НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2022 р.

Звіт з практики

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

на тему: Сервіс аутентифікації та авторизації для систем з розподіленою архітектурою

Виконав : студент __4__ курсу, групи _ІП-83_____
(шифр групи)

_____Шпильовий Роман Петрович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____асистент Іваніщев Б.В._____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) професор, д.т.н., Сімоненко В. П._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022 р.

ЗМІСТ

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ МЕТОДІВ АВТОРИЗАЦІЇ ТА АУТЕНТИФІКАЦІЇ

1.1 Визначення понять

Щодня кожен з нас по декілька разів аутентифікується у різних застосунках, на різних сайтах. У сучасному світі будь-де потрібно підтверджувати свою особистість для того щоб люди (або сервіси) розуміли з ким вони мають справу та як потрібно спілкуватися з конкретним співрозмовником (користувачем). У реальному житті ми можемо підтвердити свою особистість за допомогою документів, що засвідчують особо таких як паспорт, водійське посвідчення та інші. Але у цифровому світі потрібно використовувати інші методі (хоч часом в цьому процесі і можуть брати участь реальні документи). Найпростішим способом є введення свого імені (логіну) та паролю. Звірившись зі своєю базою застосунок може розпізнати користувача і продовжити його минулу сесію, надати йому певні права доступу, які властиві саме цьому користувачу, або якщо дані не вірні то заборонити подальше користування для запобігання несанкціонованого використання сервісу.

Основні поняття які використовуються для розпізнавання користувачів це ідентифікація, аутентифікація та авторізація. Звучать ці терміни однаково, але кожен з них має свою функцію.

Ідентифікація – заява про тем ким є користувач. Це може бути ім’я користувача, логін, електронна пошта, номер облікового запису і таке інше.

Аутентифікація – надання доказів того що користувач є тим ким він ідентифікувався. Для прикладу це може бути надання паролю від облікового запису.

Авторизація – перевірка ролі користувача в системі та надання користувачу відповідного рівня доступу. Наприклад для того щоб отримати права адміністратора потрібно аутентифікуватися в обліковий запис який має

максимальний рівень доступу і тоді при спробі видалити чужий запис користувачу буде авторизовано право на це.

Але в сучасних системах існують і більш складні схеми аутентифікації та авторизації.

1.2 Види аутентифікації

1.2.1 Аутентифікація за паролем

Цей метод ґрунтується на тому, що користувач повинен надати username (логін) та password (пароль) для успішної ідентифікації та аутентифікації в системі. Пара username/password задається користувачем при його реєстрації в системі, при цьому як username може бути ім’я, адреса електронної пошти користувача тощо.

Стосовно веб-застосунків, існує кілька стандартних протоколів для аутентифікації за паролем, які ми розглянемо нижче.

HTTP authentication

Цей протокол, описаний у стандартах HTTP 1.0/1.1[1], існує дуже давно і досі активно застосовується у корпоративному середовищі. Застосовуючи до веб-сайтів так:

Сервер, при зверненні неавторизованого клієнта до захищеного ресурсу, надсилає HTTP статус «401 Unauthorized» та додає заголовок «WWW-Authenticate» із зазначенням схеми та параметрів аутентифікації.

Браузер при отриманні такої відповіді автоматично показує діалог введення username та password. Користувач вводить деталі свого облікового запису.

У всіх наступних запитах на цей веб-сайт браузер автоматично додає HTTP заголовок «Authorization», в якому передаються дані користувача для аутентифікації сервером.

Сервер аутентифікує користувача за даними цього заголовка. Рішення про надання доступу (авторизація) здійснюється окремо на підставі ролі користувача або інших даних облікового запису.

Весь процес стандартизований та добре підтримується всіма браузерами та веб-серверами. Існує кілька схем аутентифікації, що відрізняються за рівнем безпеки:

1. **Basic** - найпростіша схема, при якій username і password користувача передаються в заголовку Authorization у незашифрованому вигляді (base64-encoded). Однак при використанні HTTPS (HTTP over SSL) протоколу є відносно безпечним. Принцип роботи можна подивитися на рисунку 1.1. [2]

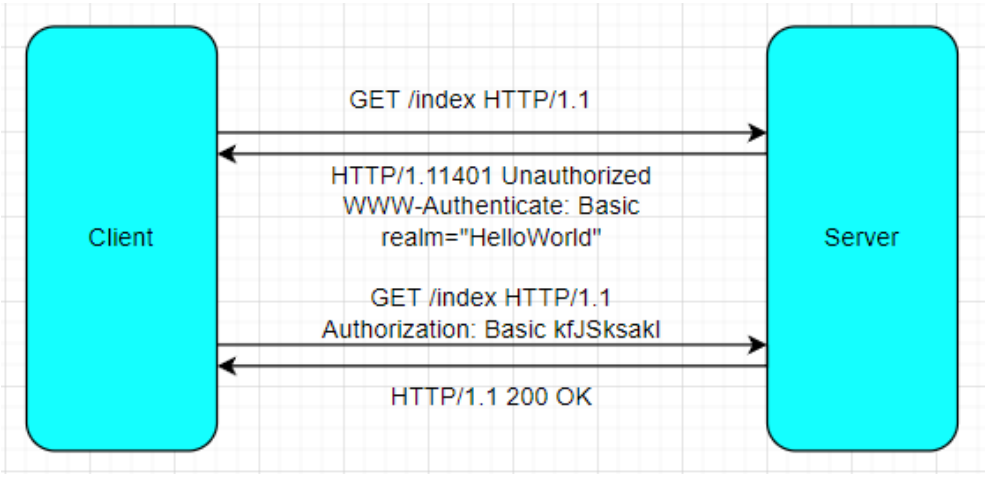

Рисунок 1.1 – Принцип роботи схеми Basic

2. **Digest** — challenge-response-схема, за якої сервер посилає унікальне значення nonce, а браузер передає MD5 хеш пароля користувача, обчислений за допомогою зазначеного nonce. Це більш безпечна альтернатива Basic схеми при незахищених з'єднаннях, але схильна до man-in-the-middle (Атака посередника – коли перехоплюються і змінюються дані якими обмінюються клієнт і сервер [3]) атак (із заміною схеми на basic). Крім того, використання цієї схеми не дозволяє використовувати сучасні хеш-функції для зберігання паролів користувачів на сервері.

3. **NTLM** (відома як Windows Authentication) - також заснована на challenge-response підході, при якому пароль не передається в чистому вигляді. Ця схема не є стандартом HTTP, але підтримується більшістю браузерів та веб-серверів. Переважно використовується для автентифікації користувачів Windows Active Directory у веб-застосунках. Вразлива до pass-the-hash атак [4].
4. **Negotiate** — ще одна схема із сімейства Windows authentication, яка дозволяє клієнту вибрати між NTLM та Kerberos автентифікацією. Kerberos — безпечніший протокол, заснований на принципі SSO. Однак він може функціонувати тільки якщо клієнт і сервер знаходяться в зоні intranet і є частиною домену Windows.

Варто зазначити, що при використанні HTTP-автентифікації у користувача немає стандартної можливості вийти з веб-програми, крім закрити всі вікна браузера.

Forms authentication

Для цього протоколу немає певного стандарту, тому всі його реалізації специфічні для конкретних систем, а точніше для модулів аутентифікації фреймворків розробки.

