O Single Sign-Off, Where Art Thou? An Empirical Analysis of Single Sign-On Account Hijacking and Session Management on the Web

Mohammad Ghasemisharif, Amrutha Ramesh, Stephen Checkoway, Chris Kanich, and Jason Polakis, ***University of Illinois at Chicago***

https://www.usenix.org/conference/usenixsecurity18/presentation/ghasemisharif

This paper is included in the Proceedings of the 27th USENIX Security Symposium.

August 15–17, 2018 • Baltimore, MD, USA

ISBN 978-1-931971-46-1

Open access to the Proceedings of the 27th USENIX Security Symposium is sponsored by USENIX.

O Single Sign-Off, Where Art Thou? An Empirical Analysis of Single Sign-On Account Hijacking and Session Management on the Web

Mohammad Ghasemisharif
Univ. of Illinois at Chicago

Amruta Ramesh
Univ. of Illinois at Chicago

Stephen Checkoway
Univ. of Illinois at Chicago

Chris Kanich
Univ. of Illinois at Chicago

Jason Polakis
Univ. of Illinois at Chicago

Abstract

The advent of Single Sign-On (SSO) has ushered in the era of a tightly interconnected Web. Users can now effortlessly navigate the Web and obtain a personalized experience without the hassle of creating and managing accounts across different services. Due to the proliferation of SSO, user accounts in identity providers are now *keys to the kingdom* and pose a massive security risk. If such an account is compromised, attackers can gain control of the user's accounts in numerous other web services.

In this paper we investigate the security implications of SSO and offer an in-depth analysis of account hijacking on the modern Web. Our experiments explore multiple aspects of the attack workflow and reveal significant variance in how services deploy SSO. We also introduce novel attacks that leverage SSO for maintaining long-term control of user accounts. We empirically evaluate our attacks against 95 major web and mobile services and demonstrate their severity and stealthy nature. Next we explore what session and account management options are available to users after an account is compromised. Our findings highlight the inherent limitations of prevalent SSO schemes as most services lack the functionality that would allow users to remediate an account takeover. This is exacerbated by the scale of SSO coverage, rendering manual remediation attempts a futile endeavor. To remedy this we propose *Single Sign-Off*, an extension to OpenID Connect for universally revoking access to all the accounts associated with the hijacked identity provider account.

1 Introduction

The creation and management of online user identities has long troubled web developers due to the complexity of such systems and the ramifications of potential vulnerabilities. This is further exacerbated by the feasibility of Sybil attacks [13] and the limitations of systems designed to prevent the automated creation of user accounts at a large scale [40, 30]. The advent of ubiquitous social and mobile platforms necessitated the deployment of technologies that could alleviate the onus of account management and offer a more integrated cross-platform and inter-service user experience. This has resulted in the proliferation of single sign-on (SSO) schemes that allow users to leverage their existing accounts in popular identity providers (IdPs) like Facebook and seamlessly access other web services or mobile apps (referred to as relying parties, or RPs) without the nuisance of repeating the account creation process or creating/managing extra passwords.

Naturally this new paradigm is not without pitfalls, and previous work has extensively explored the design and implementation flaws of SSO platforms that enable a plethora of attacks [46, 53, 49, 3, 28]. While IdPs have been recognized as single points of failure [43], there has been no systematic investigation of the deployment of SSO and how it interacts with RPs' existing techniques for session management. We highlight an underlying limitation of SSO as it is commonly deployed: while RPs universally verify the link between a local account and an IdP account at the moment of account creation, the vast majority use this process to bootstrap a local notion of identity that is not strongly tied to the IdP's account access or control. In this paper we show that even an ephemeral IdP account compromise can have significant, lasting ramifications as adversaries are able to *gain and retain access* to the victim's accounts on other services that support that IdP.

To better understand the interconnected nature of the SSO ecosystem we conduct the first, to our knowledge, large-scale measurement study of SSO adoption. We implement an automated analysis tool that crawls web services and identifies whether the account registration or log in process supports SSO, based on a manually curated list of 65 IdPs. Our study on the top 1 million websites according to Alexa found that 6.30% of websites support SSO. This highlights the scale of the threat, as attackers can gain access to a massive number of web services.

Even though compromised accounts remain a widespread and prevalent issue for major services [10]

(e.g., due to phishing [44]), we motivate part of our threat model by demonstrating a session cookie hijacking attack that allows complete account takeover in Facebook, the most prevalent IdP. This attack is completely undetectable by the user as the attacker's access does not appear in Facebook's list of active sessions. We assess the extent of this risk with a study on our university's wireless network.

Next, we investigate the capabilities and challenges that attackers face when using a hijacked IdP account to compromise the user's RP accounts, under different scenarios. We establish a systematic attack methodology and manually audit 95 of the most popular web and mobile RPs. We find that even though the specification for SSO allows an RP to request reauthentication of the user's IdP account, only two RPs consistently require this authentication during the SSO process. Thus, prior to our disclosure to Facebook, an eavesdropper would have been able to use the stolen Facebook cookies to impersonate victims at any of the other 93 RPs. We also introduce a novel hijacking attack in which the attacker preemptively creates accounts with RPs where the user does not yet have an account. By setting this long-term trap, the attacker can wait for the user to start using that service to obtain sensitive information and misuse the account's functionality.

We also evaluate the visibility of our attacks in both scenarios, and outline steps that attackers can take to minimize the digital footprints left by these attacks. Our findings further highlight the deleterious effect of SSO on account management, as we present an attack that allows the adversary to maintain access to the user's RP account, regardless of potential remediating actions taken by the user (i.e., changing passwords and killing active sessions), without making any changes visible to the user.

Finally, we identify the remediation options that RPs offer to users for preventing attackers from further accessing their accounts. Our analysis reveals that 89.5% of the RPs we evaluate do not offer options for invalidating active sessions. Moreover, manually revoking access and changing passwords is ineffective in many RPs, and practically infeasible as it cannot scale; due to the preemptive account hijacking attack (Section 5), the user would also have to check every new RP she uses in the future. For 74.7% of the RPs *users have no way to recover from our attacks.* This reflects the shortcomings of SSO schemes and the fractured state of the ecosystem; without a process for universally revoking permission across all RPs and simultaneously invalidating all existing sessions in every RP account associated with the compromised IdP account, SSO facilitates attackers in maintaining persistent and pervasive control over victims' accounts. As such, we outline *single sign-off*, an extension to SSO schemes that allows users to initiate a chain reaction of access-revocation operations that propagate across all associated accounts.

This paper makes the following contributions:

- We present the first large-scale study of the SSO ecosystem by measuring the adoption of IdPs in the Alexa top 1 million websites and quantifying the implications that stem from the prevalence of major providers. We have released our dataset to further foster research on SSO.
- We present an in-depth empirical evaluation of the implications of an IdP account compromise, and perform a systematic analysis of the subsequent account authorization and creation process under several novel attack scenarios for 95 of the most popular web and mobile RPs. Our findings offer a comprehensive evaluation of the SSO threat landscape.
- We demonstrate the inherent inability of popular SSO systems to prevent adversaries from maintaining access to users' RP accounts even after permission revocation. As such, we design *single sign-off*, a backwards-compatible extension to OpenID Connect that addresses this threat.
- We demonstrate a proof-of-concept attack against Facebook that results in complete account takeover, to further motivate part of our threat model.

Overall, the pervasiveness of SSO has created an exploitable ecosystem, further exacerbated by the lack of session management and hijacking remediation capabilities. Our analysis of how users can be harmed and how to remediate these attacks will facilitate tackling this significant yet understudied threat.

2 Background and Motivation

Here we provide an overview of how SSO schemes are implemented. We then outline the attacker capabilities assumed by our threat model, and motivate our work through a network traffic analysis study.

2.1 Single Sign-On Schemes

Broadly speaking, SSO is deployed to simplify user access to services in three categories: enterprise login, single login to a suite of distinct yet interrelated services provided by a single provider, and website/application login also called web SSO. Examples include universities using SSO to provide access to unrelated university services such as student grade systems; Google's SSO for services like YouTube; websites like Stack Overflow that support account creation and login using OpenID Connect [36]. The boundaries between these categories are fluid and all SSO schemes are similar at a high level. In this work, we are primarily concerned with web SSO and thus focus our discussion on OpenID Connect, the most recent SSO standard. However, the threats we explore are not restricted to a specific standard.

OpenID Connect is an extension to OAuth 2.0 [20] that provides a standardized method for a web service to re-

trieve identity information from an identity provider using OAuth. The protocol consists of interactions between the following parties:

- The *End-User* wishes to authenticate herself to a website or service.
- The *User Agent* is typically the End-User's browser.
- The *Identity Provider*[1] (IdP) is responsible for authenticating the End-User.
- The *Relying Party* (RP) is the website/service to whom the End-User wishes to authenticate. It is called the relying party[2] since it relies on the assertion of the End-User's identity by the Identity Provider.

OAuth is designed to cover a wide variety of authorization use cases. As such, it has a number of different protocol "flows" which are inherited by OpenID Connect. The most common flow used for authentication is the *Authorization Code Flow*. A concrete interaction between the parties when an End-User logs in is as follows. The End-User initiates logging in to an RP by clicking on a login link in her web browser (the User Agent) thus initiating a sequence of steps that, if successful, results in the End-User being logged in to the RP. Then the User Agent sends a request to the RP's web server as normal and the RP responds by directing the User Agent to visit the IdP's OAuth 2.0 *Authorization Endpoint*, e.g., using a HTTP 302 Found status code. The endpoints are URLs identifying the servers (and pages) responsible for performing the specified action. The User Agent follows the redirection by sending a request to the Authorization Endpoint. The request identifies the RP, the expected response type (i.e., an authorization code), a redirection URL, and the resources to which the RP is requesting access (e.g., basic account information like a user ID).

Now the IdP needs to perform two key steps before sending the authorization code back to the RP. The first step is authenticating the End-User. Precisely how this happens is up to the IdP but essentially:

- If the User Agent is not logged in to the IdP (or if the RP requests it) the IdP response directs the user to enter her credentials. After verifying the credentials, the IdP sets a cookie containing a unique session identifier.
- If the User Agent is already logged in to the IdP, it will already have the cookie. If so, the IdP may not interact with the End-User at all.

