

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

До захисту допущено
В.о. завідувача кафедри
Микола ГРАЙВОРОНСЬКИЙ

(підпис)

«_____» _____ 2021 р

Дипломна робота
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системи, технології та
математичні методи кібербезпеки» спеціальності: 125 «Кібербезпека»

на тему: Безпека протоколу крос-авторизації OAuth2.0 _____

Виконав (-ла): здобувач вищої освіти IV курсу групи ФБ-72
(шифр групи)

Король Михайло Сергійович
(прізвище, ім'я, по батькові)

(підпис)

Керівник к.т.н., доц. Демчинський Володимир Васильович
(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Здобувач вищої освіти _____

(підпис)

Київ - 2021 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти -перший (бакалаврський)
Спеціальність (спеціалізація) - 125 Кібербезпека
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри

_____ Микола ГРАЙВОРОНСЬКИЙ
(підпис)

«__» _____ 2021 р.

ЗАВДАННЯ
на дипломну роботу студенту

_____ Корольо Михайлу Сергійовичу _____
(прізвище, ім'я, по батькові)

1. Тема роботи « Безпека протоколу крос-авторизації OAuth2.0 » _____

керівник роботи Демчинський Володимир Васильович, к.т.н., доц.
(посада, науковий ступінь, вчене звання, прізвище та ініціали)

затверджені наказом по університету від «__» _____ 2021 р. №

2. Термін подання здобувачем вищої освіти роботи 07 червня 2021 р

3. Вихідні дані до роботи: попередні дослідження безпеки процесів крос-авторизації. _____

4. Зміст роботи: огляд процесу крос-авторизації OAuth 2.0, аналіз захищеності протоколу в рамках аналізу автоматної моделі та методу повного перекриття, побудова та реалізація атак на протокол, формування рекомендацій захисту, створення застосунку автоматизованої перевірки захищеності OAuth 2.0.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): Безпека протоколу крос-авторизації OAuth 2.0-презентація. _____

6. Дата видачі завдання: 31.01.2021. _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Отримання завдання	31.01.2021	виконано
2	Вивчення та аналіз літератури	01.02.2021 - 01.03.2021	виконано
3	Аналіз схем опису захищеності	01.03.2021 - 12.03.2021	виконано
5	Аналіз загроз та методів захисту	13.03.2021 - 21.03.2021	виконано
6	Узгодження методів аналізу з керівником	22.03.2021 - 30.03.2021	виконано
7	Проходження переддипломної практики	31.03.2021- 16.05.2021	виконано
8	Написання дипломної роботи	31.03.2021- 03.06.2021	виконано
9	Отримання допуску до захисту	03.06.2021	виконано
10	Захист дипломної роботи	15.06.2021	

Здобувач вищої освіти

(підпис)

Науковий керівник роботи

(підпис)

Михайло КОРОЛЬ

(ім'я, прізвище)

Володимир ДЕМЧИНСЬКИЙ

(ім'я, прізвище)

РЕФЕРАТ

Робота об'ємом 70 сторінок, яка містить 40 ілюстрацій, 5 таблиць, 15 джерел за переліком посилань.

Актуальність роботи визначається потребою у вдосконаленні процесу та набору специфікацій протоколу крос-авторизації OAuth 2.0.

Мета роботи полягає у дослідженні рівня безпеки OAuth 2.0.

Об'єкт дослідження: протокол OAuth 2.0 та його розширення.

Предмет дослідження є процес крос-авторизації та автентифікації у протоколі OAuth 2.0 та OpenID Connect.

Методи дослідження є аналіз та узагальнення інформаційних джерел та новітніх публікацій за темою дослідження, нечітке моделювання, системний підхід та експериментальні перевірки із використанням сучасного програмного забезпечення.

Практичне значення одержаних результатів полягає в можливості їх подальшого використання при створенні нових протоколів крос-авторизації або для покращення поточної реалізації OAuth 2.0.

Наукова новизна: побудована автоматна модель OAuth 2.0, та на її основі одержані результати теоретичного рівня захищеності протоколу, які дають поштовх для майбутніх досліджень у напрямку покращення процесу крос-авторизації та створення нових версій OAuth.

Ключові слова: потік коду авторизації, неявний потік, маркер доступу, код авторизації, OpenID Connect, клієнт, сервер авторизації, крос-авторизація.

ABSTRACT

The work consists of 3 sections, contains 40 illustrations, 5 tables, 15 references, the volume of work is 70 pages.

The task of the work is to study the process of delegative authorization within OAuth 2.0, analyze security considerations of the protocol and identify potential design flaws, describe and implement attacks that are based on identified threats, or represent the result of incorrect development. Create and form practical recommendations and recommendations for improving the current version of the protocol, implement utilite for controlling and evaluating the quality of OAuth 2.0.

The relevance of the work is determined by the need of improving the process and the set of specifications of the delegative authorization protocol OAuth 2.0.

The purpose of this is to evaluate the security and safety level of OAuth 2.0.

Object of research: OAuth 2.0 protocol and its extension.

The subject of the research is the process of delegative authorization and authentication in the OAuth 2.0 protocol and OpenID Connect.

Research methods are analysis and summarization of information sources and the latest publications related to the research topic, modeling, systematic approach and experimental tests using modern software.

The practical application of the obtained results lies in the possibility of their further use in the creation of new delegative authorization protocols or improvement of the current implementation of OAuth 2.0.

The scientific novelty: proposed finite state machine of OAuth 2.0 that served as a basis for gaining results about the theoretical level of protocol security, which can be used as a baseline for future research in the process of delegative authorization improvement and creation of new versions of OAuth.

Keywords: authorization code flow, implicit flow, access token, authorization code, OpenID Connect, client, authorization server, delegative authorization.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів.....	7
Вступ.....	8
1 OAuth 2.0. Процес крос-авторизації.....	10
1.1 Загальні положення.....	10
1.2 Потік коду авторизації.....	13
1.3 Неявний потік.....	16
1.4 OpenIDConnect.....	17
1.5 Запит до сервера ресурсів.....	19
1.6 Висновки до розділу.....	20
2 Аналіз безпеки OAuth 2.0.....	21
2.1 OAuth 2.0. Аналіз автоматної моделі.....	21
2.2 Модель порушника.....	28
2.3 Оцінка захищеності методом повного перекриття... ..	29
2.4 Атаки на OAuth 2.0.....	42
2.5 Висновки до розділу.....	59
3 OAuth 2.0. Покращення рівня безпеки.....	60
3.1 Практичні рекомендації для розробників.	60
3.2 Автоматизована перевірка захищеності OAuth 2.0.....	64
3.3 Висновки до розділу.....	66
Висновки.....	67
Перелік джерел посилання.....	68
Додаток А.....	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Власник ресурсу (resource owner) - суб'єкт, здатний надати доступ до захищеного ресурсу.

Сервер ресурсів (resource server) - сервер (як правило співпадає з сервером авторизації), що зберігає захищені ресурси, здатний прийняти та відповісти на запити до ресурсів за допомогою маркерів доступу.

Клієнт (client) - застосунок, що робить запити дозволу до захищених ресурсів.

Сервер авторизації (authorization server) - сервер, який видає клієнту маркери доступу, виконує аутентифікацію власника ресурсу.

API (Applied Programming Interface) - прикладний програмний інтерфейс.

Потоки (flows або grant types) - це сценарії, які виконує клієнт для отримання маркера доступу від сервера авторизації.

Маркер доступу (access token) - це рядок, що представляє авторизацію, видану клієнту.

Маркер оновлення (refresh token) - надаються клієнту сервером авторизації і використовуються для отримання нового маркера доступу коли поточний маркер доступу стає недійсним.

HTTP - HYPER TEXT TRANSFER PROTOCOL.

TLS - TRANSPORT LAYER SECURITY

CA - Сервер Авторизації.

CP - Сервер Ресурсів.

JWT - JSON Web Token.

ЕП - Електронний підпис.

ВСТУП

У традиційній моделі клієнт-сервер, клієнт отримує доступ до захищеного ресурсу шляхом автентифікації себе як власника ресурсу на сервері. Для того, щоб надати стороннім програмам доступ до обмежених ресурсів, власник ресурсу мав передати персональні дані (логін, пароль) третій стороні. Це створює кілька проблем та обмежень:

- Сторонні програми отримують надто широкий доступ до ресурсу власника, залишаючи його без будь-якої можливості обмеження тривалості.
- Власники ресурсів не можуть скасувати доступ до окремої третьої сторони без скасування доступу для всіх третіх сторін.
- Компрометація будь-якої сторонньої програми призводить до компрометації кінцевого користувача та всіх захищених цим обліковим записом дані [1].

У жовтні 2012 року була опублікована структура OAuth 2.0 в RFC 6749. Схема дозволила вирішити проблеми, вводячи рівень авторизації та відокремлення ролі клієнта від ролі ресурсу власника. Замість використання облікових даних для доступу, клієнт отримує маркер доступу - рядок, що позначає конкретний обсяг, термін служби та інші атрибути доступу. Маркери доступу видаються стороннім клієнтам сервером авторизації зі схвалення власника ресурсу .

Хоча сама модель є доволі гнучкою та бере до уваги різну природу (браузерний плагін, веб чи прикладна програма) застосунків та і не потребує настільки ж високого рівня довіри до третьої сторони, проте сам протокол вважається доволі абстрактним та залишає багато аспектів безпеки на відповідальність розробника.

Завданням роботи є дослідження роботи процесу крос-авторизації в рамках OAuth 2.0, аналіз захищеності протоколу, виявлення потенційних недоліків

дизайну, опис та реалізація атак, що базуються на виявленій загрозах, або є випадком некоректної розробки. Формування практичних рекомендації та рекомендації покращення поточної версії протоколу, реалізація автоматизованого механізму перевірки захищеності OAuth 2.0.

Актуальність роботи визначається потребою у вдосконаленні процесу та набору специфікацій протоколу крос-авторизації OAuth 2.0 у зв'язку з підвищенням популярності принципу Single Sign On та об'єднання даних з різних джерел в рамках одного застосунку .

Мета роботи полягає у дослідженні рівня безпеки формальної моделі OAuth 2.0 та практичних реалізацій протоколу.

Об'єктом дослідження є протокол OAuth 2.0 та його розширення.

Предметом дослідження є процес крос-авторизації та автентифікації у протоколі OAuth 2.0 та OpenID Connect.

Метод дослідження: аналіз та узагальнення інформаційних джерел та новітніх публікацій за темою дослідження, нечітке моделювання, системний підхід та експериментальні перевірки із використанням сучасного програмного забезпечення.

Наукова новизна: побудовано автоматну модель OAuth 2.0, та на її основі одержані результати теоретичного рівня захищеності протоколу, які, разом з результатами аналізу безпеки методом повного перекриття, дають поштовх для майбутніх досліджень у напрямку покращення процесу крос-авторизації та створення нових версій OAuth.

Практичне застосування полягає в можливості подальшого використання одержаних результатів при створенні нових протоколів крос-авторизації або для покращення поточної реалізації OAuth 2.0.

1 OAUTH 2.0. ПРОЦЕС КРОС-АВТОРИЗАЦІЇ

1.1 Загальні положення

Структура авторизації OAuth 2.0 дозволяє користувачеві надавати сторонньому веб-сайту або програмі доступ до захищених ресурсів користувача, не обов'язково розкриваючи їх довгострокові облікові дані або навіть свою особистість.

Протокол покладається на режими авторизації, що називаються потоками (flows), дозволяючи власнику ресурсу ділитися захищеними даними з сервера ресурсів, не передаючи свої облікові дані. З цією метою сервер OAuth 2.0 видає маркери доступу, які клієнтські програми можуть використовувати як дозвіл від імені власника ресурсу. Надається кілька потоків, придатних для різних типів клієнтів:

Таблиця 1.1 - Види потоків авторизації OAuth 2.0

№	Тип та опис
1	Потік Коду авторизації (Authorization Code Grant). Найпоширеніший потік, використовується для серверних і мобільних веб-застосунків. Адаптований до конфіденційних клієнтів.
2	Неявний потік (Implicit Flow). Бул створений для випадків, коли облікові дані користувача не можуть зберігатися на стороні клієнта, оскільки до них може отримати доступ третя сторона. Прикладами таких програм є рішення, що створені на базі фреймворків React, Angular та розширення браузера, що не містять жодного серверної обробки.
3	Облікові дані власника ресурсу (Resource Owner Password Credentials). Вимагає передачі облікових даних користувача до клієнта.

Продовження таблиці 1.1

	Оскільки в цьому випадку облікові дані будуть частиною запиту, цей потік підходить клієнтам, що мають високий ступінь довіри від власника ресурсу.
4	Клієнтські дані (“Client Credentials”) Цей потік описує підхід, коли клієнтська програма діє від свого імені, та запитує доступ до заздалегідь заданого рівня авторизації між СА та клієнтом. Тобто власник ресурси зазначає на сервері авторизації, що даному застосунку дозволений доступ до даних зах. ресурсів. Клієнт у будь-який момент часу (якщо доступ не відізнано), може передати свої облікові дані серверу авторизації та отримати маркер доступу у відповідь.

Оскільки останні два потоки вважаються допустимими реалізаціями протоколу, проте вимагають найвищого рівня довіри та потребують передачі облікових даних і тому не будуть розглядатись у цій роботі. Необхідні додаткові протоколи:

- **HTTP** - специфікація використовує протокол як базу для обміну даними, та висуває вимоги щодо підтримки запитів GET, POST на веб-серверах клієнта та сервера авторизації. Переправлення HTTP використовується сервером авторизації та клієнтом для передачі необхідних значень між собою.
- **TLS** - RFC 6749 висуває вимогу щодо обов’язкової підтримки найбільш актуальної версії протоколу на стороні клієнта, серверів авторизації та ресурсів, агенту користувача власника ресурсів під час передачі конф. даних та параметрів.

Типи клієнтів:

- **Конфіденційні:** здатні зберігати конфіденційність своїх облікових даних (“секрету” отриманому під час реєстрації).

- **Публічні:** не в змозі зберігати конфіденційність своїх облікових даних (наприклад встановлена власна програма чи браузерний плагін).

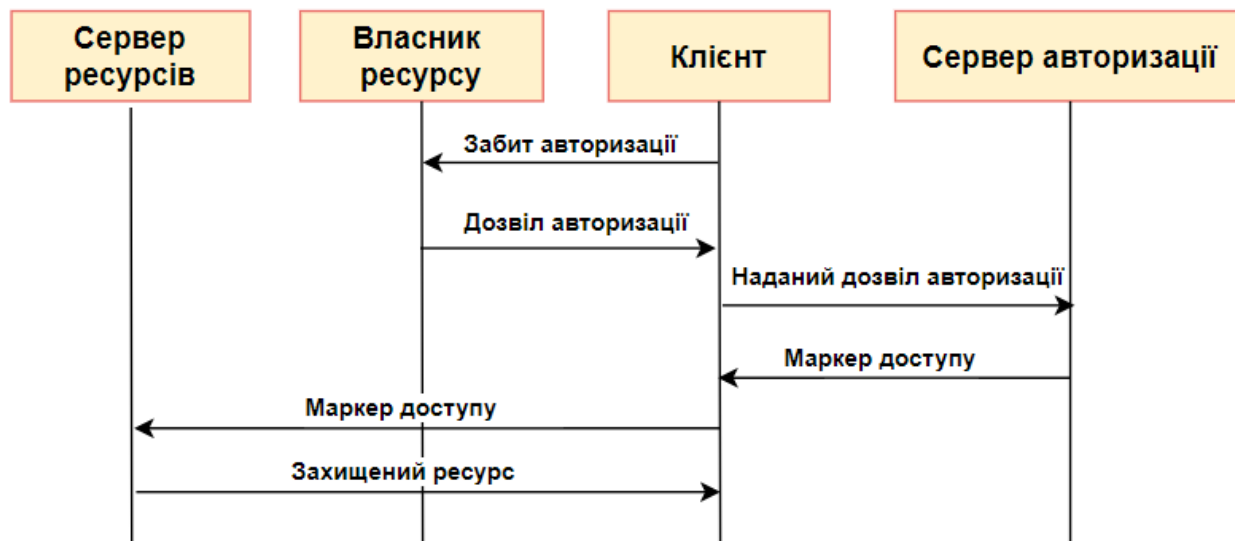


Рисунок 1.1 - Загальна схема крос-авторизації OAuth 2.0

Загальна авторизації виглядає наступним чином:

1. Клієнт вимагає авторизації у власника ресурсу.
2. Клієнт отримує дозвіл на авторизацію, у вигляді певної згоди або конф. параметру, що ідентифікує дозвіл власника ресурсу, та виражається за допомогою одного з чотирьох видів грантів. Тип надання авторизації залежить від методу, який використовує клієнт.
3. Клієнт запитує маркер доступу на сервері авторизації та пред'являючи дозвіл авторизації.
4. Сервер перевіряє дозвіл на авторизацію і, якщо він дійсний, видає маркер доступу.
5. Клієнт запитує захищені ресурси у СР, передаючи маркер доступу.
6. Сервер ресурсів перевіряє маркер доступу, і якщо він дійсний, обслуговує запит [1].

