		Relation Between
SP Session	IDP Session	*T_IDP*and *T_SP*Timers
Active	Active	None of the timers has expired
Timeout	Active	*T_SP* has expired *(T_IDP > T_SP)*
Active	Timeout	*T_IDP* has expired *(T_IDP < T_SP)*
Timeout	Timeout	Both timers have expired

The first case and the last case needs no discussion, but the second and third cases are interesting and can arise under myriad situations. As for time-out, let us assume two situations: 1) the SP does not communicate with the IDP after IDP authenticates a user and 2) SP always communicates its state to the IDP .

1) TWO TIMERS RUN INDEPENDENTLY

a: $T_{IDP} > T_{SP}$

If $T_{IDP} > T_{SP}$, then SP times-out SP session when the SP is idle for T_{SP}, but the IDP session remains active. Note that this condition is equivalent to Case 2 in *Section IV-C*. The user can sign-on to any service without further authentication (that requires authentication from the IDP if there were no active IDP session). If a user is unaware of this situation, potentially the user leaves an active IDP session that is open to any malicious actor for potential exploitation; for instance in a public library a malicious actor maybe waiting for such an opportunity.

b: $T_{IDP} < T_{SP}$

The other case arises when ($T_{IDP} > T_{SP}$), the IDP session is timed out but the SP session is still active. Note that this case is equivalent to Case 4 in *Section IV-C*. However, the security risk is lower because the user has access to the active SP, but have no access to other SPs that the user has authorization. Table 3 provides a summary of these two situations.

TABLE 3. Timeout from either SP or IDP, but not both creates different security risks.

Current Status		Can Sign-on to a SP Without Authentication
SP Session	IDP Session	
Timeout	Active	Yes
Active	Timeout	No

2) SP COMMUNICATES WITH IDP BEFORE TIME-OUT EVENT

a: $T_{SP} > T_{IDP}$

In this case the IDP session times-out while SP session is active. When the SP tries to communicate before inactivation of the session, it may not get any response from the IDP. The SP time-out will not keep any active session in the browser.

b: $T_{SP} < T_{IDP}$

In this case, the IDP can make an informed decision. If the user has no other active session, then ideally the IDP would sign-out the user. The security risk is significantly reduced.

F. ACCESSING OTHER SPs AFTER AUTHORIZATION FOR FIRST SP

After the initial sign-in, the IDP has an active session, and the user is allowed to access all the other SPs that is authenticated by this IDP, which is the main advantage of SSO. Analyzing security issues that may arise when more than one SP is active is beyond the scope of the present paper.

Next, we propose some solutions for the security risks that we have discussed in this article.

V. SOLUTIONS TO AVOID UNDESIRED ACTIVE SESSIONS

There are many advantages of SSO. However, users are still concerned about the implementation that revolves around the security and privacy of user information [36]. In this section, we discuss how application developers and SSO service developers prevent hidden active sessions.

In the following diagrams, the proposed solution of states and transitions are drawn with dotted lines.

A. NOTIFYING THE USER DURING SIGN-OUT

During a SP sign-out activity, the SP can display a message to the users, "You are signing-out from the SP, ***however, your may have other active sessions***, including an IDP session that was created by your authentication server." This alert message will create an awareness to the user that there might be other sessions that the user has to manually sign-out.

This notification will be very helpful in a tabbed browsing scenario where when the user signs-out from one tab the SP can notify that there might be other tabs that are active. The sign-out confirmation link can also include a trigger or link that invalidates the IDP session. This will reduce the risk further when the user opened multiple tabs for multiple service sessions.

B. TRIGGERING IDP SESSION INVALIDATION DURING SP SIGN-OUT

Another solution is, whenever the user signs-out of a service, the SP application sends a notification to the IDP that triggers a sign-out process of the IDP session. As shown in Fig. 12, a request from the user for a sign-out from an SP, where the initial state of the browser is S_0. The SP invalidates the browser's SP session (without invalidating IDP session status), the browser transitions to S_1. Then, the SP provides a notification to the IDP, either as an active request to invalidate the IDP session or a passive message that SP session is closing. For this transition, the browser moves to state S_2. Finally, the user is signed-out of SP and the browser moves to the state S_{END}.

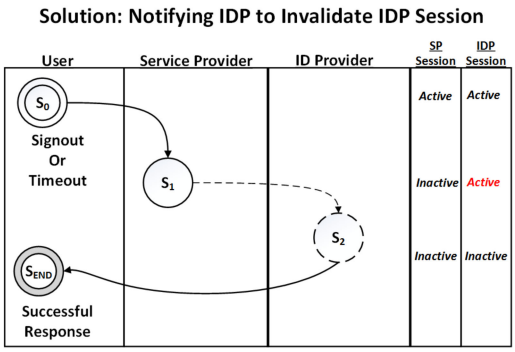

FIGURE 12. Invalidating IDP session during SP sign-out / time-out.

Thus, the IDP session becomes inactive. Any other users trying to access the service must sign-in with their own set of credentials. This solution will avoid illegitimate users accessing other user's resources as described in *Sections IV-C and IV-E.*

C. SYNCHRONIZATION OF SP AND IDP TIME-OUTS

In the implementation where the SP and the IDP are on different platforms, the time-out limits may also vary. The SPs can implement their applications in a way that the timeout threshold of their services is synchronized with the timeout threshold of the IDP of the user is associated to. In this case, if the SP times-out a user's session, the IDP also times-out the user provided other SP sessions are not active.

