Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри кібербезпеки
та захисту інформації
_____ Іван ПАРХОМЕНКО
« » травня 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

спеціальність *125 Кібербезпека*
(код і назва спеціальності)

освітній ступінь *магістр*

освітньо-наукова програма *Кібербезпека*
(назва освітньої програми)

на тему: «Метод безпарольної автентифікації на основі технології passkey»

Виконавець: студентка ІІ курсу, групи КБм-22 _____ Юлія ВЛАСЮК
(підпис) (Ім’я, ПРІЗВИЩЕ)

	Ім’я, ПРІЗВИЩЕ	**Підпис**
Науковий керівник	Іван ПАРХОМЕНКО	
Нормоконтроль	Сергій ДАКОВ	

Київ 2025

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри кібербезпеки
та захисту інформації
_____ Іван ПАРХОМЕНКО
« » листопада 2025 р

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *125 Кібербезпека та захист інформації*
(код і назва спеціальності)

освітній ступень *магістр*

Здобувача(ки) КБм-22 (група) Власюк Юлія Юріївна (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи Метод безпарольної автентифікації на основі технології passkey

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 5 від 24.10.2024 р.

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень	Процес автентифікації користувачів у мережах з контролем доступу.
Предмет досліджень	Методи автентифікації користувачів в мережі з використанням Captive Portal та OAuth.
Мета	Підвищення рівня контролю доступу за рахунок безпарольної автентифікації користувачів у мережі Wi-Fi на основі технології passkey.
Вихідні дані для проведення роботи	Методи парольної і безпарольної автентифікації у мережах.

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна	полягає у підвищенні рівня контролю доступу за рахунок безпарольної автентифікації користувачів у мережах з використанням технології passkey.
Практична цінність	можливість використання даного методу на об'єктах інформаційної діяльності для забезпечення безпарольної автентифікації

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Робота виконана у повному обсязі відповідно до теми.

5. ЕТАПИ ВИКОНАННЯ РОБОТИ

Найменування етапів робіт	Строки виконання робіт (початок-кінець)
Уточнення постановки задачі	25.10.2024 – 20.12.2024
Аналіз літературних джерел	21.12.2024 – 07.02.2025
Огляд традиційних методів автентифікації	08.02.2025 – 19.02.2025
Дослідження безпарольних методів автентифікації	20.02.2025 – 06.03.2025
Вивчення та налаштування системи Keycloak	07.03.2025 – 21.03.2025
Огляд можливостей pfSense та налаштування Captive Portal	22.03.2025 – 04.04.2025
Розробка та реалізація програмного модуля для авторизації	05.04.2025 – 25.04.2025
Інтеграція всіх компонентів у тестовому середовищі	26.04.2025 – 07.05.2025
Оцінка результатів та оформлення пояснювальної записки	08.05.2025 – 18.05.2025
Оформлення пояснювальної записки	19.05.2025

Завдання видав _____ Іван ПАРХОМЕНКО
(підпис) (Ім'я, ПРІЗВИЩЕ)

Завдання прийняв до виконання _____ Юлія ВЛАСЮК
(підпис) (Ім'я, ПРІЗВИЩЕ)

Дата видачі завдання: 25.10.2024 р.
Термін подання кваліфікаційної роботи до ЕК 19.05.2025 р.

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Метод безпарольної автентифікації на основі технології passkey»: 76 сторінок, 34 рисунків та 1 таблиця, 37 літературних джерел.

Об’єкт дослідження – процес автентифікації користувачів у мережах з контролем доступу.

Мета роботи – підвищення рівня контролю доступу за рахунок безпарольної автентифікації користувачів у мережі Wi-Fi на основі технології passkey.

У межах цієї роботи досліджено сучасні методи автентифікації користувачів у мережах з контролем доступу та проаналізовано можливості використання безпарольної технології passkey. Запропоновано та реалізовано архітектуру, що поєднує pfSense Captive Portal, Keycloak і passkey для безпечної автентифікації у Wi-Fi мережі без використання паролів.

Наукова новизна: полягає у підвищенні рівня контролю доступу за рахунок безпарольної автентифікації користувачів у мережах з використанням технології passkey.