Працює це за таким принципом: до веб-додатоку включається HTML-форма, в яку користувач повинен ввести свої username/password і відправити їх на сервер через HTTP POST для аутентифікації. У разі успіху веб-програма створює session token, який зазвичай міститься в browser cookies. При наступних веб-запитах session token автоматично передається на сервер і дозволяє програмі отримати інформацію про поточного користувача для авторизації запиту. Схема роботи forms authentication можна побачити на рисунку 1.2.[5]

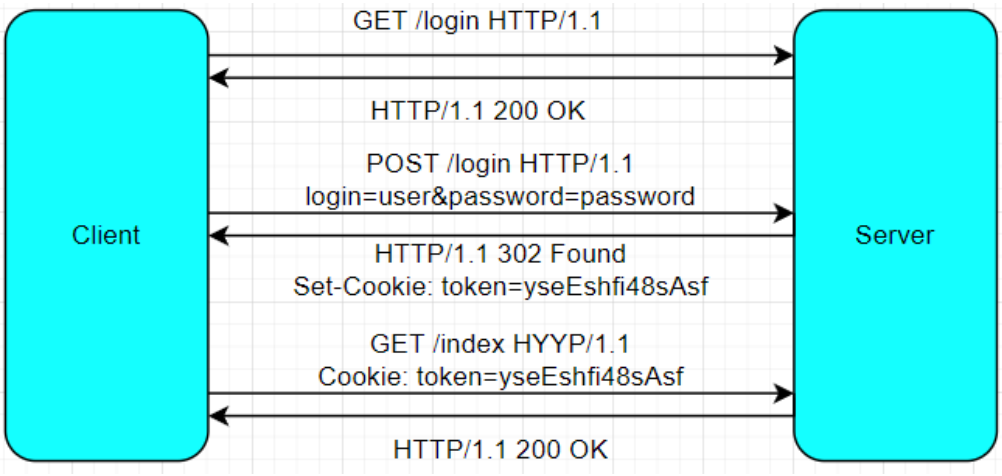

Рисунок 1.2 – Схема роботи forms authentication

Програма може створити session token двома способами:

1. Як ідентифікатор аутентифікованої сесії користувача, що зберігається у пам'яті сервера чи у базі даних. Сесія повинна містити всю необхідну інформацію користувача для можливості авторизації його запитів.
2. Як зашифрований та/або підписаний об'єкт, що містить дані про користувача, а також період дії. Цей підхід дозволяє реалізувати stateless-архітектуру сервера, проте вимагає механізму оновлення сесійного токена після закінчення терміну дії.

Необхідно розуміти, що перехоплення session token найчастіше дає аналогічний рівень доступу, як і знання username/password. Тому всі комунікації між клієнтом і сервером у разі forms authentication повинні проводитися лише захищеним з'єднанням HTTPS.

Два протоколи, описані вище, успішно використовуються для аутентифікації користувачів на веб-сайтах. Але при розробці клієнт-серверних застосунків з використанням веб-сервісів (наприклад, iOS або Android), поряд з HTTP аутентифікацією, часто застосовуються нестандартні протоколи, в яких дані для автентифікації передаються в інших частинах запиту.

1.2.2 Поширені вразливості та помилки реалізації паролів

Аутентифікація по паролю вважається не дуже надійним способом, оскільки пароль часто можна підібрати, а користувачі схильні використовувати прості та однакові паролі в різних системах, або записувати їх на клаптиках паперу. Якщо зловмисник зміг дізнатися пароль, користувач часто про це не дізнається. Крім того, розробники додатків можуть припуститися низки концептуальних помилок, що спрощують злом облікових записів.

Нижче наведено список найбільш поширених помилок при використанні аутентифікації по паролю:

- Веб-програма дозволяє користувачам створювати прості паролі.
- Веб-додаток не захищений від можливості перебору паролів (brute-force attacks[6]).
- Веб-програма сама генерує та розповсюджує паролі користувачам, однак не вимагає зміни пароля після першого входу (тобто початковий пароль десь записаний).
- Веб-програма допускає передачу паролів незахищеним HTTP-з'єднанням або в рядку URL.
- Веб-програма не використовує безпечні хеш-функції для зберігання паролів користувачів.
- Веб-програма не надає користувачам можливість зміни пароля або не нотифікує користувачів про зміну паролів.
- Веб-програма використовує вразливу функцію відновлення пароля, яку можна використовувати для отримання несанкціонованого доступу до інших облікових записів.
- Веб-програма не вимагає повторної аутентифікації користувача для важливих дій: зміна пароля, зміни електронної пошти, тощо.
- Веб-програма створює session tokens таким чином, що вони можуть бути підібрані або передбачені для інших користувачів.

- Веб-додаток допускає передачу session tokens незахищеним HTTP-з'єднанням, або в рядку URL.
- Веб-додаток вразливий для session fixation-атак[7] (тобто не замінює session token при переході анонімної сесії користувача до аутентифікованої).
- Веб-застосунок не встановлює прапори HttpOnly та Secure для browser cookies, що містять session tokens.

Веб-програма не знищує сесії користувача після короткого періоду неактивності або не надає функції виходу з аутентифікованої сесії.

1.2.3 Аутентифікація за сертифікатами

Сертифікат – це набір атрибутів, що ідентифікують власника, підписаний certificate authority (CA). CA виступає у ролі посередника, який гарантує справжність сертифікатів. Також сертифікат криптографічно пов'язаний із секретним ключем, який зберігається у власника сертифіката та дозволяє однозначно підтвердити факт володіння сертифікатом.

На стороні клієнта сертифікат разом із закритим ключем можуть зберігатися в операційній системі, браузері, файлі, на окремому фізичному пристрої (smart card, USB token). Зазвичай секретний ключ додатково захищений паролем або PIN-кодом.

У веб-застосунках традиційно використовують сертифікати стандарту X.509 [8]. Аутентифікація за допомогою сертифіката X.509 (приклад сертифікату X.509 можна побачити на рисунку 1.3) відбувається в момент з'єднання з сервером і є частиною протоколу SSL/TLS. Цей механізм також добре підтримується браузерами, які дозволяють користувачеві вибрати та застосувати сертифікат, якщо веб-сайт припускає такий спосіб автентифікації.

Під час аутентифікації сервер перевіряє сертифікат на підставі наступних правил:

1. Сертифікат має бути підписаний довіреним certification authority (перевірка ланцюжка сертифікатів).
2. Сертифікат має бути дійсним на поточну дату (перевірка терміну дії).
3. Сертифікат не повинен бути відкликаний відповідним CA (перевірка списків виключення).

