SECURE OPEN ID CONNECT IMPLEMENTATION USING AZURE ACTIVE DIRECTORY AND ASP .NET FRAMEWORK

PETRO KOLOSOV

Abstract. In this manuscript we discuss the problem of secure storage and transfer of access tokens between microservices. Web browser may store access tokens both, in local storage or in cookie files. We propose a secure implementation to store and transfer auth cookies between microservices using Azure Active Directory, OpenID Connect and ASP .NET Web Framework.

Contents

1. Statement of the problem

In this manuscript we discuss the problem of secure storage and transfer of access tokens between microservices. Web browser may store access tokens both, in local storage or in cookie files. Local storage is a web browser mechanism that allows web applications to store data locally on the user's device. It is important to note that Local storage is vulnerable to

Date: May 21, 2023.

Key words and phrases. Open ID Connect, OIDC, Azure Active Directory, PKCE, OAuth 2.0, XSS, CSRF, ASP .NET Core .

Cross Site Scripting (XSS) attacks [1]. Cross Site Scripting (XSS) is a type of attack such that malicious JavaScript code is injected into an html-page to access user's sensitive data, for example access tokens. Cross Site Scripting (XSS) attacks may be divided by following groups:

- **Reflected XSS** is a type of attack in which a malicious the script is passed to the web server via URL or form parameters and then returned back to the page's html code without proper filtering or escaping. If the user opens the page, then the script is executed in the browser, which can lead to the loss of sensitive data, such as access tokens.
- **Stored XSS** is a type of attack in which a malicious script is stored on the server, for example in the database and displayed on web pages. The script is executed in users' browsers, requesting pages with malicious code.
- **XSS in the DOM** is a type of attack such that a malicious script modifies the DOM tree of a web page, running in the user's browser. In most cases, based on modifying the URL string.

Another way to store credentials is to store them in cookie files. Cookies are small pieces of data sent by web server and stored on user's device. Storing access tokens in cookies eliminates potential XSS attacks since that HttpOnly setting makes impossible to read cookies using JavaScript code. Having credentials stored in cookies, HTTP request is performed via JavaScript. If the object { `withCredentials: true` } provided and auth cookies exist, then auth cookies are attached to request, but cannot be accessed from JS code anyway. Example of such request in TypeScript

```
return this.httpClient.post<TokensResponse>(
this.baseUrl + this.sessionsRoute,
command,
{ withCredentials: true });
```

Note that cookie files are vulnerable to Cross-Site Request Forgery (CSRF) attacks [2]. Cross-Site Request Forgery (CSRF) – is an attack such that redirects user to the resource where user has active session. It means that attacker could perform requests to resources on behalf of the user. The main idea of Cross-Site Request Forgery (CSRF) attack is illustrated below

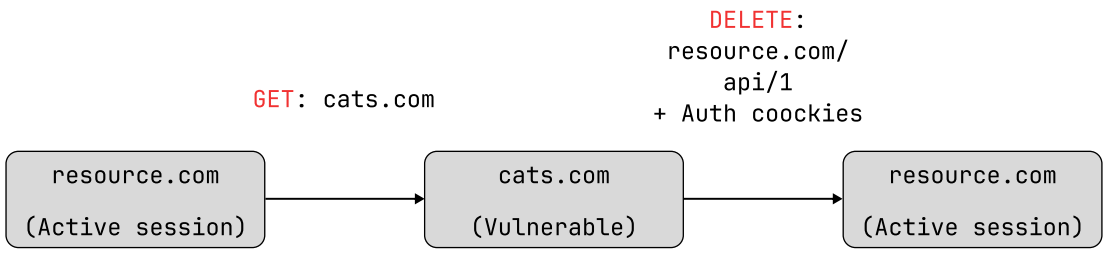

Figure 1. Cross-Site Request Forgery (CSRF) principle diagram.

Cookie files provided with `SameSite` setting that determines whether cookies will be sent along with cross domain requests.

`SameSite` setting has one of the states below:

- **None** – means no restrictions are imposed on the transfer of cookies.
- **Lax** – allows cookie transmission only by secure HTTP methods according to RFC 7231 [3]. These methods are `GET, HEAD, OPTIONS` and `TRACE`.
- **Strict** – blocks cookies from being sent with any requests from third-party resources. Cookies will only be transferred within same domain.