Актуальність теми: у сучасних умовах кіберзагроз автентифікація користувачів є критичним елементом інформаційної безпеки, особливо в мережах з обмеженим або контрольованим доступом, таких як корпоративні або публічні Wi-Fi. Традиційні методи, зокрема автентифікація за логіном і паролем, втрачають свою ефективність через численні вразливості – фішинг, брутфорс, витоки баз даних тощо.

У зв’язку з цим зростає потреба у впровадженні безпарольних технологій, зокрема passkey, що базується на відкритих криптографічних стандартах FIDO2/WebAuthn. Робота є своєчасною та актуальною з огляду на глобальний перехід до зручної та стійкої до атак автентифікації.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

2FA	-	Two-Factor Authentication
ACL	-	Access Control List
API	-	Application Programming Interface
FIDO2	-	Fast Identity Online 2
HTTPS	-	Hypertext Transfer Protocol Secure
IdP	-	Identity Provider
OIDC	-	OpenID Connect
OTP	-	One-Time Password
PK	-	Public Key
SK	-	Secret Key
SP	-	Service Provider
SSO	-	Single Sign-On
WebAuthn	-	Web Authentication
ІБ	-	Інформаційна безпека

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі питання безпеки доступу до інформаційних ресурсів займає одне з провідних місць у загальній стратегії кіберзахисту. З кожним роком зростає кількість атак, пов’язаних із компрометацією облікових записів, що, у свою чергу, підкреслює вразливість традиційних методів автентифікації, зокрема паролів. Паролі легко викрасти шляхом фішингових атак, їх можна підібрати або перехопити, а користувачі часто нехтують базовими принципами гігієни паролів. Ці фактори знижують ефективність паролів як механізму контролю доступу.

У відповідь на ці виклики все ширшого поширення набувають безпарольні технології автентифікації, які дозволяють підвищити як безпеку, так і зручність для користувачів. Однією з найперспективніших технологій у цій галузі є passkey — метод автентифікації, який базується на стандартах FIDO2 та WebAuthn і передбачає використання криптографічних пар ключів, що зберігаються на пристрої користувача або у хмарному середовищі з прив’язкою до біометричних даних. Такий підхід унеможливлює фішинг, виключає необхідність передачі чи збереження паролів на сервері та забезпечує високий рівень стійкості до атак.

В межах цієї кваліфікаційної роботи було досліджено можливості впровадження passkey у мережеву інфраструктуру з контролем доступу. Особливу увагу зосереджено на архітектурі автентифікації у Wi-Fi мережах із використанням Captive Portal, що діє як механізм обмеження доступу до мережі до моменту проходження перевірки користувача. Для реалізації безпарольної автентифікації було обрано платформу Keycloak як гнучкий і потужний інструмент керування ідентичністю, що підтримує протоколи OpenID Connect і WebAuthn.

У рамках практичної частини було реалізовано повноцінну систему автентифікації для відкритої Wi-Fi мережі. Користувач, підключаючись до точки доступу, автоматично перенаправляється через Captive Portal до спеціально розробленої проміжної програми,

яка перенаправляє його на сторінку авторизації Keycloak. Після успішної автентифікації за допомогою passkey IP-адреса користувача динамічно додається до списку дозволених на pfSense, що відкриває йому доступ до мережі.

Таким чином, ця практика дала змогу реалізувати практичну модель безпечного доступу до мережі з використанням сучасного механізму безпарольної автентифікації. Результати роботи можуть бути використані як основа для впровадження подібних рішень у корпоративному, навчальному чи публічному середовищі, сприяючи підвищенню рівня захисту інформаційних систем та зручності користувачів.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ АВТЕНТИФІКАЦІЇ В ІНФОРМАЦІЙНИХ СИСТЕМАХ

1.1 Сучасні методи авторизації користувачів

На сьогоднішній день існує велика кількість способів авторизації користувачів у системах, проте спочатку потрібно чітко розуміти термінологію.

Ідентифікація — це заява про те, ким ви є. Залежно від ситуації, це може бути ім'я, адреса електронної пошти, номер облікового запису, і тд.