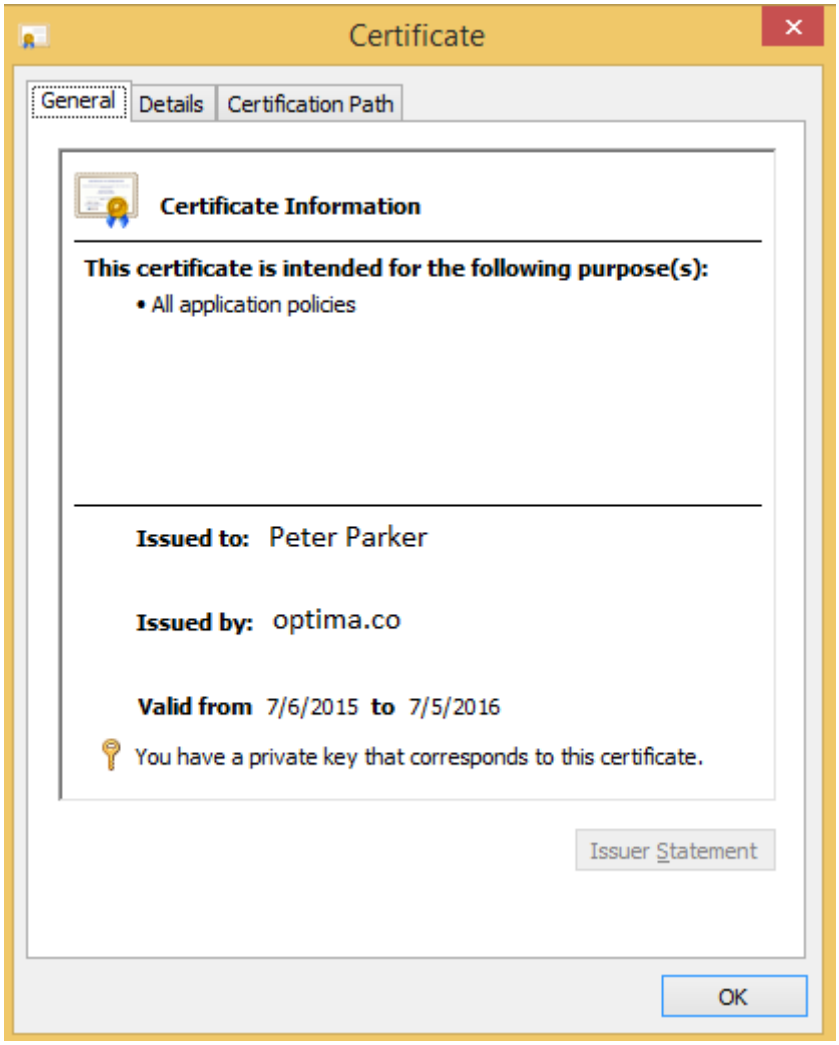

Рисунок 1.3 – Приклад сертифікату X.509

Після успішної аутентифікації веб-програма може виконати авторизацію запиту на підставі таких даних сертифіката, як subject (ім'я власника), issuer (емітент), serial number (серійний номер сертифіката) або thumbprint (відбиток відкритого ключа сертифіката).

Використання сертифікатів для аутентифікації — набагато надійніший спосіб, ніж аутентифікація за допомогою паролів. Це досягається створенням у процесі аутентифікації цифрового підпису, наявність якого доводить факт застосування закритого ключа у конкретній ситуації (non-repudiation). Однак

труднощі з поширенням та підтримкою сертифікатів робить такий спосіб автентифікації малодоступним для широкого використання.

1.2.4 Аутентифікація одноразовими паролями

Аутентифікація одноразовими паролями зазвичай застосовується додатково до аутентифікації за паролями для реалізації two-factor authentication[9] (2FA). У цій концепції користувачеві необхідно надати дані двох типів для входу в систему: щось, що він знає (наприклад, пароль) і щось, чим він володіє (наприклад, пристрій для генерації одноразових паролів). Наявність двох факторів дозволяє значною мірою збільшити рівень безпеки, що може бути потрібна для певних видів веб-додатків.

Інший популярний сценарій використання одноразових паролів – додаткова автентифікація користувача під час виконання важливих дій: переказ грошей, зміна налаштувань тощо.

Існують різні джерела створення одноразових паролів. Найбільш популярні:

1. Апаратні або програмні токени, які можуть генерувати одноразові паролі на підставі секретного ключа, введеного в них, та поточного часу. Секретні ключі користувачів, які є фактором володіння, також зберігаються на сервері, що дозволяє перевірити введені одноразові паролі. Приклад апаратної реалізації токенів - RSA SecurID[10]; програмної програми Google Authenticator [11].
2. Випадково генеровані коди, які передаються користувачеві через SMS або інший канал зв'язку. У цій ситуації фактор володіння – телефон користувача (точніше – SIM-картка, прив'язана до певного номера).
3. Роздруківка або scratch card зі списком заздалегідь сформованих одноразових паролів. Для кожного нового входу в систему потрібно ввести новий одноразовий пароль із зазначеним номером.

У веб-застосунках такий механізм аутентифікації часто реалізується за допомогою розширення forms authentication: після первинної аутентифікації за паролем, створюється сесія користувача, однак у контексті цієї сесії користувач не має доступу до додатку, поки він не виконає додаткову аутентифікацію за одноразовим паролем.

1.2.5 Аутентифікація ключами доступу

Цей спосіб найчастіше використовується для аутентифікації пристроїв, сервісів або інших програм при зверненні до веб-сервісів. Тут як секрет застосовуються ключі доступу (access key, API key) — довгі унікальні рядки, що містять довільний набір символів, які по суті замінюють комбінацію username/password.

У більшості випадків сервер генерує ключі доступу на запит користувачів, які далі зберігають ці ключі в клієнтських програмах. При створенні ключа також можна обмежити термін дії та рівень доступу, який отримає клієнтська програма під час аутентифікації за допомогою цього ключа.

Використання ключів дозволяє уникнути передачі пароля користувача стороннім програмам. Ключі мають значно більшу унікальність в порівнянні з паролями, тому їх практично неможливо підібрати. Крім того, якщо ключ був розкритий, це не призводить до компрометації основного облікового запису користувача - достатньо лише скасувати цей ключ і створити новий.

З технічної точки зору, тут не існує єдиного протоколу: ключі можуть передаватися в різних частинах запиту HTTP: URL query, request body або HTTP header. Як і у випадку аутентифікації за паролем, найоптимальніший варіант - використання HTTP header. У деяких випадках використовують HTTP-схему Bearer [2] для передачі токена в заголовку (Authorization: Bearer [token]).

Щоб уникнути перехоплення ключів, з'єднання з сервером має бути обов'язково захищене протоколом SSL/TLS.

Крім того, існують складніші схеми аутентифікації ключів для незахищених з'єднань. У цьому випадку ключ зазвичай складається з двох частин: публічної та секретної. Публічна частина використовується для ідентифікації клієнта, а секретна частина дозволяє згенерувати підпис. Наприклад, за аналогією з digest authentication схемою, сервер може надіслати клієнту унікальне значення nonce або timestamp, а клієнт повернути хеш або HMAC цього значення, обчислений з використанням секретної частини ключа. Це дозволяє уникнути передачі всього ключа в оригінальному вигляді та захищає від replay attacks [12].

1.2.6 Аутентифікація токенами

Такий спосіб аутентифікації найчастіше застосовується при побудові розподілених систем SSO, де одна програма (service provider або relying party) делегує функцію аутентифікації користувачів іншому додатку (identity provider або authentication service) [13]. Типовий приклад цього способу - вхід до програми через обліковий запис у соціальних мережах. Тут соціальні мережі є сервісами аутентифікації, а програма довіряє функцію аутентифікації користувачів соціальним мережам.

Реалізація цього способу полягає в тому, що identity provider (IP) надає достовірні відомості про користувача у вигляді токена, а service provider (SP) додаток використовує цей токен для ідентифікації, аутентифікації та авторизації користувача.

На загальному рівні весь процес виглядає наступним чином:

1. Клієнт аутентифікується в identity provider одним із способів, зручним для нього (пароль, ключ доступу, сертифікат, Kerberos або інші).

