A study on security issues related to OAuth 2.0 Project Proposal

Deukyun Nam[1], Youngjin Han[2], and Taeui Moon[2]

[1] Sungkyunkwan University, Mathematics
[2] Sungkyunkwan University, Computer Science and Engineering

Abstract. In this project, we will demonstrate the two vulnerabilities of OAuth 2.0 through a web server implementation using spring framework. First vulnerability is clickjacking and Second one is covert redirect attack. We will test the vulnerabilities after implementing authorization server. Then, we will improve our web server with well-known defense method and do research on improving the method.

Keywords:
OAuth 2.0, Clickjacking, Covert Redirect, Spring framework

1 Introduction

OAuth is an open standard of delegating resource access rights so that a third party app can request a service on behalf of the service user who is the owner of the resource. This standard is used by companies such as Google, Facebook, and Naver to permit the users to share information about their accounts.[1]

Fig. 1: Social Login with OAuth

Unlike simple client-server authorization protocol, OAuth is a authorization protocol consists of three-party - User, Client, and third party app. It This open protocol has many security precautions. Accordingly, there are a number of precautions to be taken when using the OAuth.

Nevertheless, companies that provide OAuth services such as Google, Facebook, and Twitter have also experienced exposure to OAuth 2.0-based security incidents. In 2011, Facebook was exposed to covert redirect attack due to insufficient verification of "redirect URL". [2] In 2017, Google was exposed for about 40 minutes to a phishing attack in OAuth 2.0 using Gmail. [3] For these reasons, we will do study on security issues related to OAuth 2.0.

1.1 Clickjacking

Clickjacking is a malicious technique of tricking a user into clicking on something different from what the user perceives, thus potentially revealing confidential information or allowing others to take control of their computer while clicking on seemingly innocuous objects, including web pages.

For OAuth 2.0, the attacker trick the user into granting access without their knowledge using clickjacking.

1.2 Covert Redirect

Redirect URLs are a critical part of the OAuth flow. After a user successfully authorizes an application, the authorization server will redirect the user back to the application with either an authorization code or access token in the URL. Because the redirect URL will contain sensitive information, it is critical that the service does not redirect the user to arbitrary locations.

Covert Redirect makes the server vulnerable to a variety of attacks, including Open Redirect (CWE-601).[4] The Open Redirect is a security vulnerability that can redirect a user to an untrusted website by failing to properly check the URL entered by a web application or app.

Covert Redirect is a security flaw in the implementation of OAuth by service providers. A malicious attacker specify any part of a redirect URL by intercepting a request from OAuth 2.0 client (Service user). And the attacker adds URL of his/her site by appending query parameter of normal redirect URL. The client unknowingly delivers the covert redirect URL to the authorization server. Now, the attacker's site is recognized as a normal redirect URL by the authorized server and can receive the user's resource through the user's access token and client ID.

1.3 Demonstration Environment

The Spring framework is an open source application framework and inversion of control container for the Java platform. It provides various services for developing dynamic web sites. Spring Security OAuth 2.0 is a project of Spring Boot which is the Java's most popular web Framework[5]. However, the project is still in a transitional stage and there are many unimplemented specifications.

OAuth Authorization requires three servers: Resource server, Authorization server, third-party server. The resource server host the protected resources, and handles authenticated requests after the application has obtained an access token. The authorization server authenticates the user and issues access tokens after getting proper authorization. The third-party server is in the position of a user who obtains authority by using OAuth authentication from the resource server and receives a resource. In summary, in order for the third-party application to request access to the user's account information in the resource server, the third-party application can obtain an access token through the authorization server, after that requests the user's account information from the resource server

and receive that information. In this project, we will implement resource server and authorization server for OAuth, and client side web server using Spring framework.

