Министерство образования Республики Беларусь

Учреждение образования

«Полоцкий государственный университет»

ИНФОРМАЦИОННО-КОММУНИКАЦИОННЫЕ ТЕХНОЛОГИИ: ДОСТИЖЕНИЯ, ПРОБЛЕМЫ, ИННОВАЦИИ (ИКТ-2018)

Электронный сборник статей

I Международной научно-практической конференции,

посвященной 50-летию Полоцкого государственного университета

(Новополоцк, 14–15 июня 2018 г.)

Новополоцк

Полоцкий государственный университет

2018

Информационно-коммуникационные технологии: достижения, проблемы, инновации (ИКТ-2018) [Электронный ресурс] : электронный сборник статей I международной научно-практической конференции, посвященной 50-летию Полоцкого государственного университета, Новополоцк, 14–15 июня 2018 г. / Полоцкий государственный университет. – Новополоцк, 2018. – 1 электрон. опт. диск (CD-ROM).

Представлены результаты новейших научных исследований, в области информационно-коммуникационных и интернет-технологий, а именно: методы и технологии математического и имитационного моделирования систем; автоматизация и управление производственными процессами; программная инженерия; тестирование и верификация программ; обработка сигналов, изображений и видео; защита информации и технологии информационной безопасности; электронный маркетинг; проблемы и инновационные технологии подготовки специалистов в данной области.

Сборник включен в Государственный регистр информационного ресурса. Регистрационное свидетельство № 3201815009 от 28.03.2018.

Компьютерный дизайн М. Э. Дистанова.

Технические редакторы: Т. А. Дарьянова, О. П. Михайлова.

Компьютерная верстка Д. М. Севастьяновой.

211440, ул. Блохина, 29, г. Новополоцк, Беларусь
тел. 8 (0214) 53-21-23, e-mail: irina.psu@gmail.com

ISBN 978-985-531-602-3

UDC 004.9+004.056

METHODS OF SINGLE SIGN-ON IN THE MODERN WEB

D. SAVCHENKO, O. GOLUBEVA
(Polotsk State University, Belarus)

Web applications typically require users to authenticate before starting to interact with them. A typical approach for this is so that at first users should register in applications, thus, creating personal credentials such as user names and passwords, and then as the first step of every interaction session they have to prove their identity by providing the credentials via a web form. This does not provide a decent user experience when multiple web applications are used and stimulates usage of the very same set of credentials for every web site which is unacceptable in terms of security and privacy. A different approach should be considered, when users are only required to enter credentials once and then are able to interact with all their web applications.

The described approach is called single sign-on (SSO) [1]. The main idea of it is that there exists a single identity provider — a web service that holds the knowledge of users and their credentials, enables users to register and later sign into it, and is able to provide user authentication information to third party web sites in a secure manner. This way users are only required to sign into the identity provider and they are then able to access other web sites without any additional actions. There are a number of ways to implement single sign-on and there have been a number of respective standards and protocols established.

Security Assertion Markup Language (SAML) [2] is an open standard for exchanging authentication and authorization data between parties and the most important use case this standard addresses is exactly web single sign-on. The SAML specification defines three roles: the principal (this is a web application user), the identity provider, and the service provider (this is the web application(s) that the user is going to interact with). The SAML also defines the format that is used to represent authentication requests and responses and some means by which authentication requests and responses are transferred between services and identity providers. This is done over the HTTP protocol with a web browser as a mediator, HTTP redirects, GET and POST requests and direct inter-server HTTP requests.

The general authentication flow for SAML is as follows (see picture 1).

1. A user requests a web application URL via a browser.
2. The application identifies that the user is not authenticated yet and responses with an HTTP 302 redirect to the identity provider. The redirect URL holds authentication request identifier as a query parameter, for example: *http://idprovider.example.com/redirect?auth_request_id=<somevalue>*.
3. Web browser performs an HTTP request to the redirected URL.
4. Identity provider handles the HTTP request, extracts authentication request identifier from the URL and makes a direct HTTP GET request to the web application in order to receive actual authentication request data that matches received identifier.
5. Web application responds to the HTTP request and presents authentication request data in the body of the HTTP response.
6. Identity provider reads authentication request data, performs authentication and issues another HTTP 302 redirect response to the browser redirecting it back to the web application and passing authentication response identifier in the query parameter.