Важливо зауважити, що протокол не виключає можливості існування не зареєстрованих клієнтів, проте не визначає жодної формальної процедури до їх створення та роботи. Тому неявно можна зазначити, що всі можливі типи клієнтів потребують попередньої реєстрації на сервері авторизації.

Кожен клієнт отримує унікальне значення (`client_id`), що буде слугувати ідентифікатором для сервера авторизації, після успішної реєстрації. Наступні параметри мають бути зареєстровані:

- Тип клієнту (конфіденційний чи публічний).
- URI перенаправлення (обов'язково для обох типів клієнтів).
- Додаткові дані, які сервер авторизації вважає потрібними (домен клієнту, назва, обраний метод аутентифікації тощо).

Для обох типів клієнту не виключається можливість реалізації механізму аутентифікації, проте, якщо для публічних це не є обов'язковим, конфіденційні клієнти під час реєстрації зобов'язані отримати або згенерувати облікові дані такі як: пароль, ключ або інший автентифікуючий параметр (залежить від методу автентифікації). В якості автентифікації може використовуватись будь-який протокол чи схема, яка задовольняє вимогам сервера авторизації. Типові способи доведення автентичності клієнта:

- Базова схема автентифікації HTTP.
- Передача секрету в тілі запиту.
- JWT з використанням HMAC або ЕП.
- Двосторонній (mutual) TLS.

1.2 Потік коду авторизації

Код авторизації використовується для отримання маркера доступу та маркера оновлення. Оскільки цей потік на основі перенаправлення, клієнт повинен бути спроможним взаємодіяти з користувальницьким агентом власника ресурсу (як правило веб браузер) і здатний приймати вхідні запити (через перенаправлення) з сервера авторизації. Використання цієї схеми вимагає від будь-якого типу клієнту, що має облікові дані автентифікації, обов'язково підтверджувати свою особистість.

Розглянемо етапи процесу отримання маркера доступу:

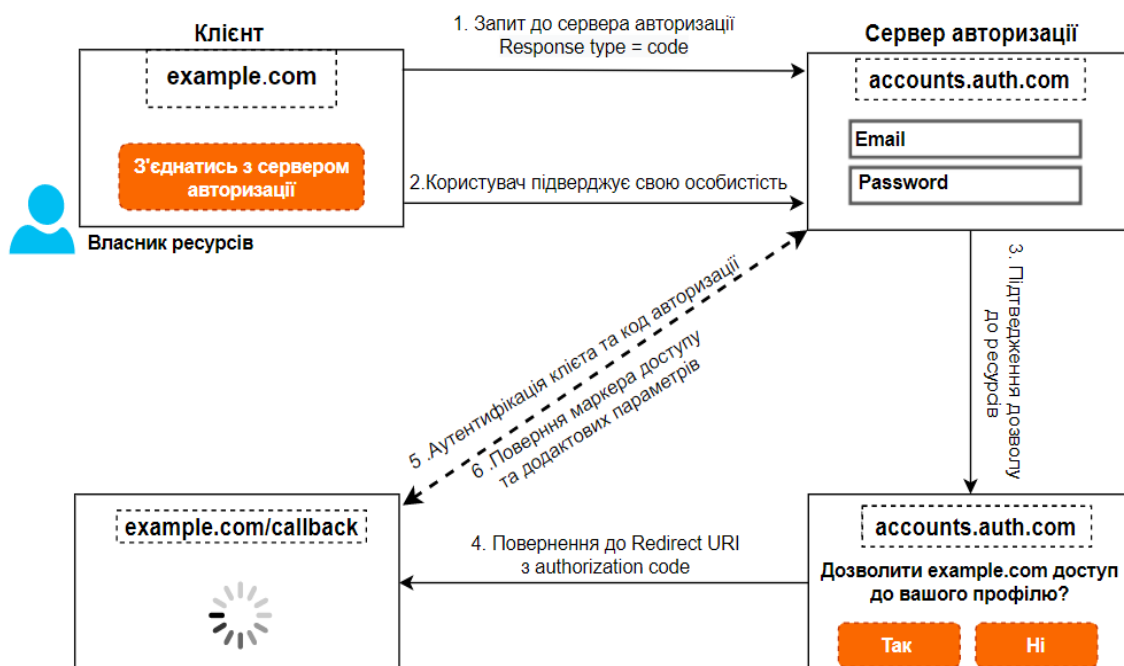


Рисунок 1.2 - Схема авторизації OAuth 2.0. Потік коду авторизації

1. Клієнт направляє власника ресурсів на сервер авторизації, щоб розпочати процес. Типові параметри:

- **response_type** - обов'язковий; має бути встановлений зі значенням code.
- **client_id** - обов'язковий; ідентифікатор клієнта.
- **redirect_uri** - опціональний; URI перенаправлення для коду авторизації або маркера оновлення.
- **scope** - опціональний; визначає рівень авторизації.
- **state** - рекомендований; значення не обмежене у форматі та має бути повернене під час перенаправлення до клієнта (етап №4).

2. Сервер авторизації перевіряє запит, щоб переконатися, що всі необхідні параметри присутні та дійсні. Вимується автентифікація власника ресурсу.

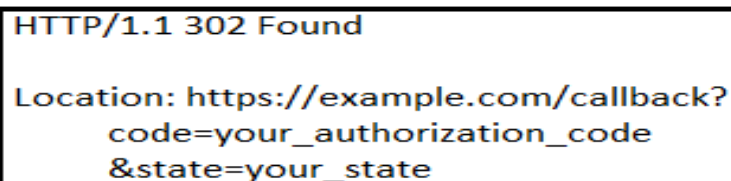
3. Власник ресурсу явним чином дає згоду чи заперечує доступ до даних, що вказані в параметрі scope (етап №1).

```
https://accounts.auth.com/authorize?
client_id=your_client_id&redirect_uri=your_redirect_url
&response_type=code
```

Рисунок 1.3 - Приклад запиту авторизації.

4. Якщо власник ресурсу надає дозвіл на доступ, сервер видає код авторизації та доставляє його клієнту, додаючи наступні параметри до компонента запиту:

- **code** - обов'язковий; одноразовий код виданий сервером авторизації для



```
HTTP/1.1 302 Found
Location: https://example.com/callback?
code=your_authorization_code
&state=your_state
```

подальшого обміну на маркер доступу (етап №6).

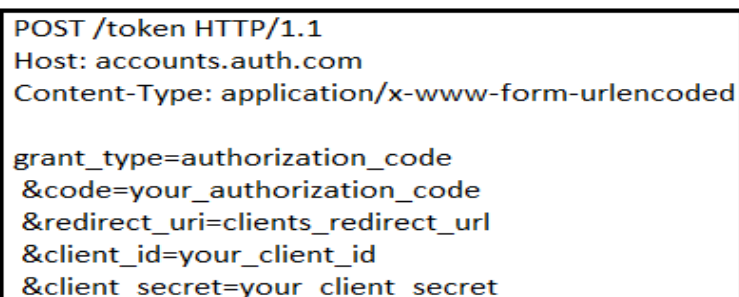
Рис.1.4 - Приклад повернення коду авторизації

- **state** - обов'язковий, якщо аналогічний параметр був присутній у запиті на авторизацію (етап №1) значення повертається.

5. Клієнт виконує запис до сервера авторизації з наступними параметрами:

- **grant_type** - обов'язковий; має мати значення "authorization_code".
- **code** - обов'язковий; передає код авторизації отриманий на минулому етапі.
- **redirect_uri** - обов'язковий; має співпадати з URI перенаправлення на етапі №1 або з зареєстрованим значенням.
- **client_id** - обов'язковий; за умови ігнорування процесу аутентифікації.

6. Якщо запит на етапі №5 є валідним (вимоги щодо автентифікації та формату виконані), сервер авторизації повертає клієнту маркер доступу та маркер оновлення



```
POST /token HTTP/1.1
Host: accounts.auth.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&code=your_authorization_code
&redirect_uri=clients_redirect_url
&client_id=your_client_id
&client_secret=your_client_secret
```

Рис 1.5 - Приклад запиту маркера доступу.

Типові параметри:

- **access_token** - обов'язковий; маркер доступу, виданий сервером авторизації.
- **token_type** - обов'язковий; тип маркеру, як правило "Bearer".
- **expires_in** - рекомендований; час життя маркера доступу (значення в секундах). Якщо значення відсутнє використовується значення за замовчуванням, яке сервер авторизації встановив заздалегідь.

- **refresh_token** - опціональний; якщо видано, використовується для отримання нового маркера доступу коли поточний перестав бути дійсним.
- **scope** - якщо виданий рівень авторизації співпадає з запитом на етапі №1 параметр є опціональним. Якщо сервер авторизації видав рівень авторизації відмінний від бажаного, параметр є обов'язковим [1].

1.3 Неявний потік

Неявний потік є менш надійним методом отримання маркера доступу, оскільки схема взагалі не має жодного етапу з механізмом аутентифікації клієнта. Отримання токена оновлення також неможливе за цим методом. Схема наявного потоку складається з перших 4-ох етапів потоку коду Авторизації, з деякими незначними змінами.

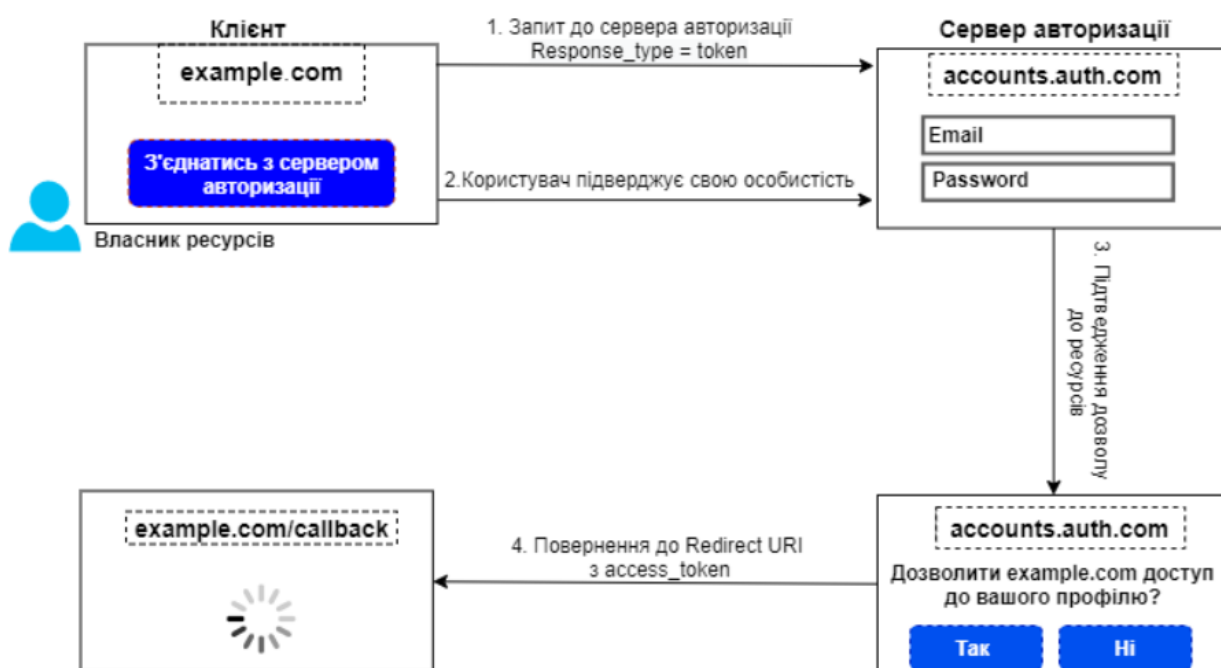


Рисунок 1.6 - Загальна схема авторизації. Неявний потік

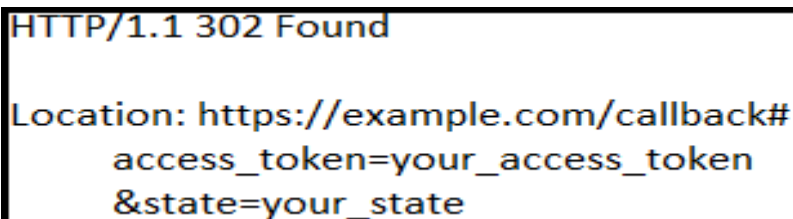
Етапи авторизації:

1. Значення та параметри співпадають з етапом №1 Коду Авторизації, єдина зміна: response_type має значення "token". Це має повідомити серверу авторизації про використання саме неявного потоку.

2. Автентифікація власника ресурсу. Значення та параметри співпадають з етапом №2 потоку Коду Авторизації. Зауваження потребує те, що на сервері авторизацію має відбуватись обов'язкова перевірка URI перенаправлення з зареєстрованим значенням.

3. Дозвіл власника ресурсу. Аналогічно етапу №3 потоку Коду Авторизації.

4. Отримання маркера доступу. Дві основні відмінності від потоку коду авторизації: маркер доступу повертається замість



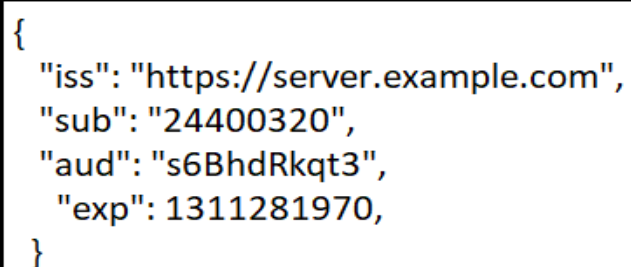
```
HTTP/1.1 302 Found
Location: https://example.com/callback#
        access_token=your_access_token
        &state=your_state
```

Рис. 1.7 - Приклад повернення маркера доступу

тимчасового коду, і його значення повертається у фрагменті URL-адреси. Роблячи це, сервер гарантує, що програма зможе отримати доступ до значення з URL-адреси, але браузер не надсилатиме маркер доступу в запиті HTTP назад до сервер.

1.4 OpenID Connect

OpenID Connect реалізує автентифікацію як розширення процесу авторизації OAuth 2.0. Клієнти вимагають використання цього розширення, включивши значення `openid` до параметру `scope` під час запиту авторизації. Інформація про виконану автентифікацію повертається у форматі JWT, що називається ID Token.



```
{
  "iss": "https://server.example.com",
  "sub": "24400320",
  "aud": "s6BhdRkqt3",
  "exp": 1311281970,
}
```

Рисунок 1.8 - Вміст ID Token

Основне розширення, яке OpenID Connect робить до OAuth 2.0, щоб забезпечити автентифікацію кінцевих користувачів, є структура даних ID токен, яка і зберігає інформацію про кінцевого користувача [2]. Наступні параметри

використовуються в ID токени для всіх потоків OAuth 2.0, що використовуються OpenID Connect:

- **iss** - обов'язковий; ідентифікує сервер авторизації що видав . Значенням є URL сервера авторизації.
- **sub** - обов'язковий; ідентифікує власника ресурсу (кінцевого користувача.
- **aud** - обов'язковий; ідентифікує клієнта.
- **exp** - обов'язковий; час, після якого токен вважається недійсним.

Передача ідентифікуючих даних не відбувається в чистому JSON форматі, замість цього передбачається використання JWT. Маркер повинен бути підписаний сервером авторизації за допомогою веб-підпису JWS, щоб досягти цілісності та неспростовності. Він також може бути зашифрований за допомогою веб-шифрування JWE для конфіденційності. Веб токен розділений на 3 частини: заголовок (містить дані формату ЕП), інформація користувача (ID token) та електронний підпис [2].

Автентифікація може йти одним із трьох шляхів: потік коду авторизації, неявний потік або гібридний потік .

Таблиця 1.2 - Потоки OpenID Connect

Потік	Відмінності від потоку oAuth 2. 0
Потік коду авторизації	Етап №1: параметр "scope" стає обов'язковим та має містити значення "openid".Етап №6: разом з маркером доступу повертається параметр "id_token" зі значенням у форматі JWT.
Неявний потік	Етап №1: параметр "scope" стає обов'язковим та має містити значення "openid"; "response_type" має містити значення "id_token". Етап №4: разом з маркером доступу повертається параметр "id_token" у форматі JWT.

Продовження таблиці 1.2

Гібридний потік	Є комбінацією неявного потоку та потоку коду авторизації. Етап №1: параметр "scope" стає обов'язковим та має містити значення "openid"; "response_type" має містити значення "id_token code". Етап №4: співпадає з відповідним етапом неявного потоку, але додатково повертається параметр "id_token" зі значенням у форматі JWT та код авторизації. Далі аналогічно потоку коду авторизації.
-----------------	--

1.5 Запит до сервера ресурсів

OAuth 2.0 визначає три способи надсилання токенів доступу, з метою отримання захищених даних власника ресурсу:

1. Передача у полі заголовку авторизації HTTP.

