УДК 681.3 https://doi.org/10.15407/pp2022.03-04.409

ЗАХИСТ ВІДКРИТИХ КЛІЄНТІВ ЗА ДОПОМОГОЮ ОДНОГО АЛГОРИТМУ АВТОРИЗАЦІЇ

Богдан Бодак, Анатолій Дорошенко

Розглянуто конфіденційні та відкриті клієнти у контексті авторизації. Проведено аналіз проблем, пов'язаних з авторизацією відкритих клієнтів, котрі не можуть забезпечити конфіденційність секретних налаштувань. Досліджено існуючі вектори атак, недоліки стандарту OAUTH, потенційно небезпечні практики. Виявлено алгоритми, моделі та методи для захищеної авторизації відритих додатків. Створено модель алгоритму авторизації на основі Proof Key for Code Exchange та методу Backend For Frontend, яка не вразлива до атак Cross-Site-Scripting та Auth Code Interception. У результаті розроблено додаток, що являє собою відкритий клієнт на технології Blazor Web Assembly, використовуючи створену модель авторизації.

Ключові слова: авторизація, аутентифікація, PKCE, Proof Key for Code Exchange, Identity Server, відкритий клієнт, OAUTH, XSS, захист інформації.

The paper focuses on authorization in public clients and provides a secure authorization model as an alternative to costly Microsoft Duende BFF solution. After providing a brief overview of confidential and public clients in terms of authorization, we have analyzed problems and potential attack vectors associated with the authorization process in public clients due to their inability to hold credentials securely. Confidential clients are implemented on secure servers or able to facilitate secure authentication by other means, while public clients lack this security. Our research discovered algorithms, models, and methods for secure authorization in public clients. As a part of our model, we have implemented high entropy Proof Key for Code Exchange generator in C# .NET 6.0. In addition, we have provided a solution to a problem of storing sensitive information in public clients using the Backend for Frontend concept. This concept leverages a reverse proxy pattern where a backend application acts as a proxy and handles all client requests. Having a proxy backend application significantly tightens security model for public clients, while restricting possible attack vectors. The authorization model being researched was based on Proof Key for Code Exchange and Backend for Frontend approach. During the testing phase of our research, we have confirmed that the model was not vulnerable to Cross-Site-Scripting and Auth Code Interception attacks. A sequence diagram outlining main actors and interactions among them in context of authorization has been designed. The diagram stands as the visual representation of the model that uses proposed methods and algorithms. As a result, we have managed to build an alternative to secure authorization solutions for public clients that do not rely on the client secret. We have summarized our key findings in a Blazor Web Assembly application, which is classified as public and uses the described authentication model.

Keywords: authorization, authentication, PKCE, Proof Key for Code Exchange, Identity Server, public application, OAUTH, XSS, information security.

Конфіденційні та відкриті клієнти

Згідно зі специфікацією OAUTH 2.0 [1], додаток може бути конфіденційним або відкритим. Конфіденційні додатки зазвичай обумовлюють безпеку облікових даних тим, що такі додатки реалізовані на захищеному сервері або можуть виконувати безпечну аутентифікацію іншими способами. З іншого боку, відкриті клієнти не можуть забезпечувати конфіденційність облікових даних, оскільки такі додатки завантажуються та виконуються безпосередньо у небезпечному середовищі: наприклад, односторінкові (Single Page) або рідні (Native) додатки. У контексті авторизації OAUTH 2.0 прийнято використовувати секрет клієнта для моніторингу та верифікації запитів на авторизацію. Секрет клієнта – це унікальний ідентифікатор, що генерується для додатка при його реєстрації.

У конфіденційних клієнтах секрет зберігається у коді або у файлах конфігурації, проте у відритих додатках неможливо безпечно зберігати секрет клієнта. Для прикладу, у мобільних додатках ми не можемо забезпечити конфіденційність секрету тому що при декомпіляції зловмисником його значення стане доступним. Мобільні додатки, які використовують секрет, вразливі і до атаки Auth Code Interception Attack [2], приклад якої наведено на Рис. 1.

