УДК 004.8
DOI: 10.31673/2409-7292.2025.014631

Дудикевич В.Б., Партика О.О., Наконечний Т.І.

ВПРОВАДЖЕННЯ СИСТЕМ ОДНОРАЗОВОГО ВХОДУ (SSO) ДЛЯ ПІДВИЩЕННЯ КІБЕРБЕЗПЕКИ

У сучасному цифровому середовищі, де безпека користувачів та їхніх даних є пріоритетом, системи одноразового входу (Single Sign-On, SSO) відіграють ключову роль у спрощенні процесу автентифікації та підвищенні рівня кібербезпеки. Дана стаття присвячена дослідженню, розробці та впровадженню централізованої системи авторизації на основі стандартів OAuth2.0 (Open Authorization) та OpenID Connect (OIDC). Автори аналізують сучасні підходи до управління ідентифікацією користувачів, а також розглядають переваги впровадження авторизаційного сервера з підтримкою JSON Web Token (JWT) та механізму PKCE (Proof Key for Code Exchange) для підвищення безпеки клієнтських додатків. Основна увага у статті приділена реалізації сервера автентифікації, який забезпечує централізовану перевірку користувачів та передачу токенів доступу між сервісами. Розглянуто Security Assertion Markup Language (SAML), OAuth2.0 та OIDC як основні стандарти ідентифікації та управління доступом. Автори дослідили переваги та недоліки кожного з підходів, підкреслюючи важливість використання Authorization Code Flow з PKCE для підвищення стійкості до атак типу "підміна клієнта" та "викрадення токенів". Також у роботі детально розглянуто механізми захисту від міжсайтового скриптингу (XSS), атаки на браузерні додатки та безпечне зберігання токенів доступу. Особливу увагу приділено питанням локальної валідації токенів для зменшення навантаження на авторизаційний сервер та підвищення продуктивності системи. Результатом дослідження є функціональна модель SSO, що підтримує OAuth 2.0, OIDC, JWT та забезпечує автентифікацію для публічних і конфіденційних клієнтів. Запропоноване рішення може бути застосоване в корпоративних та комерційних середовищах для захисту ресурсів та спрощення управління доступом. Отримані результати мають як теоретичне, так і практичне значення, оскільки сприяють розвитку безпечних цифрових ідентифікаційних систем та покращенню захисту користувачів в інтернет-середовищі.

Ключові слова: Single Sign-On (SSO), автентифікація, авторизація, кібербезпека, OAuth2.0, OpenID Connect (OIDC), JSON Web Token (JWT), Security Assertion Markup Language (SAML), PKCE (Proof Key for Code Exchange), локальна валідація токенів, міжсайтовий скриптинг (XSS), безпечне зберігання токенів, управління доступом, сервер автентифікації, цифрова ідентифікація.

Вступ

У сучасному цифровому світі, де технології відіграють ключову роль у повсякденному житті, питання автентифікації користувачів стає все більш актуальним. Збільшення кількості онлайн-сервісів вимагає ефективних механізмів ідентифікації, що забезпечують одночасно зручність і безпеку. Одним із таких рішень є Single Sign-On (SSO) – система єдиного входу, яка дозволяє користувачам автентифікуватися лише один раз і отримувати доступ до різних сервісів без повторного введення облікових даних. Це спрощує користувацький досвід, знижує ризики, пов’язані з множинними паролями, та підвищує рівень інформаційної безпеки. Попри переваги, впровадження SSO супроводжується низкою викликів. Однією з головних проблем є централізація автентифікації, яка створює єдину точку уразливості (Single Point of Failure). Уразливість постачальника ідентифікації (Identity Provider, IdP) може призвести до масової компрометації акаунтів та несанкціонованого доступу до даних. Додатковою загрозою є проблема глобального виходу (Single Logout, SLO), оскільки не всі сервіси коректно його підтримують, що може залишати користувачів авторизованими навіть після виходу з системи. Фішингові атаки, крадіжка токенів і перехоплення сесій також залишаються серйозними ризиками. Злам одного акаунта може надати зловмисникам доступ до всіх сервісів користувача. Для зменшення цих загроз у SSO застосовують багатофакторну автентифікацію (MFA) та сучасні протоколи безпеки, зокрема OAuth 2.0, OpenID Connect (OIDC) і FIDO2/WebAuthn.

Не менш важливою є проблема конфіденційності даних у SSO. Деякі сервіси запитують надмірну кількість персональних даних, що може порушувати принцип мінімізації збору інформації та знижувати рівень довіри користувачів. Важливо впроваджувати механізми контролю за передачею даних і забезпечувати можливість керування дозволами. Ще один виклик – продуктивність систем SSO у великих корпоративних середовищах. Велика кількість

запитів до IdP може перевантажувати сервери, уповільнюючи процес входу. Для вирішення цієї проблеми застосовують балансування навантаження, кешування токенів та використання легковагових протоколів, таких як JSON Web Token (JWT). Крім того, багато організацій використовують застарілі системи, що не підтримують сучасні протоколи автентифікації. Це ускладнює інтеграцію SSO, оскільки деякі сервіси потребують окремих облікових записів або ручного введення паролів. Розв'язанням може стати впровадження адаптерів, шлюзів автентифікації або поетапна міграція на сучасні технології. Таким чином, незважаючи на переваги, впровадження SSO супроводжується численними викликами, що потребують комплексного аналізу. Це дослідження спрямоване на вивчення сучасних стандартів управління ідентифікацією, аналіз методів автентифікації та розробку ефективних підходів до впровадження SSO у корпоративних і публічних інформаційних системах.

Формулювання проблеми.

Однією з головних проблем SSO є відсутність ефективного механізму глобального виходу (Single Logout, SLO). У більшості випадків вихід із сервісу через SSO не гарантує автоматичного завершення сесій у всіх підключених додатках, що може призвести до несанкціонованого доступу до даних користувача. Розв'язання цієї проблеми потребує впровадження надійного механізму SLO, який забезпечуватиме автоматичне завершення сесій у всіх сервісах, пов'язаних із IdP. Однак це завдання ускладнюється через різні механізми керування сесіями та рівень сумісності сервісів із протоколами SLO. Найбільш ефективним підходом є використання централізованих реєстрів сесій та одночасне завершення авторизації під час виходу з IdP. Попри високий рівень захисту сучасних протоколів SSO, автентифікаційні механізми залишаються вразливими до атак. Фішингові атаки, підроблені сторінки входу IdP і крадіжка токенів є основними загрозами, що дозволяють зловмисникам отримати несанкціонований доступ до ресурсів користувача. З метою підвищення рівня безпеки необхідно впроваджувати багатофакторну автентифікацію (MFA), що значно знижує ризики компрометації облікового запису. Важливими заходами також є використання токенів із коротким терміном дії, перевірка походження запитів та інтеграція з сучасними стандартами безпеки, такими як FIDO2/WebAuthn, що дозволяють реалізувати безпарольну автентифікацію.