As in Fig. 13, during the browser state S_8, the IDP can populate its timeout threshold as a part of the SAML response. When it is forwarded to the SP, browser state S_9, the SP can modify its application time limit if possible to the same limit of the IDP. This step will eliminate discrepancies arising because of different timeout thresholds of the IDP and the SP.

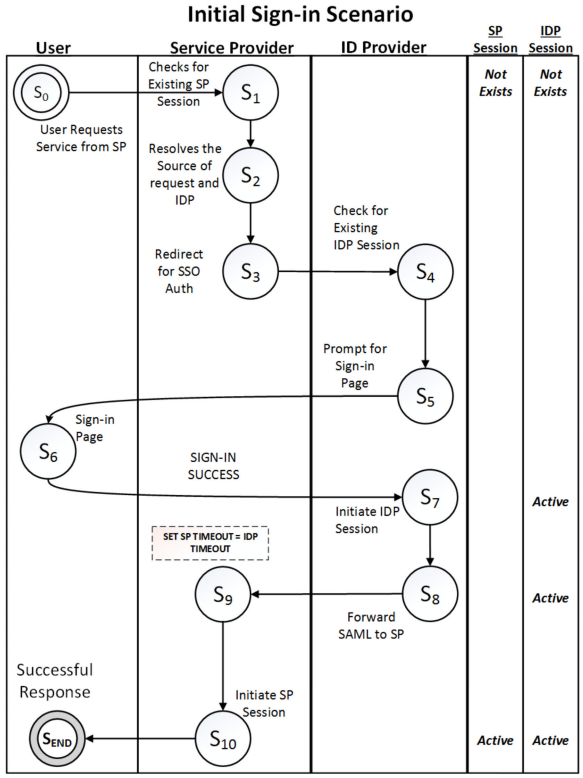

FIGURE 13. Synchronization of SP and IDP timeout limits based on IDP SAML response.

There are many commercial implementations of the IDP. If this solution is to be implemented perfectly, all the SPs should be designed to follow a common standard that specifies how the timeout limits to be set and communicated with the IDP.

D. AUTOMATICALLY SIGN-OUT AT BROWSER WINDOW CLOSING EVENT

With modern browsers and advanced browser scripting technologies, the applications can be implemented in such a way that if the user closes the browser window, the SP

session variables are invalidated. Many users believe that closing a browser window is the same as a sign-out activity. Implementing auto invalidation of a session during a window closing event helps where the users do not perform an explicit sign-out, and in situations where multiple users access a common workstations, such as, a conference room, or a public library.

Fig. 14 shows the event that causes the browser window to close, with initial state of the browser at S_0, an automatic script invokes a SP's sign-out sequence, the browser transitions to S_1, and subsequently triggering a notification to the IDP of the corresponding user's, for a sign-out request — depicted in the diagram showing that the browser's state transitions to the state S_2. At this state both the SP, and the IDP sessions are inactive.

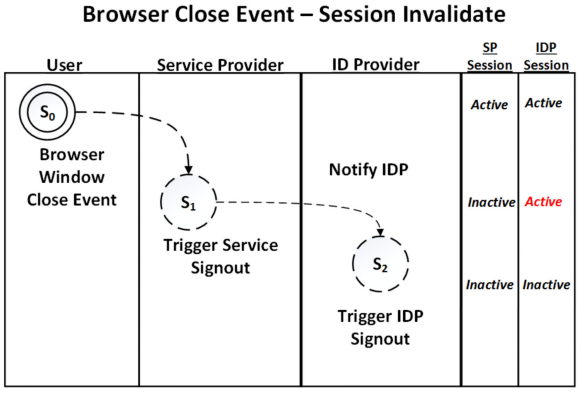

FIGURE 14. Automatically trigger sign-out during browser window close event.

The automatic trigger for IDP sign-out during a SP sign-out is effective during tabbed browsing scenarios also, where the browser can alert the user and trigger the IDP sign-out. This way, the user can feel secure about the sign-out activity that no one can access any services without their respective sign-in credentials.

Hence, when a new user opens the browser, no sessions will be valid, and the user has to follow a complete sign-in process as we have shown in Fig. 2 in *Section IV-A*.

If the user is browsing in incognito or private mode, all the site data will be erased on a window close event [15], thereby automatically removing all the session information of the IDP and the SPs.

Session-data storage methods commonly used by browsers are presented in the next section. Our two important objectives for this exercise are, the evaluation of merits and demerits of each data storage method and the selection of a data storage mechanism that is best suited for implementation of a browser extension module, to manage the information about the active SP and IDP sessions.

VI. STORAGE-METHODS COMMONLY USED IN BROWSERS

For providing the SSO service, IDPs and SPs store information in the user's browser. The commonly used data-storage methods are A. Cookies, B. Web Storage, C. IndexedDB, and D. Cache Storage. In this section, we describe each storage method, and its merits and demerits.