Assuming the authentication was successful, the IdP asks for the End-User's consent to share information with the RP, unless consent has been previously given. Having completed the necessary authentication and consent checks, the IdP directs the User Agent to the redirection URL specified during the authorization code request. This URL contains the authorization code as a query parameter. The User Agent follows the redirection thus delivering the authorization code to the RP. Note that both the RP's request for an authorization code and the IdP's response are carried by the User Agent via redirections to the other party's appropriate endpoint.

At this point, the User Agent stops mediating communication between the RP and the IdP. Instead, direct, server-to-server communication occurs. The RP sends a request to the IdP's *Token Endpoint*. The IdP responds with an ID Token and an Access Token. The ID Token contains an opaque string called the *subject identifier* which, together with the specific IdP, uniquely identifies the End-User. The RP may optionally use the Access Token to request additional information from the IdP. Having successfully authenticated, the End-User is logged in to the RP. To avoid having to engage in this protocol for every HTTP request, the RP will set a cookie in the browser. *As long as the cookie remains valid, the browser remains logged in to the RP without the need for any further communication with the IdP* (unless the RP explicitly requires SSO authentication for every session).

2.2 Threat Model

A wide range of attacks can result in users' accounts being compromised. Here we outline two different attack scenarios that capture adversaries with different levels of capabilities, and which present varying degrees of technical difficulty and attack scalability. Our goal is not to exhaustively enumerate methodologies or restrict the attacker to a specific avenue of compromise, but to highlight the diversity of alternative methods that are possible for hijacking user accounts. Moreover, each scenario presents crucial characteristics that affect the nature of the attack. Specifically, phishing can enable stealthier preemptive attacks (Section 5) while session hijacking results in the attacker "bypassing" Facebook's auxiliary detection mechanisms and not appearing in the active sessions (Section 4).

Figure 1 provides a high level overview of the attack workflow, depending on what the attacker has access to; while we use Facebook as the example IdP for the remainder of the paper, the basic transitions (solid lines) are applicable to any IdP. We describe the dotted line transition, which is specific to Facebook, in Section 4.

ⓐ **Phishing** remains the most common cause of compromise, even in major IdPs [7, 44]. By obtaining users' credentials attackers can completely take over users' IdP accounts. For the remainder of the paper we assume that phishers are able to access the victim's IdP account in spite of other mechanisms [1] that might be in place (as found in [7, 31]).

[1]The Identity Provider is referred to as the "OpenID Provider" or OP in the OpenID Connect specification [36]. For consistency with other academic work, we use the term Identity Provider.

[2]The OpenID Connect specification, somewhat confusingly, additionally refers to the RP as the "Client" [36].

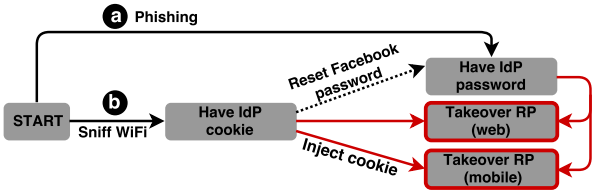

Figure 1: Workflow based on attacker's capabilities.

b Sniff WiFi (Cookie hijacking). Next we consider an eavesdropping adversary that extracts HTTP cookies that allow her to hijack user accounts [8]. This attack is less scalable than phishing as it introduces physical constraints (the attacker needs to be within WiFi range) and can be thwarted by correct deployment of HTTPS. This attacker is *less powerful* as she does not obtain the victim's password. However, as we demonstrate in Section 4, the vast majority of RPs do not require the IdP password to be re-entered, and at the outset of this study Facebook (the most prominent IdP) was transmitting session cookies over HTTP connections. This adversary *highlights the ramifications of SSO even for cautious users that do not fall victim to phishing.*

Use of SSO. For our RP takeover study (Section 4) we assume that the victim has used SSO to create or log in to the RP account at least once. For the preemptive account hijacking attack (Section 5) where the attacker creates the user's RP account, we assume that the user will eventually attempt to create the RP account using SSO. In certain cases the attacks we present work even if the user's RP account has not been associated with the IdP account (i.e., the RP account was created independently) due to how the RP implements the SSO process. For instance, after creating an account on Strava[3] through a traditional account creation process, a user can associate that account with a Facebook account (registered under the same email) using SSO without being asked to input a password. For simplicity, we assume the victim uses SSO in the remainder of the paper.

2.3 Network Traffic Study

This paper explores the security implications of the prevalence of SSO and the remediation actions available to users following account compromise. It is not focused on *how* an attacker can compromise a user's IdP account. Nevertheless, we investigated the feasibility of an IdP *cookie hijacking* attack. We selected cookie hijacking as it affects even cautious users who do not fall victim to phishing attacks.

Cookie hijacking. We audited the network traffic from all popular Facebook apps (main app, Messenger, and Instagram) on the iOS, Android, and Windows mobile platforms. We discovered that browsing in the iOS Facebook in-app browser and visiting websites that serve Facebook's static content (through the like or share button) exposed session cookies because requests for static content on the domain `staticxx.facebook.com` were not protected by HSTS and the cookies were not served with a `Secure` flag or the flag was not enforced properly. This behavior was specific to Facebook's iOS in-app browser. Thus, the initial HTTP request from the in-app browser sent session cookies in cleartext. In a controlled experiment using our own accounts, we demonstrated a successful account takeover by replaying three key values of the captured cookies (`c_user`, `datr`, and `xs`). The exposed cookies result in a complete account takeover, giving the attacker the same level of control over the account as when authenticating using the password. It is worth noting that reusing session cookies in another device does not create any unauthorized access alert, giving the attacker persistent and stealthy access.

[3]A popular service for recording and sharing athletic activities.

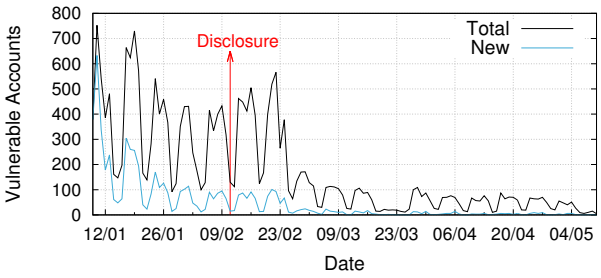

Figure 2: Number of (unique) total and previously unseen vulnerable Facebook accounts seen per day.

Ethics. Before conducting the following experiments in the wild, we had extensive communication with our Institutional Review Board clearly describing our study's objective as well as the data collection and analysis methodology. To ensure the privacy and security of users, all data collection was conducted by network operations staff who only shared aggregated, de-identified data with the research team.

Data collection. To measure the prevalence of this issue in the wild, operations staff installed our logging module on a network tap that monitored our university's wireless network. This module counted the unique values seen for the relevant Facebook HTTP cookies for a period of four months (January–May, 2017). This allowed us to differentiate between accounts and correctly quantify the number which could be compromised by an adversary.

Figure 2 shows the number of unique accounts that exposed the required cookies over an unencrypted HTTP connection each day, as well as the number of unique accounts that had not been previously seen during the experiment. Overall, we collected a total of 5,729 unique vulnerable cookies during our experiment, which were appended to requests toward 11 different Facebook (sub)domains, with `staticxx.facebook.com` being the most common. Since we do not use the exposed cookies to log into the

users' accounts, we cannot eliminate the possibility of the same user exposing different cookie values during the monitoring period. Given the infrequency with which such cookies expire, and the length of the monitoring period, we believe this number closely reflects the actual number of vulnerable users on this network. Finally, the issue affected a considerable number of versions including 28 versions of the iOS Facebook app and 14 of the iOS Messenger app. Despite the sharp decline after our disclosure and subsequent fix, cookies were still being exposed due to users not updating their apps.

This experiment aims to gauge the extent of the damage when wireless traffic is eavesdropped by adversaries. While networks encrypted with WPA2 and a strong, tightly-guarded secret key are infeasible to brute force, well-known keys and open wireless networks (which is common in free public WiFi, e.g., coffee shops, university campuses, public transit etc.) make such man-in-the-middle attacks trivial.

3 Single Sign-On Prevalence

Before exploring the security and privacy ramifications of the tightly interconnected Web, we conduct a large scale study of the proliferation of SSO.

Data collection. For our study we use a list of 65 IdPs that support the OAuth 2.0 and/or OpenID Connect standards along with their corresponding API endpoints, which we based on Wikipedia's list of OAuth providers [48]. We develop a tool for automatically processing websites and extracting information regarding which SSO IdPs are supported in a given domain. The tool is built using the Puppeteer browser automation library [18].

Upon visiting a domain, our tool first traverses all DOM elements found on the landing page. Each element is analyzed for keywords that point to account sign up or log in functionality using a set of regular expressions. If there is no match, the element is searched for sign up or log in links. The same process is repeated for all identified points of interest. If none of the elements return a result, our crawler visits and analyzes predefined link patterns which are commonly used for such functionality (e.g., `example.com/login`, `example.com/signup`) and also issues queries to DuckDuckGo to search for login pages associated with that domain. Once a log in or sign up page is identified, our tool infers which IdPs are supported through regular expressions and searching for links to known SSO API endpoints.

Data analysis. We use our tool to crawl and process the top 1M websites according to Alexa (as reported on September 14, 2017) out of which 912,206 were processed correctly; the others present various errors (e.g., time outs and DNS lookup failures). Our tool identified SSO support on 57,555 (6.30%) domains on the list. Figure 3 shows the coverage for all the IdPs that we encountered during our crawl. We find that Facebook is the most prevalent IdP covering 4.62% (42,232) of the websites, while Google and Twitter follow with 2.75% (25,142) and 1.34% (12,294), respectively. We find that more popular websites are more likely to support SSO, as shown in Figure 4, with a 10.8% coverage in the top 100K,

Cascading account compromise. Our analysis of the data collected during our large-scale study revealed an unexpectedly common behavior. Numerous major websites that function as SSO identity providers also offer functionality that allows users to log in to these sites using other services as identity providers. After manually investigating every IdP's website, we found that 52% of the IdPs exhibit a dual behavior, serving both as RPs and IdPs for other services. Figure 5 shows which identity providers are also relying parties for other identity providers. This behavior is most likely due to the usability benefits of SSO; despite the services having deployed the infrastructure for supporting account creation and management, they still allow users to log in with other services as it offers seamless integration. However, this behavior also exacerbates the security risks of the SSO ecosystem, as it increases the attack surface. Through a series of carefully selected account hijackings, the attacker can gain access to web services that do not support SSO authentication with the initial IdP. The chain of compromises also obscures the root cause, which could further hinder users' remediation efforts. Using a hijacked Facebook account an attacker could indirectly compromise an additional 226 RPs in the top 100K by first compromising the IdPs those RPs support, increasing the respective coverage by 3.1%. For instance, the attacker can first compromise the user's BitBucket account and use that to subsequently compromise the user's GitLab account.