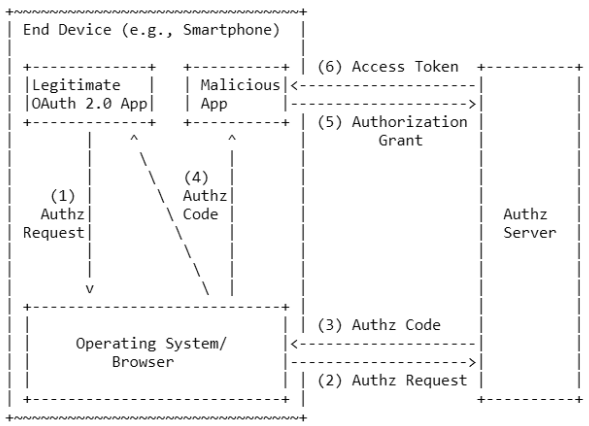

Рис. 1. Атака Auth Code Interception

© Б.В. Бодак, А.Ю. Дорошенко, 2022

Зловмисник реєструє додаткову адресу через додаток Malicious App для захоплення запитів під час переадресації та отримує можливість перехопити код авторизації від серверу. Для односторінкових додатків також неможливо використовувати секрет клієнта, оскільки код та файли конфігурації цілком доступні у браузері.

Проблема авторизації відкритих клієнтів

Із моменту публікації OAUTH 2.0 2013 року, специфікація набула значної популярності серед розробників, у тому числі і таких гігантів індустрії як Facebook, Twitter та Google, ставши по суті стандартом авторизації [3]. Беручи до уваги те, що стандарт був запропонований майже десятиліття тому, а технології постійно еволюціонують, виникає потреба у нових моделях та методах захисту від потенційних атак. Формальний аналіз популярних веб-сайтів та соціальних мереж, котрі використовують OAUTH 2.0 для авторизації, виявив декілька десятків невідомих до цього атак [3]. Серед потенційних проблем, авторами були виявлені Cross-Site Request Forgery (CSRF) та Open Redirectors, але вони не заглиблюються у методи їх вирішення. Емпіричний аналіз [4], використовуючи вектори атак [3] на прикладі 95 веб-сервісів та мобільних додатків, демонструє успішне перехоплення SSO сесії та пов'язану з цим величезну небезпеку. В статті [4] наводиться приклад перехоплення cookie аккаунту Facebook, який надалі використовується для авторизації у SSO-провайдерів за протоколом OAUTH. Запропонований авторами стандарт Single Sign-Off для єдиної «деавторизації» захопленої сесії вирішує наслідки проблеми, але не саму проблему перехоплення токену.

У статті про кращі практики безпеки OAUTH 2.0 приведено 9 основних атак та актуальні методи боротьби з ними [5]. Стандарт рекомендує використовувати PKCE [6] для відкритих та конфіденційних додатків [5], як додатковий рівень захищеності від потенційних атак, проте не надає практичних рекомендацій стосовно імплементації алгоритму, делегуючи відповідальність розробникам. Виходячи з наведених досліджень [3, 4] та практик [5, 7], перед розробниками постає суттєва проблема впровадження безпечної авторизації на базі PKCE, яка була би не вразливою до описаних атак: CSRF, Open Redirection, Auth Code Interception та інших.

Алгоритм Proof Key For Code Exchange (PKCE)

Враховуючи проблеми описані вище, для відкритих додатків специфікація OAUTH 2.0 пропонує використовувати алгоритм PKCE [6], який по суті являє собою варіант доказу володіння (proof of possession), замість секрету клієнта. Незважаючи на те, що запропонована модель побудована навколо відкритого клієнта, алгоритм доцільно застосовувати і для конфіденційних додатків. Базова реалізація алгоритму передбачає, що клієнт надає серверу авторизації доказ того, що код авторизації належить клієнтові, для того, аби сервер видав клієнтові код доступу. Алгоритм PKCE вносить додаткові параметри до звичайного процесу авторизації, зображеного на Рисунку 1: код-підтвердження (code_verifier), код-виклик (code_challenge) та тип коду-виклика (code_challenge_method). Код підтвердження - це довільний рядок, який згенеровано клієнтом відповідно до специфікації OAUTH 2.0 та використовується ним як своєрідний тип секрету. Код-виклик – це перетворений клієнтом код підтвердження за правилами специфікації, а тип коду-виклика – не обов'язковий параметр, що приймає значення "S256" (алгоритм SHA256) або "plain" (без перетворення) та вказує на тип алгоритму перетворення коду-виклику. Специфікація не рекомендує використовувати тип коду-виклика "plain", оскільки потенційно це може призвести до перехоплення коду-підтвердження та використання його як коду-виклика у чистому вигляді, що призводить до повної втрати переваг алгоритму PKCE та потенційної проблеми з безпекою. У розділі «Генерація кодів для алгоритму PKCE» надано деталі та приклад генерації кодів виклику і підтвердження за специфікацією OAUTH 2.0 з використанням C# .NET 6.0. На Рис. 2 зображено загальну модель авторизації через алгоритм PKCE.