Therefore, `SameSite` setting values such as `Lax` and `Strict` protect user from a CSRF attack blocking submission of cookies using unsecure HTTP methods and cross domain requests. There are more CSRF protection techniques in [4].

2. Introduction

Your introduction here. Include some references [2, 1, 5, 6, 3]. Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's

standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

OAuth 2.0 flow diagram

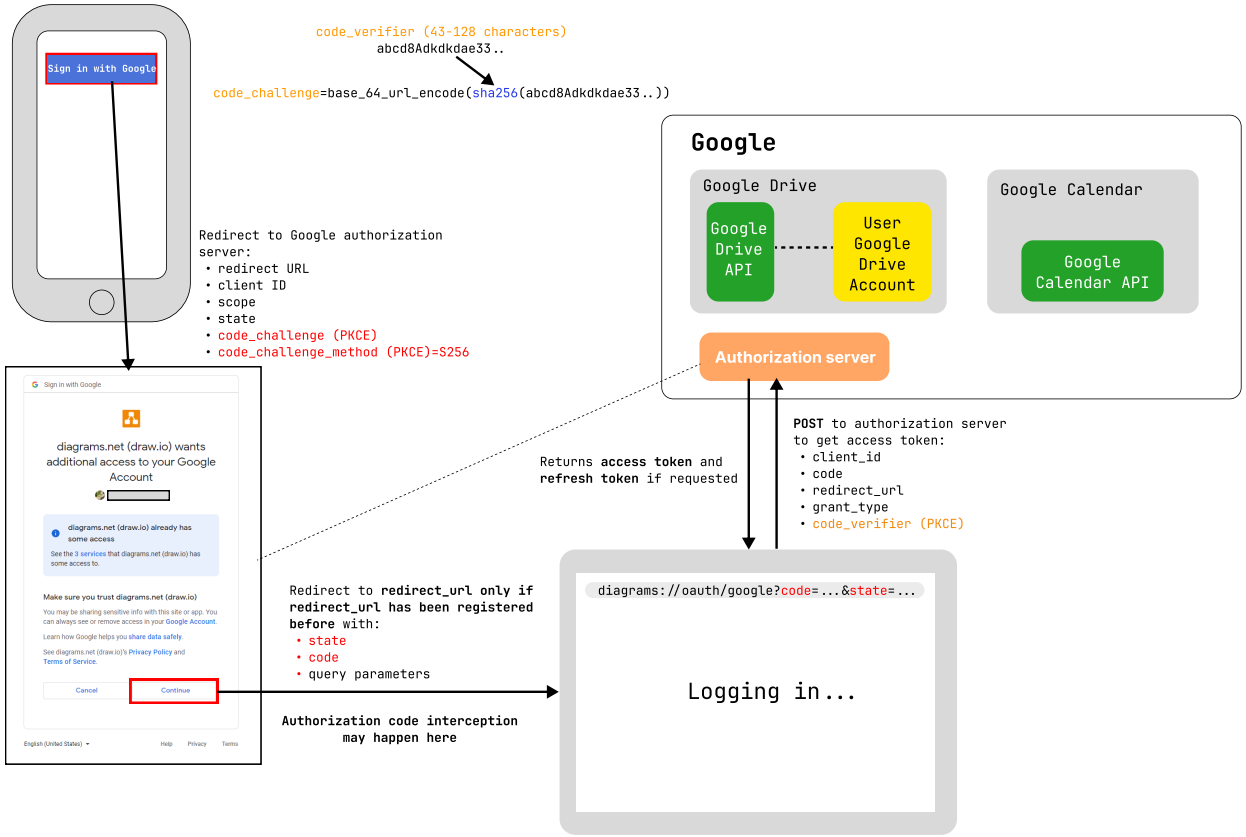

Figure 2. OAuth 2.0 with PKCE flow diagram.