Ще однією проблемою SSO є недостатня прозорість процесу передачі даних між сервісами. Багато користувачів не усвідомлюють, які саме дані передаються під час автентифікації, що може знижувати довіру до системи. Деякі сервіси запитують надлишкові персональні дані, порушуючи принцип мінімізації збору інформації. Для розв'язання цієї проблеми необхідно розробити механізми контролю, що дозволятимуть користувачам керувати своїми даними. Це можуть бути детальні сповіщення про передані атрибути, можливість вибору, які дані надавати сервісам, а також механізми відкликання дозволів. Системи SSO повинні також забезпечувати стабільну роботу навіть у середовищах із високим навантаженням. Велика кількість підключених сервісів може призводити до затримок у процесі автентифікації, особливо якщо сервери IdP перевантажені. Для оптимізації продуктивності застосовують балансування навантаження, кешування автентифікаційних токенів та механізми попередньої автентифікації, що зменшують кількість запитів до серверів авторизації.

Окремої уваги потребує впровадження легковагових протоколів обміну даними, зокрема JSON Web Token (JWT) замість XML-структур, що значно знижує навантаження на сервери. Додатковою складністю є інтеграція SSO із застарілими системами, які не підтримують сучасні протоколи автентифікації, такі як OAuth 2.0 та OIDC. Для забезпечення сумісності необхідно розробити адаптери або шлюзи автентифікації, що виконують трансляцію між старими та новими механізмами. Важливо також застосовувати поетапний підхід до міграції, щоб уникнути втрати доступу до сервісів. Перспективним напрямом є розвиток федеративної

автентифікації (Federated Identity Management), що дозволяє організаціям об'єднувати кілька незалежних систем ідентифікації в єдину екосистему з централізованим керуванням.

Огляд останніх досліджень і публікацій

У сучасному цифровому середовищі зростає потреба в безпечних та ефективних механізмах автентифікації користувачів. Традиційні системи керування обліковими записами часто вимагають введення паролів для кожного сервісу окремо, що створює значне навантаження на користувачів і підвищує ризики компрометації облікових даних [1]. Концепція Single Sign-On (SSO) вирішує цю проблему, дозволяючи користувачеві увійти один раз та отримати доступ до декількох сервісів без повторного введення облікових даних [2].

SSO значно покращує зручність роботи користувачів, знижує кількість паролів, які потрібно запам'ятовувати, і спрощує адміністрування доступу в корпоративних середовищах. Проте впровадження SSO несе й певні виклики, зокрема питання безпеки, продуктивності та конфіденційності [3]. Оскільки централізована автентифікація означає, що компрометація одного облікового запису може призвести до доступу до всіх сервісів, необхідно розробляти ефективні механізми захисту, зокрема багатофакторну автентифікацію (MFA) та захист токенів доступу [4].

Сучасні системи SSO ґрунтуються на кількох основних стандартах:

Security Assertion Markup Language (SAML 2.0) – XML-формат для обміну авторизаційними твердженнями між постачальником ідентичності (IdP) та постачальником послуг (SP) [5].

OAuth 2.0 (Open Authorization) – протокол делегованої авторизації, який дозволяє клієнтам отримувати доступ до ресурсів без передачі паролів [6].

OpenID Connect (OIDC) – розширення OAuth 2.0, яке додає механізм автентифікації користувачів через ID-токени [7].

Кожен із цих стандартів має свої сильні та слабкі сторони, що впливає на вибір конкретної реалізації для різних організацій та додатків. SAML 2.0 широко застосовується в корпоративних середовищах завдяки своїй зрілості та високому рівню безпеки, проте він вважається занадто громіздким для мобільних і веб-додатків через використання XML [8]. OAuth 2.0 є більш легковаговою альтернативою, що дозволяє реалізувати розподілену авторизацію та використовується в багатьох сучасних сервісах, зокрема Google, Facebook та Microsoft [9].

Проте дослідники відзначають, що навіть сучасні SSO-рішення мають ряд викликів, які потребують вирішення. Однією з основних проблем є відсутність ефективного механізму глобального виходу (Single Logout, SLO), що призводить до ризику збереження активних сесій навіть після виходу користувача [10]. Крім того, фішингові атаки та крадіжка токенів доступу залишаються значними загрозами для безпеки SSO [4].

Дослідження також вказують на необхідність балансування продуктивності в умовах великого навантаження на сервери IdP, що може сповільнювати процес автентифікації в масштабованих середовищах [6]. Окрему увагу приділяють інтеграції SSO у застарілі системи (legacy systems), які не підтримують протоколи OAuth 2.0 та OIDC [9].

Таким чином, впровадження SSO вимагає не лише вибору оптимального протоколу, а й врахування питань захисту даних, продуктивності, прозорості авторизації та інтеграції із застарілими системами. Подальші дослідження повинні зосереджуватися на розробці механізмів глобального виходу, удосконаленні механізмів автентифікації без паролів, а також підвищенні рівня захисту користувацьких даних під час входу через SSO.

Мета дослідження

Метою цієї наукової роботи є комплексний аналіз систем управління ідентифікацією та доступом з акцентом на такі ключові технології, як SAML, OAuth та OpenID Connect (OIDC). Робота зосереджується на дослідженні доцільності застосування різних протоколів управління доступом в залежності від специфічних сценаріїв використання. Особливу увагу

приділено розробці власного сервера SSO, що підтримує Authorization Code Flow з механізмом PKCE та локальну валідацію токенів.

Завдання дослідження

Завдання дослідження охоплюють аналіз сучасних стандартів управління доступом, реалізацію функціональної моделі SSO, порівняльний аналіз з існуючими рішеннями, а також визначення кращих практик безпеки для веб-сервісів та додатків. Дане дослідження сприяє розумінню ключових аспектів захисту цифрової ідентичності та має важливе теоретичне та практичне значення для подальшого розвитку інформаційних систем безпеки.

Поняття централізованої системи авторизації.

SSO – це метод контролю доступу, за допомогою якого користувачі можуть безпечно автентифікуватися та отримувати доступ до багатьох ресурсів та сервісів, лише один раз увійшовши в систему з одним іменем користувача та паролем. Таким чином, підхід SSO дозволяє користувачам автентифікуватися лише один раз і потім безпечно отримувати простий доступ до інших додатків. (рис. 1).

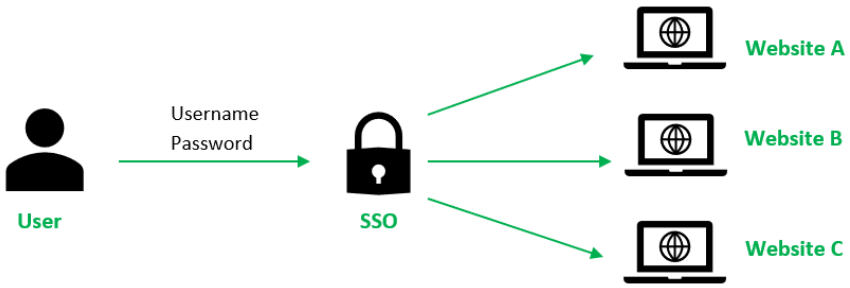

Рис. 1. Схематичне зображення основних компонентів та взаємодій в системі одноразового входу

Без рішення SSO веб-сайт підтримує базу даних облікових даних для входу – ім'я користувача та паролі. Щоразу, коли користувач входить на веб-сайт, він перевіряє облікові дані користувача зі своєю базою даних і автентифікує користувача. За допомогою рішення SSO веб-сайт не зберігає облікові дані для входу у своїй базі даних. Натомість SSO використовує загальний кластер серверів автентифікації, де користувачам потрібно лише один раз ввести свої облікові дані для автентифікації. Кожного разу, коли користувачі переходять до домену, який потребує автентифікації, вони переспрямовуються до домену автентифікації, де їх може попросити ввійти. Якщо користувач уже ввійшов у домен автентифікації, його можна негайно переспрямувати до початкового домену без входу знову.