7. Web browser makes an HTTP request to the redirected resource.

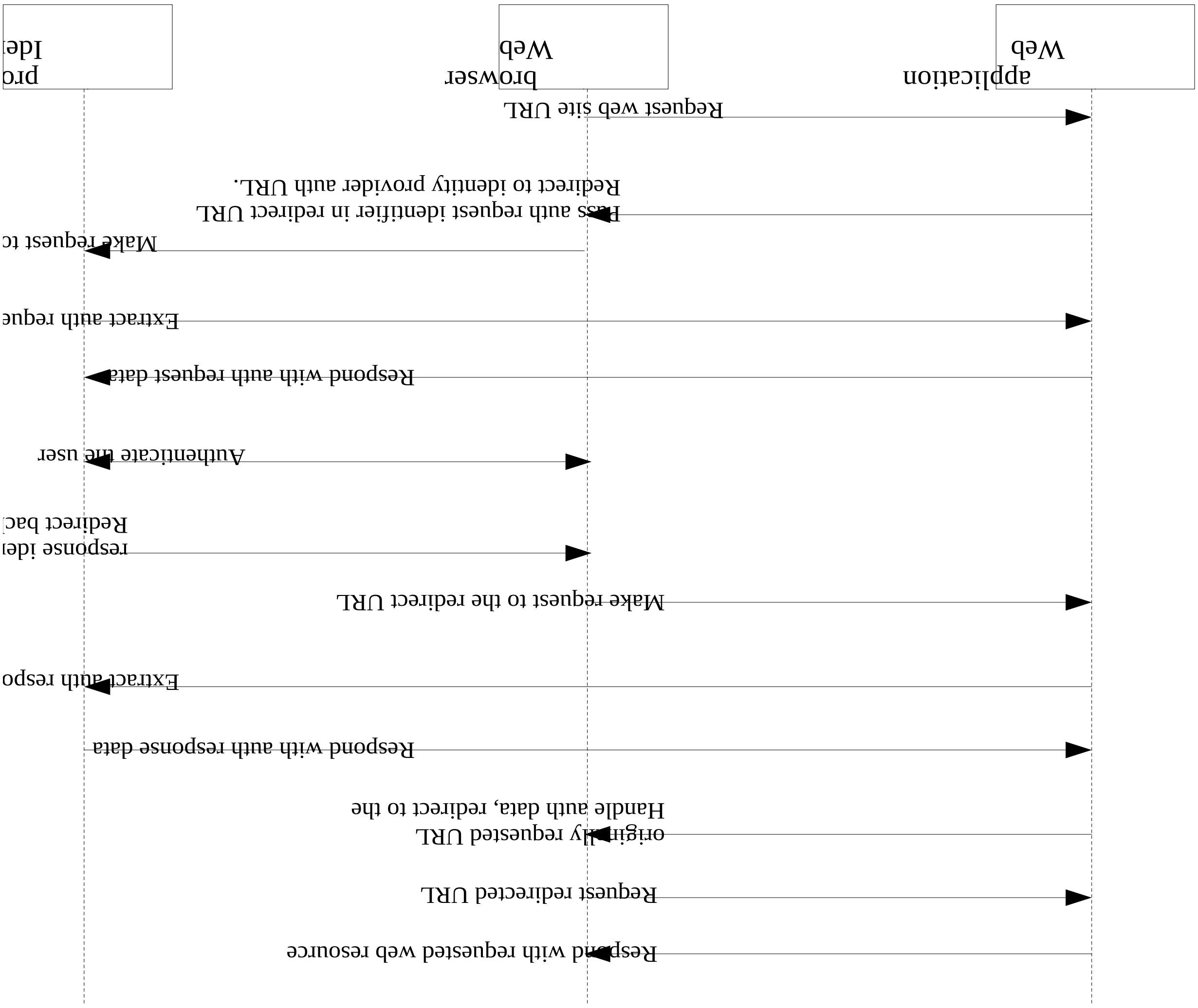
8. Web application handles the HTTP request, extracts authentication response identifier from the URL and makes a direct HTTP GET call to the identity provider in order to receive actual authentication response data that matches received identifier.

9. Identity provider responds to the HTTP request and presents authentication response data in the body of an HTTP response.

10. Web application obtains the authentication response data, authenticates the user and issues an HTTP 302 redirect response to the web browser, sending original request URL (from the first step) as a redirect URL.

11. Web browser performs an HTTP request to the redirected URL.

12. Web application successfully responses to the request as the user is authenticated now.


Ide
pro
Web
browser
Web
application
Request web site URL
Redirect to identity provider auth URL.
Pass auth request identifier in redirect URL
Make request tc
Extract auth reque
Respond with auth request data
Authenticate the user
Redirect bac
response idei
Make request to the redirect URL
Extract auth respo
Respond with auth response data
Handle auth data, redirect to the
originally requested URL
Request redirected URL
Respond with requested web resource