It is important to note that the actual increase depends on both user and website behavior. We do not have data showing how often users inadvertently create a chain of IdPs by opting to associate the account on an IdP that exhibits this dual behavior to a different IdP. On the one hand, one might expect that to be uncommon. On the other hand, the ease-of-use that motivates SSO may result in that being common behavior. Additionally, RPs that allow users to associate an IdP with their account solely through an SSO log in (as discussed in Section 2.2) remain vulnerable nonetheless. Finally, RPs that allow accounts that were created through a traditional creation process to be associated with an IdP account over SSO post facto (e.g., Strava) are also vulnerable regardless of user actions. Figure 6 depicts the impact of this cascading effect for the top 100K websites assuming that the victim's Facebook account has been compromised. The red nodes are the RPs that cannot be directly compromised using Facebook as an IdP but *can* be compromised by first using Facebook as an IdP for a second IdP.

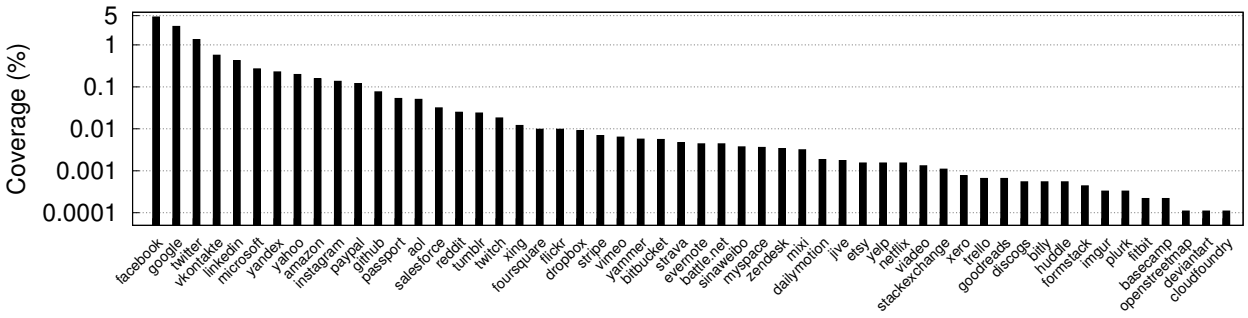

Figure 3: Percentage of websites from the top 1 million that support each identity provider.

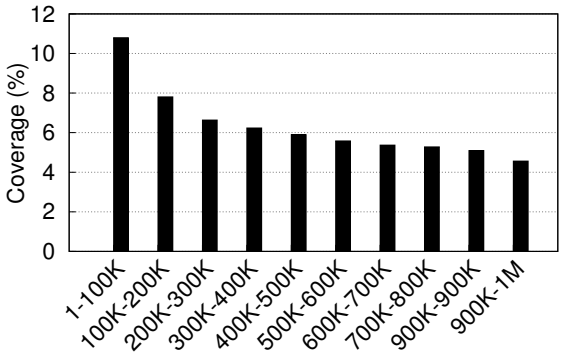

Figure 4: Percentage of websites that support SSO per website rank.

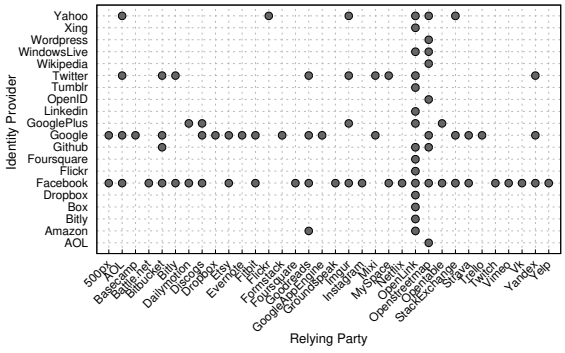

Figure 5: Dual behavior of IdPs that also operate as RPs to other IdPs.

4 Relying Party Account Takeover

Here we present our study on the feasibility of RP account hijacking. We show how attackers can leverage SSO to take over a victim's accounts across web and mobile services, and the ensuing ramifications.

Preconditions. Before any account compromise has occurred, the user creates an account in an RP using the IdP account. At some point after account creation, the attacker gains access to the user's IdP account. This can occur in several ways as captured by our threat model. To achieve her ultimate goal, whatever that may be, the attacker would like to log in to the user's account at the RP and interact with the service, thus obtaining access to whatever information or functionality is available.

Methodology. To determine the level of access the attacker has in the RP, we manually evaluated 29 websites out of the Alexa top 500 and 66 popular iOS apps that support Facebook SSO. We selected RPs from a wide range of different categories and types of functionality. For the iOS apps, we examined the top 10 apps according to the official iOS appstore from popular categories (dating, e-commerce, ride-sharing etc.) and selected those with SSO support. We also examined the Android version for a subset of these apps. See Appendix A for the complete list of RPs.

For each website, we create a new account using SSO and add any additional information the service requires (e.g., a phone number). After completing the account setup, we interact with the service in its usual manner, including sending messages, making purchases, or commenting on articles. Next, we log out of the website. At this point, we switch roles and consider what the attacker can do. We begin by injecting the user's hijacked session cookie into a clean browser session, which we then use to authenticate to the IdP during the SSO flow (see Section 2.1). Unless stated otherwise, we assume the role of the cookie hijacking attacker and do not use the user's IdP credentials in any manner. Next, we visit the RP where the user has an account and go through the normal "log in with ⟨IdP⟩" procedure. Finally, we interact with the website to determine the attacker's level of access. This includes actions like looking at the user's message or order history, sending new messages, or ordering new items.

We perform a similar experiment for each mobile app. The key difference is that there is no support in iOS or Android for adding cookies to Safari or Chrome respectively. We setup a MitM proxy and implement a cookie overwriting attack [52] to inject the hijacked IdP cookie.[4]

[4]Interestingly, while the absence of the Facebook app in iOS results in the RP apps falling back to the internal browser (Safari), in Android the RP apps predominantly rely on the Facebook app for SSO. As a result, cookie hijackers in Android may not be able to conduct the attack unless they can authenticate with the Facebook app using the cookie but not the credentials. Phishing attackers are not affected. Nevertheless, this does not affect the feasibility of the attacks mentioned throughout this paper as the underlying session management issues are independent of the access method and are valid in both iOS and Android.

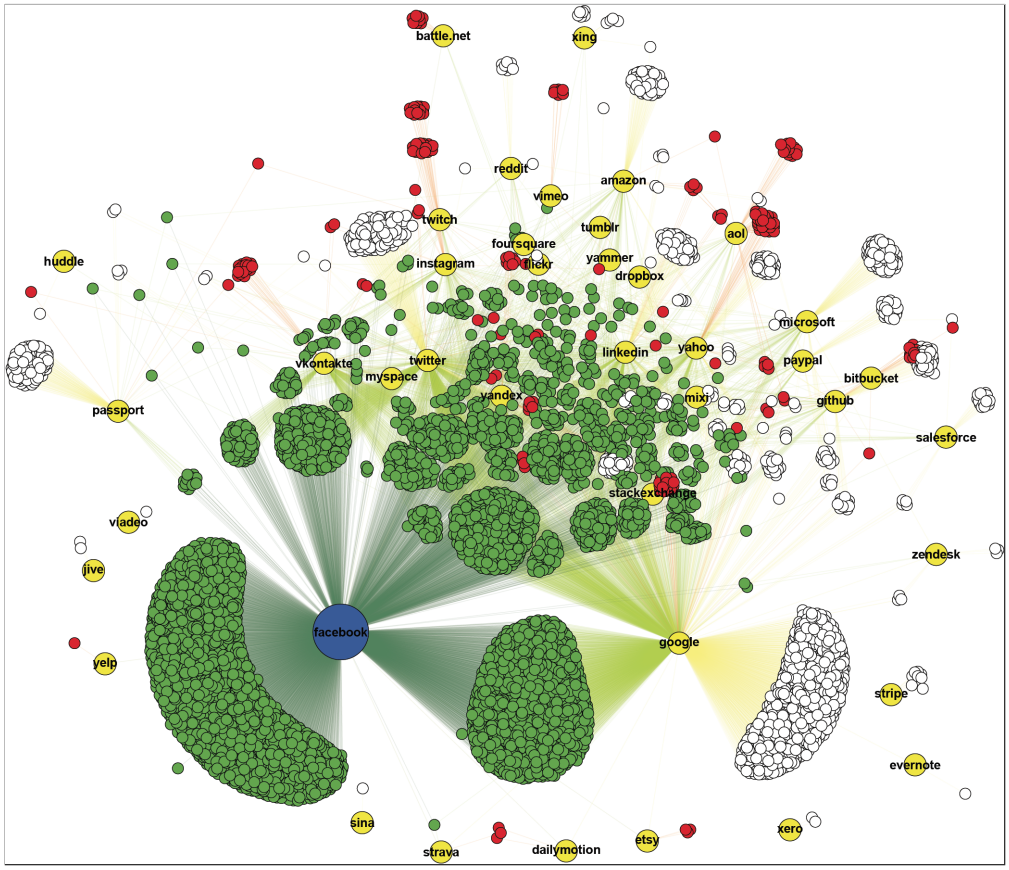

Figure 6: Effect of cascading account compromise in the top 100K websites. IdPs are depicted with yellow nodes (apart from Facebook). The 7,287 green nodes depict RPs that support Facebook login and can be directly compromised by an attacker that has hijacked the user's Facebook account. The 226 red nodes are the RPs that can be indirectly compromised due to IdPs' dual behavior. The white nodes are RPs that can not be indirectly compromised using a hijacked Facebook account.

