SCIENCE CHINA
Information Sciences

• LETTER •

July 2022, Vol. 65 179104:1–179104:3
https://doi.org/10.1007/s11432-019-2697-1

An empirical study of security issues in SSO server-side implementations

Hui WANG, Dawu GU*, Yuanyuan ZHANG & Yikun HU

Lab of Cryptology and Computer Security, Shanghai Jiao Tong University, Shanghai 200240*, China*

Received 21 April 2019/Revised 8 October 2019/Accepted 30 October 2019/Published online 26 May 2021

Citation Wang H, Gu D W, Zhang Y Y, et al. An empirical study of security issues in SSO server-side implementations. Sci China Inf Sci, 2022, 65(7): 179104, https://doi.org/10.1007/s11432-019-2697-1

Dear editor,

Single sign-on (SSO) schemes have been widely used by major companies to manage service authorization and user authentication. They can enable third-party applications to obtain user information from a service provider to identify a user. The third-party application is often referred to as the relying party (RP), and the service provider is referred to as the identity provider (IdP). According to a recent study [1], OAuth and its extension OpenID connect (OIDC) are amongst the most widespread SSO protocols; thus we focus on the two protocols in this study and use OAuth to denote them. The security of SSO systems has been studied in the literature. Prior researches [1–5] investigated SSO protocols such as SAML, OAuth, and OIDC. They were mostly targeted at the influence of web attacks or finding bugs in the client-side implementations (e.g., SDKs and mobile applications), little work has been done to assess the security of the server-side implementations of SSO systems.

In this study, an in-depth analysis of the security issues in the server-side implementations of SSO systems is presented. Our study consists of two pillars. (1) We employ a set of lightweight techniques to extract the protocol specification and pinpoint vulnerabilities in an SSO system. (2) We conduct a mobile-web comparison study to check whether RPs employ consistent implementations for their apps distributed on different platforms, and analyze the consequence of such differences. Our analysis reveals that owing to the lack of implementation guidance for authorization and use-case for authentication, developers introduced various vulnerabilities in their SSO implementations, 62.5% of the IdPs' server-side implementations and 31.9% of the RPs' server-side implementations suffer from at least one security flaw, which may lead to privacy leakage or account hijacking.

Methodology overview. Considering the lack of access to the source code on the server side of the IdPs and RPs, we mainly focus on the network traffic during an OAuth-based SSO transaction when conducting the security analysis. Our analysis includes three stages. (1) We first perform a differential traffic analysis to extract the protocol specifications, to understand how developers customize OAuth for service authorization and user authentication. (2) Based on the obtained knowledge, we mutate the requests sent from the client applications and analyze the responses returned by the servers, to assess the security of the server-side implementations in real-world SSO systems. (3) We further conduct a mobile-web comparison study on 114 popular RPs, to investigate the differences between their SSO mechanisms implemented on diverse platforms.

Protocol specification extraction. As OAuth provides no standard implementation for developers, SSO systems implemented by different IdPs and RPs may introduce different message fields and parameter names in an OAuth flow, which makes it challenging to recognize each protocol field in a given message. Therefore, we need to perform protocol reverse engineering to understand the request and response messages of a target SSO system.

Because we only need to mutate a few parameters in our security analysis, we can filter the SSO irrelevant parameters to narrow down the test scope. We first use one account (e.g., Alice) to perform SSO login twice, and collect two sets of the same request messages and responses. Then we use another account to perform SSO login, and collect the corresponding requests and responses. We adopt a message alignment and value differing technique to automatically identify the fields of our interest. By aligning and differing the same user's same request, we can filter the message-specific fields, such as the time stamp. By aligning and differing different users' same requests, we can identify the user-specific parameters by selecting the fields with different values.

Upon identifying the fields of interest, we further attempt to infer the meaning of each field. There are mainly three types of values that are security related in SSO authentication: (1) user input information, (2) IdP assigned user data such as user ID, (3) cryptographically computed data. As illustrated in Algorithm 1, we mark the fields using user input as values as public-field, the fields with specific patterns are marked as public-field as well.