Маркер доступу надсилається в заголовок запиту "Authorization", додатково клієнт має підтримувати базову схему аутентифікації HTTP RFC2617 та зобов'язаний її використовувати в рамках даного типу передачі [3].

2. Параметризована передача в тілі запиту HTTP.

Під час надсилання маркера доступу в тілі запиту HTTP, клієнт додає маркер доступу до тіла запиту за допомогою параметра "access_token". Для цього типу передачі висувається ряд вимог, щодо формату запиту, серед яких основними є: вміст має повністю складатись з символів ASCII; використання методу GET заборонено.

3. Передача параметру в URI.

Надсилання маркера доступу відбувається шляхом додавання маркер доступу до компонента запиту URI запиту. Через питання безпеки метод не є рекомендованим для використання, проте його використання не забороняється.

Лише перший метод має заявлену автентифікацію клієнта. Інші два методи запиту до сервера ресурсів її не мають. Зв'язку з цим параметризована передача в

тілі запиту або передача в URI мають містити дані автентифікації. Сервер ресурсів зобов'язаний відповідати заголовком "WWW-Authenticate" примусово ініціалізуючи використання базової автентифікації HTTP, якщо клієнт не підтверджує свою сутність під час запиту іншим чином (будь-яким, що задовольняє сервер ресурсів) [3].

Висновки до розділу 1

В даному розділі розглянуто принцип роботи протоколу OAuth 2.0. Виділені два основних режима авторизації : потік коду авторизації і неявний потік та розглянуто типові сценарії їх використання. Зазначені основні параметри, що передаються, зберігаються та перевіряються на кожному з етапів авторизації. Розглянуто розширення OpenIDConnect як надбудова для впровадження автентифікації. Таким чином, розглянутий механізм надає доступ до захищених ресурсів без необхідності передачі облікових даних власника ресурсу.

2 АНАЛІЗ БЕЗПЕКИ OAUTH 2.0

2.1 OAuth 2.0. Аналіз автоматної моделі.

У моделях безпеки на основі кінцевих автоматів (state machine model) для перевірки захищеності системи робляться деякі припущення щодо властивостей зловмисника та аналізується можливість отримання даних, які вважаються недоступними порушнику. Нехай S_i - деякий стан автомату, що містить конфіденційний об'єкт, якщо даний стан є досяжним для зловмисника та існує така послідовність дій (така, що не суперечить існуючим припущенням) яка дозволяє скомпрометувати даний об'єкт, стверджує про некоректність механізмів безпеки протоколу. Визначимо можливості та знання зловмисника згідно моделі Долева - Яо:

- зловмисник може отримати будь-яке повідомлення, що передане мережею;
- зловмисник є авторизованим користувачем мережі і тому має право встановлювати з'єднання з будь-яким іншим користувачем;
- зловмисник може стати стороною, що приймає повідомлення від будь-якої сторони, що передає;
- зловмисник може посилати будь-якому користувачеві повідомлення від імені будь-якого іншого користувача;
- зловмисник не може вгадувати випадкові числа, вибрані з досить великої множини;
- зловмисник не може розшифровувати, не маючи ключа, або коректно зашифровувати повідомлення за умови використання деякого ідеального алгоритму шифрування;
- зловмисник не може знайти таємний ключ з відкритого ключа (в асиметричних криптосистемах);

- контролюючи засоби зв'язку, зломисник, тим не менш, не може отримати доступ до закритих, внутрішніх ресурсів, наприклад, до пам'яті і жорсткого диска користувача [4].

Простіше кажучи, в моделі Долева-Яо будь-яке повідомлення, що відправляється мережею, повинно розглядатися як таке що, проходить через зломисника або як послане ним.

Кінцевий автомат Мілі OAuth 2.0.

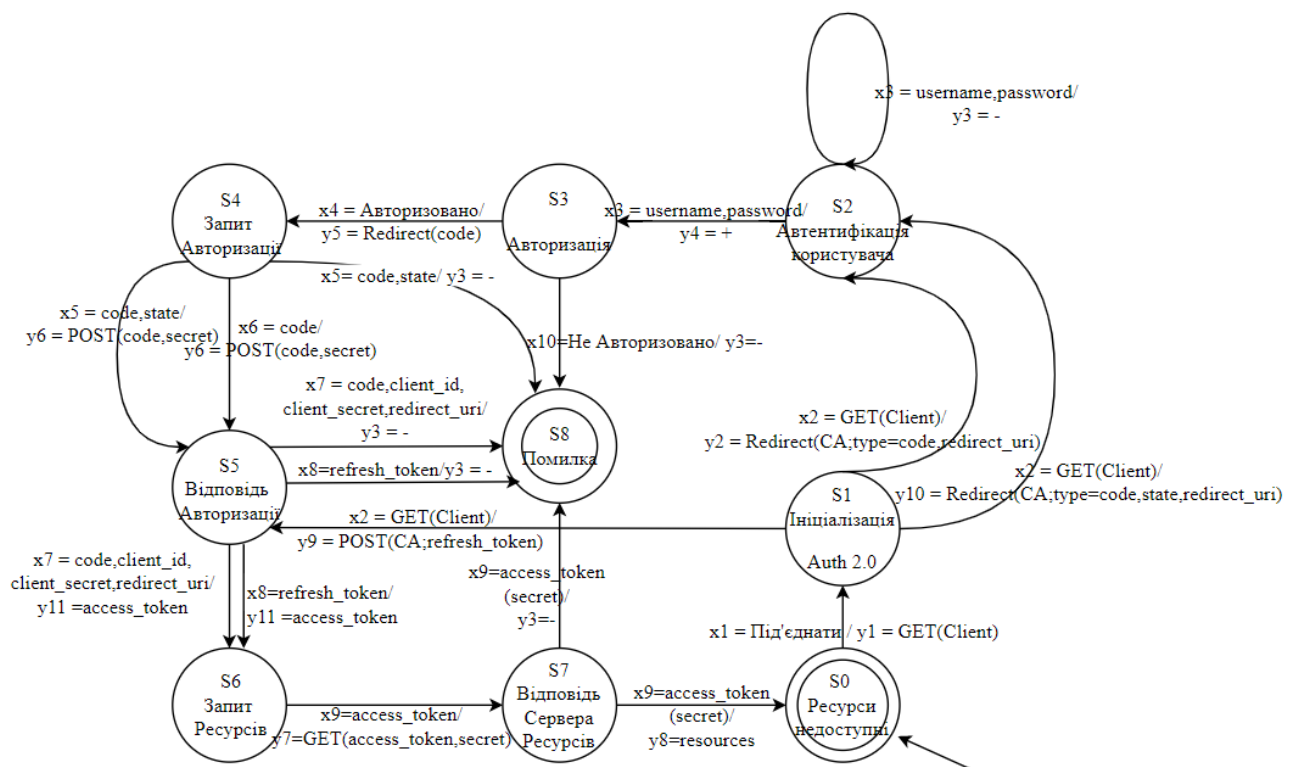


Рисунок 2.1 - Графічне представлення Автомата Мілі. OAuth 2.0

Формальний опис:

- $A = \{S, X, Y, \delta, S_0\}$; $S = \{S_0, S_1 \dots S_8\}$ - множина станів;
- $X = \{x_1, x_2, \dots x_{10}\}$ - множина входів;
- $Y = \{y_1, y_2, \dots y_{11}\}$ - множина виходів;
- δ - функція переходів (задана таблицею та графом).

Оскільки зломисник не має можливості отримати секретні дані у межах самого стану (через припущення про неможливість зчитування даних з пам'яті) доцільним лише є розглядання захищеності переходів зі стану S_i в S_j де $i > j$.

Проаналізуємо усі переходи в яких передається хоча б один конф. параметр та всі можливі дії злоумисника, для формування висновків стосовно стійкості.

Таблиця 2.1 - Функція переходів автомата Мілі OAuth.2.0

	S0	S1	S2	S3	S4	S5	S6	S7	S8
x1	S1/y1	-	-	-	-	-	-	-	-
x2	-	S2/y2 S2/y10 S5/y9	-	-	-	-	-	-	-
x3	-	-	S2/y3 S3/y4	-	-	-	-	-	-
x4	-	-		S4/y5	-	-	-	-	-
x5	-	-	-	-	S5/y6 S8/y3	-	-	-	-
x6	-	-	-	-	S5/y6	-	-	-	-
x7	-	-	-	-	-	S6/y11 S8/y3	-	-	-
x8	-	-	-	-	-	S6/y11 S8/y3	-	-	-
x9	-		-	-	-	-	S7/y7	S8/y3 S0/y8	-
x10	-	-	-	S8/y3	-	-	-	-	-

автоматної моделі OAuth 2.0. Зазначимо, що атаки зниження версії TLS є неможливим, оскільки вимагається використання найбільш актуальної версії. Типи атак, що доступні атакуючому виходячи зазначених можливостей порушника:

- 1) Атака перехвату трафіка.

- 2) Уособлення серверної частини (з точки зору TLS).
- 3) Атака уособлення клієнтської частини (з точки зору TLS).
- 4) Атака повтору.
- 5) Атака затримки запитів (є допоміжною до інших типів атак)
- 6) Атака перебору конф. параметрів.

Одразу зазначимо що для кожного розглянутого етапу є неможливими атаки що зазначені в таблиці 2.2.

Таблиця 2.2 - Аргументація неможливості реалізації атак 1,2,4

№	Причина	Аргументація
1)	Неможливо через TLS з'єднання.	Алгоритми автентифікації RSA, DSA вважаються достатньо надійними. Пакетне шифрування AES, Camellia, ARIA є стійким. Хешування SHA-256 є достатньо стійким до колізій.
2)	Неможливо через TLS з'єднання .	Властивість G_1 (автентифікація за сертифікатом) [5].
4)	Неможливо через TLS з'єднання .	Властивість G_2 (автентифікація повідомлення) та G_3 (захист від повтору) [5].

Переходи:

- $S_1 - S_5$. Відповідає переходу зі стану “Ініціація авторизації” в стан “Очікування маркера доступу” зі входом (no access token, refresh token) на рис.2.3. Конфіденційні параметри: маркер оновлення.

Атака №3 не є доцільна під час ініціалізації, оскільки формує сеанс, продовжувати який можливо лише передачею маркера оновлення, який не є відомим. Атака №5 неможлива, оскільки складність перебору 2^{128} .

Вгадування в будемо вважати успішним, якщо перевірено $\sqrt{N}$ варіантів, де N - кількість можливих значень.

Орієнтовний час перевірки: $T' = \frac{T \times \sqrt{N}}{K}$, T - час однієї операції перевірки (доцільним є припущення в 0.001 с. через існування мережевої взаємодії), K -

кількість дійсних маркерів оновлення, що видані даним СА для даного клієнта (нехай таких буде $10^6 \sim 2^{20}$). Звідси $T' \sim 557.8$ років.

- $S_2 - S_3$. Аналогічно переходу зі стану “Очікування автентифікації користувача” до стану “Очікування дозволу авторизації” на рис. 2.2 (за умови передачі вірних облікових даних). Конфіденційні параметри: облікові дані користувача.

Атака №3 неможлива під час вже встановленого з'єднання з СА, тому що потребує знання ключа шифрування існуючого сеансу, а уособлення під час ініціалізації сесії зводиться до необхідності передачі правильних облікових даних жертви. Вгадування ідентифікаторів сесії є можливим і дозволяє зловмиснику на етапі S_3 надати згоду авторизації, проте порушник не контролює клієнт тому наявність авторизації не призводить до компрометації.

Атака №5 теоретично можлива оскільки OAuth 2.0 не задає складність автентифікації.

Результат компрометації: підбір паролю веде до компрометації облікового запису власника ресурсу.

- $S_3 - S_4$. Відповідає переходу зі стану “Очікування дозволу авторизації” до “Очікування запиту”, де факт надання дозволу є входом, а code та/або state є виходом (зображено на Рис 2.2) . Конфіденційні параметри: код авторизації.

Атака №3 на сеанс з СА та з клієнтом неможлива під час встановленого з'єднання, оскільки потребує знання ключа шифрування існуючого сеансу або зводиться до перебору ідентифікаторів сесії. Вгадування ідентифікатора HTTP сесії можливе та дозволяє зловмиснику передати власний код авторизації під видом коду авторизації користувача, таким чином що веде до прив'язування захищених ресурсів порушника до жертви.

Результат компрометації: потенційно зловмисник отримує захищені ресурси користувача.

Атака №5 унеможлиблюється часом життя коду авторизації (600 с.), та

- **S₄ - S₅.** Перехід між “Очікування відповіді СА” та “Очікування маркера доступу” на Рис 2.1 . Конфіденційні параметри: код авторизації, “секрет” клієнта.

Атака №3 неможлива через передачу автентифікуючого параметра або зводиться до вгадування “секрету” та маркера доступу. Атака №5 неможлива, оскільки

складність перебору обох параметрів 2^{129} .

- **S₅ - S₆.** Характеризується станом “Очікування запиту” на Рис 2.2, де входом є (code, secret, uri, client_id) та виходом маркер доступу (опціонально - маркер оновлення). Конфіденційні параметри: маркер доступу, маркер оновлення.

Атака №3 неможлива під час встановленого з'єднання з СА. Атака №5 неможлива, адже складність перебору 2^{128} .

- **S₆ - S₇.** Відповідає переходу з “Очікування запиту” до “Авторизація ресурсів” на Рис. 2.2. Конфіденційні параметри: маркер доступу, “секрет” клієнта.

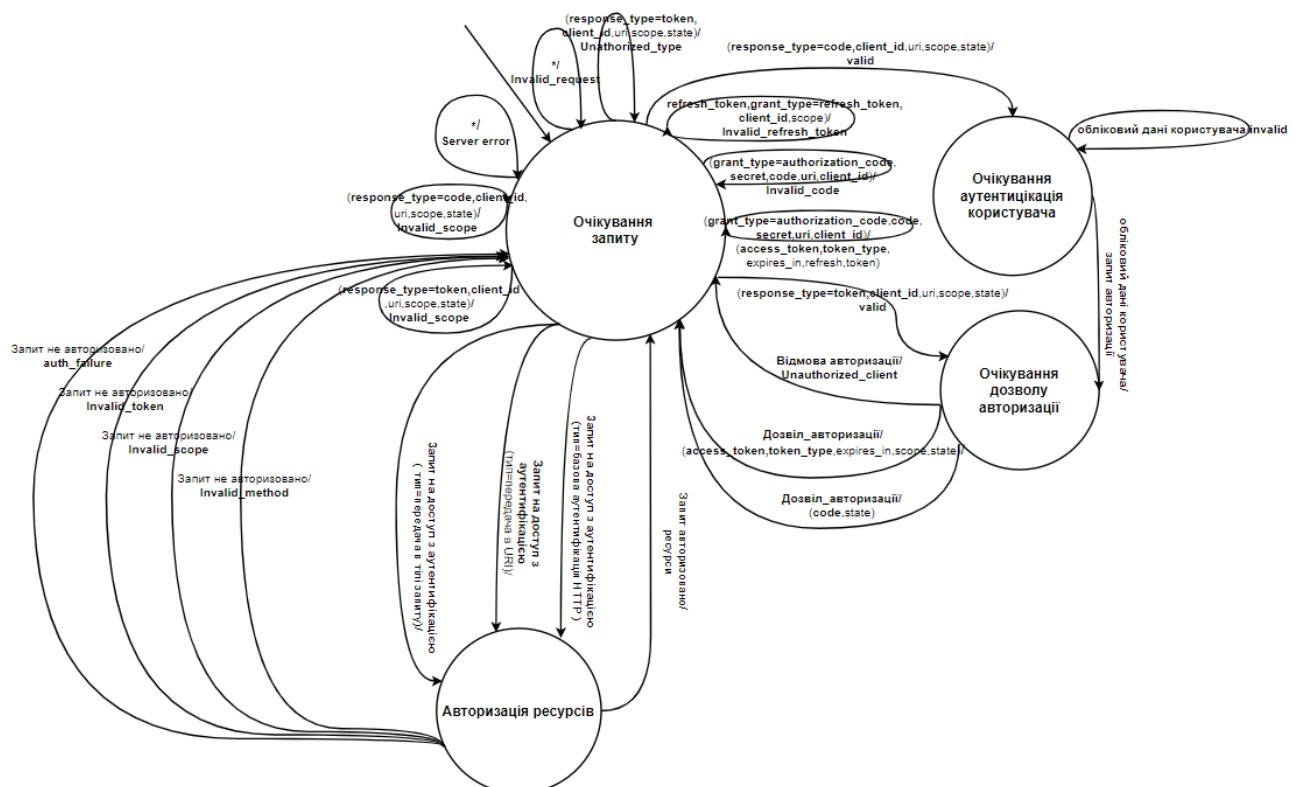


Рисунок 2.2. Структурна схема OAuth 2.0 сервера авторизації.