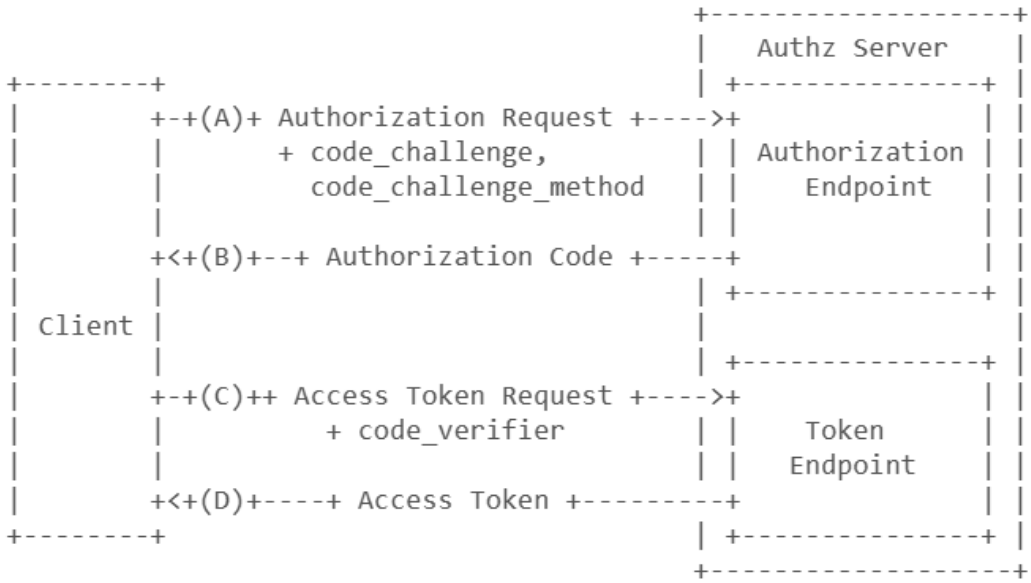

Рис. 2. Модель авторизації з використанням PKCE

Базову модель авторизації можна описати наступними кроками:

Клієнт генерує код-виклик та код підтвердження, відправляє запит на авторизацію, котрий включає код-виклик та тип коду-виклика.

Сервер авторизації зберігає код-виклика з типом та перенаправляє клієнту код авторизації.

Клієнт відправляє запит на токен доступу, включаючи код підтвердження.

Сервер перевіряє код підтвердження на основі коду-виклику та повертає токен доступу у випадку успішної верифікації.

Алгоритм успішно працює тому, що навіть у разі перехоплення коду авторизації, неможливо буде обміняти цей код на токен доступу без коду-виклику, нівелюючи атаку Auth Code Interception Attack, зображену на Рисунку 1. Пара кодів має бути унікальною для кожного нового запиту на авторизацію, а також клієнт повинен використовувати захищений протокол передачі даних (TLS).

Генерація кодів для алгоритму PKCE

Для генерації коду підтвердження (code_verifier) та коду-виклика (code_challenge) було створено окремий клас PkceGenerator із відповідними методами GenerateCodeVerifier та GenerateCodeChallenge. При створенні об'єкта класу у конструктор передається параметр, що визначає розмір згенерованого коду підтвердження, водночас мінімальний розмір за специфікацією 43 символи, а максимальний – 128. За замовчуванням аргументу присвоюється максимальне значення – 128 символів. При написанні класу використовувався C# .NET 6.0 та бібліотеки System і System.Security.Cryptography.

```
namespace DemoApp.Authentication {
  /// <summary>
  /// Provides a randomly generating PKCE code verifier and it's corresponding code challenge.
  /// </summary>
  internal class PkceGenerator {
    /// <summary>
    /// The randomly generated PKCE code verifier.
    /// </summary>
    public string CodeVerifier;

    /// <summary>
    /// Corresponding PKCE code challenge.
    /// </summary>
    public string CodeChallenge;

    /// <summary>
    /// Initializes a new instance of the Pkce class.
    /// </summary>
    /// <param name=»size»>The size of the code verifier (43 - 128 charters).</param>
    public PkceGenerator(uint size = 128) {
      CodeVerifier = GenerateCodeVerifier(size);
      CodeChallenge = GenerateCodeChallenge(CodeVerifier);
    }
   …
  }
}
```