A. COOKIES

Traditionally, browsers used to use files to store information about sessions of the users. These files are referred to as cookies. Some browsers store each cookie in a small file. However, browsers such as Chrome® store all the cookies in a single file and it is located in Chrome®'s profile folder. Information in the cookies is usually stored in key-value pair format [14]. Website services use *Set-Cookie: <cookie-name>=<cookie-value>* option to set the value of cookies in the browser.

As cookies are the earliest form of client storage, they are susceptible to various security vulnerabilities. Security attacks such as Cross Site Scripting (XSS), Cross Site Request Forgery (CSRF), Session Hijacking etc., target the information stored in the cookies. However, to provide compatibility for older versions of web browsers, many websites use cookies as a storage mechanism. A number of SSO IDPs also use cookies to store the session information of their users.

Considering the various risks in using cookies, modern web browsers provide other methods to store information on the client side.

B. WEB STORAGE

Web Storage is a container, where data is stored and accessed using Web Storage API. While the information stored in the Web Storage container is in the key-value pair format (similar to Cookies), it is more intuitive than that stored in the Cookies. However, one of the big limitation is that data must be in key-value pair format for storing in a Web Storage container. [19]

Web Storage API can be invoked via JavaScript functions. In Web Storage API, information is stored in two object like structures - *localStorage* and *sessionStorage*. They key difference between *localStorage* and *sessionStorage* is persistence of data. In *localStorage*, the data persists even after the browser is closed and opened again [16], however in *sessionStorage*, the data is stored only if the browser, or the browser-tab is open [17]. The *sessionStorage* is removed once the browser is closed.

JavaScript API methods *Storage.setItem("Item Key", "Item Value")* and *Storage.getItem ("Item Key")* are used to store and retrieve information using Web Storage API.

The information stored in a Web Storage container is more secure than cookies because, the data in a container can be accessed only by the SP (or IDP) that stored it, and not by other SPs and IDPs.

C. IndexedDB STORAGE

IndexedDB storage addresses the requirements for storing complex data in browsers. The APIs for creating and managing IndexedDB provide all elements of a NoSQL based database management system that can be used by the SPs and IDPs. Websites can store image, audio, video, and their

hierarchical structures in IndexedDB. By storing complex hierarchical data websites reduce network traffic, and enables faster webpage loading. Moreover, websites use IndexedDB for storing complex hierarchical data for reducing network traffic and loading webpages faster.

Data in IndexedDB is stored and retrieved with the set of JavaScript APIs prefixed with *window.indexedDB*. As stated in [41], it can be visualized as a database of records. Also, it is more efficient than local storage and session storage.

In spite of its efficient data storage capabilities when compared with all other storage methods, data in IndexedDB is not secure because stored data is unencrypted [22]. However, one may overcome this limitation by encrypting data in IndexedDB using Web Cryptographic API [40].

D. WEB CACHE STORAGE AND SERVICE WORKERS

Browsers utilize Cache Storage for providing faster loading and offline viewing of webpages. Cache Storage APIs can be used to store *HTTP Requests* and *HTTP Responses*. This means, Cache Storage API can store not only data objects or values, but also HTML Files, JavaScript and CSS files [18]. By storing HTML, JavaScript and CSS Files, Cache Storage API provides seamless experience to its users even when they go offline.

VII. PROPOSED BROWSER-EXTENSION FOR MANAGING SSO SESSIONS

SSO is convenient, but the possibility of hidden active session is a serious problem for the information security of the user. Especially, sign-out from a SP session does not guarantee sign-out from corresponding IDP session. While most of the SSO implementations assume that SPs and IDPs use timeouts for terminating their respective idle sessions, we could see that this situation may leave active sessions as we demonstrated in *Section IV*.

In *Section III* we have seen that in SSO implementations, communication among users, SPs, and the IDP is performed via web browsers in the form of URL redirection and assertion forwarding. Moreover, session information also stored in the browser. Therefore, browser can gather knowledge of the events and the session information of the users.

The proposed browser extension ensures sign-out from an IDP session, if the user signs-out from all the SP sessions that required authentication from the IDP. Browser extensions are software programs that add to the features of a web browser [13]. The proposed browser extension is called SSO Manager. The SSO Manager resides in the browser.

The SSO Manager uses IndexedDB storage for storing information of SP and IDP sessions [32]. The information in IndexedDB storage is updated whenever a user initiates a session of a SP, or adds a new Tab of an existing SP session, or closes a tab of an existing SP session.

A. DESCRIPTION OF THE SSO MANAGER

The SSO Manager consists of five interacting modules: *1) User Event Observer, 2) Input Controller, 3) Sign-in Handler, 4) Sign-out Handler,* and *5) Storage Data Handler*. The block diagram of the SSO manager with its components is shown in Fig. 15. A brief description of each component and major interactions among them is presented next.

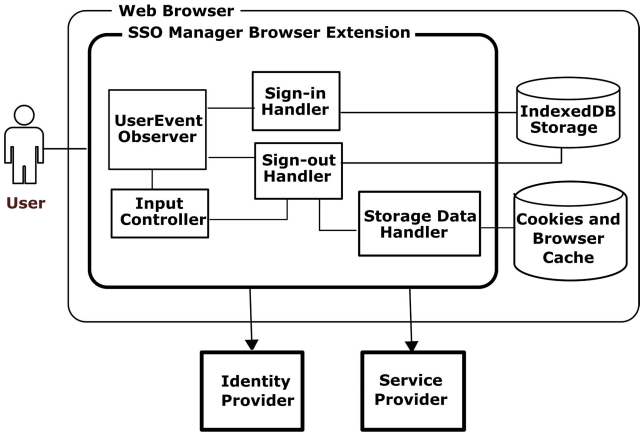

FIGURE 15. Block diagram of the proposed SSO manager.