Results. Table 1 shows a subset of the sites and apps that we tested and details regarding the attacker's requirements and capabilities. In the majority of cases, the attacker's level of access to the website or app was identical to the user's when using the hijacked IdP cookie (●). This is expected, as web site operators and app developers have an incentive to make logging in as painless as possible. In particular, the attacker is prompted to reauthenticate with the IdP in only three of the services (we have identified a workaround for one of them to bypass the restriction). We explicitly state when the hijacked cookie is not sufficient for the attack, i.e., the attacker needs the IdP password (⊛) to view certain information. Next we briefly expand on several interesting entries from the table.

Uber. We can view all account information including the details of previous rides, and can track the victim's trips in real time. The attacker has access to all app functionality; in one experiment we even tipped the driver from the attacker's device after the victim's trip completed.

Hookup. This is one of the RPs that always require reauthenticating the IdP account before getting access. However, we have found a bypass which allows us to gain access using only the IdP cookie; by selecting the *account creation* option instead of the *log in* option, if the session cookie is present the attacker will be authenticated and the system will not trigger an SSO reauthentication process.

The Guardian. We only get partial account access. To reach the settings section the attacker is asked to reauthenticate over SSO and input Facebook's password. However, we have identified a workaround: creating a password for the RP account does not require authentication, and the created password can be used to then obtain full access.

Kayak. With the Facebook cookie we can obtain booking and trip information. Payment information, email settings, and adding travelers requires reauthenticating with the password in Facebook.

Dating apps. We have full control and can view/send messages, "befriend" users etc. The attacker could also befriend an account under her control, and track the user's location in real-time [34]. In HUD, new messages are shown as unread on the victim's phone even if the attacker reads them first.

Table 1: Feasibility of various attack-related actions in a *subset* of the relying parties that we evaluated, along with some of the information or account functionality that an attack can access.

Service	Platforms	Attacker	Access	Password	Email	Messages	Locations	Purchases	User Info	Notes
Tinder	iOS	●	full			✓	N/A	N/A	N/A	Messages remain unread when read by the attacker.
InstaMessage	iOS	●	full			✗	N/A	N/A	✓	Does not support simultaneous access from two devices.
Skout	iOS	●	full			✓	N/A	N/A	✓	View favorite users who the victim swiped right.
Hookup	iOS	●⊛	full			✓	✓	N/A	✓	Found workaround for full access via hijacked cookie.
Ovia	iOS	⊛	full			✓	✓	N/A	✓	Pregnancy/health information. Requires IdP password.
Tripadvisor	iOS	●⊛	full		★	✓	✓	✓	✓	Workaround for full access in iOS: re-login using cookie.
Booking.com	iOS | web | Android	●	full		★	N/A	✓	✓	✓	Susceptible to account combination attack.
Foursquare	iOS	●	full		★†	N/A	✓	N/A	✓	Check-in history.
Yelp	iOS	●	full	†		✓	✓	N/A	✓	Check-ins, purchases, saved locations (e.g., home addr.).
Airbnb	iOS	●	full			✓	✓	✓	✓	Access to trip, reservation, and transaction history.
Expedia	iOS	●	full		★	N/A	✓	✓	✓	Passport number, TSA info, flight preferences, payments.
Kayak	iOS	●⊛	partial			N/A	N/A	✓	✓	Email set via SSO; modifiable in IdP until password is set.
Zillow	iOS | web	●	full		★	N/A	✓	N/A	✓	Credit score, home address. Creating password does not require authentication but sends notification.
Uber	iOS	●	full			N/A	✓	✓	✓	Real-time tracking. Email added w/o authentication.
Goodreads	iOS | web	●⊛	full		★	✓	✓	✓	✓	Zip code, DOB. Workaround bypasses RP's password.
ASOS	iOS | web	●	full	★†	★†	N/A	✓	✓	✓	DOB, home address, payment info, orders.
Quora	iOS | web | Android	●	full			✓	N/A	N/A	N/A	Access to private messages.
Shein	iOS	●	full			N/A	✓	✓	✓	Body measurements, orders, payment options, home address. SSO users can not set password.
Teepr Deals	web	●	full	★†	★†	N/A	✓	✓	✓	Access to recent purchases and credits.
Zoosk	iOS	●	full	†	★†	✓	N/A	✓	✓	Phone number, payments. Password reset via attacker's email.
800 Contacts	iOS | web	⊛	full			N/A	N/A	✓	N/A	Requires IdP password.
IMDB	iOS | web	●	full		★	N/A	N/A	N/A	✓	DOB, zipcode, browsing history.
Mediafire	iOS | web	●	full			N/A	N/A	✓	✓	DOB, zipcode. Access to photos and videos. Email only set via SSO and modifiable until the password is set.
4shared	iOS | web	●⊛	full		†	N/A	N/A	N/A	N/A	Cookie does not work in iOS. Access to photos and videos. IdP password required for full access in iOS.
Pinterest	iOS | web	●	full	†	★	✓	✓	✓	N/A	Creating password does not send notification.
The Guardian	iOS | web	●⊛	partial	†	★†	N/A	✓	✓	✓	Creating password does not require authentication and can bypass IdP password requirement.
WashingtonPost	iOS | web	●	full	†		N/A	✓	✓	✓	Email set via SSO. No notification for password creation.

Attacker: Cookie ● | Credentials ⊛
Email/Password: Modifiable without authentication ★ | No notification †

E-commerce. Apart from granting access to user information and account functionality, the attack enables various scams, e.g., reshipping mule scams [19], fake listings [27], and intercepting deliveries [32].

Attack visibility. An important aspect of the attack is the extent of the attack's visibility, i.e., whether the attack leaves any digital "footprints" that could potentially alert the victim to unauthorized access. While major services that act as IdPs may deploy extra detection mechanisms and show session information, that is uncommon in other services. Specifically, none of the 95 RPs actively notify the user regarding other devices or active sessions. Furthermore, only ten RPs (see Section 6) actually have an option to see the active sessions for the user's account. While a victim could potentially realize that an attack is taking place, this is unlikely for a typical user. Facebook has two security features that could affect the stealthiness of the attack; it shows the active and recent sessions in the account security page. It also offers an option to send the user an alert about logins from unrecognized devices. However during our experiments with hijacked cookies we found that no alert is sent to the victim, and *the attacker's session will not show up in the list* unless its duration exceeds one hour. Thus, in practice the victim will never become aware of an attack taking place.

Long-term access. Despite the stealthiness of our attack, the attacker could potentially lose access to the user's IdP account (e.g., due to a password change). That could prevent the cookie hijacker from accessing the account on nine RPs (two require an SSO reauthentication at the start of every session, and seven log the user out when the IdP password is reset). We design an attack that allows us to maintain access to the RP accounts even after losing access to the IdP, exemplifying the implications of SSO when compared to "traditional" account compromise. The attack entails the following steps:

(i) The attacker completes the SSO process and logs in to the user's RP account.

Table 2: RP behavior during the long-term access attack in the 29 web RPs.

Behavior	Number of RPs
No support for passwords	2
Supports both SSO and passwords	27
Password is optional	25
Password is mandatory	2
Changing email does not require password	15
– Password can be set without reset	*6*
– Password reset sent to attacker's email	*9*
Email can not be changed	5
– Email retrieved from IdP	*3*
– Does not allow change of email	*2*
Changing email requires password	7

(ii) The attacker replaces the email address associated with the RP account with her own email.
(iii) The attacker sets (or resets) the password associated with the RP account.

As a result, the attacker can maintain access to the user's RP account using the attacker's email and password to log in, while the user will still be able to continue accessing the RP account over SSO. To investigate how RPs behave in this scenario in practice, we tested all 29 web RPs from our previous experiment. In Table 2, we break down the numbers regarding how RPs affect this attack. Fifteen services allow the attacker to change the account's email without requiring the password to be entered; of these, six allow the password to be set without entering the old password whereas the remaining nine require the attacker to engage in the password reset procedure which *emails a link to the attacker's newly set email address*. Even if the attacker does not know the user's password she can leverage this process and maintain long-term access in 22 out of the 29 RPs that we tested. To make matters worse, while one would expect that RPs would notify users in the event of an email or password being changed, this is not always the case. Specifically, four services (booking.com, onedio, taringa.net, deals.teepr.com) do not notify the user of these changes and even allow the attacker to make these changes without requiring any form of authentication.

These findings also highlight a different perspective of the amplification effect that SSO can have for attackers. If the victim creates the RP accounts over SSO, only two of those accounts will definitely have a password set; given the burden of "password fatigue" [12] many users will not set passwords in RPs that do not mandate it. In such a scenario, even if the user always reuses her password across all websites, a phisher will *not* be able to compromise 93 out of the 95 RPs without using SSO.

Account linking attack. We also developed another attack that allows the attacker to obtain long-term access to the RP account in a stealthy manner. It requires the RP to support an option to de-link the IdP account (18 of the web RPs do).

(i) The attacker completes the SSO process and logs in to the RP as the user.
(ii) The attacker disconnects (de-links) the user's IdP account from the user's RP account.
(iii) The attacker logs in to her own IdP account, without logging out of the user's RP account.
(iv) While the attacker is still in the user's de-linked RP account, she links her own IdP account to the de-linked RP account.
(v) The attacker re-visits the RP while logged in to the victim's IdP and completes the SSO process.
(vi) The RP now has associated the two separate IdP accounts with the user's RP account.

As a result the attacker can maintain long-term access to the user's RP account, regardless of any changes or actions the user may conduct. We found that five of the web RPs are vulnerable to this attack (Pinterest, booking.com, Quora, 9gag, 4shared). To make matters worse, during our experiments we found that there is no warning to the user. In fact, booking.com actually sends the confirmation email to the attacker's email address; the only notification sent to the user is that the user's IdP account has been disconnected, but no information is given about the attacker's actions or accounts. When the user visits the RP there won't by any difference from prior experiences, thus remaining oblivious to the attack. We consider this design to be a significant risk to users: *under no circumstances should RPs link two different IdP accounts to the same RP account.* The victim could recover from this by logging in to the RP account using her RP credentials, de-linking and re-linking the RP account with her own IdP account. Since this attack leaves no trace, the victim would have to do this for all RP accounts. For Pinterest, users are unable to regain exclusive account control.