Аутентифікація — надання доказів, що ви насправді є той, ким ідентифікувалися (від слова "authentic" — істинний, справжній) [1].

Авторизація — перевірка, що вам дозволено доступ до запитуваного ресурсу.

Наприклад, при спробі потрапити в закритий клуб вас ідентифікують (запитають ваше ім'я і прізвище), аутентифікують (попросять показати паспорт і звірять фотографію) і авторизують (перевірять, що прізвище знаходиться в списку гостей), перш ніж пустять усередину [2].

Аналогічно ці терміни вживаються в комп'ютерних системах, де традиційно під ідентифікацією розуміють отримання вашого профілю (identity) по username або email; під аутентифікацією — перевірку, що ви знаєте пароль від цього облікового запису, а під авторизацією - перевірку вашої ролі в системі і рішення про надання доступу до запитаної сторінці або ресурсу.

Розглянуто різні підходи до аутентифікація користувачів:

- аутентифікація по паролю;
- аутентифікація по сертифікатам;
- аутентифікація по ключам-доступу;
- аутентифікація по токенах.

1.2 Аутентифікація по паролю

Цей метод ґрунтується на тому, що користувач повинен надати username і password для успішної ідентифікації і аутентифікації в системі. Пара username / password задається користувачем при його реєстрації в системі, при цьому в якості username може виступати адреса електронної пошти користувача. Стосовно до веб-застосунків, існує кілька стандартних протоколів для аутентифікації по паролю, а саме: HTTP, Forms, URL query, Request body, HTTP header.

Описаний в стандартах HTTP 1.0 / 1.1, HTTP authentification використовується вже багато років і за цей час став корпоративним стандартом. Схема взаємодії з веб-сайтами:

1. Спочатку, неавторизований клієнт робить запит до захищеного ресурсу на сервері, той у свою чергу відповідає HTTP статусом "401 Unauthorized", також у заголовок додається "WWW-Authenticate", у якому міститься схема і параметри аутентифікації;
2. Отримавши таку відповідь, браузер автоматично виводить на екран користувача діалог введення username і password, куди користувач вводить дані свого облікового запису;
3. У подальших викликах на сервер даного захищеного ресурсу, браузер додає у заголовок запиту "Authorization", в якому знаходяться всі дані користувача, які необхідні для аутентифікації сервером;
4. Сервер проводить аутентифікацію користувачів спираючись на дані отримані з цього заголовку та робить рішення допускати до захищенного ресурсу користувача, в залежності від ролі, ACL або інших даних [3].

Весь процес стандартизований і добре підтримується всіма браузерами і веб-серверами.

Існує кілька схем аутентифікації, що відрізняються за рівнем безпеки:

- Basic - найбільш проста схема, при якій username і password користувача передаються в заголовку Authorization в незашифрованому вигляді (base64- encoded). Однак при використанні HTTPS протоколу, є відносно безпечною (рисунок 1.1);
- Digest - challenge-response-схема, при якій сервер посилає унікальне значення nonce, а браузер передає MD5 хеш пароля користувача, обчислений з використанням зазначеного nonce. Більш безпечна альтернатива Basic схеми при незахищених з'єднаннях, але схильна до man-in-the-middle attacks (з заміною схеми на basic). Крім того, використання цієї схеми не дозволяє застосувати сучасні хеш-функції для зберігання паролів користувачів на сервері;
- NTLM (відома як Windows authentication) - також заснована на challengeresponse підході, при якому пароль не передається в чистому вигляді. Ця схема не є стандартом HTTP, але підтримується більшістю браузерів і веб-серверів. Переважно використовується для аутентифікації користувачів Windows Active Directory в веб-додатках. Вразлива до pass-the-hash-атакам;
- Negotiate - ще одна схема з сімейства Windows authentication, яка дозволяє клієнтові вибрати між NTLM і Kerberos аутентифікації. Kerberos - більш безпечний протокол, заснований на принципі Single Sign-On. Однак він може функціонувати, тільки якщо і клієнт, і сервер знаходяться в зоні intranet і є частиною домену Windows.