2. Клієнт просить identity provider надати йому токен для конкретного SP-програми. Identity provider генерує токен та відправляє його клієнту.
3. Клієнт аутентифікується в SP-програмі за допомогою цього токена.

Процес, описаний вище, відображає механізм аутентифікації активного клієнта, тобто такого, що може виконувати запрограмовану послідовність дій (наприклад, додаток iOS/Android). Браузер же — пасивний клієнт у тому сенсі, що він може відображати сторінки, у відповідь на запит користувачів. У цьому випадку аутентифікація досягається за допомогою автоматичного перенаправлення браузера між веб-додатками identity provider та service provider.

Існує декілька стандартів, що точно визначають протокол взаємодії між клієнтами (активними та пасивними) та IP/SP-додатками і формат підтримуваних токенів. Серед найбільш популярних стандартів - OAuth, OpenID Connect, SAML та WS-Federation. Розглянемо докладніше нижче.

Сам токен зазвичай є структурою даних, яка містить інформацію, хто згенерував токен, хто може бути одержувачем токена, термін дії, набір відомостей про самого користувача (claims). Крім того, токен додатково підписується для запобігання несанкціонованих змін та гарантій справжності.

При аутентифікації за допомогою токена програма повинна виконати такі перевірки:

1. Токен був виданий довіреним identity provider додатком (перевірка поля issuer).
2. Токен призначається поточному додатку SP (перевірка поля audience).
3. Термін дії токена ще не минув (перевірка поля expiration date).
4. Токен справжній і не змінено (перевірка підпису).

В разі успішної перевірки програма виконує авторизацію запиту на підставі даних про користувача, що містяться в токені.

Формати токенів

Існує кілька поширених форматів токенів для веб-застосунків:

1. Simple Web Token (SWT) — найбільш простий формат, який являє собою набір довільних пар ім'я/значення у форматі кодування HTML form. Стандарт визначає кілька зарезервованих імен: Issuer, Audience, ExpiresOn та HMACSHA256 [14]. Токен підписується за допомогою симетричного ключа, таким чином обидва identity provider та service provider додатки повинні мати цей ключ для можливості створення/перевірки токена. Приклад SWT можна побачити на рисунку 1.4.

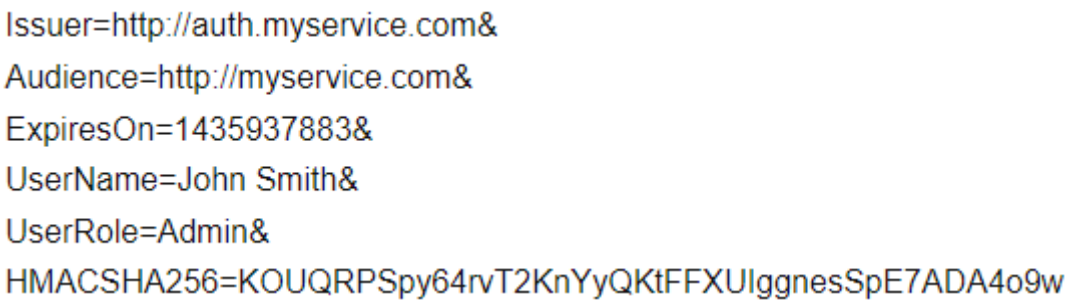

Рисунок 1.4 – приклад Simple Web Token (після декодування)

2. JSON Web Token (JWT) - містить три блоки, розділених точками: заголовок, набір полів (claims) та підпис. Перші два блоки представлені в JSON-форматі та додатково закодовані у формат base64. Набір полів містить довільні пари ім'я/значення, стандарт JWT визначає кілька зарезервованих імен (iss, aud, exp та інші) [15]. Підпис може генеруватися за допомогою симетричних алгоритмів шифрування, і асиметричних. Приклад токену можна побачити на рисунку 1.5.

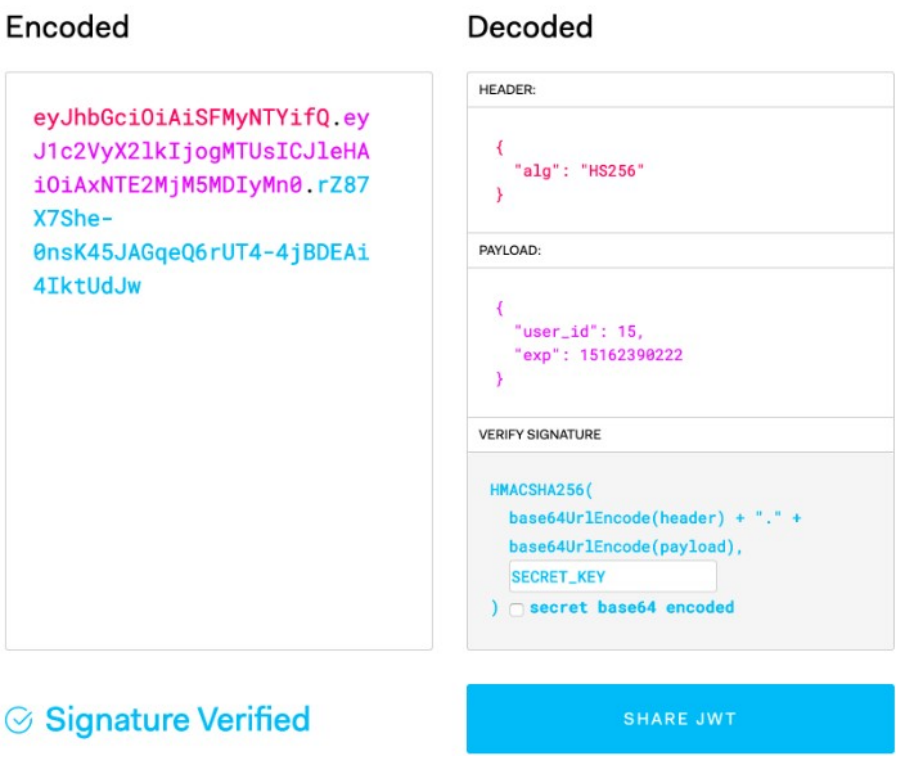

Рисунок 1.5 – Приклад JSON Web Token

3. Security Assertion Markup Language (SAML) - визначає токени (SAML assertions) в XML-форматі, що включає інформацію про емітент иа суб'єкт, а також необхідні умови для перевірки токена, набір додаткових тверджень (statements) про користувача. Підпис SAML-токенів здійснюється за допомогою асиметричної криптографії. Крім того, на відміну від попередніх форматів, SAML-токени містять механізм для підтвердження володіння токеном, що дозволяє запобігти перехопленню токенів через man-in-the-middle атаки при використанні незахищених з'єднань [16].

Стандарти WS-Trust та WS-Federation

WS-Trust і WS-Federation входять до групи стандартів WS-*, що описують SOAP/XML-веб-сервіси. Ці стандарти розробляються групою компаній, куди входять Microsoft, IBM, VeriSign та інші. Поряд із SAML, ці стандарти досить складні, використовуються переважно у корпоративних сценаріях.

Стандарт WS-Trust визначає інтерфейс сервісу авторизації, що називається Secure Token Service (STS). Цей сервіс працює за протоколом SOAP та підтримує створення, оновлення та анулювання токенів. При цьому стандарт допускає використання токенів різного формату, проте на практиці переважно використовуються SAML-токени [17].