2 Background

2.1 OAuth 2.0 authorization flow

In this section, we will describe the normal authentication flow in OAuth 2.0 and Code grant type[6]

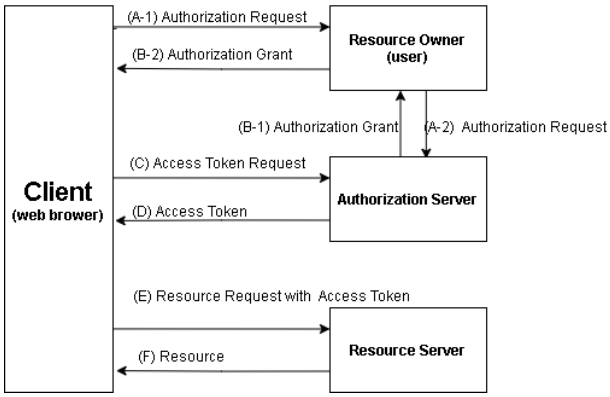


Fig. 2: OAuth 2.0 Authentication Flow

(A) The client requests authorization from the resource owner. The authorization request can be made indirectly via the authorization server as an intermediary.

(B) The client receives an authorization grant, which is a credential representing the resource owner's authorization, expressed using one of four grant types defined in OAuth 2.0. One of them is Authorization Code Grant.

(C) The client requests an access token by authenticating with the authorization server and presenting the authorization grant.

(D) The authorization server authenticates the client and validates the authorization grant, and if valid, issues an access token.

(E) The client requests the protected resource from the resource server and authenticates by presenting the access token.

(F) The resource server validates the access token, and if valid, serves the request.

2.2 Code grant type

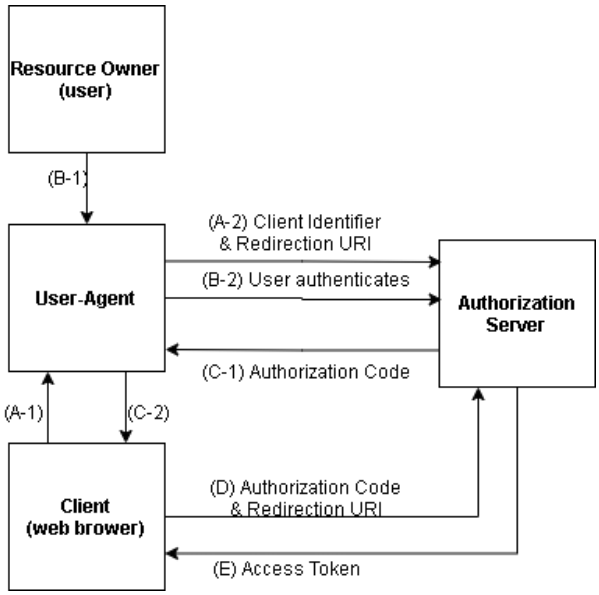


Fig. 3: Code grant type

An authorization grant is a credential representing the resource owner's authorization (to access its protected resources) used by the client to obtain an access token. The authorization code provides a few important security benefits, such as the ability to authenticate the client, as well as the transmission of the access token directly to the client without passing it through the resource owner's user-agent and potentially exposing it to others, including the resource owner.

(A) The client initiates the flow by directing the resource owner's user-agent to the authorization endpoint. The client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied).

(B) The authorization server authenticates the resource owner (via the user-agent) and establishes whether the resource owner grants or denies the client's access request.

(C) Assuming the resource owner grants access, the authorization server redirects the user-agent back to the client using the redirection URI provided earlier

(in the request or during client registration). The redirection URI includes an authorization code and any local state provided by the client earlier.

(D) The client requests an access token from the authorization server's token endpoint by including the authorization code received in the previous step. When making the request, the client authenticates with the authorization server. The client includes the redirection URI used to obtain the authorization code for verification.

(E) The authorization server authenticates the client, validates the authorization code, and ensures that the redirection URI received matches the URI used to redirect the client in step (C). If valid, the authorization server responds back with an access token.

3 Related Work

3.1 Security Vulnerabilities in the OAuth 2.0 Protocol

The common security vulnerabilities that can arise within the OAuth 2.0 protocol can be categorized into replay attack, phishing attack, and impersonation attack. The followings show the analysis of each categorized attack.