Рисунок 1.1 – Схема Basic аутентифікації

Варто відзначити, що при використанні HTTP-аутентифікації у користувача немає стандартної можливості вийти з веб-додатки, крім як закрити всі вікна браузера.

Forms authentification протокол немає певного стандарту, тому всі його реалізації специфічні для конкретних систем, а точніше, для модулів аутентифікації фреймворків розробки. Працює за наступним принципом: в веб-застосунок 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-застосунок створює session token, який зазвичай поміщається в browser cookies. При наступних веб-запитах session token автоматично передається на сервер і дозволяє застосункам отримати інформацію про поточного користувача для авторизації запиту, що зображено на рисунку 1.2.

Рисунок 1.2 – Схема Forms аутентифікації

Застосунок може створити session token двома способами:

1. Як ідентифікатор аутентифицироваться сесії користувача, яка зберігається в пам'яті сервера або в базі даних. Сесія повинна містити всю необхідну інформацію про користувача для можливості авторизації його запитів;

2. Як зашифрований і/або підписаний об'єкт, що містить дані про користувача, а також вказано термін. Цей підхід дозволяє реалізувати stateless-архітектуру сервера, однак вимагає механізму поновлення сесійного токена після закінчення терміну дії. Кілька стандартних форматів таких токенів розглядаються в секції «Аутентифікація по токену».

Необхідно розуміти, що перехоплення session token часто дає аналогічний рівень доступу, що і знання username / password. Тому всі комунікації між клієнтом і сервером у разі forms authentication повинні проводитися тільки по захищеному з'єднанню HTTPS.

Інші протоколи аутентифікації по паролю. Два протоколу, описаних вище, успішно використовуються для аутентифікації користувачів на веб-сайтах. Але при розробці клієнт-серверних додатків з використанням веб-сервісів (наприклад, iOS або Android), поряд з HTTP аутентифікації, часто застосовуються нестандартні протоколи, в яких дані для аутентифікації передаються в інших частинах запиту. Існує всього декілька місць, де можна передати username і password в HTTP запитах:

- URL query - вважається небезпечним варіантом, тоді як рядки URL можуть запам'ятовуватися браузерами, проксі і веб-серверами;
- Request body - безпечний варіант, але він застосовується лише для запитів, що містять тіло повідомлення (такі як POST, PUT, PATCH);
- HTTP header-оптимальний варіант, при цьому можуть використовуватися і стандартний заголовок Authorization (наприклад, з Basic-схемою), та інші довільні заголовки.

За простотою реалізації даного методу ховається багато недоліків і тому даний спосіб вважається не дуже надійним. Паролі часто можна підібрати, а користувачі схильні використовувати прості і однакові паролі в різних системах, або записувати їх на клаптиках паперу. Якщо зловмисник зміг з'ясувати пароль, то користувач часто про це не дізнається. Крім того, розробники застосунків можуть допустити ряд концептуальних помилок, що спрощують злом облікових записів. Нижче представлений

список вразливостей, які зустрічаються найчастіше, в разі використання аутентифікації по паролю:

- веб-застосунок дозволяє користувачам створювати прості паролі;
- веб-застосунок не захищене від можливості перебору паролів (brute-force attacks);
- веб-застосунок саме генерує і поширює паролі користувачам, однак не вимагає зміни пароля після першого входу (тобто поточний пароль десь записаний);
- веб-застосунок допускає передачу паролів по незахищеному HTTP-з'єднання або в рядку URL;
- веб-застосунок не використовує безпечні хеш-функції для зберігання паролів користувачів;
- веб-застосунок не надає користувачам можливість зміни пароля або не нотифікує користувачів про зміну їх паролів;
- веб-застосунок використовує вразливу функцію відновлення пароля, яку можна використовувати для отримання несанкціонованого доступу до інших облікових записів;
- веб-застосунок не вимагає повторної аутентифікації користувача для важливих дій: зміна пароля, зміни адреси доставки товарів і т.п.;
- веб-застосунок створює session tokens таким чином, що вони можуть бути підібрані або передбачені для інших користувачів;
- веб-застосунок допускає передачу session tokens по незахищеному HTTPз'єднання, або в рядку URL;
- веб-застосунок вразливе для session fixation-атак (не замінює session token при переході анонімної сесії користувача в аутентифицироваться);
- веб-застосунок не встановлює прапори HttpOnly і Secure для browser cookies, що містять session tokens;

- веб-застосунок не знищує сесії користувача після короткого періоду неактивності або не надає функцію виходу з аутентифицироваться сесії.