Атака №3 не є можливою, оскільки сам наявність сеансу ніяк не автентифікує клієнта, а автентифікуючий параметр відомий лише легітимному застосунку, або зводиться до його вгадування. Атака №5 неможлива, оскільки складність перебору обох параметрів 2¹²⁹.

- S₇ - S₈. Демонструється також переходом, між станами “Очікування доступу до ресурсів” та “Доступ авторизовано” на Рис. 2.3. Конфіденційні параметри: захищені ресурси.

Атака №3 неможливо реалізувати під час встановленого з'єднання (значення ключа шифрування не відоме), проте вгадування ідентифікатора сеансу користувача, що не суперечить вимогам.

Результат компрометації: зловмисник отримує доступ до захищених ресурсів користувача, після завершення процесу крос-авторизації. Атака №5 неможлива, оскільки немає відомостей про формат та складність захищеного ресурсу.

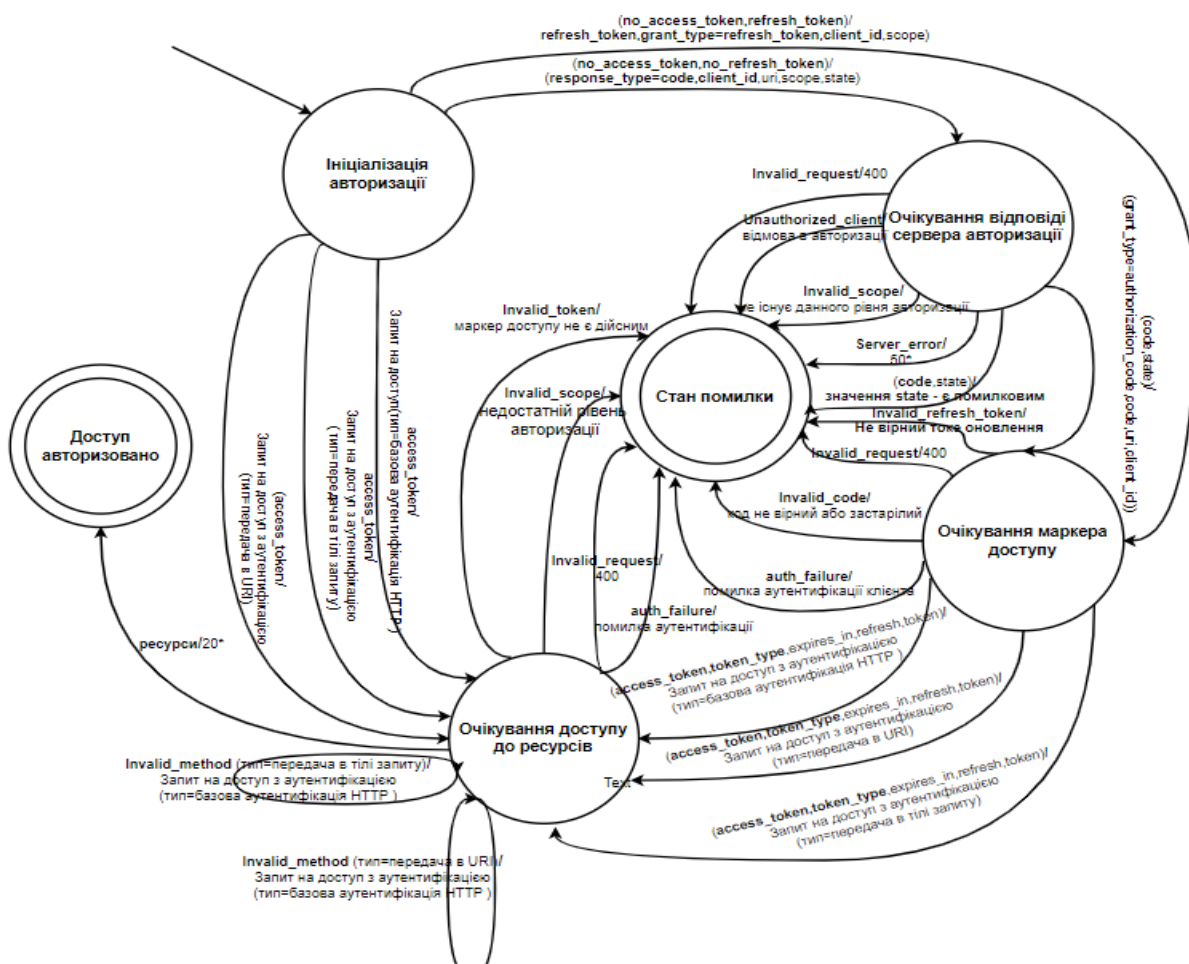


Рисунок 2.3 - Структурна схема OAuth 2.0 клієнта.

Висновки: формальна модель демонструє недоліки захисту в переходах S_2-S_3 , S_3-S_4 , S_7-S_8 , що виражаються у відсутності вимог до процесу автентифікації користувача та розміру ідентифікаторів сесії. Оскільки дані нюанси безпеки покладаються на відповідальність клієнта та СА, тому навряд чи можна стверджувати про абсолютну некоректність формальної схеми OAuth 2.0, оскільки в основу покладена ідея довіри до реалізації застосунку та сервера авторизації. До того ж в реальних випадках завдання перебору, облікових даних та ідентифікаторів HTTP сесій, теж відноситься до завдань, що мають високу оцінку складності. Підсумовуючи, можна стверджувати про недоліки дизайну OAuth 2.0, проте їх наявність рідко веде на практиці.

2.2 Модель порушника

Ціль злоумисника: несанкціонований доступ, підвищення існуючого рівня авторизації, вплив на стабільність роботи сервера авторизації та/або сервера ресурсів отримання, зміна, присвоювання, розголошення особистої інформації до якої має доступ або володіє:

1. Сервер авторизації: імена користувачів та паролі; ідентифікатори та секрети клієнта; маркери оновлення; маркери доступу ; сертифікат / приватний ключ HTTPS;
2. Клієнт: ідентифікатор клієнта (“секрет” або відповідні автентифікуючі дані клієнта); маркери оновлення та маркери доступу; код авторизації;
3. Сервер ресурсів: дані користувача; сертифікат / ключ HTTPS;

Супротивник виступає у двох ролях або їх комбінації:

- **Веб злоумисник** може ділитися шкідливими посиланнями або публікувати коментарі, що містять шкідливий вміст (наприклад, таблиці стилів або зображення) на довіреному веб-сайті. Припускається, що супротивник має можливості веб-зловмисника, тобто експлуатувати вразливості веб частини

клієнта, сервера авторизації, сервера ресурсів. Створювати будь-які запити в рамках протоколу HTTP до застосунку, СА, СР. Допускається що зловмисник здатний змусити жертву вважати шкідливі програми, додатки, веб-сторінки за легітимні, використовувати техніки фішингу та соціальної інженерії.

- **Мережевий зловмисник:** має повний доступ до мережі між клієнтом і сервером авторизації та клієнтом і сервером ресурсів відповідно. Атакуючий може підслуховувати будь-які комунікації між цими сторонами. Передбачається, що він не має доступу до зв'язку між СА та СР. Мережеві та фізичні ресурси для здійснення атаки не обмежені [6].

Таком не включається можливість, що зловмисник здатен об'єднуватись з другим учасником процесу авторизації проти третього.

2.3 Оцінка захищеності методом повного перекриття

Модель безпеки з повним перекриття, будується виходячи з постулату, що система безпеки повинна мати, принаймні, один засіб для забезпечення захисту на кожному можливому шляху впливу порушника на інформаційну систему. У моделі визначається кожен об'єкт, який потребує захисту, загрози та засоби забезпечення безпеки. З кожним об'єктом, що вимагає захисту, зв'язується деякий перелік дій, до яких може вдатися порушник для отримання несанкціонованого доступу до об'єкта [7].

Проаналізуємо протокол OAuth 2.0, використовуючи методику з повним перекриттям, з точки зору виконання всіх умов та рекомендацій специфікації, як засобів захисту. Визначимо множину об'єктів захисту:

$\mathbf{O} = \{ \text{облікові дані клієнта} - \mathbf{O}_1, \text{маркер доступу} - \mathbf{O}_2, \text{маркер оновлення} - \mathbf{O}_3, \text{облікові дані користувача} - \mathbf{O}_4, \text{обліковий запис/захищені ресурси користувача} - \mathbf{O}_5, \text{код авторизації} - \mathbf{O}_6, \text{фізичні/логічні ресурси СА} - \mathbf{O}_7 \}$. Множину загроз $\mathbf{Y} = \{ \dots \}$; множину методів захисту $\mathbf{M} = \{ \dots \}$ що пропонує OAuth 2.0, де $\mathbf{M}_0$ - відсутність засобів безпеки, та множину зовнішніх методів захисту $\mathbf{E} = \{ \dots \}$, які

не є частиною вимог протоколу, сформуємо як частину моделі загроз у табличній формі.

Таблиця 2.3 - Модель загроз [6].

Точка доступу: Клієнт				
Цільовий ресурс	Вразливість/ Слабкість	Потік	Тип додатку	STRIDE
Облікові дані клієнта	Відкритий вихідний код застосунку	Потік коду авторизації. Неявний потік	Всі	SIE
Опис U_1 : секрет може бути знайдений у вихідному коді проекту.			Наслідки: розкриття облікових даних клієнта, уособлення клієнта.	
Існуючі міри захисту: M_1 - “секрет” не видається публічним клієнтам.				
Облікові дані клієнта	Інсталяційні файли	Неявний потік	Нативні	SIE
Опис U_2 : секрет може бути добутий шляхом зворотної розробки інсталяційних файлів застосунку.			Наслідки: розкриття облікових даних клієнта, уособлення клієнта	
Існуючі міри захисту: M_1 - “секрет” не видається публічним клієнтам.				
Маркер доступу/ оновлення	Механізми захисту веб-сервера клієнта	Потік коду авторизації. Неявний потік	Веб-застосунки	IE
Опис U_3 : зловмисник може отримати маркери оновлення/доступу, шляхом компрометації веб-сервера клієнта.			Наслідки: розкриття всіх маркерів доступу/оновлення цього клієнта зловмиснику	
Існуючі міри захисту: M_2 - використання строгої автентифікації клієнта (унеможливилює використання маркерів зловмисником)				
Маркер доступу/ оновлення	Незахищене локальне сховище	Потік коду авторизації. Неявний потік	Нативні	IE

Опис $У_4$: зловмисник може спробувати прочитати локальні файли з маркерами доступу/оновлення.			Наслідки: розкриття марків користувача.Потенційна компрометація ресурсу корист.	
Існуючі міри захисту : E_1 - використання безпечного сховища даних з розмежуванням доступу.				
Маркер доступу/ оновлення	Аутентифікація на пристрої слабка або відсутня	Потік коду авторизації. Неявний потік	Всі	SIE
Опис $У_5$: пристрій жертви (наприклад, мобільний телефон) може бути викрадений.			Наслідки: зловмисник отримує доступ до всіх додатків як законний користувач.	
Існуючі міри захисту: E_2 - покладається на засоби захисту в ОС автентифікація входу в ОС).				
Облікові дані користувача	Користувач. Слабкий захист оточення.	Потік коду авторизації. Неявний потік	Веб застосунки	SIE
Опис $У_6$: шкідливий клієнт може фішингувати паролі кінцевих користувачів за допомогою використання вбудованого зловмисного браузера в процесі авторизації кінцевого користувача, або шляхом представлення власного інтерфейсу користувача замість довіреного агента.			Наслідки: уособлення користувача.	
Існуючі міри захисту: E_2 - покладається на засоби захисту в ОС та оточення.				
Код авторизації/ маркера доступу	Часткова реєстрація Redirect URI. Слабка валідація	Потік коду авторизації. Неявний потік	Всі	IE
Опис $У_7$: зловмисник змінює URI перенаправлення в запиті жертви таким чим, що пройде перевірку сервера авторизації, але буде вказувати на шкідливий ресурс.			Наслідки: компрометація коду авторизації або маркера доступу. Потенційний доступ до захищених ресурсів	

			зловмисника.	
Існуючі міри захисту: M_3 - повна валідація Redirect URI.				
Точка доступу: Узгодження авторизації (Authorization Endpoint)				
Облікові дані користувача	OAuth не вимагає перевірки справжності СА.	Потік коду авторизації. Неявний потік	Bci	SIE
Опис Y_8 : під час перенаправлення до СА, клієнт (шкідливий) переведе користувача за URL, що вказує на зловмисний веб-сайт, який повертає оманливі або неправильні відповіді з запитом ведення облікових даних.			Наслідки: розкриття облікових даних користувача, уособлення користувача.	
Існуючі міри захисту: E_3 - користувач стійкий до соц. інженерії. M_6 - TLS (автентифікація за сертифікатом)				
Дозвіл авторизації (O_5)	Користувач	Потік коду авторизації. Неявний потік	Bci	IE
Опис Y_9 : користувач не розуміє обсяг доступу, який надає .У підсумку надається доступ до додаткових ресурсів.			Наслідки: неавторизований доступ до захищених ресурсів.	
Існуючі міри захисту: M_4 - вимагається пояснення запитуємого рівня доступу.				
Повторна авторизація (O_5)	Час життя авторизації не обмежено	Потік коду авторизації. Неявний потік	Bci	IE
Опис Y_{10} : шкідливий клієнт автоматично отримує доступ, якщо користувач надавав дозвіл авторизації цьому клієнту в минулому.			Наслідки: неавторизований доступ	
Існуючі міри захисту: M_5 - вимагається постійне отримання дозволу авторизації.				
Точка доступу: Отримання маркера доступу (Token EndPoint)				

Маркер доступу/оновлення	Не захищений канал передачі	Потік коду авторизації. Неявний потік	Всі	ІЕ
Опис Y_{11} : зловмисник може підслуховувати канал під час транзиту маркера доступу з сервер авторизації на клієнт.			Наслідки: компрометація маркера доступу. Потенційний доступ до захищених ресурсів користувача.	
Існуючі міри захисту: M_6 - вимагається використання останньої версії TLS.				
Маркери/код авторизації	Сервер авторизації вразливий до SQL-ін'єкцій	Потік коду авторизації; Неявний потік	Всі	ІЕ
Опис Y_{12} : зловмисник може отримати доступ до токенів з БД сервера авторизації, шкідливими запитами типу SQL-ін'єкції.			Наслідки: розкриття всіх маркерів доступу,що є на даному сервері авторизації. Потенційний доступ до захищених ресурсів користувачів.	
Існуючі міри захисту: E_4 - валідація вхідних даних.				
Облікові дані клієнта	Незахищений канал передачі	Потік коду авторизації	Всі	SIE
Опис Y_{13} : атакуючий може підслухати процес автентифікації клієнта, отримавши його автентифікуючий параметр.			Наслідки: розкриття облікових даних клієнта, уособлення клієнта.	
Існуючі міри захисту: M_6 - вимагається використання останньої версії TLS.				
Облікові дані клієнта	Сервер авторизації вразливий до SQL-ін'єкцій	Потік коду авторизації	Всі	SIE
Опис Y_{14} : зловмисник може отримати доступ до автентифікуючих параметрів клієнта з БД сервера авторизації, шкідливими запитами типу SQL-ін'єкції.			Наслідки: розкриття облікових даних клієнта, що дозволяє фішинг або уособлення клієнта	

Існуючі міри захисту: E ₄ - валідація вхідних даних.				
Облікові дані клієнта	Нізка довжина “секретів” клієнта	Потік коду авторизації	Всі	SIE
Опис U ₁₅ : зловмисник може вгадати дійсну пару client_id /secret шляхом перебору.			Наслідки: розкриття облікових даних клієнта, особлення клієнта.	
Існуючі міри захисту: M ₇ - вимагається висока ентропія конф. параметрів.				
Отримання авторизації				
Код авторизації	Заголовки Refferer	Потік коду авторизації	Всі	IE
Опис U ₁₆ : браузері часто передають заголовок Referrer, коли веб-сторінка має вбудований вміст. Ці заголовки можуть бути надіслані навіть коли початковий сайт не довіряє цільовому сайту. Заголовок Referrer зазвичай реєструється для аналізу трафіку.			Наслідки: компрометація коду авторизації. Потенційний доступ до захищених ресурсів.	
Існуючі міри захисту: E ₅ - використання referrer-policy.				
Код авторизації	Журнали запитів веб-сервера, клієнта.	Потік коду авторизації	Всі	IE
Опис U ₁₇ : журнали запитів веб-сервера зазвичай містять параметри запитів.			Наслідки: компрометація коду авторизації. Потенційний доступ до захищених ресурсів.	
Існуючі міри захисту: M ₈ - код авторизації дійсний лише для одного використання. M ₉ - код авторизації має короткий час життя.				
Код авторизації	Історія браузера	Потік коду авторизації	Веб-застосунки	IE