The attacker's IdP account must not have been linked to any other account on that RP in the past for the attack to work. In IMDB the RP does not link the two accounts, but actually links the account to the attacker's and the victim is moved to a new empty account upon logging in. This could lead to ransom-type attacks where users will have to pay to regain access to their RP account.

IdP access escalation. We identified an attack that allows the attacker using the hijacked cookie to reset the user's Facebook password (the dotted line transition in Figure 1), by exploiting a loophole in the verification process. When adding a new phone number to the account, the attacker can add her own phone number without needing to reauthenticate via password, and then use that new phone number to reset the account password. Although an email notification is sent to the user, the user's active sessions are not logged out and the attacker can remove her email and phone number to erase any traces. This gives the cookie hijacker the ability to compromise any RPs that require IdP reauthentication.

5 Preemptive Account Hijacking

In this section we present a novel attack and conduct an empirical analysis of its feasibility. We investigate the scenario where the attacker uses the victim's IdP account to preemptively create an account for the victim on an RP at which the victim does not yet have an account. While the attacker could create such accounts for conducting other malicious actions (e.g., sending spam, or as part of an identity theft attack [5]), here we are interested in an attacker who waits for the user to join the RP and then misuses the available information and account functionality. As such, we want to answer the following research questions:

(i) *Will it be evident to the victim that their IdP account had been used to register accounts at these RPs?*
(ii) *What obstacles will the attacker face when trying to maintain access to these accounts?*
(iii) *Will the attacker be able to monitor the user's actions and use the account after the user joins the RP?*

Setup. The attacker identifies an RP of interest where the user does not have an account and uses SSO to create the user's account. After accessing the newly created account, the RP populates the attacker's device with session cookies that enable access to the account. From that moment on, the attacker can periodically check the account for any activity signifying that the user has joined the service.

Methodology. To determine the level of access that the attacker can maintain after the user joins the RP, and also identify any obstacles that the services may pose in practice, we manually recreated the attack scenario in the 95 RPs. Specifically, we visit each RP as the attacker and initiate the "Sign up with ⟨IdP⟩" process. Since the attacker is already logged in to the IdP, the SSO process completes seamlessly in most cases. Only two RPs require the attacker to set a password when creating the user's account (we found a workaround for one of them). In practice, if the attacker has knowledge of the victim's IdP account password (e.g., through phishing), she could set the same password in the RP account as well, taking advantage of the fact that many users reuse their passwords across sites [11]. Nonetheless, for the remainder of the section we consider those two services unsuitable targets for this attack and do not explore them further due to the uncertainty introduced by this factor.

Next we assume the role of the victim and evaluate the stealthiness of the attack by exploring whether there is some form of notification regarding the creation of an associated account in the RP. Then we visit the RP as the victim and initiate the account creation process and log any information shown which might prime the user that something is wrong. Once the account is created, we interact with the service and complete a series of typical user actions. Finally, we switch roles again, and complete the final phase of the attack; we attempt to access the RP account using the session cookie(s) that were created upon the initial visit and also explore what user information or account functionality we can access.

Results. This attack is indistinguishable from the RP account hijacking in regards to the information and account functionality that the attacker can access. In terms of visibility, "Sign In" and "Sign Up" over SSO redirect the user to the same point, and there is no explicit message to signify prior account activity (e.g., something akin to "Welcome back"). The only message that users may receive is that an account is already associated with that email address. Given the confusion of users regarding the SSO login and account-linking process [43] and the complicated nature of SSO in general, this is very unlikely to raise suspicions. On the other hand, during the account setup phase Quora asks the user what topics are of interest to her, which is an obstacle to the attack.

Email disassociation attack. In the straightforward preemptive attack the user will receive multiple email notifications, one for every account creation in an RP. To avoid that, we take advantage of how SSO is leveraged by services, for a stealthier attack.

(i) After gaining access to the IdP, the attacker adds her own email to the user's IdP account.
(ii) The attacker sets her own email as the primary email in the IdP account (this requires knowledge of the IdP password, or the dotted line transition of Figure 1).
(iii) The attacker creates accounts for the user in the various RPs using the common SSO workflow.
(iv) The RP accounts are created under the attacker email but associated with the user's IdP account.
(v) The attacker sets a password on the RP account (if passwords are supported – not mandatory).
(vi) (Optional, to remove traces) After the desired RP accounts have been created the attacker removes her email from the user's IdP account.
(vii) (Optional) After the user starts using a specific RP, the attacker can substitute her email in the RP with the user's email address.

The attacker can maintain access to the RP accounts using her own email and password, while the victim will be able to log in over SSO. More importantly, in terms of visibility, the victim will only receive one notification from the IdP instead of multiple account creation notifications from the RPs. For Facebook, the user will receive an email stating "*Your primary email address was changed from foo@example.com to bar@example.com*". The attacker could opt to run the attack during the night (or repeat and resume across multiple nights), which would give her enough time to create all the RP accounts and remove her email from the IdP account; when the user checks the IdP account settings the only email visible in the settings will be the user's own email (the attacker's email is only shown

during the "password reset" and "sign out of all devices" processes). Also, while the user could potentially check the settings of the RPs in the future after starting to use those services, this is unlikely for a typical user; this can be prevented with optional step (vii) for which only nine RPs send an email to verify the user's email address. This attack is similar in nature to a *login CSRF* attack [4] as the user logs into an account associated with the attacker's email address; however, it differs in practice as the user actually interacts with the account she intended to and which is associated with her IdP account.

Visibility. Typical users may simply ignore alert emails they receive due to not understanding the intricacies of account management or disregarding the emails as fake/phishing. Angulo and Ortlieb found that only 22% of hijacking victims became aware due to a warning by the service [2]. However, in practice attackers can actually prevent victims from receiving any alerts if the attacker can gain access to the user's email provider account or if the compromised IdP account is also the email provider (e.g., Google). This is a reasonable threat, as recent work has found that password reuse remains extremely common [33], and attackers can also leverage knowledge of a user's password (in this case the phisher knows the IdP password) and public PII to "guess" other passwords [45]. Specifically, the attacker can set up filters to proactively remove such alerts by redirecting those emails to the trash folder (setting up such filters is a common attacker tactic according to findings from Google's anti-abuse team [7]). More importantly, even if users become alerted, the majority of RPs lack the functionality needed for users to remediate a compromise as we show in Section 6.

6 Post-Compromise Remediation

Here we explore the remediation actions that users can take if they become aware that their IdP account has been compromised. Our goal is to explore all potential actions that users can take at the IdP or RP to prevent the attacker from further accessing their accounts. Our experiments further highlight the significant implications of SSO; apart from the absence of a standardized mechanism to revoke the attacker's access to all of the RP accounts, we find that for the majority of RPs there is no course of action available that can lock out the attacker.

Conceptually, for a website to authenticate a user with SSO, a two-link chain is created. The first link is the user's authentication to the IdP. The second link is the user's authorization for the RP to access the IdP's stored user identity. We would like for a user who becomes aware that her IdP account has been compromised to be able to sever one of those links and deny the attacker any future access to her account at the RP. In normal usage, the first time the user (or attacker) logs into an RP with a given browser session, the RP will set a persistent cookie in the

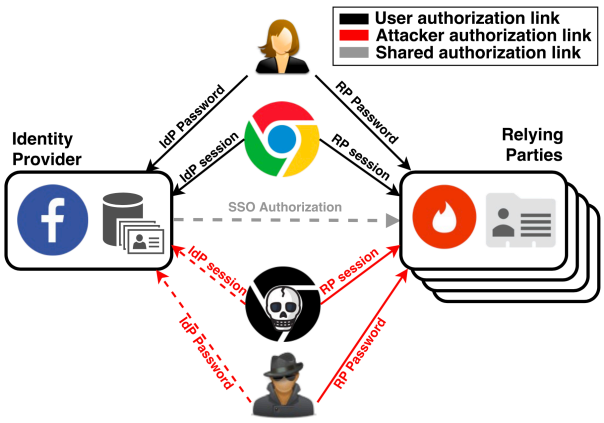

Figure 7: Access links after RP takeover. Only dashed lines can be revoked through the IdP.

browser. After the cookie has been set, the RP will trust the cookie's value to authenticate the user.

The practical consequence of using the RP cookie to authenticate the user is that once an attacker successfully authenticates as the user and receives the persistent cookie, this cookie can continue to be used until it expires regardless of any user action to break the SSO chain (unless she is also able to invalidate that RP cookie). Figure 7 depicts the conceptual connections that exist after the attacker compromising an RP account. The core of the problem is that only a subset of the attacker's connections can be severed through the IdP (shown as dashed lines). As we discuss next, our experiments show that, in practice, the majority of RPs do not offer mechanisms that can completely revoke the attacker's access. And even if such mechanisms were offered by every single RP, the sheer scale of such a manual revocation process would render it impractical. Furthermore, the inner workings of SSO authorization are too complicated for typical users to comprehend and act upon.

Methodology. We explore the options offered by RPs for users to remediate account takeover. Resulting from our investigation we have identified the following actions that a user can take: (i) logout from IdP, (ii) logout from RP, (iii) change password for IdP account, (iv) add or change password for RP account, (v) revoke RP's access to IdP account, and (vi) invalidate active RP sessions. We repeat the attack instantiation process and perform each of these actions independently, and examine how they impact the attacker's access to the RP account. We repeat the experiment for every single RP.

Results. Unfortunately, our findings paint a very bleak picture. Out of the 95 RPs we evaluated, only ten (six web, four iOS) offer some form of session management; for those RPs the user can lock the attacker out by changing the IdP password and invalidating all active sessions in the RP and IdP. In Table 3 we present one of those apps, and all the others that can somehow affect the attacker's ability

Table 3: List of RPs where the attacker's access is affected by one of the remediation actions available.