3.1.1 Reply Attack In OAuth 2.0 protocol, the authorization code means the client's access grant to the resource owner. Therefore one authorization code should be permitted per service. Otherwise a replay attack using the authorization code can be possible. The attacker can capture the authorization code redirection between the resource owner and the client, and use the captured request to replay the client to log-in as the account of the resource owner. The attacker can obtain the information of resource owner using such replay attack, and achieve the authorization grant to the resource server.

3.1.2 Phishing Attact Phishing is very easy yet effective method for the attacker. The client must go through the authorization server in order to use the information of the resource owner. During this process the attacker can generate a malicious client to steal the ID and password of the resource owner. The attacker can fish for ID and password of the resource owner, and validly be granted the access to the resource server using the taken information.

3.1.3 Impersonation Attack The attacker can intercept the authorization code in attempt to impersonate the resource owner. The attacker first intercepts the authorization code by tapping, and blocks the original authorization code in order to maintain connection using the stolen authorization code. So the attacker can impersonate a fresh session with the stolen authorization code and have access to the resource server.

3.2 Papers about OAuth

Chae, Cheol-joo proposed User permission authentication using email in the Oauth 2.0 protocol.[7] OAuth(Open Authorization) protocol that can provide restricted access to different web services has appeared. OAuth protocol provides applicable and flexible services to its users, but is exposed to reply attack, phishing attack, impersonation attack. Therefore it proposed method that after authentication Access Token can be issued by using the E-mail authentication. In proposed method, regular user authentication success rate is high when value is 5 minutes. However, in the case of the attacker, the probability which can be gotten certificated is not more than the user contrast 0.3% within 5 minutes.

Kim, Jinouk propose[8] user permission authentication using email in the OAuth 2.0 protocol. At the time a user finishes using OAuth client web service, even if he logs out of the client web service, the resource server remained in the login state may cause the problem of leaking personal information. In this paper, it proposes a solution to mitigate the threat by providing an additional security behavior check: when a user requests to log out of the Web Client service, he or she can make decision whether or not to log out of the resource server via confirmation notification regarding the state of the resource server. By utilizing the proposed method, users who log in through the OAuth Protocol in the public PC environment like department stores, libraries, printing companies, etc. can prevent the leakage of personal information issues that may arise from forgetting to check the other OAuth related services. The result shows that with this additional function, users will have a better security when dealing with resource authorization in OAuth 2.0 implementation.

3.3 Related CVE for CWE-601(Open Redirection)

3.3.1 CVE-2005-4206 URL parameter loads the URL into a frame and causes it to appear to be part of a valid page. Blackboard Learning and Community Portal System in Academic Suite 6.3.1.424, 6.2.3.23, and other versions before 6 allows remote attackers to redirect users to other URLs and conduct phishing attacks via a modified URL parameter to frameset.jsp, which loads the URL into a frame and causes it to appear to be part of a valid page.

3.3.2 CVE-2008-2951 An open redirect vulnerability in the search script in the software allows remote attackers to redirect users to arbitrary web sites and conduct phishing attacks via a URL as a parameter to the proper function.

3.3.3 CVE-2008-2052 Open redirect vulnerability in redirect.php in Bitrix Site Manager 6.5 allows remote attackers to redirect users to arbitrary web sites and conduct phishing attacks via a URL in the goto parameter.