Опис Y_{18} : інший користувач того ж самого веб-браузера може мати можливість для перегляду URL-адресу, які відвідували попередні користувачі.			Наслідки: компрометація коду авторизації. Потенційний доступ до захищених ресурсів.	
Існуючі міри захисту: E_6 - клієнт, оновлює сторінку, для очистки параметрів в історії користувацького агента.				
Код авторизації	Сервер авторизації вразливий до SQL-ін'єкцій	Потік коду авторизації	Всі	IE
Опис Y_{19} : зловмисник може отримати коди авторизації з БД сервера авторизації запустивши напад типу SQL ін'єкції.			Наслідки: розкриття всіх кодів авторизації, з відповідними значеннями "redirect_uri" та "client_id".	
Існуючі міри захисту: E_4 - валідація вхідних даних.				
Маркер доступу/оновлення	Низька ентропія та довжина коду авторизації	Потік коду авторизації	Всі	IE
Опис Y_{20} : зловмисник може вгадати дійсне значення коду авторизації та надіслати код для того, щоб отримати дійсний маркер доступу.			Наслідки: компрометація коду авторизації.Потенційний доступ до захищених ресурсів.	
Існуючі міри захисту: M_7 - вимагається висока ентропія конф. параметрів.				
Код авторизації	Користувач	Потік коду авторизації	Веб	SIE
Опис Y_{21} : шкідливий клієнт може видавати себе за дійсного та отримати доступ до коду авторизації.			Наслідки: компрометація коду авторизації. Потенційний доступ до захищених ресурсів.	
Існуючі міри захисту: M_6 - TLS (автентифікація за сертифікатом).				
Згода авторизації (O_5)	Користувач; прозорі IFrame компоненти в HTML док.	Потік коду авторизації	Веб-застосунки	IE

Опис Y_{22} : при натисканні, зловмисний сайт завантажує прозорий iFrame, накладений поверх необхідної кнопки. Коли користувач натискає видиму кнопку він фактично натискає кнопку на прихованій сторінці.			Наслідки: несанкціонована авторизація. Отримання доступу до захищених ресурсів.	
Існуючі міри захисту: E_7 - використання заголовку X-FRAME-OPTION.				
Асоціація з ресурсами жертви (O_5)	Відсутність асоціації між КА та ВР, зі сторони клієнта	Потік коду авторизації	Веб	I
Опис Y_{23} : хакер отримує код для свого облікового запису від СА після дозволу на доступ до ресурсів, але сам не продовжує потік. Він змушує користувача перейти по посиланню з кодом, що належить аккаунту зловмисника. Клієнт отримує перенаправлення з кодом, отримує токен від сервера авторизації та пов'язує сеанс клієнта жертви з ресурсами, доступними за допомогою маркера.			Наслідки: Користувач отримує доступ до ресурсів від імені зловмисника. Вплив залежить від типу доступного ресурсу.	
Існуючі міри захисту: унеможливень або послаблень не пропонується.				
Логічні ресурси СА (мн-на кодів авторизації)	Відсутність обмежень на коди або маркери доступу для користувача	Потік коду авторизації	Всі	D
Опис Y_{24} : зловмисник може вичерпати пул кодів авторизації шляхом множинних запитів авторизації.			Наслідки: неможливість подальшого використання потоку коду авторизації OAuth.	
Існуючі міри захисту: M_7 - висока ентропія та довжина конф. параметрів.				
Логічні ресурси СА	Доступність сервера авторизації	Потік коду авторизації. Неявний потік	Всі	D

Опис Y_{25} : зловмисник, що володіє бот-мережею, може робити запити з випадковими кодами авторизації, і призвести до концентрації великої кількості з'єднань HTTPS на сервері авторизації.			Наслідки: повна або часткова відмова сервера авторизації.	
Існуючі міри захисту: унеможливлень або послаблень не пропонується.				
Код авторизації	Слабка валідація URI перенаправлення	Потік коду авторизації	Всі	SIE
Опис Y_{26} : зловмисник, використовуючи відкрите перенаправлення на клієнті, здатний передати код авторизації / маркер доступу, до власного шкідливого домену, завдяки неповній валідації URI перенаправлення.			Наслідки: зловмисник отримує доступ до програми та специфічних для користувача даних у програмі. Уособлення користувача.	
Існуючі міри захисту: M_3 - вимагається повна валідація redirect_uri.				
Маркер доступу	Незахищений канал передачі	Неявний потік	Всі	IE
Опис Y_{27} : прослуховування каналу під час передачі токена.			Наслідки: компрометація маркера доступу. Потенційний доступ до захищених ресурсів користувача.	
Існуючі міри захисту: M_6 - TLS.				
Код авторизації/ маркер дост.	Історія браузера	Потік коду авторизації. Неявний потік	Всі	IE
Опис Y_{28} : атакуючий може інфектувати комп'ютер жертви шкідливим ПО, що зберігає історію браузера.			Наслідки: компрометація маркера доступу. Потенційний доступ до захищених ресурсів користувача.	
Існуючі міри захисту: E_2 - покладається на захист ОС.				

Код авторизації/ маркер дост.	Незахищений канал передачі	Потік коду авторизації. Неявний потік	Всі	ІЕ
Опис Y_{29} : Зловмисник змінює запити ініціалізації авторизації таким чином, що клієнт вважає обраним СА одну сутність зловмисну для користувача), а власник ресурсу взаємодіє з іншим сервером авторизації (довіреним).			Наслідки: компрометація маркера доступу/ коду авторизації. Потенційний доступ до захищених ресурсів користувача.	
Існуючі міри захисту: M_6 - TLS.				
Точка доступу: OpenIDConnect клієнт				
Обліковий запис користувача	Відсутність перевірки параметрів "exp" та "iat", що передаються в тілі ID Token.	Потік коду авторизації. Неявний потік	Всі	SE
Опис Y_{30}: отримання старих ID Token (наприклад, скануванням форумів підтримки або веб-пошуку) для подальшого використання.			Наслідки: уособлення користувача.	
Існуючі міри захисту: M_{10} - повна перевірка параметрів ID Token				
Обліковий запис користувача	Відсутність перевірки ЕП після отримання JWT	Потік коду авторизації. Неявний потік	Всі	SIE
Опис Y_{31} : зловмисник, перехватує передачу ID Token, та змінює його зміст. Він також здатний сформував повністю новий ID token використовуючи специфікований формат JWT, використовуючи в якості параметрів відомі дані про жертву.			Наслідки: потенційно, атакуючий отримує доступ до ресурсів під обліковими записами усіх існуючих користувачів на сервері авторизації.	
Існуючі міри захисту: M_{10} - повна перевірка параметрів ID Token				

Обліковий запис користувача	Відсутність перевірки параметру "aud"	Потік коду авторизації. Неявний потік	Всі	SIE
Опис Y_{32} : зловмисник може створити власного клієнта, наприклад, шкідлива служба прогнозу погоди і заманити жертву на вхід. Потім зловмисник отримує ID Token, який призначений для використання в цій службі прогнозу погоди, але повторно використовувати його в іншому клієнті.			Наслідки: зловмисник отримує доступ до програми та специфічних для користувача даних на вразливому клієнті.	
Існуючі міри захисту: M_{10} - повна перевірка параметрів ID Token				
Обліковий запис користувача	Контроль параметру "iss", що передаються в тілі ID Token відсутній або невірний.	Потік коду авторизації. Неявний потік	Всі	SIE
Опис Y_{33} : клієнт додає можливість крос-автентифікації через шкідливий СА . Жертва $ID_V = sub_V: iss_V$, $ID_A = sub_A: iss_A$ - зловмисник. Атакуючий, що контролює СА випускає ID Token зі значенням iss_V .Клієнт вважає що, токен був виданий iss_V [15].			Наслідки: Зловмисник може увійти як довільний кінцевий користувач іншого сервера авторизації	
Існуючі міри захисту: M_{10} - повна перевірка параметрів ID Token.				
Обліковий запис користувача	Відсутність перевірки автентичності ЕП, якою підписано JWT.	Потік коду авторизації. Неявний потік	Всі	SIE
Опис Y_{34} : у випадку, якщо Клієнт не перевіряє надійність ключа використаного для підписання ID Token, зловмисник може маніпулювати ним в рамках неявного потоку та вводити інші облікові дані.			Наслідки: потенційно, атакуючий отримує доступ до ресурсів під обліковими записами усіх існуючих користувачів на сервері авторизації.	

Існуючі міри захисту: M_{10} - повна перевірка параметрів ID Token				
Точка доступу: OpenIDConnect сервер авторизації				
Обліковий запис користувача	Некоректна перевірка запитів зі встановленими значеннями	Потік коду авторизації. Неявний потік	Всі	SIE
Опис Y_{35} : OpenID Connect дає можливість запитувати повернення додаткових параметрів (claims) в тілі ID Token з встановленими значеннями (клієнт сам встановлює значення деяких параметрів ID Token та посилає їх як частину запиту на автентифікацію).Нехай у межах запиту на автентифікацію OpenID Connect, клієнт має можливість запросити окремі параметри щодо кінцевого користувача зі встановленими ним значеннями. Жертва: $ID_V = sub_V: iss$, зловмисник $ID_A = sub_A: iss$. Якщо параметр "sub" вимагається з певним значенням для ID Token, сервер авторизації повинен надіслати позитивну відповідь лише в тому випадку, якщо кінцевий користувач визначений цим "sub" має активний сеанс з сервером авторизації або успішно автентифікувати.Однак, зловмисник додає параметр "sub" із значенням {”id_token”: {”Sub”: {”value”: sub _V }}}} до запиту на автентифікацію та виконує вхід в акаунт жертви за власними обліковими даними [15].			Наслідки: зловмисник отримує доступ всіх облікових записів даного СА.	
Існуючі міри захисту: M_{11} - перевірка відповідності встановлених значень				

Як демонструє таблиця та графічне представлення (Рис.2.4), багато проблем, зазначених у моделі загроз, можна унеможливити або пом'якшити шляхом вбудованих в протокол офіційних рекомендації. Однак немає достатньої захищеності від загрози Y_{23} - асоціація з акаунтом жертви. В якості захисту

потрібно впроваджувати деяку прив'язку між параметром state та сеансу користувача у клієнтському застосунку. Протокол також є вразливим до загрози Y_{25} - DoS-атака, що спрямована сервері ресурсів, але запобігання відмови в обслуговуванні не є головним акцентом в цьому випадку. Підсумовуючи, OAuth 2.0 демонструє базовий рівень захисту при правильній реалізації, проте у багатьох випадках покладається на зовнішні засоби захисту та концентрується на контролі захищених ресурсів, засобами автентифікації та маркерами доступу. Безпеці самих маркерів доступу, кодів авторизації, “секретів” клієнта приділяється менша увага.

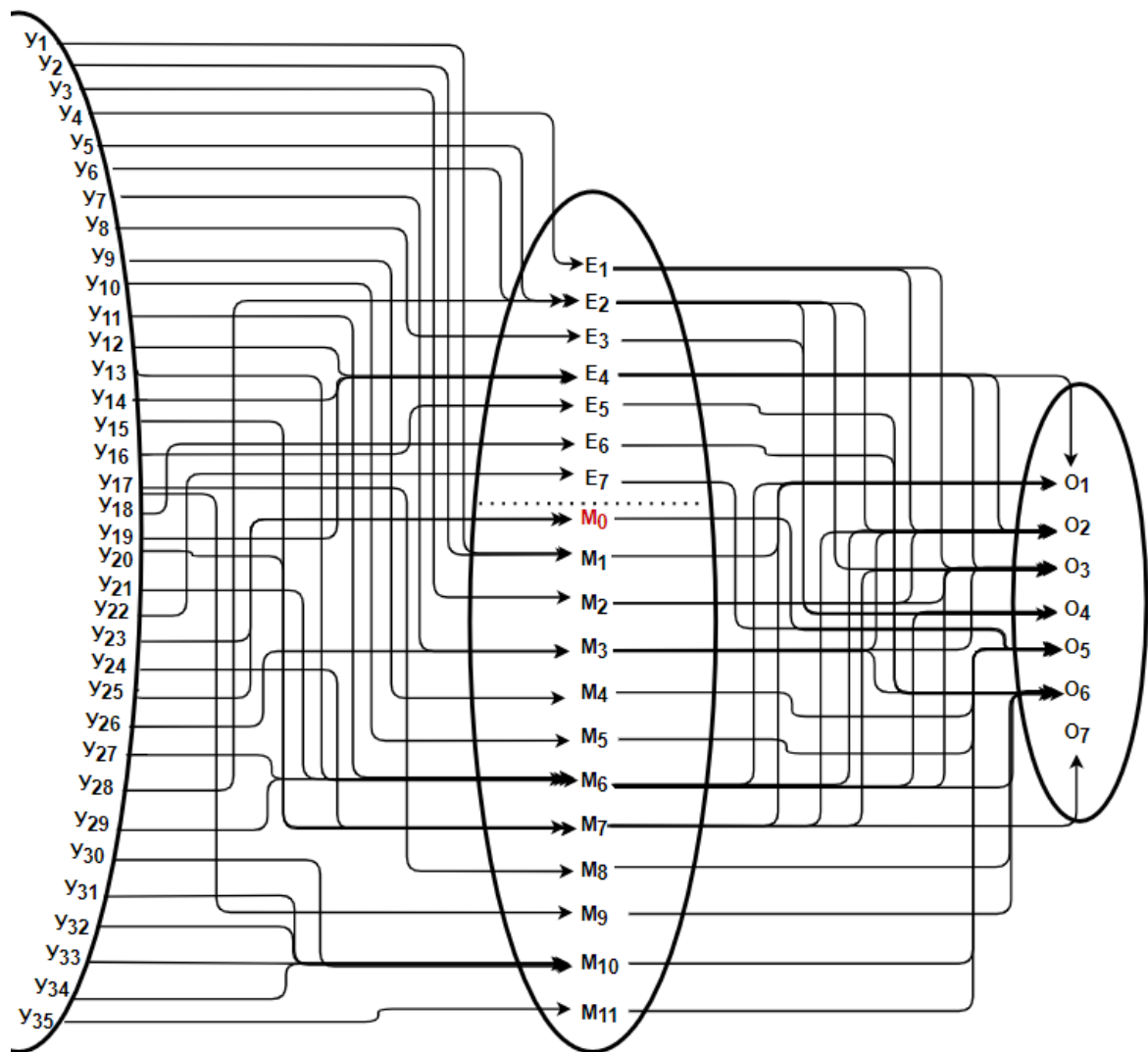


Рисунок 2.4 - Модель з повним перекриттям у вигляді тридольного графу

2.4 Атаки на OAuth 2.0

Атака перенаправлення 307.

Дана атака, порушує властивості авторизації, зломисник (запустивши вразливий клієнт) отримує облікові дані користувача, коли той авторизує ресурси або входить до системи сервера авторизації, який використовує неправильний статус код перенаправлення HTTP. Сама атака ґрунтується на простій помилці реалізації.