Єдиний вхід можливий за допомогою механізму сесій. За допомогою SSO центральний домен виконує автентифікацію, а потім ділиться сесією з іншими доменами. Спосіб спільного використання сесій може відрізнятися для різних протоколів SSO, але загальна концепція однакова.

Наприклад, домен автентифікації може генерувати підписаний веб-токен JSON (JWT) (зашифрований за допомогою JSON Web Encryption (JWE)), який містить усю інформацію, необхідну для ідентифікації користувача для будь-якого іншого домену, що вимагає автентифікації. Цей токен передається клієнту, але оскільки він підписаний, клієнт не може його будь-яким чином змінити. Токен може бути переданий до вихідного домену за допомогою перенаправлення та використаний доменом автентифікації та будь-якими іншими доменами для ідентифікації користувача.

Сучасні стандарти управління доступом та обліковими даними

Існує декілька протоколів і стандартів, про які слід знати під час ідентифікації та роботи з SSO. До них належать:

© Дудикевич В.Б., Партика О.О., Наконечний Т.І. (2025). Впровадження систем одноразового входу (SSO) для підвищення кібербезпеки. Сучасний захист інформації, 1(61), 60–73. https://doi.org/10.31673/2409-7292.2025.014631.

- Security Access Markup Language (SAML)
- Open Authorization (OAuth): OAuth
- OpenID Connect (OIDC): OIDC

SAML (Security Assertion Markup Language) представляє собою мову розмітки для передачі облікових даних про авторизацію від постачальників ідентичності (Identity Provider) до постачальників послуг (Service Provider). Іншими словами, це забезпечує безпечний обмін даними між додатками та дозволяє користувачам отримувати доступ з одним набором облікових даних.

Учасники SAML

- Identity providers автентифікують користувачів: ці системи відповідають за підтвердження того, що користувач є тим, за кого себе видають, а потім надсилають ці дані (і права доступу користувача) постачальнику послуг (Service Providers)
- Service providers авторизують користувачів: ці системи використовують дані автентифікації від Identity Provider, щоб надати доступ до ресурсу

Таким чином, SAML є сполучною ланкою між автентифікацією особи користувача та авторизацією на використання сервісу. Це мова, яка допомагає постачальникам ідентифікаційних даних (idPs)і постачальникам послуг (SPs) спілкуватися.

Якщо коротко, протокол описує транзакцію, у якій користувач може ввійти в систему в одному місці, а потім показати підтвердження входу в іншому і отримати доступ.

Програма додає проміжне програмне забезпечення (middleware), яке стоїть між програмою та клієнтом і перехоплює весь трафік перед тим, як запити потраплять до кінцевої точки (endpoint).

Для того, щоб встановити безпечне з’єднання між Identity provider та service provider вводиться поняття довіри (trust). Ми говоримо, що ресурс довіряє identity provider або authority, коли цей ресурс готовий вірити тому, що authority говорить про своїх користувачів. Постачальник ідентичності (Identity Provider, IdP) є авторитетним джерелом, якому ресурс (постачальник послуг, Service Provider, SP) довіряє. Ресурс може вірити IdP, коли останній надає певну ідентифікаційну інформацію про користувачів, яку ресурс перевіряє та використовує для надання доступу до своїх послуг.

Якщо IdP говорить, що певний користувач є його клієнтом і успішно пройшов процес автентифікації, ресурс може повірити цій інформації і надати користувачеві доступ. Довіра полягає в тому, що ресурс вірить відомостям, які надає авторитетний IdP. SAML використовує механізми обміну повідомленнями та підпису для забезпечення цілісності та безпеки обміну даними між IdP та SP. Це дозволяє ресурсу впевнитись у достовірності та актуальності інформації, що надається IdP.

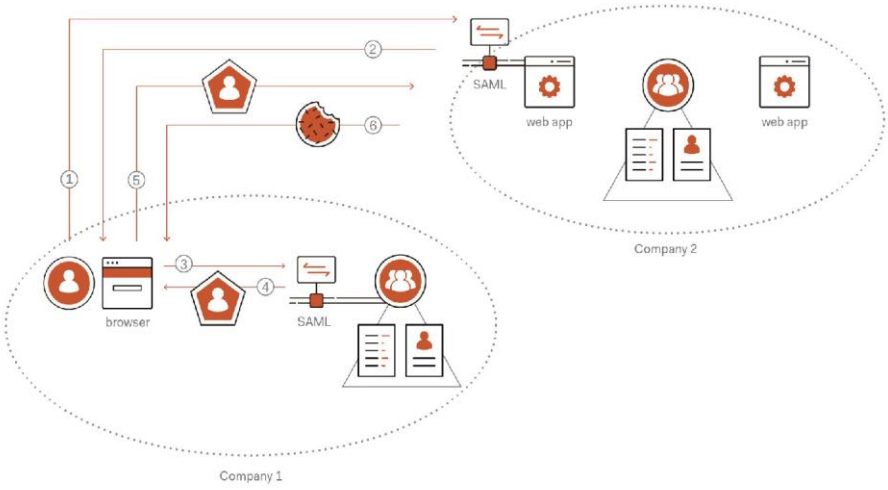

Рис. 2. Опис процесу автентифікації користувача через систему SSO

© Дудикевич В.Б., Партика О.О., Наконечний Т.І. (2025). Впровадження систем одноразового входу (SSO) для підвищення кібербезпеки. Сучасний захист інформації, 1(61), 60–73. https://doi.org/10.31673/2409-7292.2025.014631.

У першому етапі діаграми користувач направляє браузер до додатку та намагається отримати сторінку.[1] Проміжне програмне забезпечення перехоплює запит, бачить, що користувач не автентифікований, і перетворює запит на запит на автентифікацію до постачальника ідентичності (IdP), оскільки він налаштований як один з довірених IdP [2]. На практиці проміжний програмний засіб створює певне повідомлення, ймовірно, URL зі специфічними параметрами рядка запиту(query string), та перенаправляє браузер до певної кінцевої точки на стороні Identity Provider [3].

Далі, постачальник ідентичності перевіряє, що користувач вже правильно пройшов автентифікацію, і переконується, що ресурс є одним з ресурсів, які були зареєстровані та схвалені. Завдяки цим позитивним перевіркам IdP видає користувачу те, що ми називаємо токен безпеки (security token) [4]. Токен безпеки є артефактом, набором бітів, який використовується для передачі конкретного доказу успішної автентифікації користувача. Токени безпеки мають цифровий підпис.