Відповідно до специфікації OAUTH 2.0 RFC-7636, код підтвердження має бути криптографічним рядком, згенерованим із високою ентропією, розміром від 43 до 128 символів. Допустимими символами є A-Z, a-z, 0-9, -, ., _, ~. Запропонована реалізація методу для генерації коду підтвердження представлена нижче.

```
/// <summary>
/// Generates a code_verifier according to RFC-7636.
/// </summary>
/// <param name=»size»>The size of the code verifier (43 - 128 charters).</param>
/// <returns>A code verifier.</returns>
public static string GenerateCodeVerifier(uint size = 128) {
  if (size < 43 || size > 128)
    size = 128;
    const string unreservedCharacters = «ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvw
xyz0123456789-._~»;
      Random random = new Random();
      char[] highEntropyCryptograph = new char[size];
```

```
    for (int i = 0; i < highEntropyCryptograph.Length; i++) {
      highEntropyCryptograph[i] = unreservedCharacters[random.Next(unreservedCharacters.Length)];
    }

    return new string(highEntropyCryptograph);
}
```

Код-виклик створюється за допомогою хешування значення коду підтвердження алгоритмом SHA256. Метод, наведений нижче, створює код-виклик використовуючи SHA256 та заміняє небажані символи після хешування, роблячи значення доступним для передачі через URL (так званий URL Encoding).

```
/// <summary>
/// Generates a code_challenge according to RFC-7636.
/// </summary>
/// <param name=»codeVerifier»>The code verifier.</param>
/// <returns>A code challenge.</returns>
public static string GenerateCodeChallenge(string codeVerifier) {
    using var sha256 = SHA256.Create();
    var challengeBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(codeVerifier));

    return Convert.ToBase64String(challengeBytes)
    .Replace(‘+’, ‘-’)
    .Replace(‘/’, ‘_’)
    .Replace(«=», «»);
}
```

Використання класу у програмі нетривіальне: користувачу необхідно створити об’єкт класу та звернутися до відповідного поля щоб отримати код підтвердження та код-виклик.

```
PkceGenerator pkceGenerator = new ();
string codeVerifier = pkceGenerator.CodeVerifier;
string codeChallenge = pkceGenerator.CodeChallenge;
```

Водночас слід мати на увазі, що пара code_challenge та code_verifier повинна використовуватися лише один раз для авторизації. Не варто повторно використовувати згенеровану пару кодів, оскільки це може призвести до потенційного перехоплення та використання зловмисником кодів для створення токену авторизації.

Специфікація OAUTH 2.0 та RFC-7636 не надає інформації щодо практик зберігання code_verifier на клієнтах. У наступних розділах ми розглянемо розповсюджені методи збереження коду підтвердження, оскільки це є фундаментальним для безпеки у використанні алгоритму PKCE.

Методи збереження code_verifier у відкритих клієнтах

Як було зазначено, специфікація алгоритму PKCE не передбачає аналіз методів за якими буде збережено код підтвердження між запитами. Оскільки клієнт отримує код авторизації через зворотній виклик (callback), не є можливим зберігання коду підтвердження у форматі змінної чи будь-яким іншим способом у коді. Ми будемо розглядати методи для збереження значення code_verifier лише у контексті WEB-додатків (React, Angular, Web Assembly). Додаткові методи безпечного зберігання інформації для мобільних додатків Android [8] та iOS [9] відрізняються і виходять за межі цієї статті.

Збереження у сховищі браузера. Очевидним рішенням для збереження будь-якої інформації у роботі з WEB-клієнтами є сховище браузера. Існує два види такого сховища: Local Storage та Session Storage [10]. Так, за допомогою дуже простого JavaScript коду, розробники можуть зберігати певні дані та налаштування користувача. Різниця між Local Storage та Session Storage полягає у тривалості зберігання даних. Session Storage не зберігає значення між вкладками та вікнами браузера, надаючи хибне відчуття безпеки розробникам, які збираються записати значення у такому сховищі. На жаль, якщо сайт виявиться вразливим до XSS [11] атак, зловмисник зможе зчитати дані зі сховища браузера не дивлячись на те, що вони в Local Storage або Session Storage.

Переваги зберігання значення у сховищі браузера:

- Швидкість роботи.
- Простота реалізації.

Недоліки такого підходу:

- Вразливість до XSS атак.
- Потенційний витік інформації.

Варто зазначити, що сучасні фреймворки по типу Angular та React пропонують набір методів для запобігання XSS атак, але все ж існує вірогідність небезпеки пов'язаної зі зберіганням чутливої інформації у середовищі браузера. Також не є доцільним використання будь-яких алгоритмів шифрування на клієнті, оскільки код додатку цілком завантажується браузером, і будь-хто, в тому числі і зловмисник може переглянути алгоритм для того, щоб розшифрувати значення.