Service	User Action: IdP logout	RP logout	IdP passw	RP passw	Revoke RP	RP sessions
Tinder	✓	✓	✗	N/A	✗	N/A
Zoosk	✓	✓	✓	✗	✗	N/A
Skout	✓	✓	✗	✓	✗	N/A
GetDown	✗	✓	✗	✓	✓	N/A
Meetme	✓	✓	✗	✓	✗	N/A
Hookup	✗	✓	✗	✓	✓	N/A
Down	✓	✓	✗	N/A	✗	N/A
GoodReads	✓	✓	✓	✓	✓/✗	✓
Yelp	✓	✓	✓	✗	✓	N/A
Expedia	✓	✓	✗	✗	✗	N/A
Kayak	✓	✓	✓/✗	✓/✗	✓/✗	N/A
HomeAway	✓	✓	✓	✓	✗	N/A
Wish	✗	✓	✗	N/A	✓	N/A
Cartwheel	✓	✓	✓	N/A	✓	N/A
Geek	✗	✓	✗	N/A	✓	N/A

Attacker maintains access: ✓ | Attacker loses access: ✗

to maintain access to the account. *For the remaining 71 RPs, the user does not have any course of action to revoke attacker access to the accounts.*

Logging out from the IdP does not affect the attacker if she is already connected to the RP. The attacker will have an issue only if she attempts to reconnect after the RP cookie has expired. Only five of the web RPs have short-lived sessions that could pose an obstacle. It is important to note that for Facebook, the default option presented when changing the password does not affect the attacker. However, we assume a more cautious user that selects the option to log out from all active sessions. Below we provide more details on two interesting cases.

GoodReads. Revoking RP access and logging out from all active sessions logs the attacker out from the web version. The attacker still maintains access in the app.

Kayak. The attacker retains partial read access to the account no matter what actions are taken.

7 Single Sign-Off

Prevalent SSO schemes do not provide functionality for an IdP to universally revoke access to all RP accounts created or accessed from a compromised IdP account. Since such a scenario is not covered by the current OAuth and OpenID specifications,[5] it is crucial to develop a mechanism for mitigating this threat.

[5]The SAML specification describes Single Logout, however it is difficult to implement and breaks under common run time issues [6] and lacks support by major libraries like Shibboleth [38]. Also, it is ineffective when the attacker has a different IdP session from the user [9] (e.g., attacker connects to IdP with user's password). There is a draft specification for IdP-initiated logout for OpenID Connect that is under development. We discuss this in Section 7.2.

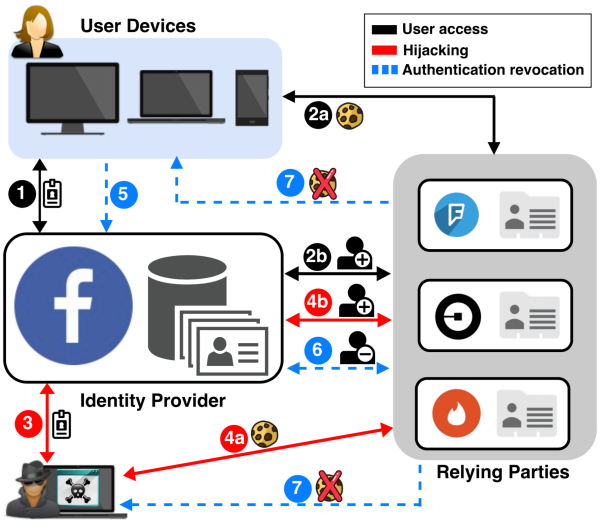

Figure 8: Simplified workflow of an SSO account hijacking attack and the subsequent access revocation.

We present a protocol for universal access revocation designed to enable post-compromise remediation of IdP account hijacking. While we consider the implementation of the single sign-off protocol as part of our future work, we present our current design to kickstart a discussion within the security community on this inherent limitation of SSO and a first step in addressing this significant threat.

Universal revocation. Figure 8 presents the workflow of the hijacking attack and the subsequent steps of the single sign-off universal access revocation protocol. For ease of presentation, we describe a simplified version of the SSO authorization process.

(1) The user creates an account on the IdP and connects from multiple devices by supplying her credentials. This has populated all the required cookies in the respective browsers and apps on each device, allowing the user to seamlessly access the account in the future without the need to reauthenticate.

(2a) (2b) The user visits various sites/apps that support SSO with that IdP, and creates accounts associated to her IdP account through SSO. These services also populate her devices with the required cookies.

(3) The attacker hijacks the user's IdP account through any of our threat model scenarios.

(4a) (4b) The attacker visits the relying parties and leverages the single sign-on functionality to gain access to the user's accounts on those web services and mobile devices. Accordingly, all the required cookies for connecting to the accounts will be populated in the attacker's browser and apps. The attacker now has the same level of access as the user, and will be able to freely access any information or account functionality offered by the RPs. The attacker may also pre-emptively create accounts on other RPs, as described in Section 5.

5 After realizing that her account on the IdP has been compromised, the user connects to her account and initiates the single sign-off revocation process in the IdP. This will first require the user to change her password on the IdP and complete a two-factor authentication step, e.g., over SMS, if it is enabled for the account. Then it will simultaneously invalidate all active IdP sessions on all connected devices.

6 The IdP maintains a list of RPs that have completed authentication or authorization over SSO for that account and revokes their access permission. As aforementioned, this does not sever both edges of the two-link chain created by SSO. To prevent the attacker from having access to the user's RP accounts, the IdP also issues Authentication Revocation Requests to all the RPs that are associated with that account.

7 Once an RP receives a valid Authentication Revocation Request for a specific user account from a supported IdP, it logs out active sessions on all the connected devices, and invalidates all access tokens. The user's accounts on the RPs will be temporarily inaccessible until the user successfully reauthenticates through an SSO process, and will require the user to set a new password (if the RP supports passwords). This also works against the email disassociation preemptive account hijacking attack (Section 5). However, it will not work against the account linking attack (Section 4), and RPs should never implement such functionality. For cases where the RP is also an IdP (Section 3), it will in turn issue Authentication Revocation Requests to all the relying parties that are associated with that user account.

7.1 OpenID Connect Auth. Revocation

Here we detail our proposed backwards-compatible extension to OpenID Connect to support single sign-off by adding support for authentication revocation. To ease implementation, our extension adds a single callback endpoint to each RP and uses standard OpenID Connect messages and data structures.

Client Registration. RPs register with IdPs by sending JSON containing client metadata via HTTP POST to the Client Registration Endpoint [35, § 3.1]. This metadata includes the client name and URIs for redirection callbacks used as part of the authentication flow (Section 2). Our extension adds an authentication revocation URI that the IdP uses to notify the RP that a user's authentication has been revoked and user sessions should be expired. The revocation URI must use TLS. We extend the Client Registration Request [35, § 3.1] to include an additional revocation URI After successful registration, the Client Registration Endpoint returns JSON containing, among other fields, a `client_id` value which uniquely identifies the RP [35, § 3.2]. The `client_id` is used as an audience identifier in the standard OpenID Connect ID Token [36, § 2] and in the Revocation Token described below. Listing 1 depicts an example client registration request.

Listing 1: Example Client Registration Request

```
{"client_name": "Example Client",
 "redirect_uris":
  ["https://client.example.org/callback1",
   "https://client.example.org/callback2"],
   "revocation_uri":
   "https://client.example.org/revocation",
   // Other metadata.
}
```

Listing 2: Example Revocation Token

```
{"iss": "https://server.example.org",
 "sub": "24400320",
 "aud": "s6BhdRkqt3",
 "exp": 0,
 "iat": 1510873662 }
```

Authentication Revocation. Once a user regains control of her IdP account and initiates the single sign-off procedure, the IdP will notify all the RPs for which ID Tokens have been issued, unless the token has already expired, as well as revoke all relevant Refresh Tokens. The IdP will send JSON containing a Revocation Token to the revocation URI specified during Client Registration.

The Revocation Token is a JSON Web Token [24] containing all of the required claims for an ID Token [36, § 2]. Specifically, the Revocation Token contains the issuer identifier (`iss`) which identifies the IdP; the subject identifier (`sub`) which—coupled with the issuer identifier—uniquely identifies the user; the audience (`aud`) whose value contains the `client_id` for the RP; the expiration time (`exp`) whose value must be 0; and the time the JWT was issued (`iat`). The Revocation Token must be signed (and optionally encrypted) using a JSON Web Signature [23] (and optionally JSON Web Encryption [25]) in the same manner, using exactly the same algorithm and keys as the standard ID Token [36, § 2]. Listing 2 shows an example of a Revocation Token.

Upon receiving an Authentication Revocation Request, the RP validates the Revocation Token using the procedure for validating ID Tokens [36, § 3.1.3.7]. If valid, the RP logs that user out of all active sessions, e.g., by expiring all authentication cookies in the user's browsers. The RP responds to a valid Authentication Revocation Request with an HTTP 200 OK status code and to an invalid request with an OAUTH 2 error response [20, § 5.2]. If the RP is itself an IdP, after receiving a valid request, it sends Authentication Revocation Requests to its own RPs. Listing 3 gives an example of our proposed Authentication Revocation Request. The `revocation_token` is a signed JSON Web Token [24]. The line breaks are for visual reasons only. The signature may be verified using the example ECDSA P-256 key given in the JWS standard [23, Appendix A.3].

Listing 3: Example Authentication Revocation Request

```
POST /revocation HTTP/1.1
Content-Type: application/json
Host: client.example.org

{
 "revocation_token":
   "eyJraWQiOiJTU09mZiIsImFsZyI6IkVTMjU2In0.ey
    Jpc3MiOiJodHRwczovL3NlcnZlci5leGFtcGxlLm9y
    ZyIsInN1YiI6IjI0NDAwMzIwIiwiYXVkIjoiczZCaG
    RSa3F0MyIsImV4cCI6MCwiaWF0IjoxNTEwODczNjYy
    fQ.GfUwDTJ-kWFHQo9QyYAkBhvfIeO2o8jji8jUwNl
    KljhMiHRGZxFp2m-kF6LVLkMBJ08Q952djqNr7IQUF
    YS_aw"
}
```

7.2 Alternative Proposal

In independent work, Jones and Bradley [22] describe a back-channel logout mechanism for OpenID Connect. Similar to our proposed Authentication Revocation Request, their approach uses a signed JSON Web Token sent from the IdP to the RP as an HTTP POST request. The two designs are quite similar with a few key differences that we highlight in this section.