1.3 Аутентифікація по сертифікатам

Сертифікат являє собою набір атрибутів, що ідентифікують власника, підписаний certificate authority (CA). CA виступає в ролі посередника, який гарантує справжність сертифікатів. Також сертифікат криптографічно пов'язаний з закритим ключем, яких зберігається у власника сертифіката і дозволяє однозначно підтвердити факт володіння сертифікатом. На стороні клієнта сертифікат разом з закритим ключем можуть зберігатися в операційній системі, в браузері, в файлі, на окремому фізичному пристрої (smart card, USB token). Зазвичай закритий ключ додатково захищений паролем або PIN-кодом. У веб-застосунках традиційно використовують сертифікати стандарту X.509. Аутентифікація за допомогою X.509-сертифіката відбувається в момент з'єднання з сервером і є частиною протоколу SSL / TLS. Цей механізм також добре підтримується браузерами, які дозволяють користувачеві вибрати і застосувати сертифікат, якщо веб-сайт допускає такий спосіб аутентифікації (рисунок 1.3).

Рисунок 1.3 – Схема аутентифікації по сертифікатах

Під час аутентифікації сервер виконує перевірку сертифіката на підставі наступних правил:

- Сертифікат повинен бути підписаний довіреною certification authority (перевірка ланцюжка сертифікатів).
- Сертифікат повинен бути дійсним на поточну дату (перевірка терміну дії).
- Сертифікат не повинен бути відкликаний відповідним CA (перевірка списків виключення).

Після успішної аутентифікації веб-застосунок 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 - куди більш надійний спосіб, ніж аутентифікація за допомогою паролів. Це досягається створенням в процесі аутентифікації цифрового підпису, наявність якої доводить факт застосування закритого ключа в конкретній ситуації (non-repudiation). Однак труднощі з поширенням і підтримкою сертифікатів робить такий спосіб аутентифікації малодоступним в широких колах.

Аутентифікація за одноразовими паролями зазвичай застосовується додатково до аутентифікації по паролів для реалізації two-factor authentication (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 в значній мірі збільшити рівень безпеки, що може вимагатись для певних видів веб-застосунків.

Інший популярний сценарій використання одноразових паролів - додаткова аутентифікація користувача під час виконання важливих дій: переказ грошей, зміна налаштувань і т.п. Існують різні джерела для створення одноразових паролів. Найбільш популярні:

- апаратні або програмні маркери, які можуть генерувати одноразові паролі на підставі секретного ключа, введеного в них, і поточного часу. Секретні ключі користувачів, які є фактором володіння, також зберігаються на сервері, що дозволяє виконати перевірку введених одноразових паролів. Приклад апаратної реалізацій токенов - RSA SecurID; програмної - додаток Google Authenticator;
- випадково генеруються коди, що передаються користувачеві через SMS або інший канал зв'язку. У цій ситуації фактор володіння - телефон користувача (точніше - SIM-карта, прив'язана до певного номера);
- роздруківка або scratch card зі списком заздалегідь сформованих одноразових паролів. Для кожного нового входу в систему потрібно ввести новий одноразовий пароль з зазначеним номером.

У веб-застосунках такий механізм аутентифікації часто реалізується за допомогою розширення forms authentication: після первинної аутентифікації по паролю, створюється сесія користувача, проте в контексті цієї сесії користувач не має доступу до додатка до тих пір, поки він не виконає додаткову аутентифікацію за одноразовим паролем.

1.4 Аутентифікація по ключам доступу

Цей спосіб найчастіше використовується для аутентифікації пристроїв, сервісів або інших застосунків при зверненні до веб-сервісів.

Тут в якості секрету застосовуються ключі доступу (access key, API key) - довгі унікальні рядки, що містять довільний набір символів, по суті замінюють собою комбінацію username / password.