Стандарт WS-Federation стосується механізмів взаємодії сервісів між компаніями, зокрема протоколів обміну токенів [18]. При цьому WS-Federation розширює функції та інтерфейс сервісу STS, описаного у стандарті WS-Trust. Серед іншого стандарт WS-Federation визначає:

- Формат та способи обміну метаданими про послуги.
- Функцію єдиного виходу із усіх систем (single sign-out).
- Сервіс атрибутів, що надає додаткову інформацію користувача.
- Сервіс псевдонімів дозволяє створювати альтернативні імена користувачів.
- Підтримка пасивних клієнтів (браузерів) за допомогою переадресації.

Можна сказати, що WS-Federation дозволяє вирішити ті самі завдання, що й SAML, проте їх підходи та реалізація певною мірою відрізняються.

JSON Web Token

JSON Web Token (JWT) відрізняються від інших веб-токенів тим, що містять набір вимог. Вимоги використовуються для передачі інформації між двома сторонами. Що це за вимоги, залежить від конкретного випадку використання. Наприклад, вимога може стверджувати, хто видав токен, як довго він дійсний або які права надано клієнту.

JWT — це рядок, що складається з трьох частин, розділених крапками (.), і серіалізований за допомогою base64. У найбільш поширеному форматі серіалізації, компактній серіалізації, JWT виглядає приблизно так: xxxxx.yyyyy.zzzzz. Приклад можна побачити на рисунку 1.5.

Після декодування ви отримаєте два рядки JSON:

1. Заголовок і корисне навантаження.
2. Підпис.

Заголовок JOSE (JSON Object Signing and Encryption) містить тип токену — в даному випадку JWT — і алгоритм підпису.

Корисне навантаження містить вимоги. Це рядок JSON, який зазвичай містить не більше десятка полів, для того щоб JWT був компактним. Ця інформація зазвичай використовується сервером для перевірки того, що користувач має дозвіл виконувати запитувану дію.

Немає обов'язкових вимог для JWT, але стандарти накладання можуть зробити вимоги обов'язковими. Наприклад, при використанні JWT як маркера доступу носія за OAuth2.0 повинні бути присутніми iss, sub, aud і exp, деякі зустрічаються частіше, ніж інші [19].

Підпис гарантує, що токен не був змінений. Сторона, яка створює JWT, підписує заголовок і корисне навантаження секретом, який відомий як емітенту, так і одержувачу, або закритим ключем, відомим лише відправнику. Коли токен використовується, сторона, що отримує, перевіряє, що заголовок і корисне навантаження відповідають підпису.

Коротше кажучи, JWT використовуються як безпечний спосіб аутентифікації користувачів і обміну інформацією.

Як правило, приватний ключ або секрет використовується емітентом для підписання JWT. Одержувач JWT перевірить підпис, щоб переконатися, що токен не був змінений після того, як його підписав емітент. Неаутентифікованим джерелам важко вгадати ключ підпису та спробувати змінити претензії в JWT.

Проте не всі алгоритми підписання створені однаковими. Наприклад, деякі алгоритми підпису використовують секретне значення, яке спільно використовують емітент і сторона, яка перевіряє JWT. Інші алгоритми використовують відкритий і закритий ключ. Закритий ключ відомий лише емітенту, тоді як відкритий ключ може бути широко поширений. Для перевірки підпису можна використовувати відкритий ключ, але для створення підпису можна використовувати лише закритий ключ. Це більш безпечно, ніж загальний секрет, оскільки приватний ключ повинен існувати лише в одному

місці. Через це серверу не потрібно зберігати базу даних з інформацією, необхідною для ідентифікації користувача. Для розробників це дуже зручно — сервер, який видає JWT, і сервер, який його перевіряє, не повинні бути однаковими.

Стандарти OAuth та OpenID Connect

На відміну від SAML та WS-Federation, стандарт OAuth (Open Authorization) не описує протокол автентифікації користувача. Натомість він визначає механізм отримання доступу однієї програми до іншої від імені користувача. Однак існують схеми, що дозволяють здійснити аутентифікацію користувача на базі цього стандарту.

Перша версія стандарту розроблялася у 2007 – 2010 роках, а поточна версія 2.0 опублікована у 2012 році. Версія 2.0 значно розширює і водночас спрощує стандарт, але несумісна з версією 1.0. Зараз OAuth 2.0 дуже популярний і використовується всюди для надання делегованого доступу та третьої сторонньої аутентифікації користувачів.

І так, як працює OAuth? Він описує, як програма яка робить запит на інформацію (client) може отримати доступ до потрібної інформації користувача (resource server) з дозволу самого користувача (resource owner). Схему взаємодії компонентів у стандарті OAuth можна побачити на рисунку 1.6. [20] У загальному вигляді весь процес складається з кількох кроків:

1. Користувач (resource owner) дає дозвіл програмі (client) на доступ до певного ресурсу у вигляді гранту.
2. Програма звертається до сервера авторизації та отримує токен доступу до ресурсу в обмін на свій грант. Під час виклику програма додатково аутентифікується за допомогою ключа доступу, виданого йому під час попередньої реєстрації.
3. Програма використовує цей токен для отримання необхідних даних від сервера ресурсів.

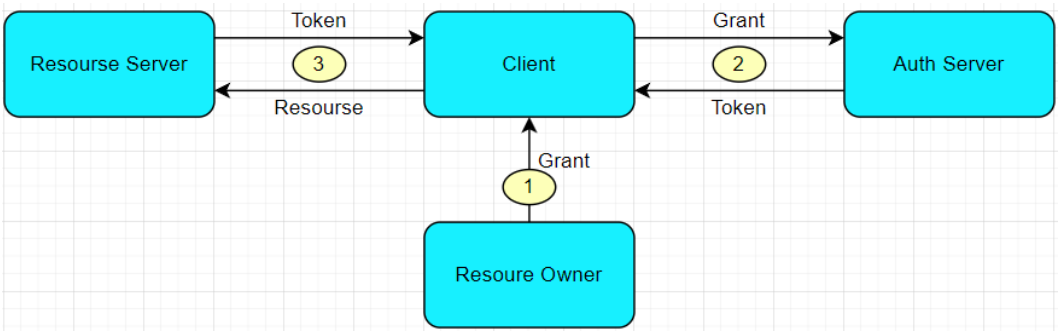

Рисунок 1.6 – Схему взаємодії компонентів у стандарті OAuth

Стандарт має чотири види грантів, які визначають можливі сценарії застосування:

Authorization Code — цей грант користувач може отримати від авторизованого сервера після успішної аутентифікації та підтвердження згоди на надання доступу. Такий спосіб найчастіше використовується у веб-додатках. Процес отримання гранту дуже схожий на механізм аутентифікації пасивних клієнтів SAML і WS-Federation.

Implicit — застосовується, коли програма не має можливості безпечно отримати токен від сервера авторизації (наприклад, JavaScript додаток у браузері). У цьому випадку грант – це токен, отриманий від сервера авторизації, а крок № 2 виключається зі сценарію вище.

Resource Owner Password Credentials — грант є парою username/password користувача. Може застосовуватися, якщо програма є «інтерфейсом» для сервера ресурсів (наприклад, програма - мобільний клієнт для Gmail).

Client Credentials — у цьому випадку немає жодного користувача, а програма отримує доступ до своїх ресурсів за допомогою своїх ключів доступу (виключається крок № 1).