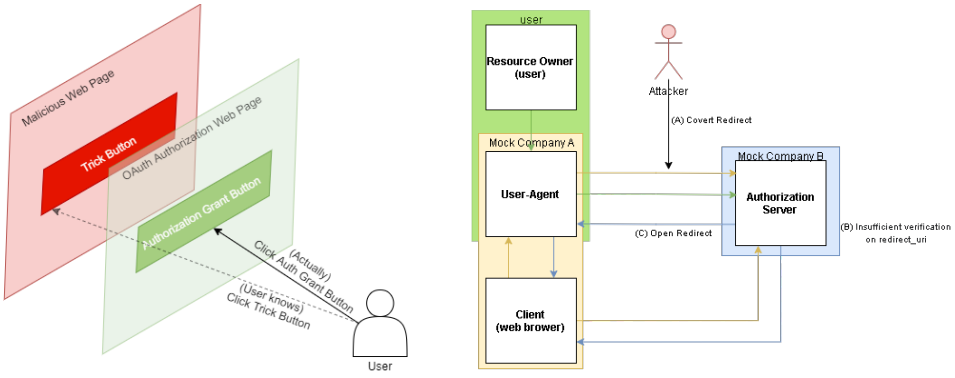
4 Problem Formulation

4.1 Clickjacking

Assume that the OAuth authorization server is vulnerable to a clickjacking attack. Therefore, the authorization server does not use the HTTP header 'X-Frame-Option'. We creates a malicious website in which it loads the OAuth authorization server URL in a transparent iframe above the web page. The web page is stacked below the iframe, and has some innocuous looking buttons or links, placed very carefully to be directly under the authorization server's confirmation button. When the user clicks the misleading visible button, they are actually clicking the invisible button on the authorization page, thereby granting access to the attacker's application. This allows us to trick the user into granting access without their knowledge.

4.2 Covert Redirect

It is known that, to prevent covert redirect, authorization Server must perform Full Path verification for the redirect URL value.[6] If only the domain is verified or only a part of the path is verified. Many vendor sites such as Google and Facebook prevent these threats through strict URL validation. [9, 10]
Nevertheless, there are cases where the framework including errors in the URL verification process was release to the full version and registered as CVE. That is CVE-2019-11269 identified in Spring Security OAuth 2.36 and priors.[11] In this project, we will do proof of concept for the CVE and research on how Spring handle that issue with version update.
Figure 4.(b) describe the attacking scenario in Covert Redirect. (A) Attacker covert his/her attacking URL inside of the user's redirect URL. (B) The mock company B do not check the full path of redirect URL. (C) Open Redirect occurs and the user's client redirect to an unintended URL. In this vulnerability. We will focus on POC of CVE-2019-11269 mentioned above.



(a) Clickjacking Scenario

(b) Covert Redirect Scenario

Fig. 4: Attack Scenario

4.3 Analysis Environment

We need two mock companies and three servers to exploit vulnerabilities. The mock company A is a web service company which has a web-server which uses OAuth 2.0 with vulnerable authorization server for login function. The mock company B acts like Google. It has two servers that is a authorization server which authenticates the client and validates the authorization grant and a resource server which contains resource and validate request on resource using access token.

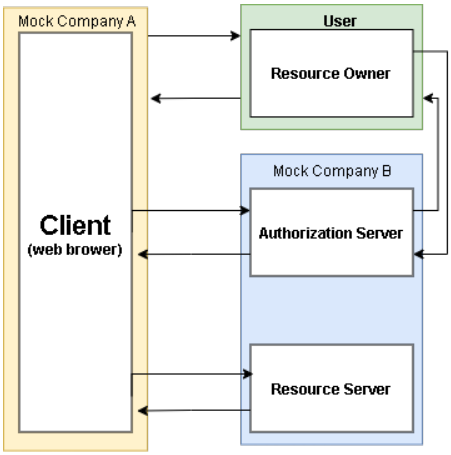


Fig. 5: Authorization flow between two companies

We will implement three servers using Spring Framework. Especially the web-server of mock company A will mainly dependent on Spring Security OAuth2 Client. The two servers of mock company B will mainly dependent on Spring Security OAuth2 Resource Server.

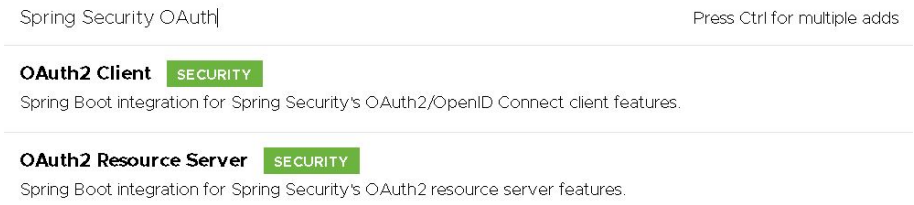


Fig. 6: Available projects in Spring Security. capture from start.spring.io

5 Project Milestone

We will start implementing it from week 6 onwards. Weeks 6 to 8 will be Client(web server) implementation. Weeks 7 through 10 will implement Authorization Server and Resource Server. From the 9th week onwards, attack simulation will be conducted in earnest, and at the same time, if there are any shortcomings in the server, corrections and supplements will be made.