Збереження через JavaScript cookies. Іншим підходом до збереження інформації в браузері є cookies [12]. Як і у попередньому методі, використовується простий JavaScript код для збереження та доступу до даних через document.cookie. Переваги та недоліки такі ж, як і в попереднього підходу. Якщо додаток встановлює cookies через JavaScript, то зловмисник має змогу зчитати їх та відправити собі на сервер.

Збереження за допомогою Server cookies. Сервер може встановлювати cookies у відповіді до клієнта, які неможливо зчитати через JavaScript. Такі cookies додаються до відповіді на запит клієнта та позначаються як HttpOnly [13]. Однак обов'язково рекомендується встановлювати атрибут SameSite зі значенням Lax або Strict [13] для того, щоб cookies не передавались на сторонні ресурси при запитах із браузера. Якщо не встановити атрибут SameSite та якщо браузер не підтримує автоматичну установку SameSite. Lax для SameSite.None cookies, то це уможливлює так звану атаку з прихованої iframe (Hidden iframe attack). Враховуючи, що додаток вразливий до XSS атаки та має HttpOnly cookies без атрибуту SameSite, зловмисник може вбудувати прихований iframe, який надсилає запит на сторонній сервіс. Усі cookies із невстановленим SameSite або SameSite.None будуть передані зловмиснику.

Переваги підходу з Server cookies:

- Робить XSS атаку неможливою за правильної конфігурації.
- Найбільш безпечний метод зберігання чутливої інформації.

Недоліки такого підходу:

- Обов'язкова наявність сервера в одному домені з клієнтом для того щоб записати cookie та встановити атрибут SameSite.Strict.
- Потенційне ускладнення процесу публікації додатку, оскільки потрібно публікувати і клієнт, і сервер.

Використання методу Backend For Frontend (BFF) при авторизації через PKCE

Застосувавши підхід до збереження даних через Server cookies у контексті авторизації PKCE, ми отримаємо метод, який має назву Backend For Frontend (BFF) та набуває популярності серед розробників додатків, що класифікуються як відриті. Деякі розробники пропонують готові рішення BFF для додатків. Наприклад, розробники Identity Service – компанія Duende, котра не є частиною Microsoft, пропонує використовувати Duende BFF [14] як їх модуль від Identity Service, доступний компаніям з підпискою Enterprise. Перевагою такого рішення є функціонал та здатність працювати з існуючими системами Identity Service. Серед недоліків можна назвати високу ціну на Enterprise підписку та тісну інтеграцію лише з провайдерами Identity Service 5.

Враховуючи відсутність інших рішень на ринку та високу ціну корпоративної підписки Microsoft Enterprise, було ухвалено рішення розробити власну версію BFF на C# .NET 6.0, яка працює з клієнтами стандарту OAUTH 2.0. Розглянемо приклад послідовності авторизації у системі, яка складається з клієнту Blazor Web Assembly [15] та сервера BFF написаному на C# .NET 6.0. Приклад сервера авторизації стандарту OAUTH 2.0 розглядатися не буде, оскільки такий матеріал виходить за межі цієї статті.

Послідовність процесу авторизації з алгоритмом PKCE та використанням методу BFF наступна:

1. Користувач розпочинає авторизацію, натиснувши на кнопку «Увійти» у додатку.
2. Додаток генерує код-виклик та код підтвердження (детальніше у розділі «Генерація кодів для алгоритму PKCE»).
3. Додаток надсилає код-виклик, код-підтвердження та адресу зворотного виклику на BFF через POST запит.
4. BFF додає код-підтвердження до Response.Cookies, встановивши атрибути HttpOnly, Secure, та SameSite. Strict.
5. BFF перенаправляє запит до сервера авторизації, методу /authorize разом із параметрами коду-виклику та адресу зворотного виклику.
6. Сервер авторизації направляє користувача на сторінку авторизації, де той вводить свій логін та пароль.
7. Результат обробляється на сервері авторизації, який зберігає код-виклик та генерує код авторизації.
8. Сервер авторизації перенаправляє запит на адресу зворотного виклику клієнта, додаючи разом з тим код авторизації.
9. Клієнт надсилає POST запит до BFF на отримання токену, додаючи код авторизації до параметрів.
10. BFF зчитує код-підтвердження через cookies, додає до запиту код підтвердження, код авторизації та викликає сервіс авторизації для отримання токену. BFF на даному етапі може видалити код-підтвердження із cookies.
11. Сервер авторизації перевіряє отриманий код підтвердження з кодом виклику, а також код авторизації та IP-адресу запиту.
12. Після успішної перевірки, сервер генерує токен авторизації та відправляє його як результат запиту.