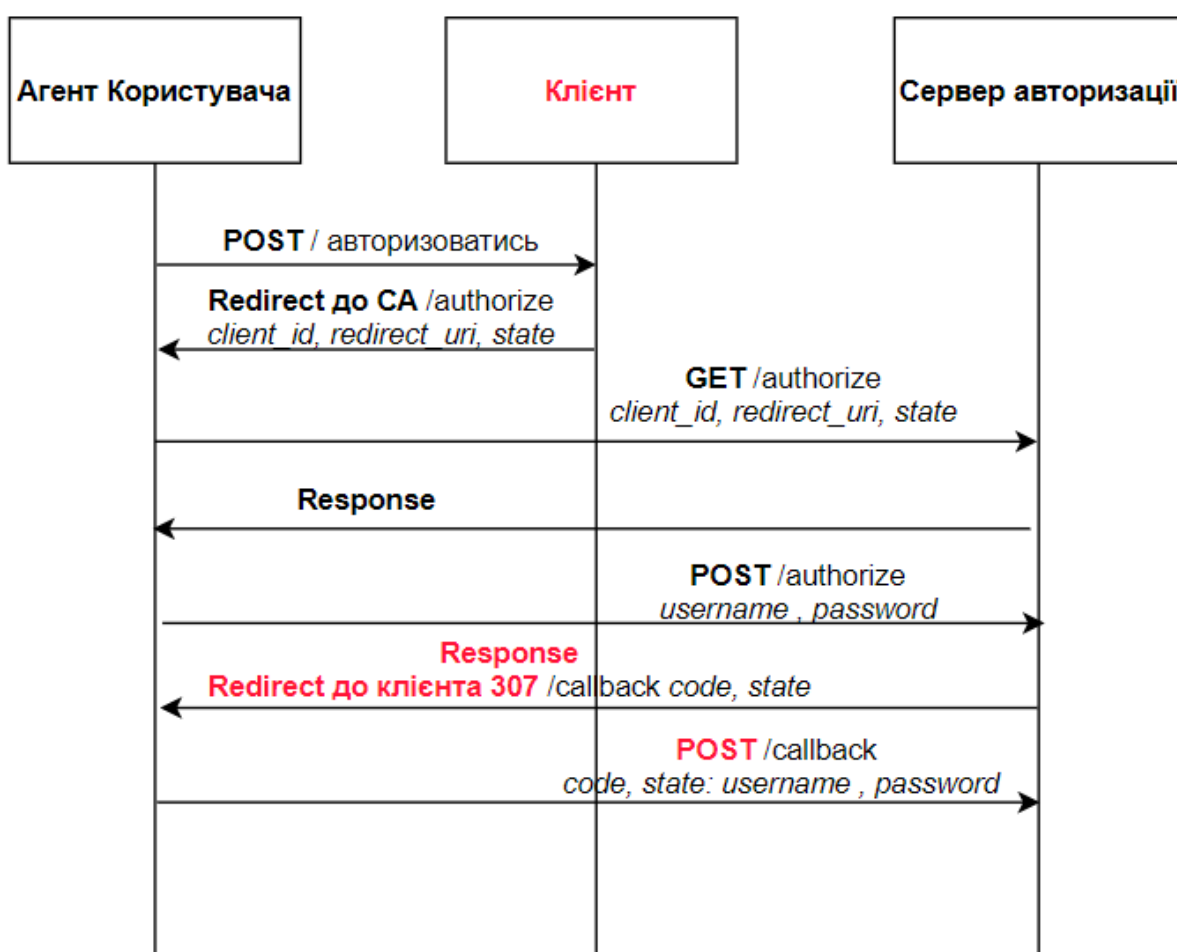


Рисунок 2.5 - Схема атаки перенаправлення 307.

Припущення:

- 1) Сервер авторизації, який використовується для входу, використовує 307

містить посилання на веб-сайт зломисника або на ресурс, розташований на веб-сайті зломисника. Коли користувач натискає посилання або сторінка завантажується, браузер користувача надсилає запит. Цей запит містить заголовок `Referer` із повним URI сторінки на який було перенаправлено користувача, що в цьому випадку містить `state` та `code`. Оскільки значення `state` мало захищати сеанс від атак CSRF, зломисник тепер може використовувати отримані дані здійснення атаки. Наприклад, він може перенаправити браузер жертви на кінцеву точку перенаправлення клієнта (знову) і цим переписати

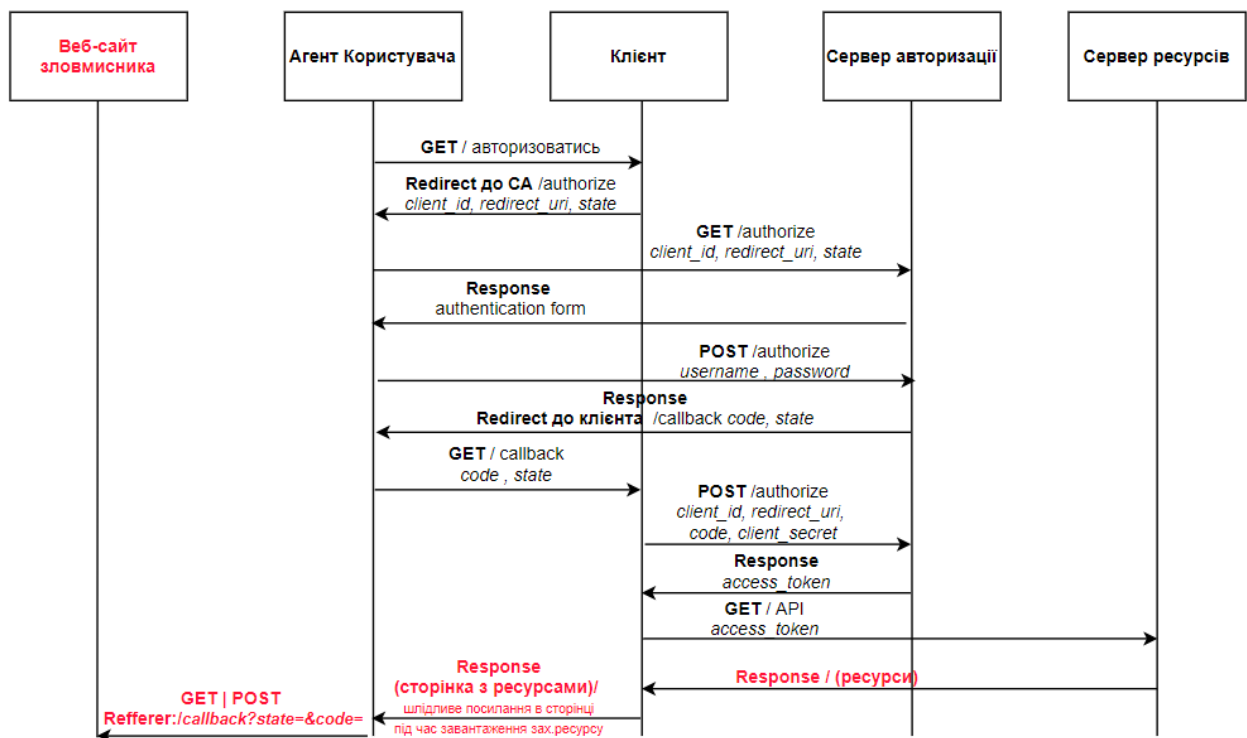


Рисунок 2.9 - Схема атаки витоку коду з заголовку `Referer`

попередньо виконану авторизацію (використовуючи же свій код авторизації і значення `state` жертви). Користувач буде використовувати ресурси атакуючого. Код авторизації призначений лише для одноразового використання і він вже є застарілим до моменту отримання зломисником (хоча Facebook та Slack мали вразливість подвійного використання коду авторизації). Однак `state` часто не обмежується одноразовим використанням, що полегшує реалізацію цієї атаки на практиці [9].

Рішення. Обмеження state та code одноразовим використанням та використовувати referrer-policies, щоб уникнути витоку коду авторизації ЗЛОВМИСНИКУ.

Реалізація:

- 1) Агент користувача отримує перенаправлення до клієнта.

```
GET /authorization-code/callback?code=ZwcH9JUEKXVF64sUrrI01DTe0y4I0UpI71k7uTL6VtU&state=eyJjc3JmX3Rva2VuIjojInUzNjZOMTh4VjBHZy16WkM3ZXdsADNIX2N1dkQ1WG5UIiwgImRlc3RpbmF0aW9uIjojImV5S2h1R2NpT21KSXV6STFOaUo5Lk1taDBkSEET2Hk5c2IyTmhiR2h2YzNRNk9EQTRmZlZyJkGjPpS5S5wX01GQzZkQ2hGM2V1SjklOVoyX3BaZWp3VUZXSXFM1VZLXZ0WHhFVHQ4In0%3D HTTP/1.1
Host: localhost:8080
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: navigate
Sec-Fetch-Dest: document
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90"
sec-ch-ua-mobile: ?0
Accept-Encoding: gzip, deflate
Accept-Language: uk-UA,uk;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: session=eyJvaW9uIjojInUzNjZOMTh4VjBHZy16WkM3ZXdsADNIX2N1dkQ1WG5UIiwgImRlc3RpbmF0aW9uIjojImV5S2h1R2NpT21KSXV6STFOaUo5Lk1taDBkSEET2Hk5c2IyTmhiR2h2YzNRNk9EQTRmZlZyJkGjPpS5S5wX01GQzZkQ2hGM2V1SjklOVoyX3BaZWp3VUZXSXFM1VZLXZ0WHhFVHQ4In0%3D
Connection: close
```

Рисунок 2.10 - Відправка коду авторизації клієнту.

- 2) Відповіддю до запиту №1, є сторінка зі шкідливим запитом.

```
<body id="samples">
<nav class="navbar navbar-default">
  <div class="container-fluid">
    <ul class="nav navbar-nav">
      <li>

<a href="/">Home</a></li>
```

Рисунок 2.11 - Шкідливий запит у відповіді на запит рис. 2.10

- 3) Шкідливий запит має значення Referer зі значеннями code та state.

```
Request URL: http://localhost/evil.png
Request Method: GET
Status Code: 404 Not Found
Remote Address: 127.0.0.1:8081
Referrer Policy: unsafe-url

Response Headers
View source
Connection: close
Content-Length: 295
Content-Type: text/html; charset=iso-8859-1
Date: Tue, 11 May 2021 11:44:41 GMT
Server: Apache/2.4.47 (Win64) OpenSSL/1.1.1k PHP/8.0.5

Request Headers
View source
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: uk-UA,uk;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: session=eyJvaW9uIjojInUzNjZOMTh4VjBHZy16WkM3ZXdsADNIX2N1dkQ1WG5UIiwgImRlc3RpbmF0aW9uIjojImV5S2h1R2NpT21KSXV6STFOaUo5Lk1taDBkSEET2Hk5c2IyTmhiR2h2YzNRNk9EQTRmZlZyJkGjPpS5S5wX01GQzZkQ2hGM2V1SjklOVoyX3BaZWp3VUZXSXFM1VZLXZ0WHhFVHQ4In0%3D
Host: localhost
Proxy-Connection: keep-alive
Referer: http://localhost/authorization-code/callback?code=ZwcH9JUEKXVF64sUrrI01DTe0y4I0UpI71k7uTL6VtU&state=eyJjc3JmX3Rva2VuIjojInUzNjZOMTh4VjBHZy16WkM3ZXdsADNIX2N1dkQ1WG5UIiwgImRlc3RpbmF0aW9uIjojImV5S2h1R2NpT21KSXV6STFOaUo5Lk1taDBkSEET2Hk5c2IyTmhiR2h2YzNRNk9EQTRmZlZyJkGjPpS5S5wX01GQzZkQ2hGM2V1SjklOVoyX3BaZWp3VUZXSXFM1VZLXZ0WHhFVHQ4In0%3D
```

Рисунок 2.12. Виток коду авторизації з заголовку Referrer

Атака підміни сервера авторизації.

Веде до порушення авторизації, зловмисник заплутує клієнта, який СА користувач вибрав на початку процесу входу / авторизації для того, щоб отримати код авторизації або маркер доступу, за допомогою якого можна видавати себе за користувача або отримати доступ до даних користувачів. Реалізація існує для обох потоків. Для запуску атаки зловмисник маніпулює першим запитом користувача таким чином, що клієнт вважає, що власник ресурсу хоче використовувати СА зловмисника. В результаті, надсилає зловмиснику код авторизації або маркер доступу, виданий легітимним сервером авторизації.

Припущення:

- 1) Зловмисник присутній в мережі, який може маніпулювати запитами між власником ресурсів та додатком на етапі до встановлення з'єднання між власником ресурсу та сервером авторизації. Припущення є доцільним оскільки OAuth 2.0 потребує наявності TLS лише при передачі конфіденційних даних (до встановлення з'єднання з СА вони не передаються). Зазначення потребує те, що атака спрацює навіть при початковому використанні HTTPS між клієнтом і користувачем (треба заздалегідь реалізувати атаку SSLstrip), проте в цьому випадку атакуючий отримає код авторизації ще до того як той потрапить до застосунку, підслуховуючи з'єднання [10].
- 2) Клієнт дозволяє користувачам входити в систему із ідентифікаційними даними, наданими деякими СА (у тому числі за ідентифікаційними даними зловмисного сервера авторизації).
- 3) Клієнт видає той самий URI-перенаправлення для всіх СА.

Атака на режим коду авторизації.

Як у звичайному потоці, все починається, коли користувач вибирає, що він бажає увійти за допомогою легітимного СА (етап спуфінгу СА не описується) . Тепер, зловмисник перехоплює запит, призначений для клієнта, і змінює вміст

цього запиту, замінюючи СА на свій. Відповідь клієнта (що містить перенаправлення на зловмисний сервер) знову перехоплюється і модифікуються таким чином, що він перенаправляє користувача на дійсний СА. Зловмисник також замінює `client_id` застосунка на зловмисному сервері ідентифікатором клієнта легітимного сервера авторизації (що є загальнодоступною інформацією). Користувач аутентифікується і перенаправляється до клієнта. На цій стадії застосунок вважає, що `code`, який міститься в перенаправленні, був виданий довіреним СА, а не шкідливим, і намагається отримати маркер доступу у зловмисного сервера авторизації. Код авторизації отримується зловмисником.

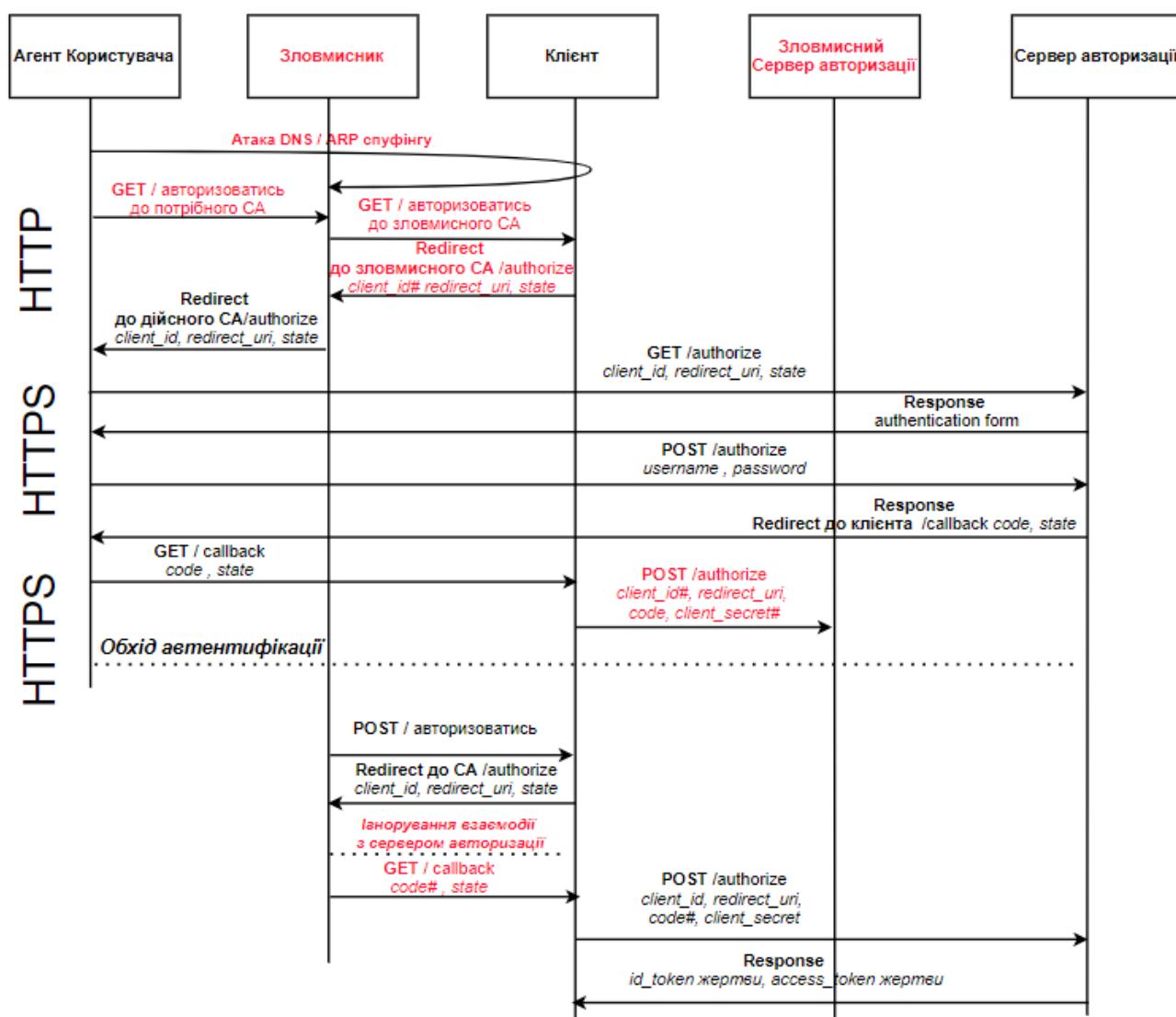


Рисунок 2.13 - Схема атаки підміни сервера авторизації

Розширення атаки. Обхід аутентифікації. Зловмисник, отримавши код, запускає новий процес входу. Він використовує справжній сервер авторизації, ігноруючи перенаправлення до нього. Тепер атакуючий відправляє code (отримання зазначено вище) клієнту, імітуючи реальний вхід (використовуючи значення state від попереднього запиту). Потім застосунок отримує маркер доступу або ID Token чесного користувача. Атакуючий автентифікується під чужою особистістю.