1) USER EVENT OBSERVER

This module tracks all URLs requested by the browser. This includes a user accessing a service for the first time, accessing the same service via multiple tabs or windows, and accessing other services for the same IDP. When it detects a sign-in event, it sends the information to Sign-in Handler. Similarly, when it detects a sign-out event, it sends that information to the Sign-out Handler for further handling. Moreover, when it detects a timed-out event, the information is communicated to the Sign-out Handler.

Algorithm 1 shows an outline of an ObserveEvent(*SP*) procedure that User Event Observer utilizes to perform the tasks described above.

Algorithm 1 User Event Observer: Observe Event

```
1: procedure ObserveEvent(SP)
2:     for All user events do
3:         if sign-in request or new tab event then forward
   info to Sign-In Handler
4:         else
5:             if sign-out request or tab close event then
   forward info to Sign-out Handler
6:             end if
7:         end if
8:     end for
9: end procedure
```

2) INPUT CONTROLLER

After a user signs-out from a SP or IDP, the Input Controller module displays on the screen a list of active sessions the user still has, if there is any, and request the user to click a button that it displays. If the user does not respond within a predetermined time window, it communicates with the Sign-Out handler to terminate all active sessions of the user.

3) SIGN-IN HANDLER

The Sign-in Handler manages the sign-in events. The inputs to this module come from User Event Observer module after it executes procedure ObserveEvent(SP) in **Algorithm 1**. After receiving information of a Sign-in event, this module writes the information in the IndexedDB Storage using ManageSignIn procedure shown in **Algorithm 2**.

Algorithm 2 Sign-In Handler: Manage Sign-In

```
1: procedure ManageSignIn
2:     if IDP Sign-in URL request then creates a new data entry in IndexedDB
3:     end if
4:     if New Page Request or Tab Request for SP then Iterates through the IndexedDB entries and add the SP Entry to the corresponding IDP
5:     end if
6: end procedure
```

4) SIGN-OUT HANDLER

The Sign-out Handler manages the Sign-out events. Once a sign-out event's information is received from the User Event Observer module, the Sign-out Handler request the Input Controller module to inform the user about the sign-out activity. Based on the user input, the Sign-out Handler removes the data entry from IndexedDB storage, invokes IDP sign-out URLs to request IDP sign-out, and sends relevant information to Storage Data Handler module to remove all the cookies and cache information from the browser. These tasks are executed using ManageSignOut procedure in **Algorithm 3**.

Algorithm 3 Sign-Out Handler: Manage Sign-Out

```
1: procedure ManageSignOut
2:     if IDP SignOut URL request then Invokes Input Controller to prompt user to confirm sign-out
3:     end if
4:     if User confirms IDP sign-out then
5:         for all active service sessions do
6:             Invoke sign-out URL
7:             Request Storage Data Handler to remove all temp data of SPs
8:         end for
9:         Invoke IDP Sign-out URL
10:        Request Storage Data Handler to remove all temp data of IDP
11:    end if
12: end procedure
```

5) STORAGE DATA HANDLER

Storage Data Handler module is responsible for removing cookies and information from the browser storage that are stored during initiation of the IDP and SP sessions.

B. DATA STRUCTURE FOR THE BROWSER STORAGE

The proposed SSO Manager stores the information in IndexedDB browser storage for two reasons: a) IndexedDB supports hierarchical format for data storage and b) data for the IDP and SP sessions can be organized in a nested or tree-like structure.

Fig. 16 illustrates a nested view of the schema of data stored by the SSO Manager module when a user has one IDP session for a SP on a Tab. The outermost rectangular box encloses all SP sessions that are authenticated by the IDP. See Fig. 18 for one SP session from two Tabs, and Fig. 20 for two SP sessions from three Tabs.

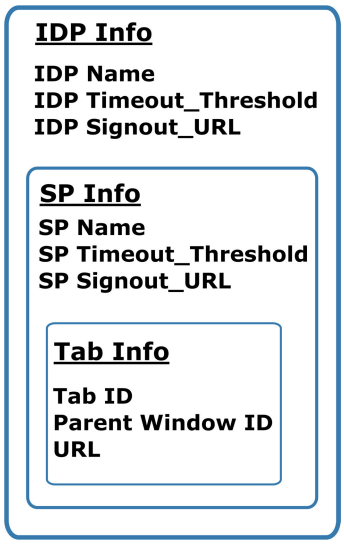

FIGURE 16. Active SSO Sessions - schema design.

Every record holds metadata about the IDP, including name of the IDP, IDP's sign-out URL, and IDP's timeout threshold. The record also contains a list of SP's metadata. As there is a many-to-one relationship between SP and the IDP, the record holds the list of SP metadata for the IDP. Every SP metadata contains the information about the SP such as name or ID, SP's timeout threshold and SP's sign-out URL. The SP information data contains the list of Tabs that the SP is currently accessed from. Tabs are browser tabs, and the data record contains Tab's IDs, parent window's ID, and the URL accessed by the user from the tab.