13. BFF отримує токен авторизації, записує його у cookies з параметрами HttpOnly, Secure, SameSite.Strict.
14. BFF повертає успішний результат клієнтові.
15. Клієнт посилає запит на інформацію по користувачу через BFF, відновлює стан.

Процес авторизації ілюструє діаграма послідовності, зображена на Рис. 3.

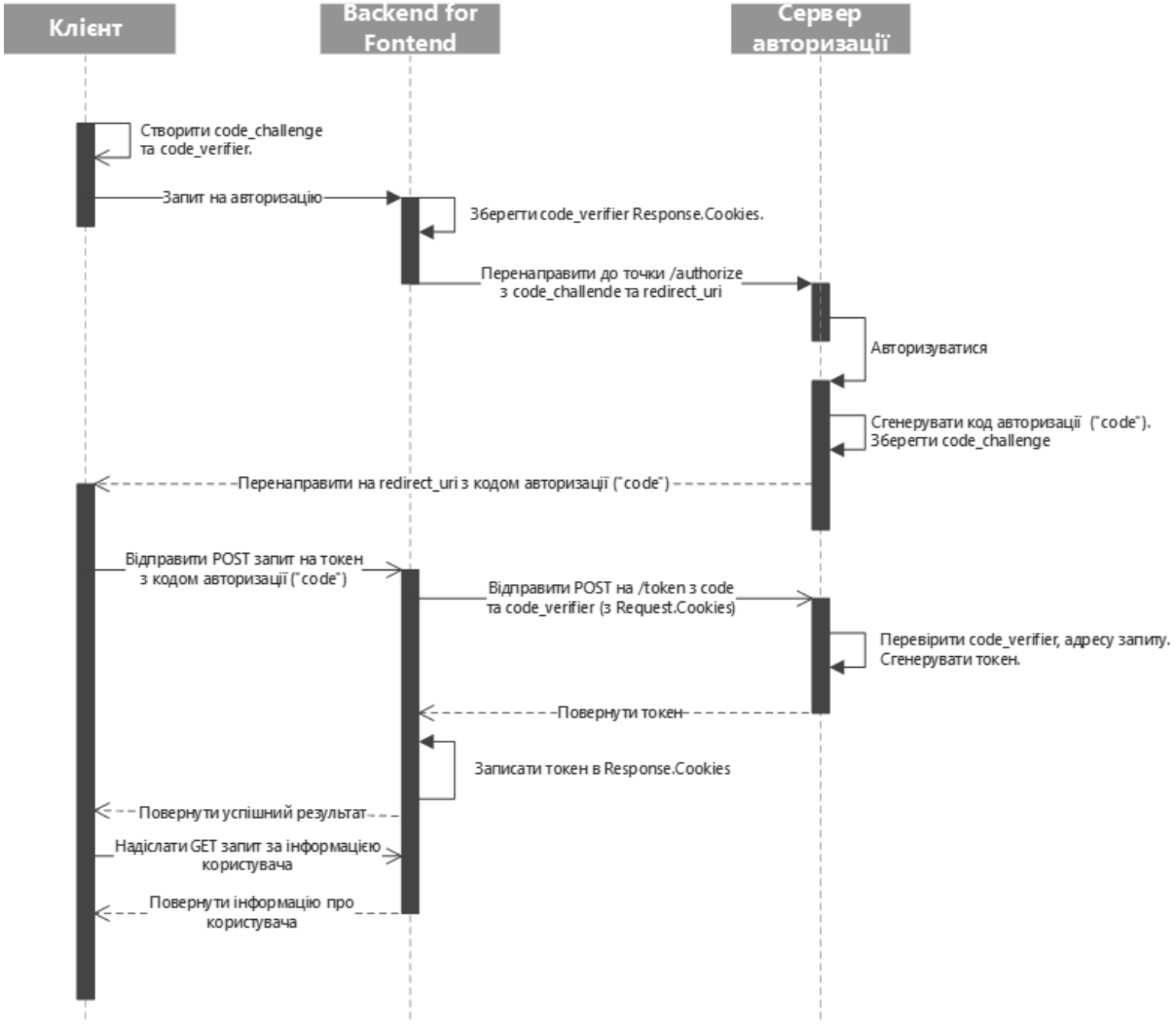

Рис. 3. Діаграма послідовності для авторизації з використанням PKCE та BFF

Таким чином, після успішної авторизації, клієнт авторизований та має змогу викликати методи BFF призначені для авторизованих користувачів. При цьому на клієнті не має жодної інформації вразливої до XSS-атак. Всі чутливі дані як-то code_verifier чи токен зберігаються через HttpOnly cookies з атрибутом SameSite. Strict, як описано у розділі вище. У даній моделі BFF сервіс по суті являє собою реалізацію шаблону зворотного проксі (Reverse Proxy) [16], пропускаючи запити крізь себе та оброблюючи авторизацію клієнта.