Атрибути, які пересилаються усередині токенів, називаються клеймами (claims). Клейм просто є атрибутом в контексті, що дозволяє одержувачу вирішити, чи вірити, що користувач насправді володіє цим атрибутом. Клейми можуть містити різноманітну інформацію про користувача, таку як ідентифікатор, роль, електронну адресу, атрибути профілю тощо. Приймаючи токен з клеймами, ресурс може використовувати цю інформацію для прийняття рішень щодо доступу користувача до ресурсу. Наприклад, ресурс може перевірити клейм, який містить роль "адміністратор", і надати розширені привілеї цьому користувачу.

Після того, як Identity Provider видає SAML токен, він повертає його до браузера в межах HTML-форми, разом з деяким JavaScript, який виконується при завантаженні сторінки - відправляє токен до додатку за допомогою POST-запиту, де його перехоплює проміжне програмне забезпечення[5]. Проміжне програмне забезпечення аналізує токен, перевіряє, чи він надійшов з довіреного джерела, перевіряє, чи не був порушений підпис. Якщо всі перевірки пройдені успішно, код генерує те, що ми називаємо session cookie [6].

OAuth 2.0, що розшифровується як «Open Authorization», — це стандарт, який дозволяє веб-сайту або програмі отримувати доступ до ресурсів, розміщених іншими веб-програмами, від імені користувача. Він замінив OAuth 1.0 у 2012 році і тепер є фактичним галузевим стандартом для онлайн-авторизації. OAuth 2.0 надає контрольований доступ та обмежує дії, які може виконувати клієнтський додаток від імені користувача, ніколи не передаючи облікові дані користувача.

Хоча веб є основною платформою для OAuth, специфікація також описує, як обробляти цей вид делегованого доступу для інших типів клієнтів (програми на основі веб-переглядача, веб-програми на стороні сервера, мобільних/нативних додатків, підключених пристроїв).

OAuth– це протокол авторизації, а не протокол автентифікації. Таким чином, він призначений для надання доступу до набору ресурсів, наприклад, віддалених API або даних користувача. OAuth використовує токени доступу(Access Token). Токен доступу є частиною даних, яка представляє авторизацію для доступу до ресурсів від імені кінцевого користувача. OAuth не визначає конкретного формату токенів доступу. Однак, у деяких контекстах часто використовується формат JSON Web Token (JWT). Це дозволяє видавцям токенів включати дані безпосередньо в сам токен. Крім того, з міркувань безпеки, токени доступу можуть мати термін дії, після закінчення якого вони стають недійсними.

Ідея ролей є частиною основної специфікації фреймворка авторизації OAuth. Вони визначають основні компоненти системи OAuth і включають наступне:

- Власник ресурсу (Resource Owner): Користувач або система, яка володіє захищеними ресурсами і може надати до них доступ.
- Клієнт (Client): Система, яка потребує доступу до захищених ресурсів. Для доступу до ресурсів клієнт повинен мати відповідний токен доступу (Access Token).
- Авторизаційний сервер (Authorization Server): Цей сервер отримує запити від клієнта

на отримання токенів доступу і видає їх після успішної автентифікації та згоди власника ресурсу. Авторизаційний сервер надає дві кінцеві точки: кінцева точка авторизації (Authorization endpoint), яка відповідає за інтерактивну автентифікацію та згоду користувача, і кінцева точка токенів (Token endpoint), яка використовується для взаємодії між системами.

● Ресурс Сервер (Resource Server): Сервер, який захищає ресурси користувача і отримує запити на доступ від клієнта. Він приймає та перевіряє токен доступу від клієнта і повертає йому відповідні ресурси.

OpenID Connect або OIDC — це протокол ідентифікації, який використовує механізми авторизації та авте нтифікації OAuth 2.0.OIDC був розроблений OpenID Foundation, до якого входять такі компанії, як Google і Microsoft. У той час як OAuth 2.0 є протоколом авторизації, OIDC є протоколом автентифікації особи, який може використовуватися для перевірки ідентичності користувача в клієнтському сервісі, яку також називають "Перевіряючою Стороною" (Relying Party). Крім того, клейми користувачів, як-от ім’я, адреса електронної пошти тощо, також можуть бути надані за запитом.

Широкий спектр клієнтів може використовувати OpenID Connect (OIDC) для ідентифікації користувачів, від одно сторінкових програм (SPA) до нативних і мобільних програм. Його також можна використовувати для єдиного входу (SSO) у різних програмах. OIDC використовує веб-токени JSON (JWT), сценарії HTTP та уникає передачі облікових даних користувача сервісам.

OpenID Connect включає в себе механізм згоди користувача. Це важливо, оскільки OIDC часто використовується в клієнтських сервісах, (наприклад, перевіряюча сторона), де для обміну персональними даними потрібна чітка згода користувача.

Постачальник OIDC (зазвичай його називають постачальником OpenID або постачальником ідентифікаційної інформації або IdP) виконує автентифікацію користувача, отримання згоди користувача та видачу токенів. Клієнт або сервіс, які запитують ідентифікацію користувача, зазвичай називають "Перевіряючою стороною " (Relying Party). Це може бути, наприклад, веб-програма, а також програма JavaScript або мобільна програма.

OIDC використовує OAuth 2.0 як базовий протокол. Основними розширеннями є спеціальне значення («openid»), використання додаткового токена (ID Token, який інкапсулює клейми ідентифікації у форматі JSON) і акцент на автентифікації, а не на авторизації. Основним використанням цього токена у форматі JWT є надання інформації про результат операції автентифікації. За запитом він може надати ідентифікаційні дані, що описують профіль користувача. Дані про результат автентифікації та інформацію профілю користувача називаються клеймами (claims). Клейми щодо профілю користувача можуть бути будь-якими даними, які мають відношення до "Перевіряючої сторони " для цілей ідентифікації, як-от постійний ідентифікатор, адреса електронної пошти, ім’я тощо.

Ці функції разом із простотою реалізації роблять OpenID Connect корисним протоколом, коли потрібна ідентифікація користувача, і потужною альтернативою більш складному SAML 2.0. Він також особливо підходить для мобільних додатків.

Реалізація технік безпеки від підміни та викрадення токена для доступу до даних. Розглянемо деякі унікальні виклики, з якими стикаються браузери при роботі з OAuth для односторінкових додатків (Single Page Application). Односторінкові додатки є поширеним патерном для створення додатків у вебі сьогодні. Існує кілька варіацій таких архітектур, але всі вони мають одну спільну річ - браузер. Одно сторінковий додаток може бути розгорнутий на статичному веб-хостингу, наприклад, S3, але ви також можете використовувати фреймворк одно сторінкового додатку, який обслуговується динамічним веб-сервером, таким як .Net або Java. Одним з простіших обмежень JavaScript-додатків є відсутність можливості передавати будь-які облікові дані в додатку.