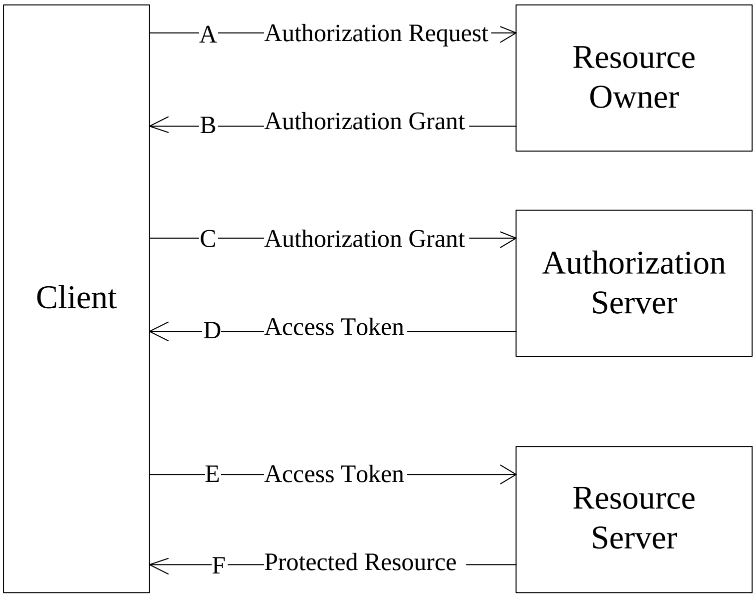
Picture 1. – SAML Flow

The **OAuth 2.0** authorization framework enables a third-party application to obtain limited access to an HTTP service on behalf of a user (resource owner) without knowing authorization credentials by orchestrating an approval interaction between the resource owner and the web service [3]. This is achieved through separating the role of the client from that of the resource owner; instead of using the resource owner’s credentials to access protected resources, the client obtains an access token — a string denoting a specific scope of access, lifetime, and other access attributes; the access token is then used to access the protected resources hosted by the server.

While OAuth 2.0 is an authorization framework it can still be used for authentication provided that the web service used for authentication provides some API to request some kind of user identification data, an email for instance. This data can be used to identify and thus authenticate a user.

General abstract OAuth 2.0 flow is presented in the figure 2 and includes the following steps:

1. the client requests authorization from the resource owner; the request can be made indirectly via the authorization server as an intermediary as it is done in authorization code grant type;
2. the client receives an authorization grant, which is a special credential representing the resource owner's authorization;
3. the client requests an access token by authenticating with the authorization server and presenting the authorization grant;
4. the authorization server authenticates the client and validates the authorization grant and issues an access token for valid authorization grant;
5. the client requests the protected resource from the resource server and authenticates by presenting the access token;
6. the resource server validates the access token and serves the request for valid access token.



Picture 2. – OAuth 2.0 Protocol Flow

It is not necessary for an authorization server and a resource server to be separate entities; they may be the one same server.

The authorization code grant type is used to obtain access tokens and is optimized for confidential clients [3]. This is a redirecting-based flow and thus the client must be capable of interacting with the resource owner's user-agent (typically a web browser) and capable of receiving incoming requests (via redirection) from the authorization server.

The flow includes the following steps:

1. the client directs the resource owner's user-agent to the authorization endpoint; the client includes its client identifier, requested scope, local state, and a redirection URL for authorization server to send user-agent back to after access is granted or denied;
2. the authorization server authenticates the resource owner via the user-agent and establishes whether he or she grants or denies the client's access request;
3. assuming access has been granted by the resource owner, the authorization server redirects the user-agent to the client using the redirected URI provided earlier and including an authorization code and local state provided by the client earlier;
4. the client requests an access token from the authorization server by including the previously received authorization code and redirection URI previously used for verification;
5. the authorization server authenticates the client, validates the authorization code and redirection URI and, if valid, responds with an access token.

Obtained access token then can be used to form the Authorization HTTP header when accessing protected resources.

OpenID Connect [4] is an authentication layer built on top of the OAuth 2.0 authorization framework. OpenID Connect standardizes, besides other details, a special *userinfo* endpoint that should be used by web applications to obtain user authentication information. In order to be able to obtain it applications should first run the OAuth 2.0 authorization flow and obtain an access token. This token can later be used to form the Authorization HTTP header when performing requests to the *userinfo* endpoint.

Obviously, for each case when HTTP protocol is assumed it is highly recommended to use SSL encryption and digital signing on top of it for security and privacy considerations, thus effectively using HTTPS.

Modern web applications tend to prefer OpenID Connect when implementing single sign-on functionality. This is explained by the fact that OpenID Connect has in fact some advantages over SAML or plain OAuth 2.0: it is much simpler (but by no means less secure) compared to SAML which has rather complicated authentication flow and verbose XML exchange data format, and it is designed specifically for authentication unlike OAuth 2.0. So, OpenID Connect should be recommended for all new web application that need to implement single sign-on.

References

1. Single sign-on [Electronic Resource] / Wikimedia Foundation, Inc. – Mode of access: https://en.wikipedia.org/wiki/Single_sign-on. – Date of access: 14.05.2018.
2. Security Assertion Markup Language (SAML) V2.0 Technical Overview [Electronic Resource] / N. Ragouzis [et al.]. – Mode of access: https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf. – Date of access: 14.05.2018.
3. RFC 6749 — The OAuth 2.0 Authorization Framework [Electronic Resource] / Dick Hardt. – Mode of access: https://tools.ietf.org/pdf/rfc6749.pdf. – Date of access: 14.05.2018.
4. OpenID Connect Core 1.0 [Electronic Resource] / N. Sakimura [et al.]. – Mode of access: http://openid.net/specs/openid-connect-core-1_0-final.html. – Date of access: 14.05.2018.