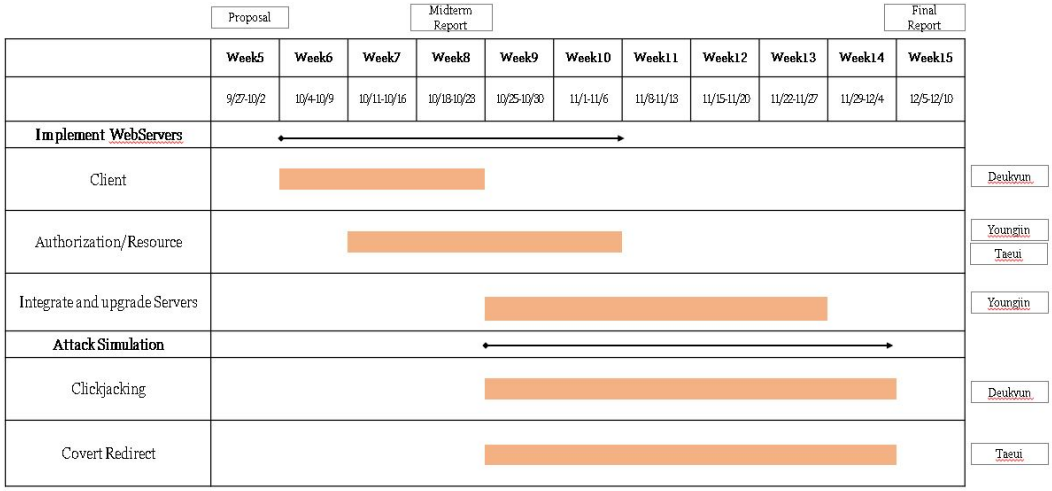


Fig. 7: Project Plan

References

1. Wikipedia OAuth Retrieved from https://en.wikipedia.org/wiki/OAuth
2. John B. (2014.05.03) Covert Redirect and its real impact on OAuth and OpenID Connect ¡http://www.thread-safe.com/2014/05/covert-redirect-and-its-real-impact-on.html¿
3. Duo Security (2017.05.05) Google OAuth Phishing Attack: Live QA Retrived from https://www.youtube.com/watch?v=2fU1YUHtB8A
4. CVE® CWE-601: URL Redirection to Untrusted Site ('Open Redirect'), https://cwe.mitre.org/data/definitions/601.html
5. Jet Brains (2021) Java Retrived from https://www.jetbrains.com/lp/devecosystem-2021/java/
6. IETF RFC 6749 "The OAuth 2.0 Authorization Framework ", https://datatracker.ietf.org/doc/html/rfc6749, 2012.
7. Chae, Cheol-Joo, Choi, Kwang-Nam, Choi, Ki-seok, Yae, Yong-Hee, Sin, Yong-Ju. (2015). User permission authentication using email in the oauth 2.0 protocol. Journal of Internet Computing and Services (JICS), 16(1), 21–28. (https://doi.org/10.7472/JKSII.2015.16.1.21)
8. Kim, Jinouk, Park, Jungsoo, Nguyen-Vu, Long, Jung, Souhwan. (2017). A Study on Vulnerability Prevention Mechanism Due to Logout Problem Using OAuth . Journal of The Korea Institute of Information Security and Cryptology (JKIISC), 27(1), 5–14. (https://doi.org/10.13089/JKIISC.2017.27.1.5)
9. Phillip Hodgson (2017.12.19) Strict URI Matching Retrieved from https://developers.facebook.com/blog/post/2017/12/18/strict-uri-matching
10. Using OAuth 2.0 for Web Server Applications Retrieved from (2021.09.21) https://developers.google.com/identity/protocols/oauth2/web-server#uri-validation
11. NIST CVE-2019-11269, https://nvd.nist.gov/vuln/detail/CVE-2019-11269
12. Nguyen, Q., & Baker, O. F. (2019). Applying Spring Security Framework and OAuth2 To Protect Microservice Architecture API. J. Softw., 14(6), 257-264.