Це означає, що JavaScript-додатки вважаються публічними клієнтами з точки зору OAuth. Якщо вставити ключ API в JavaScript-додаток, будь-хто, хто використовує додаток,

може легко переглянути вихідний код у браузері і знайти ключ. Будь-які спроби сховати ключ можуть бути оброблені зворотною розробкою. Тим не менш, якщо є динамічний серверний компонент, який хостить одно сторінковий додаток, то цей серверний компонент може бути конфіденційним клієнтом. Тобто немає способу включити ключі API або секрети клієнта в JavaScript-частину додатка, якщо ви хочете, щоб вони залишилися секретними. Тому, не маючи секрета клієнта, ми повинні мати можливість виконувати OAuth-потік, який не вимагає секрета, що точно надає розширення PKCE.

Ще одним викликом браузерного середовища є велика кількість атак, якими вразливі браузери. Одна з найнебезпечніших атак - це міжсайтовий скриптинг, відома також як XSS. Якщо зловмисник може здійснити атаку на міжсайтовий скриптинг у вашому додатку, це означає, що він може виконувати код у вашому додатку, який виглядає як законний код. Уразливості міжсайтового скриптингу є дуже небезпечні, оскільки зловмисник може отримати доступ до будь-яких даних, які додаток зберігає самостійно, наприклад, до даних користувача або до токена доступу. Зловмисник також може надсилати запити API з використанням токена доступу, навіть якщо він не може бачити сам токен доступу.

Одним з найкращих способів захисту від атак на міжсайтовий скриптинг є використання суворої політики безпеки вмісту (Content Security Policy). Це дозволяє браузерам вказати, з яких доменів можна завантажувати JavaScript. Але навіть тоді створення досить сильної політики безпеки вмісту може бути викликом, оскільки потрібно дозволити функціонування аналітичних систем або рекламних мереж. Це дещо складна задача, оскільки багато розробників додатків сьогодні покладаються на сторонній JavaScript-код, чи то для реклами, аналітики, або навіть для фреймворка CSS, який вимагає JavaScript для активації анімацій. Усі ці сторонні скрипти є потенційними векторами атак, які можуть скомпрометувати ваш додаток. Ви можете підключати ці JavaScript-файли з інших доменів або включати їх до вашого процесу збірки, компілюючи їх в один JavaScript-файл. Але в будь-якому випадку цей сторонній код може бути скомпрометований і зашкодити вашому додатку. Це не означає, що ви повністю не можете використовувати сторонній JavaScript, просто треба пам'ятати про потенційні ризики при його використанні.

Тісно пов'язано з атаками на міжсайтовий скриптинг є те, що користувачі можуть встановити будь-яку кількість розширень браузера, які також отримують доступ до сторінки. Це дуже складне питання, оскільки ви ніколи не можете бути повністю впевненими, який код насправді виконується в вашому додатку. Навіть якщо у вас є сувора політика безпеки вмісту і ви не завантажуєте жодного зовнішнього JavaScript у ваш додаток, користувач може зробити це самостійно, використовуючи розширення, які вставляють JavaScript на сторінку. Це означає, що будь-яке розширення, яке користувачі встановили, також може спостерігати за даними вашого додатка.

І на кінець, одним з серйозних обмежень браузерів є те, що насправді не існує ніякого безпечного API для зберігання даних. Вже відомо, що немає способу включити секрет клієнта в браузерний додаток, але навіть якби він був, браузеру не було б де його зберігати в безпечному місці. Зловмисники можуть переглянути локальне сховище, кукі, локальну базу даних або будь-яке інше місце, де додаток може зберігати дані. Існує декілька способів ускладнити заволодіння даними, таких як шифрування, використання додаткових шарів захисту, використання міцних механізмів автентифікації і авторизації, але це не може гарантувати 100% безпеки.

Отже, при розробці браузерних додатків слід пам'ятати про ці унікальні виклики і застосовувати відповідні заходи безпеки для забезпечення найвищого рівня безпеки для користувачів і додатків.

Реалізація власного Single Sign On Server. В залежності від типу клієнта, ми будемо використовувати різні гранти з різними властивостями. Зокрема, ми хочемо зосередитись на сценаріях, коли односторінковий додаток викликає API браузера. З цією метою ми

використовуємо грант авторизації OAuth 2.0 з кодом підтвердження. (Authorization Code Flow with PKCE) Грант авторизації з кодом дає клієнту можливість отримати доступ до API від імені користувача і в межах того, на що користувач надав згоду. Проте, в гранті авторизації з кодом використання ідентифікатора додатка, такого як секрет клієнта, необхідне. Кожного разу, коли додаток обмінює авторизаційний код, він повинен автентифікуватися як клієнт перед авторизаційним сервером.

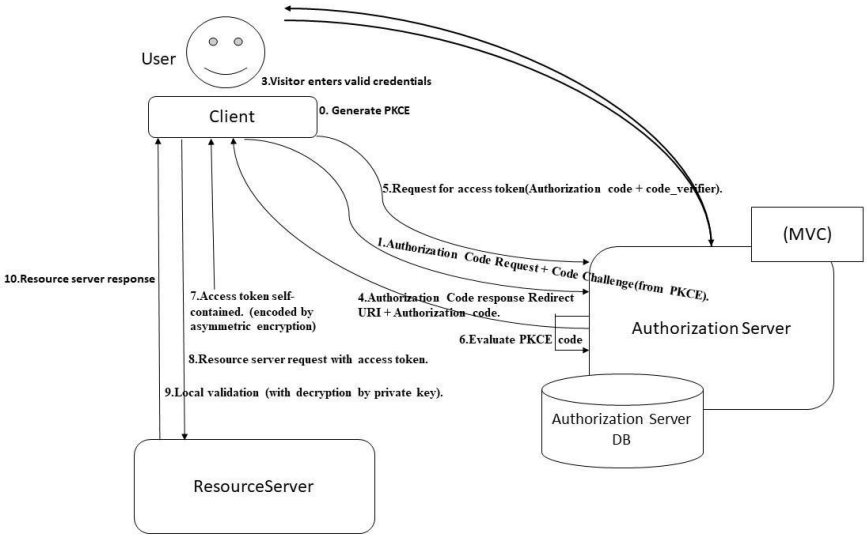

Рис. 3. Схема взаємодії у процесі OAuth 2.0

На діаграмі зображені учасники процесу авторизації:

- Угорі зображений користувач та його браузер.
- Авторизаційний сервер.
- Клієнт, що приблизно знаходиться посередині.
- API-сервер ресурсів, який клієнт повинен викликати в рамках нашого сценарію.

Покроковий опис процесу

0. Generate PKCE

Процес розпочинається з того, що користувач натискає кнопку входу, і це подібно до того, як користувач говорить: "Я хочу використовувати цей додаток". Перед тим, як додаток перенаправить користувача, він створює новий секрет для цього конкретного потоку. Це не секрет клієнта. Це випадковий рядок, що генерується додатком та повинен бути різним кожного разу, коли розпочинається процес авторизації. Це називається PKCE (Proof Key for Code Exchange) – кодовий верифікатор PKCE.