Стандарт не визначає формат токена, який отримує програма: у сценаріях, адресованих стандартом, додатку немає необхідності аналізувати токен, тому що він лише використовується для отримання доступу до ресурсів. Тому ні токен, ні грант не можуть буди використані для аутентифікації користувача. Однак якщо програмі необхідно отримати достовірну інформацію про користувача, існують кілька способів зробити це:

1. Найчастіше API сервери ресурсів включають операцію, що надає інформацію про користувача (наприклад, /me в Facebook API). Програма може виконувати цю операцію щоразу після отримання токена для ідентифікації клієнта. Такий метод іноді називають псевдоаутентифікацією.
2. Використовувати стандарт OpenID Connect, розроблений як шар облікових даних навколо OAuth (опублікований у 2014 році). Відповідно до цього стандарту, сервер авторизації надає додатковий identity token на кроці №2. Цей токен у форматі JWT міститиме набір певних полів (claims) з інформацією про користувача.

Варто зауважити, що OpenID Connect, який замінив попередні версії стандарту OpenID 1.0 та 2.0, також містить набір необов'язкових додатків для пошуку серверів авторизації, динамічної реєстрації клієнтів та керування сесією користувача.

1.3 Альтернативи паролів

1.3.1 Недоліки паролів

Ми розглянули основні способи аутентифікації та авторизації для сучасних застосунків та веб-програм, але який з них найкращий? Відповідь на це питання не можуть однозначно дати найкращі вчені всього світу. Основна проблема в тому що для різних ситуацій краще використовувати різні рішення. Але хоч і не існує «найкращого» способу аутентифікації є найпопулярніший і це – аутентифікація по паролю. Це зумовлено як історично (це один з перших методів аутентифікації), так і простотою імплементації цього способу. Та не можна забувати про головний недолік цього способу – незахищеність перед будь-якими видами атак. Тому така популярність паролів над усіма іншими методами аутентифікації є серйозною вадою для безпеки застосунків. До того ж розробники часто нехтують базовими кроками для збільшення рівня безпеки, наприклад, такими як salting і hashing [21].

Ще один недолік паролів – те що вони засновані лише на елементі пам’яті людей. Це потребує зусиль від користувачів для того щоб їх не забути. Рідко користувачі підходять з повним усвідомленням відповідальності до підбору паролів, тому що складно для кожного сайту придумати окремий пароль та ще й постійно його пам’ятати, тому часто використовуються однакові паролі.

Одне дослідження [22] показало, що більшість користувачів мають багато облікових записів, до яких вони забули свої паролі, а також облікові записи які вони взагалі не пам’ятають що зареєстрували. Інше дослідження [23] використовувало розширення для браузера для спостереження за тисячами звичок користувачів стосовно паролів, знаходячи в середньому 25 облікових записів і 6 унікальних паролів на користувача. Тобто користувачі масово використовують однакові паролі для своїх облікових записів.

Чим же можна замінити безпосереднього введення паролю?

1.3.2 Візуальний пароль

До візуальних паролів можна віднести малювання різних рисунків і подальша перевірка схожості положень точок з початковим патерном. Дехто вважає запам’ятовування рисунків легшим ніж звичайних паролів, тому підносять такий спосіб аутентифікації на рівень вище за звичайні паролі. Але на мою думку набагато ефективнішими є системи які використовують модель «спільного секрету» (shared-secret[24]). Це віддалено нагадує Zero Knowledge Protocol, але насправді модель спільного секрету набагато простіший спосіб і, що варто зазначити, менш безпечний.

Найкраще суть такого методу демонструє аутентифікація на основі методу GrIDsure[25].

Розглянемо приклад. При реєстрації користувачеві надається сітка (наприклад, 5×5) і він обирає шаблон тобто послідовність клітинок. Наприклад, існує 254 можливих візерунка довжиною 4. Кожного разу, коли користувач бажає аутентифікуватися на захищеному ресурсі, йому представляється сітка виклику, що містить випадкові символи. Потім користувач вводить символи з клітинок, які відповідають його унікальному візерунку (шаблону). Оскільки асоціація цифр із клітинками є рандомізованою, то рядок введений користувачем, відрізняється кожного разу. Таким чином він розкриває знання про свою таємницю, не вводячи сам секрет. Наглядніше можна побачити на рисунку 1.7.

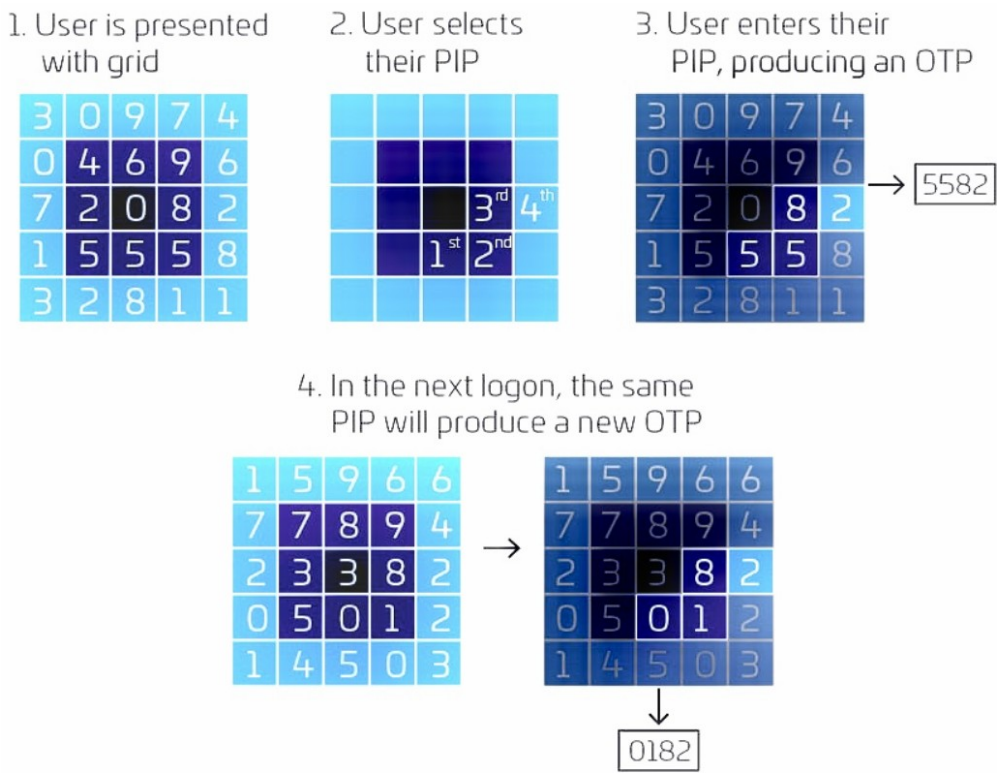

Рисунок 1.7 – Приклад роботи GrIDsure

Ця схема зручніша за паролі адже можна візуально запам’ятати шаблон і це спростить введення кожного паролю. Але у цього способу також є недоліки: на відміну від паролів, які часто можна вводити за допомогою пам’яті м’язів, транскрибування цифр із клітинок сітки вимагає зусиль і уваги і, ймовірно, буде повільніше. Також такий спосіб повність недоступний для незрячих користувачів.