Висновки

У даній статті було розглянуто різницю між конфіденційними та відкритими клієнтами, здійснено аналіз проблем, пов'язаних із авторизацією відкритих клієнтів за протоколом OAUTH 2.0. У результаті аналізу було виявлено алгоритми, моделі та методи для реалізації процесу авторизації у відкритих додатках. Проведено роботу над моделюванням та практичною реалізацією алгоритму Proof Key for Code Exchange з методом Backend For Frontend. Результатом роботи є відкритий додаток, що використовує розроблену модель та алгоритми для авторизації, та не є вразливим до атак, виділених при теоретичному аналізі: Cross-Site-Scripting (XSS), Auth Code Interception, CSRF та інших.

Література

1. The OAuth 2.0 Authorization Framework. *Microsoft Internet Engineering Task Force (IETF).* URL: https://datatracker.ietf.org/doc/html/rfc6749#section-2.1 (дата звернення 01.08.2022).
2. Testing for OAuth Client Weaknesses. *OWASP Project.* URL: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/05.2-Testing_for_OAuth_Client_Weaknesses (дата звернення 01.08.2022).

3. Bansal, C., Bhargavan, K., Delignat-Lavaud, A. and Maffeis, S., 2014. Discovering concrete attacks on website authorization by formal analysis. *Journal of Computer Security*, 22(4), pp.601-657.
4. Ghasemisharif, M., Ramesh, A., Checkoway, S., Kanich, C. and Polakis, J., 2018. O Single {Sign-Off}, Where Art Thou? An Empirical Analysis of Single {Sign-On} Account Hijacking and Session Management on the Web. *In 27th USENIX Security Symposium (USENIX Security 18)* (pp. 1475-1492).
5. Lodderstedt, T., Bradley, J., Labunets, A. and Fett, D., OAuth 2.0 Security Best Current Practice (draft-ietf-oauth-security-topics-16). *Internet Engineering Task Force (IETF).* Available from: http://www.watersprings.org/pub/id/draft-ietf-oauth-security-topics-06.html [Accessed 1/08/2022].
6. Proof Key for Code Exchange by OAuth Public Clients. *Google Internet Engineering Task Force (IETF)* URL: https://datatracker.ietf.org/doc/html/rfc7636 (дата звернення 01.08.2022).
7. Lodderstedt, T., McGloin, M. and Hunt, P., 2013. RFC 6819: OAuth 2.0 threat model and security considerations. Internet Engineering Tast Force (IETF), pp.1-71.
8. App security best practices. *Android Developers Documentation.* URL: https://developer.android.com/topic/security/best-practices#safe-data (дата звернення 01.08.2022).
9. Encrypting Your App's Files. Protect the user's data in iOS by encrypting it on disk. *Apple Developers Documentation.* URL: https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy/encrypting_your_app_s_files (дата звернення 01.08.2022).
10. Window.localStorage. *Mozilla Development Documentation.* URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage (дата звернення 01.08.2022).
11. Cross Site Scripting (XSS). *OWASP Community.* URL: https://owasp.org/www-community/attacks/xss/ (дата звернення 01.08.2022).
12. Using HTTP cookies. *Mozilla Development Documentation.* URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#security (дата звернення 01.08.2022).
13. Same Site cookies. *Mozilla Development Documentation.* URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite (дата звернення 01.08.2022).
14. BFF Security Framework. *Duende Software.* URL: https://docs.duendesoftware.com/identityserver/v5/bff/ (дата звернення 01.08.2022).
15. ASP.NET Core Blazor. *Microsoft Documentation.* URL: https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0 (дата звернення 01.08.2022).
16. A. Chiarelli. Building a Reverse Proxy in .NET Core. *Auth0 Blog.* URL: https://auth0.com/blog/building-a-reverse-proxy-in-dot-net-core/ (дата звернення 01.08.2022).