Whenever a user tries to sign-in via SSO, the SSO Manager gathers information from the browser and stores it in its schema. The functionality of the SSO Manager module is described in the following sections.

C. ILLUSTRATION OF OPERATIONS OF PROPOSED SSO MANAGER

1) INITIAL SIGN-IN SCENARIO

Let us consider a scenario of a user Jane, an employee of an organization (say, X). To access the services such as payroll systems, work hours system, or shared file repository, Jane has to use the credentials provided to her by the organization.

The organization has its own IDP service for authenticating all users.

When Jane tries to access the organization's payroll service's URL, the payroll system as a SP, redirects the browser to the organization X's IDP server.

During the sign-in activity, the User Event Observer identifies the activity and sends IDP and SP information to the Sign-in Handler. During the sign-in process, a SP session is initiated and a record of the session is created, which is as shown in Fig. 17 as **listing 1**. After the sign-on is complete, the Sign-in Handler stores the information in IndexedDB of the web browser. The structure of data is shown in Fig. 16.

Listing 1 : Database Entry with single SP

```
{
 "IDP Name" : "Organization X - ID and Auth",
 "IDP Timeout Threshold " : "45 m",
 "IDP Sign-out URL" : "https://OrgX.com/auth/signout",
 "SP Info" : [
  {
    "SP Name": "Smartpay Payroll System",
    "SP Timeout Threshold" : "35 m",
    "SP Sign-out URL":"https://pay.smartpay.com/signout",
    "Tabs" : [
      {
        "Tab ID": "001",
        "Parent Window ID" : "001",
        "URL" : "https://OrgX.smartpay.com/compensation"
      }
    ]
  }
 ]
}
```

FIGURE 17. Active SSO sessions - schema code listing.

2) ADDITIONAL TABS

Users may want to use multiple tabs to access the same service. Consider the situation where a user accesses the salary information page, and the tax forms page of the payroll management system simultaneously. The user chooses to use two tabs. Once the user opens a new tab and starts a new instance of the same SP, the User Event Observer sends the information to Sign-In Handler, which recognizes the new tab opening event. After receiving the information, the Sign-In Handler adds this information; Fig. 18 shows the updated schema; the corresponding entry in the database is shown as **listing 2** in Fig. 19.

3) ADDITIONAL SERVICES

Similar to the additional tabs, additional services are handled by the SSO Manager. If the employee Jane opens the work hours application (SP_2) on another tab, it (the (SP_2) also utilizes the organization's IDP. Since the IDP session is existing and active, work hours application does not require Jane to sign-in again. State transitions for this type of event has been illustrated in Fig. 6 in *Section IV-B*.

After detecting this new event, the User Event Observer sends the event's information to the Sign-in Handler, which adds SP_2's information to the IndexedDB entry. The schema structure for this case is shown in Fig. 20, and updated database entry is shown as **listing 3** in Fig. 21.

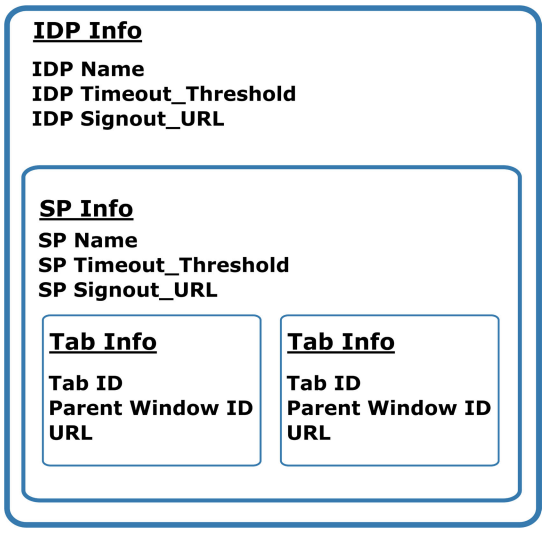

FIGURE 18. Schema - SP with multiple tabs.

Listing 2 : Database Entry with single SP - Multiple Tabs

```
{
 "IDP Name" : "Organization X - ID and Auth",
 "IDP Timeout Threshold " : "45 m",
 "IDP Sign-out URL" : "https://OrgX.com/auth/signout",
 "SP Info" : [
  {
    "SP Name": "Smartpay Payroll System",
    "SP Timeout Threshold" : "35 m",
    "SP Sign-out URL":"https://pay.smartpay.com/signout",
    "Tabs" : [
      {
        "Tab ID": "002",
        "Parent Window ID" : "001",
        "URL" : "https://OrgX.smartpay.com/compensation"
      } ,
      {
        "Tab ID": "003",
        "Parent Window ID" : "001",
        "URL" : "https://OrgX.smartpay.com/taxforms"
      }
    ]
  },
 ]
}
```

FIGURE 19. Active SSO sessions - schema code listing.