PKCE спочатку розроблявся для мобільних додатків для захисту авторизаційного коду, оскільки в мобільних додатках також немає секрета клієнта. Але виявляється, що PKCE також захищає від деяких атак, навіть якщо у вас є секрет клієнта. І це також працює добре в браузерах з JavaScript. Таким чином, додаток зберігає цей кодовий верифікатор в браузері, зазвичай, зберігаючи його в LocalStorage або SessionStorage, і потім обчислює хеш від нього, який називається кодовим викликом. Хеш, звичайно ж, є односторонньою операцією. Тому, якщо хтось знає значення хешу, він не зможе його розшифрувати і визначити секрет.

1. Authorization Code Request + Code Challenge(from PKCE). Тепер додаток перенаправляє користувача на сервер зі значним набором даних у рядку запиту, включаючи цей хеш, ідентифікатор клієнта, URL перенаправлення і scope. Окрім токена доступу, необхідного для доступу до API, ми також запитуємо ID-токен. ID-токен корисний у цьому сценарії, оскільки сам токен доступу є непрозорим для клієнта і не містить багато інформації про транзакцію. Наступний параметр - client_id - представляє ідентифікатор клієнта, який

ідентифікує веб-додаток на сервері авторизації. Значення response_type для цього конкретного гранту є code. Ми хочемо отримати код від кінцевої точки авторизації, який клієнтський додаток пізніше обміняє на токен за допомогою кінцевої точки обміну токенами. Нам не потрібно вказувати response mode параметр, оскільки ми погоджуємося зі стандартним режимом відповіді, який у випадку типу відповіді code є рядок запиту(query) - це означає, що ми очікуємо, що сервер авторизації поверне код авторизації як параметр рядка запиту. Далі ми знаходимо параметр scope. Це повідомлення містить всі ті самі значення scopes, які були вказані раніше - openid, profile і email, що означає, що ми вимагаємо ID-токен разом із кодом. Наступний параметр представляє URI перенаправлення, але він повинен збігатися з одним з URL-адрес перенаправлення, які було додано при реєстрації клієнта на сервері авторизації. Якщо подивитися, що насправді відбувається тут, це перше повідомлення, яке відправляється по передньому каналу. Тому, через те, що це запит по передньому каналу, додаток надсилає лише хеш створеного ним секрету, а не сам секрет.

2. Ask for credentials. Тепер користувач перебуває на сервері OAuth, і сервер OAuth просить його увійти в систему.

3. Visitor enters valid credentials. Користувач вводить свої дані для входу і проходить усі необхідні процедури багатофакторної автентифікації. Після отримання запиту на авторизацію сервер авторизації бере на себе відповідальність за інтерактивну частину потоку. Кінцева точка авторизації визначає, що потрібно для автентифікації користувача і пропускає його через цей процес. Потім сервер пропонує користувачу підтвердити, що він намагається увійти до цього додатку і показує йому екран згоди.

4. Authorization Code response Redirect URI + Authorization code. У момент, коли користувач надає згоду, кінцева точка авторизації повертає відповідь з запитаним кодом авторизації у рядку запиту, відповідно до вказаного response_type. Крім того, відповідь включає звичайну команду set-cookie, за допомогою якої сервер авторизації реєструє у браузері, що була встановлена автентифікаційна сесія. На цей момент браузер просто виконує перенаправлення, яке передасть код авторизації клієнтському додатку.

5. Request for access token(Authorization code + code_verifier). Клієнтський додаток поєднує код авторизації зі своїми власними обліковими даними клієнта і надсилає їх у повідомленні до кінцевої точки отримання токена (token endpoint). Повідомлення до кінцевої точки отримання токена відправляється у формі HTTP POST-запиту, в якому додаток надає свої client_id і PKCE, отриманий з минулого кроку код авторизації та новий параметр - grant_type. Кожного разу, коли додаток спілкується з кінцевою точкою отримання токена, він повинен вказати бажаний тип гранту, щоб повідомити сервер авторизації, як інтерпретувати запит. У цьому конкретному випадку, процесом є грант authorization_code. Це говорить серверу авторизації шукати код авторизації в повідомленні і враховувати client ID і PKCE у контексті цього конкретного гранту. Останнім параметром повідомлення є redirect_uri. На цьому етапі сервер авторизації фактично не має можливості виконувати перенаправлення, оскільки клієнт спілкується з сервером авторизації через зворотній канал. Замість цього, параметр redirect_uri використовується як захисний захід, щоб запобігти маніпулюванню перенаправленням URI - сервер авторизації перевірить, що представлений тут redirect_uri є ідентичним тому, що був вказаний під час запиту коду авторизації, запобігаючи заміні URI зловмисником.

6. Evaluate PKCE code. Так, сервер переглядає цей запит, кажучи: "Добре, я бачу, що цей код щойно був виданий, він ще не використовувався і був призначений для певного client ID. А коли запит почався, я отримав цей хеш(code challange). Тому сервер обчислює хеш коду-верифікатора в цьому запиті, порівнює хеші, і якщо вони збігаються, то він знає, що об'єкт, який використовує код авторизації, є тим самим об'єктом, який розпочав процес".

7. Access token self-contained. (encoded by asymmetric encryption) with public key + refresh token. Припускаючи, що запит приймається сервером авторизації та обробляється без

проблем, процес завершується відповіддю, яка містить артефакт, у цьому випадку - токен доступу.

Вміст відповіді:

• Запрошений токен доступу

• ID-токен у відповідь на наявність openid в списку запитаних значень scopes

• Тип токена, який завжди є Bearer

• Параметр expires_in, який виражає час, протягом якого токен доступу повинен вважатися дійсним. Хоча іноді токен доступу може містити цю інформацію та перебувати у форматі, який можна перевірити, клієнти повинні завжди розглядати токени доступу як непрозорі. Тому параметр expires_in надається як параметр у відповіді, щоб клієнт міг використовувати цю інформацію (наприклад, для визначення того, наскільки тривалий час кешувати токен доступу).

Resource server request with access token. Після того, як клієнт отримав запрошений токен доступу, він може нарешті викликати API: все, що йому потрібно зробити, це включити біти токену доступу в класичний REST-виклик. Ключовою особливістю в цьому повідомленні є заголовок HTTP Authorization, який використовує схему автентифікації Bearer та містить біти токену доступу.

9. Local validation (with decryption by private key).
10. Resource server response.

Специфікація JSON Web Token (JWT) надає механізм для підпису JSON-даних та кодування їх у форматі, який легко передавати та безпечно використовувати в HTTP-заголовках. Специфікацію JWT можна використовувати для багатьох різних цілей. Вона використовується в OpenID Connect, а також для цілей, що повністю виходять за контекст веб-додатків. Це корисний будівельний блок, який також можна використовувати як токени доступу.

Отже, спочатку розглянемо основний формат JSON Web Token. JSON Web Token виглядає приблизно так. Це дані в кодуванні base64, які складаються з трьох частин, і якщо уважно придивитись, ви побачите крапку, що розділяє їх на три частини. Якщо розділити це на частини за крапками, то перші дві частини - це базовий 64-розрядний код, JSON, а остання частина - це підпис. Таким чином, після декодування з base64 ви побачите прості рядки JSON. Отже, верхня частина є заголовком. Це говорить про токен. Вона описує те як використаний алгоритм підпису та може також вказувати, який ключ було використано для його підпису. Середня частина – це корисне навантаження, яке містить дані, які вас насправді цікавлять. Зазвичай це короткі назви властивостей, за якими йдуть значення, також відомі як клейми.