We identify the cryptographically computed fields by measuring the degree of similarities between two corresponding fields in Alice's request and Bob's request. If the Leven-

* Corresponding author (email: dwgu@sjtu.edu.cn)

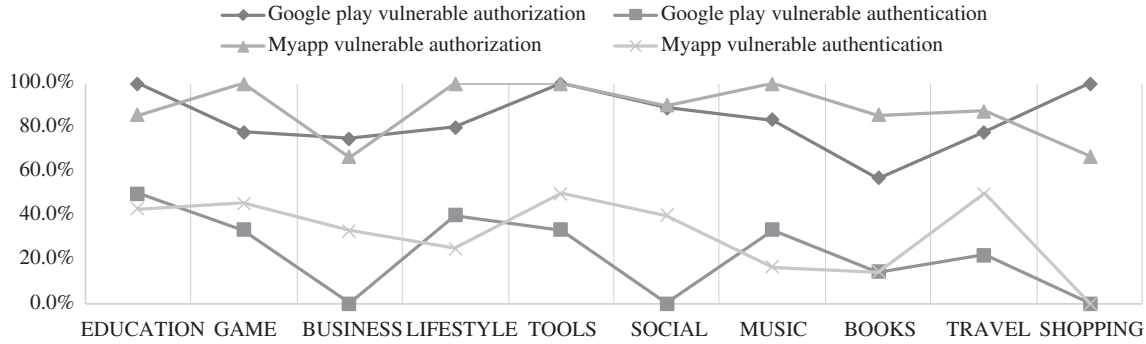

Figure 1 Percentages of SSO-enabled Apps with vulnerabilities based on App category.

Algorithm 1 Target field inference algorithm

Input: TargetFileds, RegisterParams;
Output: Forgeable, Public, Crypto;
Forgeable ← ∅;
Public ← ∅;
Crypto ← ∅;
for param ∈ TargetFileds **do**
  if param ∈ RegisterParam **then**
    Public ← Public ∪ param;
    TargetFileds ← TargetFileds\param;
  end if
  if RegexMatching(param) **then**
    Public ← Public ∪ param;
    TargetFileds ← TargetFileds\param;
  end if
end for
for param ∈ TargetFileds **do**
  if LSR(param) $<$ 0.5 **then**
    Crypto ← Crypto ∪ param;
  else
    Forgeable ← Forgeable ∪ param;
  end if
end for

shtein similarity ratio (LSR) is below 0.5, we mark the field as crypto-field. The remaining fields are marked as forgeable-field.

Message request manipulation. Because we aim to assess the security of the server-side implementation, we need to manipulate the request message to test the server behavior. We perform a fuzzing test in this step and mutate the value of fields of interest. (1) For the public-fields and forgeable-fields, we substitute the value with the value in another user's request. (2) For the protocol specific fields, such as the "scope" field and the API field, we substitute the value with the NULL value and all the available values provided by the IdPs in their SSO specifications. (3) For the crypto-fields, we substitute the value with the value in another user's request, as well as an expired value, a malformed value, and an invalid value.

By comparing the responses, we can decide whether the IdP's server is vulnerable. If we substitute a field in Alice's request with the value of Bob's, and the field is not access token, but the server's response message is identical to Bob's original response, it is vulnerable. If the access token in Alice's request is expired or invalid, and the server still returns information identical to Alice's original response, it is vulnerable. The fuzzing test can audit the access control policy of an IdP, i.e., whether an RP can access APIs outside the scope of the requested permissions or obtain user information of an unauthenticated user.

Mobile-Web comparison. Different versions of a certain RP's app distributed on different platforms share the same user data, and different implementations on different platforms can expand the attack surface, leading to vulnerabilities that threaten the overall security of authorization and authentication. To shed light on this problem, we analyze a set of popular RPs who distribute apps on different platforms. We first extract the authorization and authentication schemes of the apps distributed on diverse platforms of each SSO system, and examine whether these apps employ a consistent SSO mechanism. If a difference exists, we further attempt to explore the root causes of the difference, and investigate whether such a difference may compromise the target SSO system.