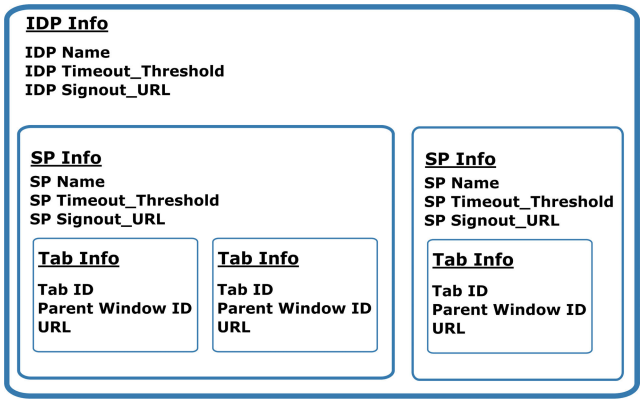

FIGURE 20. Schema structure - multiple *SPs*.

4) SIGN-OUT SCENARIO

The main objective of the SSO Manager browser extension is to efficiently manage the sign-out scenarios. Following are the steps that happen within the SSO Manager, whenever a user signs-out of a service from a tab. User Event Observer interprets the request and forwards this information to Sign-out Handler. The Sign-out Handler gathers the list of

Listing 3: Database Entry with multiple SPs

```
{
 "IDP Name" : "Organization X - ID and Auth",
 "IDP Timeout Threshold " : "45 m",
 "IDP Sign-out URL" : "https://OrgX.com/auth/signout",
 "SP Info" : [
  {
    "SP Name": "Smartpay Payroll System",
    "SP Timeout Threshold" : "35 m",
    "SP Sign-out URL":"https://pay.smartpay.com/signout",
    "Tabs" : [
      {
        "Tab ID": "002",
        "Parent Window ID" : "001",
        "URL" : "https://OrgX.smartpay.com/compensation"
      } ,
      {
        "Tab ID": "003",
        "Parent Window ID" : "001",
        "URL" : "https://OrgX.smartpay.com/taxforms"
      }
     ]
  },
  {
    "SP Name": "TimesheetApplication",
    "SP Timeout Threshold" : "30 m",
    "SP Sign-out URL":"https://OrgX.workhours.com/signout",
    "Tabs" : [
      {
        "Tab ID": "004",
        "Parent Window ID" : "001",
        "URL" : "https://OrgX.workhours.com/home"
      }
     ]
  }
 ]
}
```

FIGURE 21. Active SSO sessions - schema code listing.

other active sessions of the IDP, and forwards it to the Input Controller module. Input Controller module notifies the user of the remaining active sessions, and displays the list of all the active services that were authenticated by the IDP.

Considering the employee user example, if the employee chooses to sign-out from the salary tab of payroll service, SSO Manager prompts a message for input from the user regarding all the active tabs and services for the IDP. In this case, SSO Manager prompts a display where the user can confirm about signing out of all the services.

The action mentioned above, enables the user to determine, whether he or she would like to continue using the services authenticated by the IDP, or completely sign-out of all the services and IDP. The input is received by the prompt displayed by the Input Controller and sent to Sign-out Handler module. The Sign-out Handler module updates the information in the IndexedDB accordingly. If the user opts to sign-out from all the sessions, and the IDP session, the entry in the IndexedDB is removed by the Sign-out Handler module. The module also requests the Storage Data Handler to remove any temporary information such as cookies, from the browser's storage. However, if the user does not intend to sign-out from all the sessions and would like to close the tab or window of that particular session, the Sign-out Handler removes the corresponding entry from the data store.

The SSO Manager also handles accidental closing of browser tabs or entire browser window. If there are no browser windows in open state, the SSO Manager erases all session information from its storage and it triggers a routine for sending sign-out requests to the IDP and all SPs.

The main limitation of the proposed browser extension is that the user must install browser-extension. Another limitation is availability of the proposed data storage APIs. The proposed implementation would not be possible if necessary APIs are unavailable. Also, the browser can only erase data from its storage and send necessary information to SPs and IDP for removing information that they maybe maintaining.

In summary, the SSO Manager constantly monitors all active IDP and SP sessions and invokes sign-out URL based on the situation. It also ensures that the users are made aware of existing active sessions.

VIII. CONCLUSION

SSO or single sign-on gives users a convenient way to access a variety of services of an organization after a single authentication. However, a few scenarios in the implementation of how the applications coordinate with the authentication provider, makes the implementation vulnerable, and may unintentionally leave active SP and IDP sessions. We illustrated how sign-out and time-out from the IDP and the SPs might lead to vulnerable information security situations. We also provided information about how tabbed browsing feature of the modern browsers adds more complexity where there can be multiple tabs accessing one service and sharing the IDP session and one sign-out may add to information vulnerabilities.

The solutions to reduce the vulnerabilities of a SSO feature, mentioned above are not complex to implement, and if implemented, they will reduce the risk of unauthorized access to the resources, because incidences of unintentionally left active SP or IDP sessions will be avoided. Also, if two or more users are actively logged in different tabs the information security of both the active users is at risk of being compromised. Either user who completes the session first might not be aware of his or her own active session, and the other active user may maliciously gain authorization to access the first user's resources.

We also have proposed a browser extension for removing active SP and IDP sessions. We are implementing our proposed extension.

ACKNOWLEDGMENT

The authors thank the reviewers for critically reading the manuscript, and for providing constructive suggestions that have improved the paper substantially. They also thank the journal editor for obtaining timely reviews for the paper, from competent scholars.