Prior work shows that developers often fail to understand the full implications of security mechanisms in practice [26, 39]. This suggests that new security mechanisms should contain as few variants and options as is practicable. Following this principle, we explicitly opted for a straightforward design that minimizes the implementation burden and avoids optional features that may lead to implementation inconsistencies. In contrast, the back-channel logout draft contains several options as well as implementation choices about which user sessions are logged out.

Specifically, the back-channel logout specification draft states that "Refresh tokens with the `offline_access` property normally SHOULD NOT be revoked" and that an open issue is whether to define another optional parameter that would signal that `offline_access` tokens should be revoked. If such a parameter is not defined, then there is potential for attackers to maintain access to the user's accounts through such tokens. The potential risk of this situation is exacerbated by the frequency of access control flaws on the web [41]. If such a parameter is defined, the increased complexity of the specification increases the risk of incorrect and inconsistent implementations across RPs. In contrast, we propose that all user sessions be logged out and refresh tokens revoked.

The back-channel logout proposal is also more flexible than our proposal in that it allows the IdP to specify which user sessions at the RP are to be terminated. Our proposal explicitly states that *all* active sessions on *all* devices *must* be terminated. Although the flexibility of terminating single sessions might be useful under normal operations, it increases the implementation complexity and the likelihood of improper deployment. Offering a user multiple options for session termination may lead to incomplete post-compromise remediation if the user makes the wrong choices.

The similarity of the back-channel logout proposal and our proposal suggests that both approaches are substantially correct. Our findings in this work demonstrate the need for a standardized, universal authentication revocation mechanism, be it our proposal, the back-channel logout proposal, or some other related approach. Although the back-channel logout proposal is a concrete—and much-needed—step toward mitigating the threat of IdP compromise, we believe a simple design with little flexibility is preferable.

8 Limitations and Discussion

SSO coverage. Our crawler attempts to recognize common SSO implementation methods, but developers may use arbitrary methods that it does not recognize or support IdPs that are not in our list. As such, we believe that our results constitute a lower bound but offer a significant step toward better understanding the SSO ecosystem and provide a valuable quantification of SSO adoption.

Single sign-off. An attacker could potentially initiate the revocation process and shut the user out of all RPs. However, apart from the user becoming aware of the attack, the attacker is automatically locked out of all the RP accounts and the user can initiate an account recovery process in the IdP. As such, the attacker actually lacks the incentives to do this. Furthermore, from the users' perspective, temporary lockout is preferable to attackers maintaining account access. Thus, our mechanism offers a remediation strategy against a massive security threat for which users currently lack a defense, and presents benefits that significantly outweigh the potential inconvenience.

Disclosure. The severity of our attacks necessitates their disclosure to the affected parties. We submitted a detailed report to Facebook which led to the subsequent fix of the cookie exposure. We have also notified most of the RPs from our experiments, and provided a description of our presented attacks. As some RPs lack contact info, we have not been able to contact all of them.

9 Related Work

Previous work has extensively demonstrated how web services fail to correctly implement SSO in practice and also conducted formal analysis of the security guarantees of existing protocols. Wang and Chen studied popular SSO implementations and identified flaws that allowed attackers to gain access to user accounts [46]. Zhou and Evans built SSOScan an automated vulnerability checker that analyzed web applications that used Facebook SSO [53]. In [49] the authors presented OAuthTester, an adaptive model-based testing framework for automatically evalu-

ating implementations of OAuth 2.0 systems in practice. They also explored how SSO implementation flaws in dual role IdPs could lead to the amplification of attacks. Bai et al. [3] also demonstrated an automated analysis tool for discovering flaws in SSO implementations. Sun and Beznosov provided an empirical analysis of the implementation flaws of three major OAuth identity providers [42]. Shernal et al. [37] presented a study on the implementation of OAuth 2.0 in popular sites and their vulnerability to CSRF attacks due to the non-compliant implementations. Zuo et al. [54] created a tool for detecting server-side access control implementation flaws.

Fett et al. [15] presented a formal analysis of the OAuth 2.0 specification, and were able to demonstrate four novel attacks against OAuth. The authors had previously explored the privacy limitations of existing SSO schemes and proposed SPRESSO, a privacy-preserving SSO system [16] with provable properties [17]. Sun et al. [43] explored SSO from the perspective of users and identified usability challenges they faced as well as their privacy concerns that stem from RPs accessing their information on the IdP.

Wang et al. [47] uncovered significant flaws in three SDKs provided by major IdPs, by applying a systematic process for uncovering implicit assumptions required for ensuring security. Their analysis showed how these assumptions are violated by app developers in practice, leading to web applications that do not satisfy the required security properties. Recently, Mainka et al. [29] presented a systematic analysis of attacks against OpenID Connect, and demonstrated how techniques used against other SSO systems could be adapted to also attack OpenID Connect. The authors had previously evaluated OpenID and discovered novel attacks that would allow a malicious IdP to compromise the security of all accounts on a vulnerable service provider [28].

Hu et al. [21] focused on social networks and common API designs that leverage OAuth 2.0 for providing access. Their evaluation highlighted an inherent limitation of OAuth's design, which enables an app impersonation attack that can lead to unauthorized data access. Yue conducted a user study to demonstrate how SSO could lead to more effective phishing attacks [50], while Zhao et al. [51] explored how to make the appearance and functionality of SSO phishing websites reflect those of the legitimate websites. Recently, Farooqi et al. [14] studied how collusion networks in Facebook exploit popular apps with weak security settings to obtain OAuth tokens. Sivakorn et al. [41] demonstrated how the lack of ubiquitous HTTPS resulted in the exposure of HTTP cookies granting attackers access to sensitive user data and account functionality in major services.

In contrast to prior work on design and implementation issues of SSO, we explore the security risks surrounding the deployment of SSO, which *would persist even if implementations were complete and correct*. Thus, our study complements prior work by highlighting the ramifications of using SSO alongside traditional local account management techniques.

10 Conclusions

While the SSO paradigm enables seamless integration and effortless navigation, it also epitomizes the single point of failure which the Internet's architects have strived to avoid since its inception. And even though this property is not a vulnerability in and of itself, we have shown that SSO as it is currently implemented exposes users to numerous dangerous and stealthy attacks, some of which extend to services not connected to the original provider. Our novel preemptive account hijacking technique and the feasibility of long-term access to victims' accounts highlight the obstacles to mitigating these attacks and revoking an adversary's access. Even worse, the vast majority of RPs lack functionality for victims to terminate active sessions and recover from such an attack. Even if such functionality were available, the scale of such a remediation would render it a Sisyphean task for users. Guided by our findings and the significant threat posed by these attacks, we designed single sign-off, an access revocation extension to OpenID Connect that enables users to efficiently recover from an IdP account hijack. We hope this will help initiate a discussion within the community, and kick-start efforts to address the shortcomings of existing SSO schemes.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful feedback. We would also like to thank Yan Xuan, Himanshu Sharma and the Academic Computing and Communications Center at UIC for their technical support throughout this project. Finally, we would like to thank Michalis Diamantaris for his assistance. This material is based in part upon work supported by the U.S. National Science Foundation under award CNS-1409868 and a gift from the Mozilla Foundation. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government or the NSF.

Data availability

The dataset from our SSO coverage study can be found at: `https://www.cs.uic.edu/~sso-study/`

References

[1] Alaca, F., and van Oorschot, P. Device fingerprinting for augmenting web authentication: classification and analysis of methods. In *Proceedings of ACSAC 2016* (Dec. 2016).

[2] Angulo, J., and Ortlieb, M. "WTH..!?!" experiences, reactions, and expectations related to online privacy panic situations. In *Proceedings of SOUPS 2015* (June 2015).

[3] Bai, G., Lei, J., Meng, G., Venkatraman, S. S., Saxena, P., Sun, J., Liu, Y., and Dong, J. S. Authscan: Automatic extraction of web authentication protocols from implementations. In *Proceedings of NDSS 2013* (Feb. 2013).

[4] Barth, A., Jackson, C., and Mitchell, J. C. Robust defenses for cross-site request forgery. In *Proceedings of CCS 2008*.

[5] Bilge, L., Strufe, T., Balzarotti, D., and Kirda, E. All your contacts are belong to us: Automated identity theft attacks on social networks. In *Proceedings of WWW 2009* (Apr. 2009).

[6] Browinski, G. Saml single logout - what you need to know. `https://www.portalguard.com/blog/2016/06/20/saml-single-logout-need-to-know/`, June 2016.

[7] Bursztein, E., Benko, B., Margolis, D., Pietraszek, T., Archer, A., Aquino, A., Pitsillidis, A., and Savage, S. Handcrafted fraud and extortion: Manual account hijacking in the wild. In *Proceedings of IMC 2014* (Nov. 2014).

[8] Butler, E. Firesheep. `http://codebutler.com/firesheep`, 2010.

[9] CA Technologies. Single logout overview (saml 2.0). `https://docops.ca.com/ca-single-sign-on/12-52-sp2/en/configuring/partnership-federation/logging-out-of-user-sessions/single-logout-overview-saml-2-0/`, 2017.

[10] Cao, Q., Yang, X., Yu, J., and Palow, C. Uncovering large groups of active malicious accounts in online social networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), CCS '14.

[11] Das, A., Bonneau, J., Caesar, M., Borisov, N., and Wang, X. The tangled web of password reuse. In *Proceedings of NDSS 2014* (Feb. 2014).

[12] Dhamija, R., and Dusseault, L. The seven flaws of identity management: Usability and security challenges. *IEEE Security & Privacy 6*, 2 (2008).

[13] Douceur, J. R. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems* (2001), IPTPS '01.

[14] Farooqi, S., Zaffar, F., Leontiadis, N., and Shafiq, Z. Measuring and mitigating oauth access token abuse by collusion networks. In *Proceedings of IMC 2017* (Nov. 2017).

[15] Fett, D., Küsters, R., and Schmitz, G. A comprehensive formal security analysis of oauth 2.0. In *Proceedings of CCS 2016*.