References

1. The OAuth 2.0 Authorization Framework. *Microsoft Internet Engineering Task Force (IETF).* Available from: https://datatracker.ietf.org/doc/html/rfc6749#section-2.1 [Accessed 1/08/2022].
2. Testing for OAuth Client Weaknesses. *OWASP Project.* Available from: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/05.2-Testing_for_OAuth_Client_Weaknesses [Accessed 1/08/2022].
3. Bansal, C., Bhargavan, K., Delignat-Lavaud, A. and Maffeis, S., 2014. Discovering concrete attacks on website authorization by formal analysis. *Journal of Computer Security*, 22(4), pp.601-657.
4. Ghasemisharif, M., Ramesh, A., Checkoway, S., Kanich, C. and Polakis, J., 2018. O Single {Sign-Off}, Where Art Thou? An Empirical Analysis of Single {Sign-On} Account Hijacking and Session Management on the Web. *In 27th USENIX Security Symposium (USENIX Security 18)* (pp. 1475-1492).
5. Lodderstedt, T., Bradley, J., Labunets, A. and Fett, D., OAuth 2.0 Security Best Current Practice (draft-ietf-oauth-security-topics-16). *Internet Engineering Task Force (IETF).* Available from: http://www.watersprings.org/pub/id/draft-ietf-oauth-security-topics-06.html [Accessed 1/08/2022].
6. Proof Key for Code Exchange by OAuth Public Clients. *Google Internet Engineering Task Force (IETF)* Available from: https://datatracker.ietf.org/doc/html/rfc7636 [Accessed 1/08/2022].
7. Lodderstedt, T., McGloin, M. and Hunt, P., 2013. RFC 6819: OAuth 2.0 threat model and security considerations. Internet Engineering Tast Force (IETF), pp.1-71.
8. App security best practices. *Android Developers Documentation.* Available from: https://developer.android.com/topic/security/best-practices#safe-data [Accessed 1/08/2022].
9. Encrypting Your App's Files. Protect the user's data in iOS by encrypting it on disk. *Apple Developers Documentation.* Available from: https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy/encrypting_your_app_s_files [Accessed 1/08/2022].
10. Window.localStorage. *Mozilla Development Documentation.* Available from: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage [Accessed 1/08/2022].
11. Cross Site Scripting (XSS). *OWASP Community.* Available from: https://owasp.org/www-community/attacks/xss/ [Accessed 1/08/2022].
12. Using HTTP cookies. *Mozilla Development Documentation.* Available from: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#security [Accessed 1/08/2022].
13. Same Site cookies. *Mozilla Development Documentation.* Available from: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite [Accessed 1/08/2022].
14. BFF Security Framework. *Duende Software.* Available from: https://docs.duendesoftware.com/identityserver/v5/bff/ [Accessed 1/08/2022].
15. ASP.NET Core Blazor. *Microsoft Documentation.* Available from: https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0 [Accessed 1/08/2022].
16. A. Chiarelli. Building a Reverse Proxy in .NET Core. *Auth0 Blog.* Available from: https://auth0.com/blog/building-a-reverse-proxy-in-dot-net-core/ [Accessed 1/08/2022].

Одержано 03.08.2022

Про авторів:

Бодак Богдан Вікторович,
аспірант кафедри автоматики і управління
в технічних системах
НТУУ «КПІ імені Ігоря Сікорського»,
Кількість публікацій в українських виданнях – 3,

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії
комп'ютерних обчислень Інституту
програмних систем НАН України,
професор кафедри автоматики і управління
в технічних системах
НТУУ «КПІ імені Ігоря Сікорського».
Кількість публікацій в українських виданнях – понад 180.
Кількість публікацій в зарубіжних виданнях – понад 70.
Індекс Гірша – 6.
Кількість публікацій в українських виданнях – 28.
Кількість зарубіжних публікацій – 12.
http://orcid.org/0000-0002-8435-1451.

Місце роботи авторів:

Інститут програмних систем НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (38)(044) 526-21-48
E-mail: bohdan.bodak@outlook.com, doroshenkoanatoliy2@gmail.com

Прізвища та ініціали авторів і назва доповіді англійською мовою:
Bodak B. V., Doroshenko A.Yu..
Protecting public clients using an authorization algorithm

Прізвища та ініціали авторів і назва доповіді українською мовою:
Бодак Б. В., Дорошенко А. Ю.
Захист відкритих клієнтів за допомогою одного алгоритму авторизації

Контакти для редактора: Бодак Богдан Вікторович,
аспірант кафедри автоматики та управління в технічних системах НТУУ «КПІ»,
e-mail: bohdan.bodak@outlook.com, тел.: (38)(067) 291-50-45 (Telegram, Viber),
(1)(519) 465-97-37 (WhatsApp, iMessage)