Збоку безпеки є невеликий недолік відносно звичайних паролів. Зловмисникам доволі легко запам’ятати шаблон який вводить жертва у випадку коли є зоровий контакт з клітинками та полем вводу. До того ж обмежана кількість клітинок робить можливим злом шляхом перебору (нагадаю що для поля 5 на 5 кількість 4-значних кодів - всього 254), тому необхідно реалізовувати додатковий захист від такого виду атак.

1.3.3 Менеджер паролів

Останнім часом набирають популярності програми які можуть зберігати всі ваші паролі в одному місці. Для користувачів iOS така функція взагалі вбудована в їх телефони. Перевагою такого способу є те, що користувачу потрібно пам’ятати один пароль (у випадку с телефоном можна використати відбиток пальця або обличчя) для того, щоб отримати доступ до всіх ресурсів відразу. Це дозволяє створити унікальний пароль для кожного ресурсу і таким чином це збільшує безпеку користувача в цілому. Але при виборі менеджера паролів потрібно підійти з відповідальністю.

Існують варіанти з відкритим та закритим кором. У першому випадку користувач може сам переглянути код і точно буде знати які методи використовуються, як саме паролі зберігаються і це надає впевненості. Але з іншого боку ми маємо вільний доступ зловмисників до вихідного коду програми. Тобто переглянувши код вони можуть знайти вразливість в коді і тоді вони зможуть легко атакувати користувачів.

Якщо обирати рішення з закритим кодом, то треба з розумом обирати компанію, що підтримує програму, щоб вона бережно зберігала ваші паролі, ніколи їх не поширювала, а також вчасно випускала оновлення безпеки.

З точки зору безпеки – рішення неоднозначне. З одного боку ми маємо завжди різні паролі, які окремо дуже важко зламати. Але з іншого боку існує один пароль, який відкриває доступ відразу до всього. На мою думку, з боку користувачів, при використанні менеджера паролів потрібно також використовувати двофакторну аутентифікацію для важливих дій (переказ коштів, зміна паролю) для того щоб додатково вберегти свій обліковий запис.

1.3.4 Біометричні паролі

Кожному з нас відомо що у кожної людини унікальні відбитки пальців, форма обличчя, сітчатка ока, свій тембр голосу та інше. Це особливості можна використовувати для аутентифікації в різних програмах. Останнє десятиліття датчик відбитку пальця поширився по світу і використовується майже у кожному смартфоні. Точність таких датчиків доволі висока і хибні спрацьовування або помилки скоріше виняток ніж правило. Розпізнавання обличчя дуже поширене в Китаї для проведення банківських операцій, останнім часом виробники телефонів почали додавати у свої смартфони датчики для розпізнавання обличчя, хоча технологія ще розвивається.

Головна перевага такого способу аутентифікації – це зручність для користувача. Не потрібно придумувати та пам’ятати паролі, адже у будь-який момент можна показати своє обличчя і легко довести хто ти є насправді (як з людьми в реальному житті). Також швидкість розпізнавання біометричних даних на багато швидша ніж введення паролю.

Із недоліків такий спосіб має важкість фізичного втілення та поширення серед користувачів. Це пов’язано з тим, що він потребує додаткових зовнішніх датчиків для зчитування біометричної інформації. Також існує ймовірність того що датчик буде неякісним, що збільшить кількість помилок при аутентифікації та знизить рівень безпеки. Також є способи для обману таких датчиків, наприклад, зробити підробку відбитка за допомогою відбитку зі скла та гелевого тіла.

1.3.5 Об’єднаний єдиний вхід

SSO [26] дозволяє виконати аутентифікацію користувача, перенаправивши його на надійний сервер, який зможе підтвердити особистість користувача.

Про те як це працює вже згадувалося вище, на прикладі OAuth.

Найбільшою перевагою такого способу є зручність для користувача, адже треба пам’ятати лише один пароль від ресурсу, що зможе аутентифікувати користувача на інших. Також цей спосіб показує себе більш швидким за звичайні паролі, адже може використовуватись поточна активна сесія з cookies[27], та й вводити один і той самий пароль (нам потрібно аутентифікуватися лише в одному акаунті, який потім вже нас аутентифікує на усіх інших сервісах) завжди швидше.

Основний недолік – незахищеність від зловмисників. Адже отримавши доступ до одного акаунту шахраї автоматично мають доступ до всіх акаунтів сервісів що використовують об’єднаний єдиний вхід.

1.4 Порівняння альтернатив

Ми розглянули 4 альтернативи звичайним паролям: візуальний пароль, менеджер паролів, біометричні паролі, об’єднаний єдиний вхід. Порівняймо їх у таблиці 1.1. Для оцінювання використаємо шкалу від 1 до 5, де 5 – найкраще задовольняє обрану властивість, тобто найкращий варіант для застосування, а 1 – взагалі суперечить властивості, найгірший в цьому плані варіант, 3 – щось середнє. За відправну точку візьмемо звичайні паролі.

Таблиця 1.1 – Порівняльна характеристика

Назва → / Властивості ↓	Звичайні паролі	Візуальні паролі	Менеджер паролів	Біометричні паролі	Об’єднаний єдиний вхід
Легкий для запам’ятовування	3	4	5	5	5
Незалежність від сторонніх пристроїв/програм	5	5	3	1	4
Час користувача для аутентифікації	3	4	5	5	5

Кінець таблиці 1.1

Ціна для розробника	5	3	2	1	5
Час спрацювання алгоритму	5	3	4	5	2
Безпека	2	2	2	3	3

Як бачимо вони паролі є лідером в швидкості спрацювання алгоритму, не потребують сторонніх програм для реалізації та є дуже легкими для імплементації розробниками.

Візуальні паролі в свою чергу дають є більш зручними для користувача та легшими для запам’ятання, але натомість їх важче реалізувати розробникам.

Менеджер паролів максимально полегшує для користувачів роботу з паролями так як сам «запам’ятовує» і використовує потрібні паролі. Проблема лише в тому, що потрібно використовувати сторонні програми, які в свою чергу передають та зберігають паролі на третій стороні, це збільшує ризики витоку інформації та злому, але завдяки тому що кожен сервіс має свій унікальний пароль – це збільшує захищеність користувача в цілому, тому менеджер паролів отримує такий самий бал безпеки як і звичайні паролі. Також реалізація самого менеджера паролів потребує чималих зусиль з боку розробників.

Біометричні методи аутентифікації дозволяють користувачу взагалі не придумувати пароль, що є дуже зручним. Натомість для такого методу необхідні додаткові датчики для розпізнання біометричних даних, а для розробників реалізація цієї схеми потребує дуже багато зусиль, адже потрібно використовувати machine learning і штучний інтелект. Але коли все налаштовано то це забезпечує вищий рівень безпеки ніж звичайні паролі, адже біометрію важче підробити.

Метод об’єднаного єдиного входу пропонує найбільше переваг, на мою думку. Це дуже зручно для користувача – потрібно пам’ятати лише один пароль, і зазвичай він зберігається в кукі, тому на його введення взагалі не потрібно витрачати час. Також це легко для розробників, адже існує купа бібліотек, які реалізують цей метод і є захищеними. Майже зберігається

незалежність від сторонніх програм, хочу аутентифікація повністю залежить від довіреного сервісу аутентифікаці, ймовірність виходу його з ладу дуже мала (якби ставались часті падіння сервісу, його б не обрали «довіреним). Але потрібно відносно довго чекати аутентифікації, адже відбувається звернення до стороннього сервісу, що в свою чергу зумовлює затримку. Стосовно безпеки цей спосіб можна назвати відносно безпечним тому, що аутентифікація здійснюється на сторонньому сервісі який є перевіреним і довіреним. Та не забуваємо про те, що потрібно берегти пароль від цього сервісу щоб він не потрапив до зловмисників.