Evaluation. Our test includes 400 real-world Android applications and 114 websites. Our selection is unbiased: we select top 200 apps from 10 popular categories in Google Play, and top 200 apps from Myapp, a famous third-party app market in China. 312 out of the 400 Android applications provide login service, 135 apps of them support OAuth-based SSO login, and 114 of the OAuth-capable apps have a counterpart on the Web platform. We note that 99.3% of the RPs integrate SSO services provided by eight notable IdPs, including Facebook, Google, Twitter, Yahoo, QQ, Weibo, WeChat, and Alipay, so we mainly test these IdPs in our analysis.

Of all the 135 OAuth-enabled Android apps, 88.1% suffer from at least one vulnerability. As shown in Figure 1, we can observe the percentage of apps suffering from vulnerable authentication in Myapp is higher than that in Google Play. The root cause is that three major IdPs (i.e., Google, Facebook, Yahoo) supported by apps in Google Play utilize OIDC for SSO login, which is more secure for authentication than OAuth, while none of the four Chinese IdPs support OIDC. Meanwhile, the percentage of apps suffering from vulnerable authorization is higher than the percentage of apps suffering from vulnerable authentication. The reason is that most of the authorization vulnerabilities are introduced by IdP's incorrect implementation, and an RP often supports SSO services provided by various IdPs, pitfalls in any one of them can compromise the security of the RP app. Whereas the majority of the authentication vulnerabilities are caused by faulty implementations on RP's server side, which only affect the RP app itself.

SSO implementations of websites and Android apps have differences in both authorization and authentication. The authorization pages provided by IdPs vary significantly between the Web and mobile platforms. Six of the tested IdPs provide different authorization pages for websites and An-

droid apps. Such differences and IdP's vague descriptions of the requested scopes can cause user's misinterpretation of the permissions granted and lead to unexpected privacy leakage. Most of the websites request the same permission scope as their counterparts on Android, with only four exceptions. The eight IdPs in our dataset deploy consistent authentication mechanisms on different platforms. However, the RPs identify users in different ways. Developers of web apps are inclined to use the crypto-fields as authenticators, only 4% of the tested web apps use forgeable-fields or public-fields for authentication. Upon getting the authorization code or access token, they request user information with them through the server to server communication. While on the mobile platform, 28% of the Android apps use forgeable-fields or public-fields as authenticators, 54% of the apps exchange the authorization code for an access token through the client side, and leak the client secret inadvertently, such practices increase the risk of account hijacking.

Conclusion. We report an empirical study of the security issues in the server-side implementations of real-world SSO systems. The study shows that both the IdP developers and the RP developers misunderstood several key concepts of OAuth and OIDC, they did not know which information is suitable for authentication, had confusions about the difference between different tokens, as well as the difference between app permissions and user permissions. As a result, 62.5% of the IdPs' server-side implementations and 31.9% of the RPs' server-side implementations suffer from at least one security flaw, which can lead to privacy leakage or account hijacking.

References

1 Yang R H, Lau W C, Chen J Y, et al. Vetting single sign-on SDK implementations via symbolic reasoning. In: Proceedings of the USENIX Security Symposium, Baltimore, 2018. 1459–1474
2 Bai G, Lei J, Meng G, et al. AUTHSCAN: automatic extraction of web authentication protocols from implementations. In: Proceedings of the Network and Distributed System Security Symposium, San Diego, 2013. 1–20
3 Ghasemisharif M, Ramesh A, Checkoway S, et al. O single sign-off, where art thou? An empirical analysis of single sign-on account hijacking and session management on the web. In: Proceedings of the USENIX Security Symposium, Baltimore, 2018. 1475–1492
4 Wang H, Zhang Y, Li J, et al. Vulnerability assessment of oauth implementations in android applications. In: Proceedings of the Annual Computer Security Applications Conference, Los Angeles, 2015. 61–70
5 Navas J, Beltrán M. Understanding and mitigating OpenID Connect threats. Comput Secur, 2019, 84: 1–16