Рішення. Основною проблемою коду авторизації та неявного потоку протоколу OAuth є відсутність достовірної інформації у перенаправленні. Клієнт не отримує інформацію звідки було ініційовано перенаправлення. Отже, до не може перевірити автентичність сервера авторизації. Рішенням є включення ідентифікатора CA до URI перенаправлення у певній формі, яка не може бути під впливом зловмисника. Тобто існує потреба в покращенні специфікації щодо автентифікації суб'єкта ініціалізації перенаправлення.

Реалізація. Нехай, сервер авторизації Okta є довіреним, а CA Github виконує роль зловмисного.

- 1) Користувач, відправляє запит ініціалізації авторизації дійсним СА.

```
GET /login/okta? HTTP/1.1
Host: localhost:9090
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90"
sec-ch-ua-mobile: ?0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:9090/
```

Рисунок 2.14. Атака підміни СА. Етап 1.

- 2) Зловмисник, що знаходиться посередині, змінює запит таким чином, що тепер вміст вказує на використання шкідливого сервера авторизації.
- 3) Клієнт вважає, що власник ресурсу бажає використовувати Github, та повертає перенаправлення до СА.
- 4) Зловмисник, знову змінює запит, та повертає на агент користувача, Redirect 302, до сервера авторизації Okta.

```
GET /login/github? HTTP/1.1
Host: localhost:9090
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90"
sec-ch-ua-mobile: ?0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:9090/
Accept-Encoding: gzip, deflate
Accept-Language: uk-UA,uk;q=0.9,en-US;q=0.8,en;q=0.7
```

Рисунок 2.15. Атака підміни СА. Етап 2.

```
HTTP/2 302 Found
Date: Wed, 02 Jun 2021 05:48:13 GMT
Content-Length: 0
Location:
https://github.com/login/oauth/authorize?state=SsJ4MUIkWDlvMGxiGHJaakV0d3M0eFZUdKpPeWchdkFacldeEWdPeW8vrcVud0RISnJ5VaVMTS9samlYB115TQcclient_id=beba61f37fa6580b4841&redirect_uri=https%3A%2F%2Fokta-devok12.okta.com%2Foauth2%2Fv1%2Fauthorize%2Fcallback&response_type=code&scope=user
Server: nginx
Public-Key-Pins-Report-Only: pin-sha256="r5EfnZxQVvQpKo3AgYRaT7X2bD0/kj3ACvmaxfdT2st8="; pin-sha256="MaqlcUgh2mvY/EFSGeSwBRkI+rZ6/dxe/DuQfBT/vnQ="; pin-sha256="7CG5IEvDWN+8Thf3qjR7/bQSWaS22SLqolhn06iyJI="; pin-sha256="rrV6CLCCvqmk89gWibYT0J06HNQ8cClt7CGGoiVTjC0g="; max-age=60; report-uri="https://okta.a.report-uri.com/r/default/bkpr/reportOnly"
X-Okta-Request-Id: YLchnUYC5eyI2-hKgv8aXvAADv0
X-Xss-Protection: 0
P3p: CP="HONK"
```

Рисунок 2.16. Атака підміни СА. Етап 3.

```
HTTP/1.0 302 FOUND
Content-Type: text/html; charset=utf-8
Content-Length: 1186
Location: https://dev-48793024.okta.com/oauth2/default/v1/authorize?client_id=0oas88qa3VJxHHSNq5d6&redirect_uri=http%3A%2F%2Flocalhost%3A9090%2Fauthorize-code%2Fcallback
Set-Cookie: oicd_id_token=; Expires=Thu, 01 Jan 1970 00:00:00 GMT; HttpOnly; Path=/
Vary: Cookie
Server: Werkzeug/2.0.0 Python/3.7.2
Date: Wed, 02 Jun 2021 05:42:56 GMT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>
Redirecting...
</title>
<h1>
Redirecting...
</h1>
<p>
You should be redirected automatically to target URL: <a href="https://dev-48793024.okta.com/oauth2/default/v1/authorize?client_id=0oas88qa3VJxHHSNq5d6&redirect_uri=https%3A%2F%2Flocalhost%3A9090%2Fauthorize-code%2Fcallback">https://dev-48793024.okta.com/oauth2/default/v1/authorize?client_id=0oas88qa3VJxHHSNq5d6&redirect_uri=http%3A%2F%2Flocalhost%3A9090%2Fauthorize-code%2Fcallback</a>
```

Рисунок 2.17. Атака підміни СА. Етап 4.

- 5) Клієнт вважає, що користувач дає доступ до захищених ресурсів Github, а власник ресурсу впевнений, що асоціюється з довіреним для нього СА (Okta).

Атака асоціації з ресурсами жертви

Порушення авторизації. Зловмисник заплутує клієнта, шляхом підміни власника ресурсу. Вплив такої атаки залежить від типу ресурсу. Наприклад, користувач може завантажувати приватні дані в застосунок, думаючи, що завантажує інформацію у свій власний профіль на цьому клієнті. Реалізація можлива для обох потоків авторизації.

Припущення:

- 1) Клієнт не контролює, чи дійсно отримане значення state асоціюється з ідентифікатором HTTP сесії у запиті передачі коду авторизації.

Атака. Зловмисник спочатку отримує код або маркер доступу, що стосується власних захищених ресурсів. Скасувавши потік на власному пристрої та зберевши значення state, code (або access_token), атакуючий шляхом розміщення URI перенаправлення до клієнта з власними даними, обманює жертву виконати перехід на шкідливу сторінку. Далі потік продовжується без змін. В результаті користувач-жертва отримує доступ до ресурсів від імені зловмисника.

Рішення. Параметр state не допоможе в даній атаці, оскільки для зловмисника не є перешкодою вбудувати значення в URI перенаправлення, обійшовши перевірку на клієнті. Захист CSRF також не ефективним оскільки усі запити з агенту користувача є типу GET. Зниження часу дії коду авторизації, дозволить значно ускладнити можливість такого нападу, оскільки код авторизації зловмисника вже може бути не актуальним. Рішення також може бути досягнуто на стороні клієнта: генерування параметру state на базі ідентифікатора сеансу (cookie) між застосунком та власником ресурсу та передача його хешованої форми. Таким чином, клієнт при отриманні перенаправлення від СА з відповідними параметрами, здатен перевірити значення ідентифікатора сеансу з хешем, що передається в state [11].

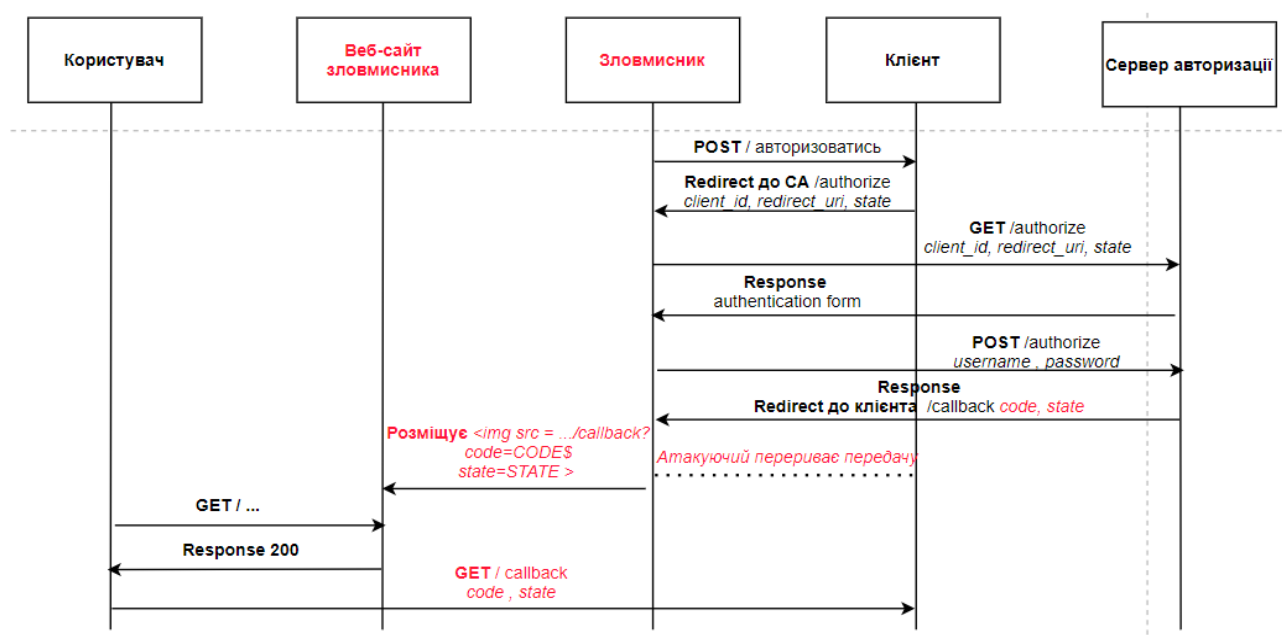


Рисунок 2.18 - Схема атаки асоціації з ресурсами жертви

Реалізація:

- 1) Атакуючий, автентифікується на сервері авторизації.
- 2) Перенаправлення до клієнта з браузера зловмисника зберігається та розміщується на шкідливому сайті.

```
GET /authorization-code/callback?code=3xgQ0tEW7NxDFc5n4NM-kaZreJ-Fs3dAVBpPUYYMM0I&state=eyJjc3JmX3Rva2VuijogIkVma0TV4cHQ0TkdMUGFxd2R2cHpEek91TjQhUGlR0FZ0IiwgImRlc3RpbmF0aW9uIjogImV5S2hiR2NpT21KSUV6STF0aUo5LkltadBkSEETThk5c2IyTaWQ9STTW1lSkdZUnNln0%3D HTTP/1.1
Host: localhost:9090
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: navigate
Sec-Fetch-Dest: document
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90"
sec-ch-ua-mobile: ?0
Accept-Encoding: gzip, deflate
Accept-Language: uk-UA,uk;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: session=eyJvaWRjX2NzcmZfdG9rZW4iOiJFZjkleHBONE5HTFBhcXdkWXB6RHpPdU1zYVBpRDhWTiJ9.E5f5oQ.agMcI3L9Qq_tvHskWhRPvtMep18
Connection: close
```

Рисунок 2.19. - Атака асоціації. Етап 1.

```
POST /api/v1/authn HTTP/2
Host: dev-5982310.okta.com
Cookie: DT=DIOZJxtc7hwS9GadjamciB-Cw; t=default; JSESSIONID=866215D885CC713D8B8R0R0EB07B3232; oktaStateToken=00Cb4hRjtIw7lE1l_Mu3SLdt6P33T91US0t0eVqyxj
Content-Length: 192
Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="90"
Accept: application/json
X-Okta-User-Agent-Extended: okta-sigin-widgit-5.5.4
Accept-Language: uk
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36
Content-Type: application/json
Origin: https://dev-5982310.okta.com
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate
Connection: close

{
  "password": "Attaker1",
  "username": "korol5156@gmail.com",
  "options": {
    "warnBeforePasswordExpired": true,
    "multiOptionalFactorEnroll": true
  },
  "stateToken": "00Cb4hRjtIw7lE1l_Mu3SLdt6P33T91US0t0eVqyxj"
}
```

Рисунок 2.20 - Атака асоціації. Етап 2.

- 3) Користувач відвідує зловмисний сайт, на якому знаходиться запит до клієнта зі значенням state та code.

116	http://localhost	GET	/applications.html		200	42
117	http://localhost:9090	GET	/authorization-code/callback?co...	✓	302	12
118	http://localhost:9090	GET	/login		302	42

Request

Response

Pretty

Raw

Render

\n

Actions

```

18 </head>
19 <body>
20  10, Entropy = 5.174535214100417:: PASSED!
127.0.0.1:59691: GET https://okta-devok12.okta.com/oauth2/v1/authorize/cal
<< 302 0b
127.0.0.1:59696: clientconnect
Code double usage check:: PASSED!
Error! Another Flow type is available for usage!
Redirect Uri fuzzing Check:: PASSED!
```

Рисунок 3.1 Перевірка якості OAuth 2.0. CA - Github, Клієнт - Okta.

```
Evaluating Auth Flow ...
Flow Type: Auth Grant: PASSED!
TLS checking : PASSED!
State presence: PASSED!
307 Redirect Attack checking: PASSED!
Normal flow control: PASSED!
Referer policy presence: PASSED!
Open Redirect attack checking: PASSED!
Quality check of code Length > 10, Entropy = 5.192646647654387:: PASSED!
Quality check of state Length > 10, Entropy = 5.107390591881611:: PASSED!
127.0.0.1:55348: GET https://okta-devok12.okta.com/oauth2/v1/authorize/cal1ba
&code=4%2F0AY0e-g6gSXRqChsIuwZCTLOqmjqwDQSFgu4ZZyGwwmsiseS7ea0VYuxowvKv95vKv
s%3A%2F%2Fwww.googleapis.com%2Fauth%2Fuserinfo.email&authuser=0&prompt=none
<< 302 0b
127.0.0.1:55353: clientconnect
Code double usage check:: PASSED!
Error! Another Flow type is available for usage!
Redirect Uri fuzzing Check:: PASSED!
```

Рисунок 3.2. Перевірка якості OAuth 2.0. CA - Google, Клієнт - Okta.

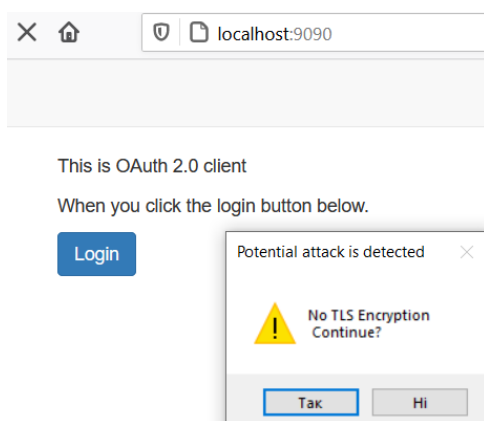


Рис 3.3 - Зупинка OAuth. HTTP.

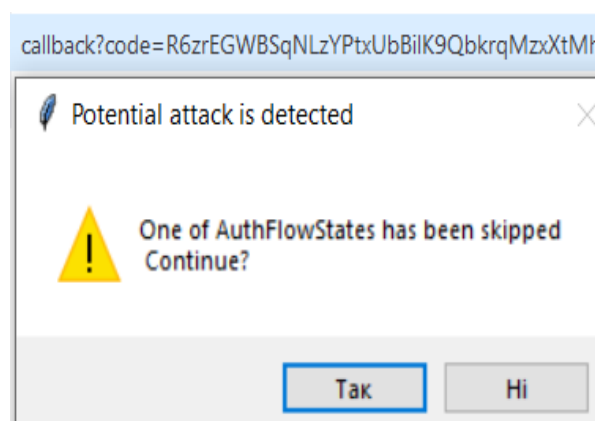


Рис 3.4 - Порушення стадії потоку

Висновки до розділу 3

В даному розділі запропоновано практичні рекомендації щодо реалізації клієнта і сервера авторизації та CP OAuth 2.0, ціль яких полягає в примушенні розробників слідувати існуючим рекомендаціям безпеки та звуженні варіативності специфікації. Запропоновано варіант посилення поточної схеми крос-авторизації спираючись на унеможливлення поверхні атаки з розділу №2 та виявленні недоліків архітектури. Розроблено застосунок, що оцінює процес крос-авторизації OAuth 2.0, який відбувається в поточний момент часу в агенті користувача, на предмет виявлення нападу та відомих недоліків реалізації.

ВИСНОВКИ

OAuth 2.0 є гнучким протоколом крос-авторизації, що легко реалізується розробниками клієнтів та серверів авторизації. В рамках роботи розглянуто повний цикл процесу крос-авторизації OAuth 2.0, який має дві основні альтернативи: потік коду авторизації та неявний потік. Дослідивши функціональні можливості, рівень захищеності та сучасні варіації впровадження, було зроблено висновки щодо придатності OAuth 2.0, як протоколу крос-авторизації, що відповідає високому рівню захисту.