REFERENCES

[1] A. Armando, R. Carbone, L. Compagna, J. Cuellar, G. Pellegrino, and A. Sorniotti, "From multiple credentials to browser-based single sign-on: Are we more secure?" in *Future Challenges in Security and Privacy for Academia and Industry*, J. Camenisch, S. Fischer-Hübner, Y. Murayama, A. Portmann, and C. Rieder, Eds. Berlin, Germany: Springer, 2011, pp. 68–79.

[2] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, G. Pellegrino, and A. Sorniotti, "An authentication flaw in browser-based single sign-on protocols: Impact and remediations," *Comput. Secur.*, vol. 33, pp. 41–58, Mar. 2013.

[3] E. B. Campbell, *Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants*, document RFC-6819, Internet Engineering Task Force, Oct. 2015. [Online]. Available: https://tools.ietf.org/html/rfc7522

[4] V. Beltran and E. Bertin, "Identity management for Web business communications," in *Proc. 18th Int. Conf. Intell. Next Gener. Netw.*, 2015, pp. 103–107.

[5] M. Bigler, "Single sign-on," *Internal Auditor*, vol. 61, no. 6, pp. 31–34, Dec. 2004.
[6] S. Cantor, J. Kemp, R. Philpott, and E. Maler. (2015). Assertions and protocols for the OASIS security assertion markup language (SAML) v2.0. OASIS. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
[7] E. Y. Chen, Y. Pei, S. Chen, Y. Tian, R. Kotcher, and P. Tague, "OAuth demystified for mobile application developers," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2014, pp. 892–903.
[8] *Chromium Browser Project, the Chromium Project*. Accessed: Mar. 23, 2020. [Online]. Available: https://www.chromium.org/Home
[9] J. De Clercq, "Single sign-on architectures," in *Infrastructure Security*, G. Davida, Y. Frankel, and O. Rees, Eds. Berlin, Germany: Springer, 2002, pp. 40–58.
[10] P. Dubroy and R. Balakrishnan, "A study of tabbed browsing among Mozilla Firefox users," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst. (CHI)*, New York, NY, USA, 2010, pp. 673–682.
[11] D. Fett, R. Küsters, and G. Schmitz, "A comprehensive formal security analysis of OAuth 2.0," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, New York, NY, USA, 2016, pp. 1204–1215.
[12] OWASP Foundation. (2012). *Session Timeout*. [Online]. Available: https://www.owasp.org/index.php/Session_Timeout
[13] Google. *Google Chrome—Browser Extensions*. Accessed: Apr. 16, 2020. [Online]. Available: https://developer.chrome.com/extensions
[14] Google. *Cookies*. Accessed: Apr. 16, 2020. [Online]. Available: https://developers.google.com/web/tools/chrome-devtools/storage/cookies
[15] Google. *How Private Browsing Works in Chrome*. Accessed: Apr. 16, 2020. [Online]. Available: https://support.google.com/chrome/answer/7440301
[16] Google. *Local Storage*. Accessed: Apr. 16, 2020. [Online]. Available: https://developers.google.com/web/tools/chrome-devtools/storage/localstorage
[17] Google. *Session Storage*. Accessed: Apr. 16, 2020. [Online]. Available: https://developers.google.com/web/tools/chrome-devtools/storage/sessionstorage
[18] Google. *Using the Cache API*. Accessed: Apr. 16, 2020. [Online]. Available: https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/cache-api
[19] Google. *Web Storage Overview*. Accessed: Apr. 16, 2020. [Online]. Available: https://developers.google.com/web/fundamentals/instant-and-offline/web-storage
[20] T. Groß, "Security analysis of the SAML single sign-on browser/artifact profile," in *Proc. 19th Annu. Comput. Secur. Appl. Conf.*, 2003, pp. 298–307.
[21] D. Hardt, *The OAuth 2.0 Authorization Framework*, document RFC 6749, Internet Engineering Task Force, Oct. 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749
[22] S. Kimak, J. Ellman, and C. Laing, "Some potential issues with the security of HTML5 IndexedDB," in *Proc. 9th IET Int. Conf. Syst. Saf. Cyber Secur.*, 2014, pp. 1–6.
[23] K. Kumar and J. Bose, "Segregating user data by tabs in Web browsers," in *Proc. IEEE Asia Pacific Conf. Wireless Mobile*, Aug. 2014, pp. 322–327.
[24] F. Layouni and Y. Pollet, "An ontology-based architecture for federated identity management," in *Proc. Int. Conf. Adv. Inf. Netw. Appl.*, 2009, pp. 162–166.
[25] C. Mainka, V. Mladenov, and J. Schwenk, "Do not trust me: Using malicious IdPs for analyzing and attacking single sign-on," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Mar. 2016, pp. 321–336.
[26] E. Maler and D. Reed, "The Venn of identity: Options and issues in federated identity management," *IEEE Secur. Privacy Mag.*, vol. 6, no. 2, pp. 16–23, Mar. 2008.
[27] A. McPeak and A. M. McPeak. (Jan. 2018). *A Brief History of Web Browsers and How they Work*. [Online]. Available: https://crossbrowsertesting.com/blog/test-automation/history-of-web-browsers/
[28] A. Nair, A. Madhu, and J. J. Kizhakkethottam, "Security issues of single sign on Web services," in *Proc. Int. Conf. Soft-Comput. Netw. Secur. (ICSNS)*, Feb. 2015, pp. 1–4.
[29] NetApplications.com. *Net Market Share—Browser Market Share*. Accessed: Nov. 11, 2019. [Online]. Available: https://netmarketshare.com/browser-market-share.aspx
[30] V. Radha and D. H. Reddy, "A survey on single sign-on techniques," *Procedia Technol.*, vol. 4, pp. 134–139, Jan. 2012.
[31] L. Ramamoorthi and D. Sarkar, "Single sign-on demystified: Security considerations for developers and users," in *Trends and Advances in Information Systems and Technologies*, Á. Rocha, H. Adeli, L. P. Reis, and S. Costanzo, Eds. Cham, Switzerland: Springer, 2018, pp. 185–196.
[32] L. Ramamoorthi and D. Sarkar, "Single sign-on implementation: Leveraging browser storage for handling tabbed browsing sign-outs," in *Developments and Advances in Defense and Security*, Á. Rocha and R. P. Pereira, Eds. Singapore: Springer, 2020, pp. 15–28.
[33] S. Shi, X. Wang, and W. C. Lau, "MoSSOT: An automated blackbox tester for single sign-on vulnerabilities in mobile applications," in *Proc. ACM Asia Conf. Comput. Commun. Secur. (Asia CCS)*. New York, NY, USA: Association for Computing Machinery, 2019, p. 269–282.
[34] *Shibboleth Consortium*. Accessed: Dec. 12, 2020. [Online]. Available: https://www.shibboleth.net/
[35] S. Simpson and T. Groß, *A Survey of Security Analysis in Federated Identity Management*. Cham, Switzerland: Springer, 2016, pp. 231–247.
[36] S.-T. Sun, E. Pospisil, I. Muslukhov, N. Dindar, K. Hawkey, and K. Beznosov, "What makes users refuse Web single sign-on?: An empirical investigation of openid," in *Proc. 7th Symp. Usable Privacy Secur. (SOUPS)*, New York, NY, USA, 2011, pp. 4:1–4:20.
[37] E. T. Lodderstedt, *OAuth 2.0 Threat Model and Security Considerations*, document RFC-6819, Internet Engineering Task Force, Oct. 2013. [Online]. Available: https://tools.ietf.org/html/rfc6819
[38] H. Wang, Y. Zhang, J. Li, and D. Gu, "The achilles heel of OAuth: A multi-platform study of OAuth-based authentication," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl. (ACSAC)*, New York, NY, USA, 2016, pp. 167–176.
[39] J. Wiele, "Pebkac revisited—Psychological and mental obstacles in the way of effective awareness campaigns," in *ISSE 2011 Securing Electronic Business Processes*. Wiesbaden, Germany: Vieweg+Teubner Verlag, 2011, pp. 88–97.
[40] World Wide Web Consortium. (Jan. 2017). *W3C Docs—Web Cryptography API*. [Online]. Available: https://www.w3.org/TR/WebCryptoAPI/
[41] World Wide Web Consortium. (Jan. 2018). *W3C Docs—Indexed Database API 2.0*. [Online]. Available: https://www.w3.org/TR/IndexedDB/
[42] F. Yang and S. Manoharan, "A security analysis of the OAuth protocol," in *Proc. IEEE Pacific Rim Conf. Commun., Comput. Signal Process. (PACRIM)*, Aug. 2013, pp. 271–276.