Специфікація токенів доступу JWT визначає кілька вимог щодо того, що має бути у токені. Багато з цих вимог насправді є зарезервованими в специфікації основного JSON Web Token. Все ж таки серверам OAuth також можна додавати свої власні вимоги до зарезервованих. Але найменше, це ті, які повинні бути в маркері. По-перше, ми маємо is, скорочення від issuer (видавець). Це ідентифікатор сервера, який видав цей маркер. Зазвичай це буде базова URL-адреса сервера OAuth, де ви можете отримати discovery документ(інформацію про сервер авторизації) на основі цієї URL-адреси. exp — це скорочення від терміну дії. Це Unix -часова мітка, коли токен доступу більше не повинен вважатися дійсним. Це необхідний параметр. Таким чином, токени доступу JWT не можуть видаватися з необмеженою тривалістю життя. iat - скорочено від issued at (видано в). Це часова мітка, коли цей токен був виданий. aud - скорочено від audience (аудиторія). Це призначено для ідентифікації сторони, яка буде читати та перевіряти цей токен. Це може бути ідентифікатор сервера з ресурсами. sub - скорочено від subject (предмет). Це ідентифікатор того, кого представляє токен. У разі, якщо відбувається взаємодія з користувачем, це буде ідентифікатор користувача. Якщо учасник не залучений, як у випадку з грантом облікових даних клієнта (client credentials grant), то в цьому місці має бути ідентифікатором клієнта

цього додатка. Далі маємо client_id. Це насправді ідентифікатор клієнта додатка, якому був виданий цей токен. Останнім є jti, скорочено від JSON Web Token ID. Це унікальний ідентифікатор для даного токена, який ресурсний сервер може використовувати, щоб перевірити, чи використовується токен більше одного разу. Існує також кілька необов'язкових параметрів. Scope може бути включений, якщо токен доступу було видано з діапазоном у запиті. auth_time - це часова мітка, коли користувач останній раз автентифікувався на сервері авторизації. Наприклад, ваше API може використовувати це для того, щоб вимагати, щоб користувач вводив свій пароль протягом останньої години перед виконанням важливої операції, і воно відхилить запит API, якщо ця мітка часу є занадто старою.

Надалі у цьому розділі ми розглянемо спосіб валідації токенів доступу JSON Web Token (JWT). Цей спосіб полягає у тому, що перевірку токена доступу можна виконати без передачі мережевого трафіку, просто розглядаючи сам токен. Це також називається локальною перевіркою, оскільки вона відбувається локально без звернення до мережі.

Спочатку слід пам'ятати, що лише ваші API будуть перевіряти токени доступу. Застосунки, які отримують токени доступу після входу користувача, ніколи не повинні проводити перевірку токенів доступу. Вони просто використовують їх у запитах до API. Тому ми розглядаємо кроки, які API повинен виконати для прийняття рішення про те, чи є токен доступу дійсним, і чи повинен метод API повертати дані або виконувати операцію.

Розпочнемо з розгляду прикладу маркера доступу JSON Web Token. JSON Web Token складається з трьох частин: заголовку, корисного навантаження та підпису. Заголовок та корисне навантаження закодовано в форматі Base64 JSON. Тому спочатку ми розкодуємо ці дані та подивимося, що міститься всередині.

У заголовку згадується про токен та надається опис використаного алгоритму для підпису, також заголовок може містити інші дані, наприклад, ідентифікатор ключа, яким було підписано токен. Один важливий момент тут полягає в тому, що специфікація токенів доступу JSON Web Token явно зазначає, що необхідно уникати використання алгоритму "none". Це правильно, оскільки "none", ймовірно, було найгіршою ідеєю, яку коли-небудь включали до специфікації маркерів JSON Web Tokens. Це стало джерелом багатьох вразливостей протягом років. Однак слід мати на увазі, що ви читаєте дані з цього заголовка до перевірки підпису. Тому ви повинні ставитися до всіх цих даних як до ненадійних на цьому етапі. Основний наслідок полягає в тому, що ви повинні приймати тільки алгоритми підпису, які очікуєте від сервера. Класичним прикладом тут, окрім "none", є випадок, коли ваш сервер OAuth зазвичай використовує асиметричний алгоритм, наприклад, RS256, але у заголовку ви бачите симетричний алгоритм HS256, і ви використовуєте його для перевірки підпису. В такому випадку ви фактично використовуєте публічний ключ як спільний секрет, і зловмисник може створювати токени з будь-якими даними. Тому на практиці ви, ймовірно, створите закодований список відомих алгоритмів підпису, які використовує ваш сервер, і прийматимете лише ці значення.

Отже, дізнаємось, як знайти ці ключі. Специфікація токенів доступу JSON Web Token рекомендує, щоб сервери OAuth публікували свої публічні ключі та ідентифікатор видавця за URL-адресою, яку можна знайти в метаданих сервера. Цю URL-адресу метаданих можна зазвичай знайти в документації для використовуваного сервера OAuth, або ви можете побудувати її, додавши до URL-адреси видавця шлях ".well-known". Після отримання цих метаданих ви побачите багато JSON-даних, які описують сервер OAuth. Тут є речі, такі як ідентифікатор видавця, URL-адреса для кінцевої точки авторизації, кінцева точка маркера доступу та інші дані, які зараз нас не цікавлять. Але також є властивість "jwks_uri", яка є URL-адресою, де можна знайти ключі. Після завантаження цієї URL-адреси ви отримаєте фактичні дані ключів. Вони зазвичай представлені у форматі JSON Web Key Set (JWKS), який містить набір публічних ключів, які сервер використовує для підпису маркерів доступу.

Кожен ключ містить алгоритм підпису, матеріал ключа та його ідентифікатор. Зазвичай ви отримуєте декілька ключів, оскільки сервер може періодично оновлювати ключі для забезпечення безпеки.

Спочатку, ви завантажуєте JWKS з цієї URL-адреси та зберігаєте ключі в якомусь внутрішньому сховищі. Важливо пам'ятати, що потрібно періодично оновлювати ці ключі, оскільки сервер може їх змінювати. Також можна встановити час життя для цих ключів, і якщо ключі старіші за цей час життя, потрібно їх оновити. Загалом, обробка ключів - це важливий аспект управління безпекою токенів доступу.

Після отримання JWKS та ключів, ви готові перейти до перевірки підпису. Візьмемо алгоритм підпису з заголовка токена та відповідний ключ з JWKS, використаємо його для перевірки підпису токена та порівнюєте його з підписом у токені. Якщо підписи співпадають, то токен доступу вважається дійсним, і можна продовжити перевірку.