Аналіз безпеки показує, що протокол є надто простим, щоб бути повністю захищеним. Дослідження автоматної моделі продемонструвало, що схема авторизації OAuth 2.0 не є достатньо суворою та має недоліки архітектури, які потребують уваги розробників. Модель загроз, що була побудована в рамках аналізу методом повного перекриття, виявила значну кількість потенційних вразливостей протоколу, проте майже всі вони унеможливлюються або послаблюються специфікованих реалізацією з дотриманням всіх рекомендації безпеки щодо розробки. Потребує уваги те, що під час моделювання захищеності робилися лише базові припущення про можливості зловмисника і тому існування навіть несуттєвих вразливостей є ознакою лише базової або навіть слабкої схеми крос-авторизації в OAuth 2.0. В рамках опису та реалізації поверхні атаки можна зробити висновок про відсутність додаткових методів або рівнів захисту протоколу, адже навіть незначні відхилення від специфікації можуть призводити до компрометації процесу авторизації. Крім того, можна спростувати твердження, що потік коду авторизації є значно більш захищеним ніж неявний потік, оскільки домінуюча кількість виявлених загроз та атак стосується саме першої альтернативи OAuth 2.0.

Результати проведеної роботи дають основу для майбутніх досліджень процесів і схем крос-авторизації та можуть бути використані з метою аналізу та створення нових версій OAuth 2.0.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. D. Hardt - Microsoft. Internet Engineering Task Force (IETF) Request for Comments: 6749: RFC6749: The OAuth 2.0 Authorization Framework [Електронний ресурс]. — October 2012. — С. 8,23-30. — № 2070-1721. — Режим доступа: <https://datatracker.ietf.org/doc/html/rfc6749>.
2. N. Sakimura - NRI, J. Bradley - Ping Identity, M. Jones - Microsoft, B. de Medeiros - Google, C. Mortimore - Salesforce: OpenID Connect Core 1.0 incorporating errata set 1 [Електронний ресурс]. — November 8, 2014 — Режим доступа: https://openid.net/specs/openid-connect-core-1_0.html.
3. M. Jones - Microsoft, D. Hardt. Internet Engineering Task Force (IETF) Request for Comments: 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage [Електронний ресурс]. — October 2012. — С. 4-7. — № 2070-1721. Режим доступа: <https://datatracker.ietf.org/doc/html/rfc6750>.
4. Модель угрозы Долева — Яо [Електронний ресурс]. — 3 августа 2014 — Режим доступа: https://ru.wikipedia.org/w/index.php?title=Модель_угрозы_Долева_—_Яо&oldid=64582489.
5. А. В. Черемушкин: КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ: ОСНОВНЫЕ СВОЙСТВА И УЯЗВИМОСТИ. — Ноябрь 2009. — С. 119-120. — Приложение №2.
6. T. Lodderstedt, Ed -Deutsche Telekom AG, M.McGloin - IBM, P. Hunt - Oracle Corporation: OAuth 2.0 Threat Model and Security Considerations [Електронний ресурс]. — January 2013 — С. 15-40. — Режим доступа: <https://datatracker.ietf.org/doc/html/rfc6819>.
7. Н.А. Гайдамакин: ТЕОРЕТИЧЕСКИЕ ОСНОВЫ КОМПЬЮТЕРНОЙ БЕЗОПАСНОСТИ. — 2008. — С. 178-180. — Учебное пособие. Режим доступа: https://elar.urfu.ru/bitstream/10995/1778/5/1335332_schoolbook.pdf.

8. Daniel Fett, Ralf Küsters, Guido Schmitz: A Comprehensive Formal Security Analysis of OAuth 2.0*. — 8 August 2016. — С. 9-15. Режим доступа: <https://arxiv.org/pdf/1601.01229.pdf>.
9. Egor Homakov: How I hacked Github again [Електронний ресурс]. — 7 February 2014. Режим доступа: <http://homakov.blogspot.com/2014/02/how-i-hacked-github-again.html>.
10. T. Lodderstedt, J. Bradley, A. Labunets, D. Fett: Auth 2.0 Security Best Current Practice draft-ietf-oauth-security-topics-18 [Електронний ресурс]. — 12 April 2021 — С. 19-23. — №18. Режим доступа: <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics-18>.
11. Егор Хомяков, Андрей Лабунец: OAUTH 2 — Я УЗНАЮ ТЕБЯ ПО ТОКЕНАМ [Електронний ресурс].// ХАКЕР 08 /175/ — август 2013. — С. 72-74. — №175. Режим доступа: <https://xakep.ru/pdf/xa/175>.
12. San-Tsai Sun, Konstantin Beznosov: The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems [Електронний ресурс]. — 2015. — С. 10-11. Режим доступа: <https://css.csail.mit.edu/6.858/2015/readings/oauth-ss0.pdf>.
13. Теоретичні і прикладні проблеми фізики, математики та інформатики: матеріали ХІХ Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених (13 – 14 травня 2021 р., м. Київ, Україна) / Уклад.: Василенко О. Д., Пономаренко С. М., Бех С. В., Степаненко В. М., Мирошникова І. Ю., Деркач О. Г., Козленко О. В. – Київ : КПІ ім. Ігоря Сікорського, Видавництво «Політехніка», 2021. –347 с.
14. OAuth by Sakurity [Електронний ресурс]. Режим доступа: <https://sakurity.com/oauth>.
15. Vladislav Mladenov Christian Mainka: OpenID Connect Security Considerations [Електронний ресурс]. — January 13, 2017. — С. 18-21. Режим доступа: https://www.nds.ruhr-uni-bochum.de/media/ei/veroeffentlichungen/2017/01/13/OIDCSecurity_1.pdf.

ДОДАТОК А

Виконуваний код застосунку оцінки захищеності OAuth 2.0.

interceptor.py

```
import mitmproxy.http, mitmproxy.ctx as ctx, threading
import re as regexp, concurrent.futures, requests, json
from copy import copy
from urllib.parse import unquote
from math import log as lg
from sys import exit
from random import randint
from tkinter import *
from tkinter.messagebox import askquestion as ASK
from time import sleep
from Colors import bcolors as clr
from urllib.parse import urlparse
import subprocess

def SendRequestHelpingFunc(URI, method = "GET", param = None):
    if method == "GET" :

        Request = requests.get( url = URI )
        return "error" in Request.url.lower() or Request.status_code >= 400

    if not param:

        Request = requests.post( url = URI )
        return "error" in Request.url.lower() or Request.status_code >= 400
```

```
Request = requests.post(url = URI, params = param)
return "error" in Request.url.lower() or Request.status_code >= 400
```

```
def Printout(output):
```

```
    if "warning" in output.lower():
```

```
        print(f"{clr.BOLD}{clr.WARNING}{output}{clr.ENDC}")
```

```
    elif "Critical" in output.lower():
```

```
        print(f"{clr.BOLD}{clr.FAIL}{output}{clr.ENDC}")
```

```
    elif "error" in output.lower():
```

```
        print(f"{clr.BOLD}{clr.RED}{output}{clr.ENDC}")
```

```
    else:
```

```
        print(f"{clr.BOLD}{output}{clr.OKGREEN}: PASSED!{clr.ENDC}")
```

```
class Auth:
```

```
    def __init__(self):
```

```
        self.FlowData, self.Critical = {}, []
```

```
        self.States = { "ClientToCA": False,
```

```
                        "RedirectClient": False,
```

```
                        "CAToClient": False
```

```
        }
```

```
        self.RequestStateSignatures = {
```



```
def ChangeClientTypeCheck(self, Flow):
```

```
    if "code" in Flow["URL"]:
```

```
        URL = Flow["URL"].replace("code", "token")
```

```
    else:
```

```
        URL = Flow["URL"].replace("token", "code")
```

```
    return SendRequestHelpingFunc(URL)
```

```
def AuthFlowCodeDoubleUsageTest(self, Flow):
```

```
    if self.SearchParameter("access_token", Flow):
```

```
        return None
```

```
    URL = Flow["URL"]
```

```
    if Flow["Method"] == "GET":
```

```
        return SendRequestHelpingFunc(URL)
```

```
    else:
```

```
        params = Flow["Query"]
```

```
        return SendRequestHelpingFunc(URL, params)
```

```
def RedirectURIValidationCheck(self, Flow):
```

```

RedUri = self.SearchParameter("redirect_uri",Flow)
DomainPortion = urlparse(RedUri).netloc
results = []

with open("URICheck.json") as json_file:
    read = json.load(json_file)

    with concurrent.futures.ThreadPoolExecutor( max_workers=25 )
as executor:

    UrlCheck = {

        executor.submit(
            SendRequestHelpingFunc ,
            Flow["URL"].replace(DomainPortion,
DomainPortion + part),
                                10):    part    for    part    in
read["URIEndings"]["DomainChanges"]

    }

    PathCheck = {

        executor.submit(
            SendRequestHelpingFunc ,
            Flow["URL"].replace(RedUri, RedUri + part),
                                10):    part    for    part    in
read["URIEndings"]["PathChanges"]

```

```
}
```

```
UrlCheck.update(PathCheck)
```

```
for check in concurrent.futures.as_completed(UrlCheck):
```

```
    try:
```

```
        if not check.result():
```

```
            results.append(URICheck[check])
```

```
    except Exception as exc:
```

```
        print('%r generated an exception: %s' %
```

```
(UrlCheck[check], exc))
```

```
    return results
```

```
def ControlAttackFlow(self, Error, AuthFlow):
```

```
    AuthFlow.intercept()
```

```
    root = Tk().withdraw()
```

```
    while True:
```

```
        res = ASK('Potential attack is detected',
```

```
                  f'{Error}\n Continue?',
```

```
                  icon='warning')
```

```
        if res == 'yes':
```

```
            AuthFlow.resume()
```

```
            root.destroy()
```

break

elif res == 'no':

root.destroy()

exit(Error)

continue

def EstimateParamsQuality(self, value):

Dictionary, Entropy = {}, 0

for i in value:

try:

Dictionary[i] += 1

except Exception:

Dictionary[i] = 1

for key in Dictionary.keys():

*Entropy -= (Dictionary[key] / len(value)) * (lg(Dictionary[key] /
len(value), 2))*

return len(value) >= 10, Entropy

def TLSCheck(self, Data):

Error = "No TLS Encryption"

```

if Data["Signature"] in self.RequestStateSignatures.keys():

    if "http://" in Data["Headers"]["referer"] + Data["URL"]:
        return Error

    elif regexp.match(r"http://",Data["Headers"]["location"]):
        return Error

    return None

def StateFlowControl(self, Data):

    CurrentState = ""

    for Signature in self.States.keys():

        if self.States[Signature] == False:
            CurrentState = Signature
            break

    if Data["Signature"] == CurrentState:
        self.States[CurrentState] = True
        return None

    if not CurrentState:
        return None

    return "One of AuthFlowStates has been skipped"

```

```
def CheckOpenRedirectInUri(self, Data):
```

```
    if "redirect_uri" in Data["Query"].keys():
```

```
        if 'http' in Data["Query"]["redirect_uri"][5:]:
```

```
            return "Open Redirect Attack is detected"
```

```
    return None
```

```
def CheckReffererValue(self, Data):
```

```
    if "referer" in Data["Headers"].keys():
```

```
        if                                regex.search(r'(access_token|code).+',  
Data["Headers"]["referer"]):
```

```
            return "Access_token or code leakage through Refferer  
Header"
```

```
    return None
```

```
def RedirectCodeCheck(self, Data):
```

```
    if Data["Code"] == 307:
```

```
        return "307 Redirect is detected.\n Potential leakage of credentials"
```

```
    return None
```

```

def StateParamCheck(self, Data):
    if "state" in Data["Query"].keys():

        return None
    return "No state parameter!"

```

```

def FormDataForProccessing(self, Signature, AuthFlow):

    data = {}
    data["Signature"] = Signature

    try:
        data["Method"] = AuthFlow.method
        data["URL"] = unquote(AuthFlow.url)
        data["Query"] = { key : value for key, value in
AuthFlow.query.items() }

    except Exception as e:
        data["Code"] = AuthFlow.status_code

    finally:
        data["Headers"] = { key : unquote(value) for key, value in
AuthFlow.headers.items() }

    return data

```

```

def request(self, flow: mitmproxy.http.HTTPFlow) :

```

```

        for flowSignature in self.RequestStateSignatures.keys():
            try:

                if not
                regexp.search(self.RequestStateSignatures[flowSignature]["Method"],
                flow.request.method):

                    raise ValueError(
                        'No Auth Request Signatures have been found'
                    )

                if not
                regexp.search(self.RequestStateSignatures[flowSignature]["Signature"],
                flow.request.url ):

                    raise ValueError(
                        'No Auth Request Signatures have been found'
                    )

                form_data = self.FormDataForProccessing(flowSignature,
                flow.request)

                with concurrent.futures.ThreadPoolExecutor(max_workers=
                5) as executor:

                    self.Critical.append(
                        executor.submit( self.TLSCheck, form_data))

                    self.Critical.append(
                        executor.submit( self.StateFlowControl,
                form_data ))

```

```

        self.Critical.append(
            executor.submit( self.CheckOpenRedirectInUri,
form_data ))

```

```

        self.Critical.append(
            executor.submit( self.CheckReffererValue,
form_data ))

```

```

        self.Critical.append(
            executor.submit( self.StateParamCheck,
form_data))

```

```

        for attack in
concurrent.futures.as_completed(self.Critical):
            result = attack.result()

```

```

            if result is not None:
                self.ControlAttackFlow(result,flow)

```

```

        form_data["Critical"] = [ critical.result() for critical in
self.Critical]

```

```

        self.FlowData[flowSignature], self.Critical = form_data, []

```

```

    except Exception as NoNeededParams:
        continue

```

```

def response(self, flow: mitmproxy.http.HTTPFlow):

```

```

    for flowSignature in self.ResponseStateSignatures.keys():

```

```

        try:

            if not
regexp.search(self.ResponseStateSignatures[flowSignature]["Code"],str(flow.response.
status_code)):

                raise ValueError(
                    'No Auth Response Signatures have been found'
                )

            if not
regexp.search(self.ResponseStateSignatures[flowSignature]["Location"],
flow.response.headers["location"]):

                raise ValueError(
                    'No Auth Respose Signatures have been found'
                )

            form_data = self.FormDataForProccessing(flowSignature,
flow.response)

            with concurrent.futures.ThreadPoolExecutor(max_workers =
4) as executor:

                self.Critical.append(

                    executor.submit( self.TLSCheck, form_data))

                self.Critical.append(

                    executor.submit( self.StateFlowControl,
form_data ))

```

```

        self.Critical.append(
            executor.submit(self.RedirectCodeCheck,
                form_data))

        self.Critical.append(
            executor.submit(        self.CheckReffererValue,
                form_data ))

        for                attack                in
            concurrent.futures.as_completed(self.Critical):

                if attack.result() is not None:

                    self.ControlAttackFlow(attack.result(),flow)

                    form_data["Critical"] = [ critical.result() for critical
                        in self.Critical]

                    self.FlowData[flowSignature],        self.Critical        =
                        form_data, []

            except Exception as NoNeededParams:
                continue

    def EstimateAuthQuality(self):

        while False in self.States.values():
            sleep(1)

    InternalChecks = ["code", "state", "access_token" ]
    Type = lambda x: "Implicit Grant" if x is None else "Auth Grant"

```

```

Flow = Type(self.SearchParameter('code',self.FlowData['CAToClient']))
Printout(f'Evaluating Auth Flow ...\nFlow Type: {Flow}')
if "Auth" in Flow:
    InternalChecks = InternalChecks[:-1]
else:
    InternalChecks = InternalChecks[1:]
Checker = True
for key in self.States.keys():

    for criticals in self.FlowData[key]["Critical"]:

        if criticals is not None:
            Checker = False
            Printout(f'Critical! Current Auth Flow have
suscipicious implementation or was under attack!\nReason:{criticals}')
            if Checker:
                Printout("TLS checking ");Printout("State presence")
                Printout("307 Redirect Attack checking");Printout("Normal flow
control")

                Printout("Referer policy presence");Printout("Open Redirect attack
checking")

for key in InternalChecks:

    param = self.SearchParameter(key, self.FlowData["CAToClient"])
    if param:
        length, Entropy = self.EstimateParamsQuality(param)
        if not length or Entropy < 4:

```

```

        Printout(f"WARNING: {key} has low Length or low
Entropy {Entropy}!")

        continue

    Printout(f"Quality check of {key} Length > 10, Entropy =
{Entropy}:")

    Checker =

self.AuthFlowCodeDoubleUsageTest(self.FlowData["CAToClient"])

    if Checker:

        Printout(f"Code double usage check:")

    elif Checker is False:

        Printout(f"Error! Code value can be used for second time!")

    Checker = self.ChangeClientTypeCheck(self.FlowData["ClientToCA"] )

    if Checker:

        Printout(f"Change Flow type Check:")

    elif Checker is False:

        Printout(f"Error! Another Flow type is available for usage!")

    if not Checker:

        Printout(f"Redirect Uri fuzzing Check:")

    else:

        for fuzz in Checker:

            Printout(f"Error! Redirect URI can be changed with: {fuzz}")

    exit("Exiting...")

addons = [ Auth() ]

```

main.py

from os import system as sys

def main():

try:

sys('mitmdump -s interceptor.py')

except Exception as Error:

raise Error

if __name__ == "__main__":

main()