3. Authentication flow

Authentication flow diagram

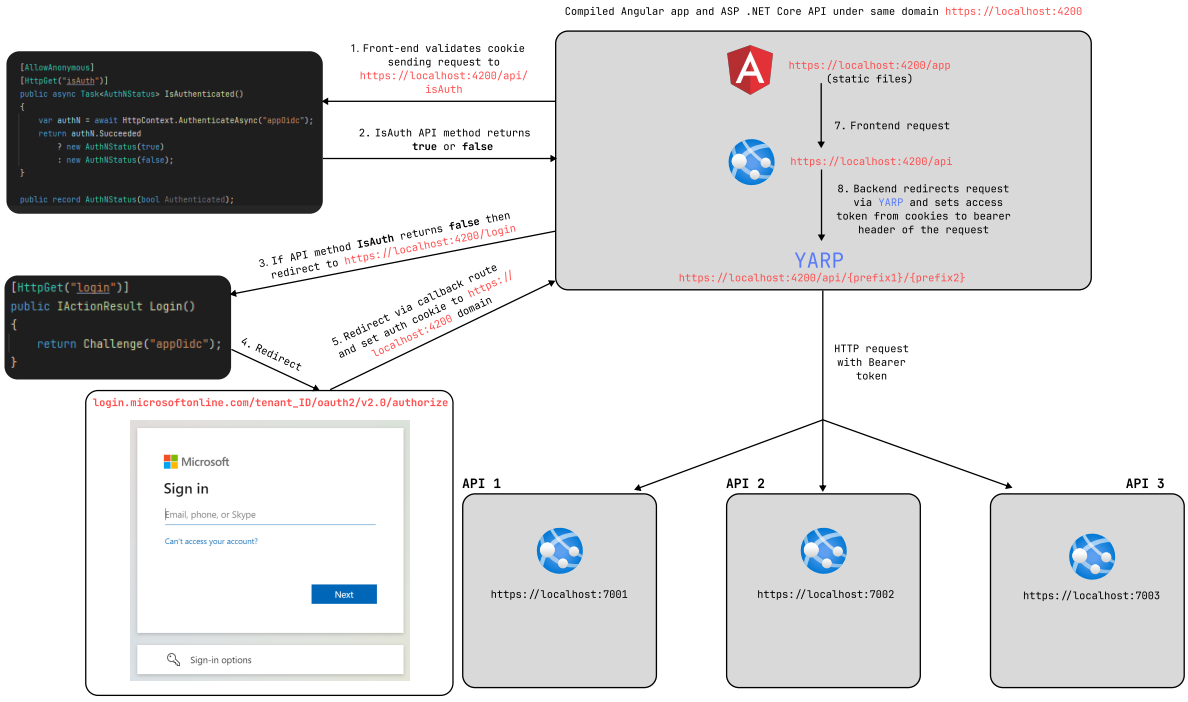

Figure 3. Authentication flow diagram.

(1) Frontend application sends request to `https://localhost:4200/api/isAuth` API method to validate authentication

(2) API method `https://localhost:4200/api/isAuth` returns `true` if authentication valid, otherwise `false`

(3) If authentication invalid then browser redirects to `https://localhost:4200/login`, otherwise no action taken

(4) Login redirects browser to authorize url `login.microsoftonline.com/tenant/oauth2/v2.0/authorize`

(5) If user is logged in then browser is redirected to fallback url with cookies already set for `https://localhost:4200` domain

(6) Request is sent to validate authentication `https://localhost:4200/api/isAuth`, now it returns `true`

(7) Frontend at `https://localhost:4200/app` sends request to the `https://localhost:4200/api/OtherApi1/products`

(8) Cookie exists for `https://localhost:4200/app` so that `https://localhost:4200/api` attaches it as header to request via YARP and sends request with token to the external resource `OtherApi1/products`

(9) If response code is 401 at step (8) then repeat step (1)

4. Refresh token flow

5. Conclusions

Conclusions of your manuscript.

6. Acknowledgements

Thanks someone and somebody for help and useful comments.

References

[1] Kevin Spett. Cross-site scripting. *SPI Labs*, 1(1):20, 2005.

[2] Mohd Shadab Siddiqui and Deepanker Verma. Cross site request forgery: A common web application weakness. In *2011 IEEE 3rd International Conference on Communication Software and Networks*, pages 538–543. IEEE, 2011.

[3] Roy Fielding and Julian Reschke. RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): semantics and content. https://www.rfc-editor.org/rfc/rfc7231, 2014.

[4] OWASP Cheat Sheet Series. Cross-Site Request Forgery Prevention Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html, 2023.

[5] J Bradley and N Agarwal. RFC 7636: Proof Key for Code Exchange by OAuth Public Clients. https://www.rfc-editor.org/rfc/rfc7636, 2015.

[6] Dick Hardt. The OAuth 2.0 authorization framework. https://www.rfc-editor.org/rfc/rfc6749, 2012.

Email address: kolosovp94@gmail.com

URL: https://kolosovpetro.github.io