Однак перевірка підпису сама по собі недостатня. Важливо перевірити ряд інших речей, таких як термін дії токена, аудиторія, прив'язка до відповідного користувача, а також, можливо, деякі додаткові налаштування, які ви ввели для таконе доступу. Однак ці кроки вимагають доступу до додаткових джерел інформації, таких як база даних користувачів або конфігураційне сховище. Зазвичай ці джерела даних підтримуються серверами OAuth і надаються як складові системи.

Отже, це загальний огляд того, як виконується перевірка маркера доступу JSON Web Token в сервері OAuth. Це складний процес, і його безпеку необхідно ретельно враховувати при розробці та впровадженні.

Висновки

У цій статті було проведено комплексний аналіз систем єдиного входу (SSO), їхніх переваг, викликів та перспектив розвитку. SSO значно покращує користувацький досвід, спрощуючи автентифікацію та знижуючи ризики, пов’язані з керуванням численними паролями. Проте впровадження таких систем супроводжується низкою проблем, зокрема питаннями безпеки, конфіденційності, продуктивності та сумісності з застарілими системами.

Однією з ключових загроз є централізація автентифікації, що створює єдину точку уразливості (Single Point of Failure). Крім того, проблема глобального виходу (Single Logout, SLO) залишається невирішеною для багатьох сервісів, що може призводити до несанкціонованого доступу до облікових записів. Загрози фішингу, крадіжки токенів і перехоплення сесій вимагають додаткових заходів безпеки, таких як багатофакторна автентифікація (MFA), використання короткоживучих токенів і інтеграція сучасних стандартів безпеки (OAuth 2.0, OpenID Connect, FIDO2/WebAuthn).

Ще однією важливою проблемою є недостатня прозорість процесу обміну даними між сервісами. Для підвищення довіри користувачів необхідно запроваджувати механізми контролю за передачею особистої інформації, зокрема можливість вибору переданих даних і відкликання дозволів.

Оптимізація продуктивності систем SSO є критичною для великих організацій, оскільки велика кількість запитів до IdP може спричиняти затримки у доступі до сервісів. Балансування навантаження, кешування автентифікаційних токенів і використання легковагових протоколів, таких як JSON Web Token (JWT), є ефективними рішеннями для підвищення швидкодії.

Інтеграція SSO у застарілі системи залишається значним викликом, оскільки не всі сервіси підтримують сучасні протоколи автентифікації. Використання адаптерів, шлюзів автентифікації та поетапна міграція на нові стандарти можуть забезпечити поступове впровадження SSO без ризику збоїв у роботі інформаційних систем.

У перспективі розвиток SSO має орієнтуватися на посилення безпеки, підвищення прозорості управління даними та покращення інтеграції з іншими сервісами. Значний

потенціал має федеративна автентифікація (Federated Identity Management), що дозволяє організаціям створювати єдині екосистеми автентифікації з централізованим управлінням.

Загалом, для ефективного впровадження та використання SSO необхідний комплексний підхід, що враховує як технічні аспекти, так і потреби користувачів. Подальші дослідження повинні зосередитися на розробці механізмів удосконалення Single Logout, підвищенні рівня захисту персональних даних та покращенні продуктивності автентифікаційних систем.

Перелік посилань

1. Чирський Ю.В. Запровадження системи електронного документообігу в Україні. Режим доступу: http://old.minjust.gov.ua/7546.

2. Morkonda S. G., Chiasson S., van Oorschot P. C. Influences of displaying permission-related information on web single sign-on login decisions / Srivathsan G. Morkonda, Sonia Chiasson, Paul C. van Oorschot // Computers & Security. - April 2024. - Vol. 139. - P. 103666. - Available online 20 December 2023. - DOI: https://doi.org/10.1016/j.cose.2023.103666

3. Wilson Y., Hingnikar A. Single Sign-On. In: Solving Identity Management in Modern Applications / Y. Wilson, A. Hingnikar. – Berkeley, CA: Apress, 2023. – DOI: https://doi.org/10.1007/978-1-4842-8261-8_11

4. Suoranta S., Manzoor K., Tontti A., Ruuskanen J., Aura T. Logout in single sign-on systems: Problems and solutions / Sanna Suoranta, Kamran Manzoor, Asko Tontti, Joonas Ruuskanen, Tuomas Aura // Journal of Information Security and Applications. – February 2014. – Vol. 19, Issue 1. – P. 61-77. – DOI: https://doi.org/10.1016/j.jisa.2014.03.005

5. Surya M., Anithadevi N. Single Sign-on Mechanism Using Attribute-Based Encryption in Distributed Computer Networks / M. Surya, N. Anithadevi // Procedia Computer Science. – 2015. – Vol. 47. – P. 441-451. – DOI: https://doi.org/10.1016/j.procs.2015.03.228

6. Heckle R. R., Lutters W. G. Tensions of network security and collaborative work practice: Understanding a single sign-on deployment in a regional hospital / Rosa R. Heckle, Wayne G. Lutters // International Journal of Medical Informatics. – August 2011. – Vol. 80, Issue 8. – P. e49-e61. – DOI: https://doi.org/10.1016/j.ijmedinf.2011.02.001

7. Cusack B., Ghazizadeh E. Evaluating single sign-on security failure in cloud services / Brian Cusack, Eghbal Ghazizadeh // Business Horizons. – November–December 2016. – Vol. 59, Issue 6. – P. 605-614. – DOI: https://doi.org/10.1016/j.bushor.2016.08.002

8. Pérez Méndez A., Marín López R., López Millán G. Providing efficient SSO to cloud service access in AAA-based identity federations / Alejandro Pérez Méndez, Rafael Marín López, Gabriel López Millán // Future Generation Computer Systems. – May 2016. – Vol. 58. – P. 13-28. – DOI: https://doi.org/10.1016/j.future.2015.12.002

9. Федынишин Т., Михайлова О., Опірський І. Метод визначення потенційно небезпечних осіб по даних Bluetooth // Ukrainian Scientific Journal of Information Security. – 2023. – Том 29, Випуск 3. – С. 5.

10. Yevseiev, S., Hryshchuk, R., Molodetska, K., et al. (2022). Modeling of Security Systems for Critical Infrastructure Facilities. PC Technology Center. Available at: [Link or URL if available, e.g., polissiauniver.edu.ua]

11. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., & Strub, P.-Y. (2014). Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. *IEEE Symposium on Security and Privacy (SP)*, 98–113. https://doi.org/10.1109/SP.2014.15

12. Cantor, S., Kemp, J., Philpott, R., & Maler, E. (2005). Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) v2.0. OASIS Standard. Retrieved from https://www.oasis-open.org

13. Choudhary, A., & Kesswani, N. (2022). A comparative analysis of OAuth 2.0 and OpenID Connect security protocols. *Future Generation Computer Systems, 130*, 254–266. https://doi.org/10.1016/j.future.2022.01.018

14. Cusack, B., & Ghazizadeh, E. (2016). Evaluating single sign-on security failure in cloud services. *Business Horizons, 59*(6), 605–614. https://doi.org/10.1016/j.bushor.2016.08.002

15. Hardt, D. (2012). The OAuth 2.0 Authorization Framework. Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc6749.

Надійшла 23.01.2025