LOKESH SARAVANAN RAMAMOORTHI (Member, IEEE) received the Master of Science degree in software engineering from the Coimbatore Institute of Technology, Coimbatore, India, in May 2004, and the Master of Business Administration degree from the Miami Business School, University of Miami, Coral Gables, FL, USA, in May 2015. He is currently a Lecturer of software engineering and cybersecurity with the Department of Electrical and Computer Engineering, College of Engineering, University of Miami. In his professional experience, he has worked in software design and development, project management, and technical operations management for Fortune 100 clients. His research interests include cybersecurity, designing secure software, software engineering, software architecture, and health care 4.0. He also holds the following certifications such as the Project Management Professional (PMP), the Certified Information Systems Auditor (CISA), and the Six Sigma Green Belt.

DILIP SARKAR (Senior Member, IEEE) received the B.Tech. degree (Hons.) in electronics and electrical communication engineering from IIT Kharagpur, India, in May 1983, the M.S. degree in computer science from the Indian Institute of Science, Bengaluru, India, in December 1984, the Ph.D. degree from Washington State University, Pullman, WA, USA, in 1986, and the Ph.D. degree in computer science from the University of Central Florida, Orlando, FL, USA, in May 1988. He is currently an Associate Professor of computer science with the University of Miami, Coral Gables, FL, USA. His research interests include concurrent transport protocols, parallel and distributed processing, neural networks, wireless sensors networks and their applications, and security of virtual machines. In these areas, he has guided several theses and has authored numerous articles. He has served on the program committees of the IEEE International Conference on Communications, IEEE Globecom, IEEE International Conference on Multimedia and Expo, the International Conference on Computer Communications and Networks, and the IEEE INFOCOM for many years.