ВИСНОВОК

У першому розділі були розглянуті різні методи аутентифікації та авторизації. Існує декілька основних методів для того щоб аутентифікати користувача. Аутентифікація за паролем дозволяє підтвердити достовірність даних за допомогою введення секретного набору символів,який повинен бути відомий лише одному користувачу. Такий спосіб є доволі швидким, але не дуже надійним, адже доволі легко піддається злому і потребує більше уваги розробників, аби уникнути поширених помилок безпеки, які ми розглянули.

Аутентифікація сертифікатами частіше за все здійснюється за допомогою сертифіката X.509. Цей метод надійніший завдяки трьом перевіркам сертифікату під час авторизації, але через труднощі з поширенням та підтримкою сертифікатів він не дуже поширений.

Одноразові паролі використовуються частіше за все для забезпечення 2FA аутентифікації. Тобто як допоміжні до інших методів.

Аутентифікація ключами доступу дозволяє уникнути передачі пароля користувача стороннім програмам. Це забезпечує високий рівень безпеки, але потно подбати про те що ключі не буде перехоплено.

Токени використовуються в сучасному світі дуже широко. Було розглянуто найпопулярніші формати токенів: SWT, JWT, SAML.

На прикладі стандарту OAuth було розглянуто можливість аутентифікуватися в різних сервісах за допомогою одного облікового запису, шляхом звернення до стороннього сервісу, який є перевіреним та довіреним.

Було запропоновано декілька альтернатив звичайному введенню паролю, а саме: Візуальні паролі (на основі GrIDsure), Менеджери паролів, Біометрична аутентифікація, Об’єднаний єдиний вхід. Порівнявши ці методи з паролями можна зробити висновок, що жоден із цих способів не може забезпечити максимальної захищеності для даних користувача.

Таким чином актуальною є задача розроблення повністю безпечного методу аутентифікації. Це можливо завдяки методу криптографії zero-

knowledge proof. Перевагою використання такого методу є те, що він сумісний як зі звичайними паролями, так і з візуальними або біометричними. Завдяки йому ми можемо приховати початкові дані користувачів від сторонніх ресурсів (навіть від самого сервісу де відбувається аутентифікація) і як результат рівень безпеки наближається до максимального.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hypertext Transfer Protocol -- HTTP/1.0 [Електронний ресурс] // www.w3.org. W3C. – 1996. – Режим доступу до ресурсу: https://www.w3.org/Protocols/HTTP/1.0/spec.html#BasicAA.
2. MDN contributors. HTTP authentication [Електронний ресурс] / MDN contributors. – 2022. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication.
3. Attack74: Name:man-in-the-middle [Електронний ресурс] // The All.Net Security Database. – 2003. – Режим доступу до ресурсу: http://all.net/CID/Attack/Attack74.html.
4. Hacker R. Active Directory глазами хакера / Ralf Hacker // Active Directory глазами хакера / Ralf Hacker., 2021. – С. 60.
5. Mitchell S. An Overview of Forms Authentication (C#) [Електронний ресурс] / Scott Mitchell. – 2021. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/web-forms/overview/older-versions-security/introduction/an-overview-of-forms-authentication-cs.
6. Brute Force Attack: Definition and Examples [Електронний ресурс] // kaspersky. – 2021. – Режим доступу до ресурсу: https://www.kaspersky.com/resource-center/definitions/brute-force-attack.
7. Understanding Session Fixation Attacks [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://secureteam.co.uk/articles/web-application-security-articles/understanding-session-fixation-attacks/.
8. What Is an X.509 Certificate & How Does It Work? [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://sectigo.com/resource-library/what-is-x509-certificate.
9. What Is Two-Factor Authentication (2FA)? [Електронний ресурс] // Twilio – Режим доступу до ресурсу: https://authy.com/what-is-2fa/.

10. Multi-Factor Authentication [Електронний ресурс] // SecurID. – 2022. – Режим доступу до ресурсу: https://www.securid.com/products/multi-factor-authentication/.
11. Баранова Н. Google Authenticator — приложение для двухэтапной аутентификации [Електронний ресурс] / Наталья Баранова // Теплица социальных технологий. – 2018. – Режим доступу до ресурсу: https://te-st.ru/2018/02/08/google-authenticator/
12. What Is a Replay Attack? [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.kaspersky.com/resource-center/definitions/replay-attack.
13. What Is Token-Based Authentication? [Електронний ресурс] // Okta. – 2020. – Режим доступу до ресурсу: https://www.okta.com/identity-101/what-is-token-based-authentication/
14. HMACSHA256 Class [Електронний ресурс] // Microsoft. – 2022. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.hmacsha256?view=net-6.0
15. Introduction to JSON Web Tokens [Електронний ресурс] // jwt.io – Режим доступу до ресурсу: https://jwt.io/introduction
16. What is Security Assertion Markup Language (SAML)? [Електронний ресурс] // Oracle – Режим доступу до ресурсу: https://www.oracle.com/security/cloud-security/what-is-saml/
17. WS-Trust 1.4 [Електронний ресурс] // OASIS. – 2012. – Режим доступу до ресурсу: http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html
18. Authenticate users with WS-Federation in ASP.NET Core [Електронний ресурс] // Microsoft – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/ws-federation?view=aspnetcore-6.0
19. Ti Zhang. How to Use JWT with OAuth [Електронний ресурс] / Ti Zhang – Режим доступу до ресурсу:

https://www.loginradius.com/blog/engineering/using-jwt-with-oauth2-when-and-why/

20. dmitrybitman. OAuth 2.0 простым и понятным языком [Електронний ресурс] / dmitrybitman. – 2011. – Режим доступу до ресурсу: https://habr.com/ru/company/vk/blog/115163/
21. J. Bonneau and S. Preibusch, “The password thicket: technical and market failures in human authentication on the web,” in Proc. WEIS 2010, 2010.
22. S. Gaw and E. W. Felten, “Password Management Strategies for Online Accounts,” in ACM SOUPS 2006: Proc. 2nd Symp. on Usable Privacy and Security, pp. 44–55.
23. D. Florêncio and C. Herley, “A large-scale study of web password habits,” in WWW ’07: Proc. 16th International Conf. on the World Wide Web. ACM, 2007, pp. 657–666.
24. Menezes A. Discrete Mathematics and Its Applications / A. Menezes, P. van Oorschot, S. Vanstone // Handbook of Applied Cryptography / A. Menezes, P. van Oorschot, S. Vanstone. – https://cacr.uwaterloo.ca/hac/. – C. chapter 10 and 12.
25. Grid Authentication [Електронний ресурс] // Thales – Режим доступу до ресурсу: https://cpl.thalesgroup.com/access-management/authenticators/grid-authentication
26. How Does Single Sign-On Work? [Електронний ресурс] // onelogin – Режим доступу до ресурсу: https://www.onelogin.com/learn/how-single-sign-on-works
27. Что такое файлы cookies и зачем они нужны [Електронний ресурс] // ssl.com. – 2019. – Режим доступу до ресурсу: https://ssl.com.ua/blog/what-are-cookies/