[16] Fett, D., Küsters, R., and Schmitz, G. Spresso: A secure, privacy-respecting single sign-on system for the web. In *Proceedings of CCS 2015*.

[17] Fett, D., Küsters, R., and Schmitz, G. An expressive model for the web infrastructure: Definition and application to the browser id sso system. In *Proceedings of IEEE Symposium on Security and Privacy 2014* (May 2014), IEEE, pp. 673–688.

[18] Google. Puppeteer. `https://github.com/GoogleChrome/puppeteer`, 2017.

[19] Hao, S., Borgolte, K., Nikiforakis, N., Stringhini, G., Egele, M., Eubanks, M., Krebs, B., and Vigna, G. Drops for stuff: An analysis of reshipping mule scams. In *Proceedings of CCS 2015* (Oct. 2015), ACM, pp. 1081–1092.

[20] Hardt, D. The OAuth 2.0 authorization framework. RFC 6749, RFC Editor, Oct. 2012.

[21] Hu, P., Yang, R., Li, Y., and Lau, W. C. Application impersonation: problems of oauth and api design in online social networks. In *Proceedings of COSN 2014*, ACM.

[22] Jones, M. B., and Bradley, J. OpenID Connect Back-Channel Logout 1.0 - draft 04, 2017.

[23] Jones, M. B., Bradley, J., and Sakimura, N. JSON Web Signature (JWS). RFC 7515, RFC Editor, May 2015.

[24] Jones, M. B., Bradley, J., and Sakimura, N. JSON Web Token (JWT). RFC 7519, RFC Editor, May 2015.

[25] Jones, M. B., and Hildebrand, J. JSON Web Encryption (JWE). RFC 7516, RFC Editor, May 2015.

[26] Kranch, M., and Bonneau, J. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *22nd Annual Network and Distributed System Security Symposium, NDSS* (2015).

[27] Krebs, B. How cybercrooks put the beatdown on my beats. https://krebsonsecurity.com/tag/amazon-hacked-seller-account/, Apr. 2017.

[28] Mainka, C., Mladenov, V., and Schwenk, J. Do not trust me: Using malicious idps for analyzing and attacking single sign-on. In *Proceedings of EuroS&P 2016* (Mar. 2016), IEEE, pp. 321–336.

[29] Mainka, C., Mladenov, V., Schwenk, J., and Wich, T. Sok: Single sign-on security–an evaluation of openid connect. In *Proceedings of EuroS&P 2017* (Aug. 2017).

[30] Motoyama, M., Levchenko, K., Kanich, C., McCoy, D., Voelker, G. M., and Savage, S. Re: Captchas: Understanding captcha-solving services in an economic context. In *Proceedings of USENIX Security 2010* (Aug. 2010).

[31] Onaolapo, J., Mariconti, E., and Stringhini, G. What happens after you are pwnd: Understanding the use of leaked webmail credentials in the wild. In *Proceedings of IMC 2016* (Nov. 2016).

[32] Panther, L. Cyber crooks hack into amazon accounts to place pricey orders and steal the goods. *Mirror* (July 2016).

[33] Pearman, S., Thomas, J., Naeini, P. E., Habib, H., Bauer, L., Christin, N., Cranor, L. F., Egelman, S., and Forget, A. Let's go in for a closer look: Observing passwords in their natural habitat. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017).

[34] Polakis, I., Argyros, G., Petsios, T., Sivakorn, S., and Keromytis, A. D. Where's wally?: Precise user discovery attacks in location proximity services. In *Proceedings of CCS 2015* (Oct. 2015).

[35] Sakimura, N., Bradley, J., and Jones, M. B. OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1, Nov. 2014.

[36] Sakimura, N., Bradley, J., Jones, M. B., de Medeiros, B., and Mortimore, C. OpenID Connect Core 1.0 incorporating errata set 1, Nov. 2014.

[37] Shernan, E., Carter, H., Tian, D., Traynor, P., and Butler, K. More guidelines than rules: Csrf vulnerabilities from noncompliant oauth 2.0 implementations. In *Proceedings of DIMVA 2015* (July 2015).

[38] Shibboleth Contributors. Sloissues. https://wiki.shibboleth.net/confluence/display/CONCEPT/SLOIssues, 2017.

[39] Sivakorn, S., Keromytis, A. D., and Polakis, J. That's the way the cookie crumbles: Evaluating https enforcing mechanisms. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society* (2016), WPES '16.

[40] Sivakorn, S., Polakis, I., and Keromytis, A. D. I am robot: (deep) learning to break semantic image CAPTCHAs. In *Proceedings of EuroS&P 2016* (Mar. 2016).

[41] Sivakorn, S., Polakis, J., and Keromytis, A. D. The cracked cookie jar: HTTP cookie hijacking and the exposure of private information. In *Proceedings of IEEE Symposium on Security and Privacy 2016* (May 2016).

[42] Sun, S.-T., and Beznosov, K. The devil is in the (implementation) details: An empirical analysis of oauth sso systems. In *Proceedings of CCS 2012*.

[43] Sun, S.-T., Pospisil, E., Muslukhov, I., Dindar, N., Hawkey, K., and Beznosov, K. What makes users refuse web single sign-on?: An empirical investigation of openid. In *Proceedings of SOUPS 2011* (July 2011).

[44] Thomas, K., Li, F., Zand, A., Barrett, J., Ranieri, J., Invernizzi, L., Markov, Y., Comanescu, O., Eranti, V., Moscicki, A., Margolis, D., Paxson, V., and Bursztein, E. Data breaches, phishing, or malware? understanding the risks of stolen credentials. In *Proceedings of CCS 2017* (Oct. 2017), ACM.

[45] Wang, D., Zhang, Z., Wang, P., Yan, J., and Huang, X. Targeted online password guessing: An underestimated threat. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), CCS '16.

[46] Wang, R., and Chen, S. Signing me onto your accounts through facebook and google: a traffic-guided security study of commercially deployed single-sign-on web services. In *Proceedings of IEEE Symposium on Security and Privacy 2012*.

[47] Wang, R., Zhou, Y., Chen, S., Qadeer, S., Evans, D., and Gurevich, Y. Explicating sdks: Uncovering assumptions underlying secure authentication and authorization. In *Proceedings of USENIX Security* (Aug. 2013).

[48] Wikipedia Contributors. List of oauth providers. https://en.wikipedia.org/wiki/List_of_OAuth_providers, 2017.

[49] Yang, R., Li, G., Lau, W. C., Zhang, K., and Hu, P. Model-based security testing: An empirical study on oauth 2.0 implementations. In *Proceedings of ASIACCS 2016* (May 2016), ACM, pp. 651–662.

[50] Yue, C. The devil is phishing: Rethinking web single sign-on systems security. In *Proceedings of LEET 2013* (Aug. 2013), USENIX.

[51] Zhao, R., John, S., Karas, S., Bussell, C., Roberts, J., Six, D., Gavett, B., and Yue, C. The highly insidious extreme phishing attacks. In *Proceedings of ICCCN 2016* (Aug. 2016), IEEE.

[52] Zheng, X., Jiang, J., Liang, J., Duan, H., Chen, S., Wan, T., and Weaver, N. Cookies lack integrity: Real-world implications. In *USENIX Security 2015* (Aug. 2015).

[53] Zhou, Y., and Evans, D. SSOScan: Automated testing of web applications for single sign-on vulnerabilities. In *Proceedings of USENIX Security 2014*.

[54] Zuo, C., Zhao, Q., and Lin, Z. Authscope: Towards automatic discovery of vulnerable authorizations in online services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Oct. 2017), CCS '17.

A List of Services

In Table 4 we detail all the web and mobile RPs that we audited throughout our experiments.

Table 4: Complete list of all web services and mobile apps that we audited during our experiments.

Service	Platform	Service	Platform
IMDB	web	Uber	iOS
Pinterest	web	Tinder	iOS
Imgur	web	Yelp	iOS
NY Times	web	Expedia	iOS
Booking	web	TripAdvisor	iOS
Wikihow	web	Kayak	iOS
Guardian	web	GasBuddy	iOS
WashingtonPost	web	Hotels.com	iOS
BlastingNews	web	HomeAway	iOS
Quora	web	AirBnB	iOS
Mediafire	web	Wish	iOS
Hclips	web	OfferUP	iOS
Gfycat	web	LetGo	iOS
9gag	web	Groupon	iOS
FoxNews	web	AliExpress	iOS
LiveJournal	web	RetailMeNot	iOS
WittyFeed	web	CartWheel	iOS
Zillow	web	Shein	iOS
Onedio	web	Geek	iOS
Giphy	web	5miles	iOS
Taringa	web	Clover	iOS
GoodReads	web	Zoosk	iOS
Fiverr	web	Bumble	iOS
Asos	web	Skout	iOS
Teepr Deals	web	Coffee Meets Bagel	iOS
4shared	web	Get Down	iOS
USArtToday	web	InstaMessage	iOS
TheFreeDictionary	web	HUD	iOS
WashingtonStreetJournal	web	MocoSpace	iOS
800 Contacts	web	Happn	iOS
IMDB	iOS	MeetMe	iOS
Pinterest	iOS	Mingle2	iOS
Imgur	iOS	Hookup	iOS
NY Times	iOS	Mingle	iOS
Booking	iOS	Down	iOS
The Guardian	iOS	Mingle	iOS
Washington Post	iOS	Tagged	iOS
Quora	iOS	Sudy	iOS
Mediafire	iOS	Ovia	iOS
9gag	iOS	Pregnancy+	iOS
LiveJournal	iOS	800 Contacts	iOS
Wittyfeed	iOS	Nurse Grid	iOS
Zillow	iOS	NCLEX RN	iOS
Onedio	iOS	Quora	Android
Giphy	iOS	Uber	Android
Goodreads	iOS	Tinder	Android
Fiverr	iOS	Ovia	Android
Asos	iOS	Pregnancy+	Android
Thefreedictionary	iOS	Booking	Android
Foursquare	iOS	Mediafire	Android
Realtor	iOS	Lyft	Android
Trulia	iOS	Yelp	Android
MapMyWalk	iOS	Groupon	Android
4shared	iOS	Skout	Android