У більшості випадків, сервер генерує ключі доступу за запитом користувачів, які далі зберігають ці ключі в клієнтських додатках.

При створенні ключа також можливо обмежити термін дії і рівень доступу, який отримає клієнтську програму при аутентифікації за допомогою цього ключа.

Гарний приклад застосування аутентифікації по ключу - хмара Amazon Web Services.

Припустимо, у користувача є веб-застосунок, що дозволяє завантажувати і переглядати фотографії, і він хоче використовувати сервіс Amazon S3 для зберігання файлів.

В такому випадку, користувач через консоль AWS може створити ключ, що має обмежений доступ до хмари: тільки читання / запис його файлів в Amazon S3.

Цей ключ в результаті можна застосувати для аутентифікації веб-додатки в хмарі AWS. (рисунок 1.4) [4].

Рисунок 1.4 – Схема аутентифікації по ключу

Використання ключів дозволяє уникнути передачі пароля користувача стороннім додаткам (в прикладі вище користувач зберіг в веб-додатку не свій пароль, а ключ доступу). Ключі мають значно більшу ентропією в порівнянні з паролями, тому їх

практично неможливо підібрати. Крім того, якщо ключ був розкритий, це не призводить до компрометації основний облікового запису користувача - достатньо лише анулювати цей ключ і створити новий.

З технічної точки зору, тут не існує єдиного протоколу: ключі можуть передаватися в різних частинах HTTP-запиту: URL query, request body або HTTP header. Як і в випадку аутентифікації по паролю, найбільш оптимальний варіант - використання HTTP header. У деяких випадках використовують HTTP-схему Bearer для передачі токена в заголовку (Authorization: Bearer [token]). Щоб уникнути перехоплення ключів, з'єднання з сервером має бути обов'язково захищене протоколом SSL / TLS. (рисунок 1.5)

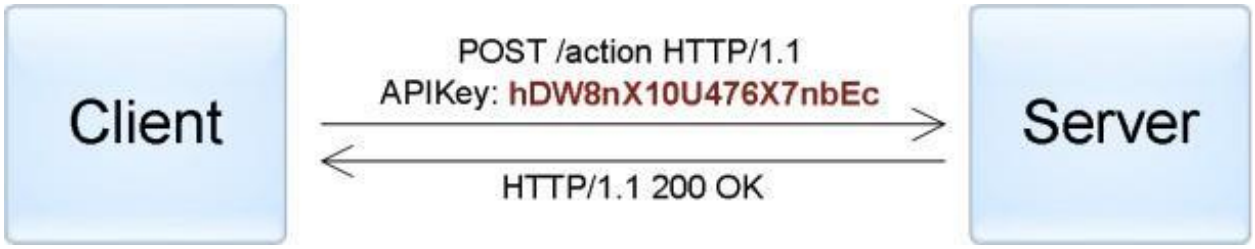


Рисунок 1.5 – Схема аутентифікації по ключу доступу, переданого в HTTP заголовку

Крім того, існують більш складні схеми аутентифікації по ключам для незахищених з'єднань. В цьому випадку, ключ зазвичай складається з двох частин: публічної і приватної. Публічна частина використовується для ідентифікації клієнта, а приватна частина дозволяє згенерувати, так званий електронний підпис, що використовується для декодування запиту направленного на сервер.

Наприклад, за аналогією з digest authentication схемою, сервер може послати 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 і захищає від атак, що роблять повторні запити на сервер.

1.5 Аутентифікація по токенах

Найчастіше застосовується при побудові розподілених систем Single Sign-On (SSO), де один застосунок (service provider або relying party) делегує функцію аутентифікації користувачів іншому застосун (identity provider або authentication service). Типовий приклад цього способу - вхід в застосунок через обліковий запис в соціальних мережах.

Тут соціальні мережі є сервісами аутентифікації, а додаток довіряє функцію аутентифікації користувачів соціальних мереж.

Реалізація цього способу полягає в тому, що identity provider (IP) надає достовірні відомості про користувача в вигляді токена, а service provider (SP) додаток використовує цей токен для ідентифікації, аутентифікації і авторизації користувача (рисунок 1.6) [5].

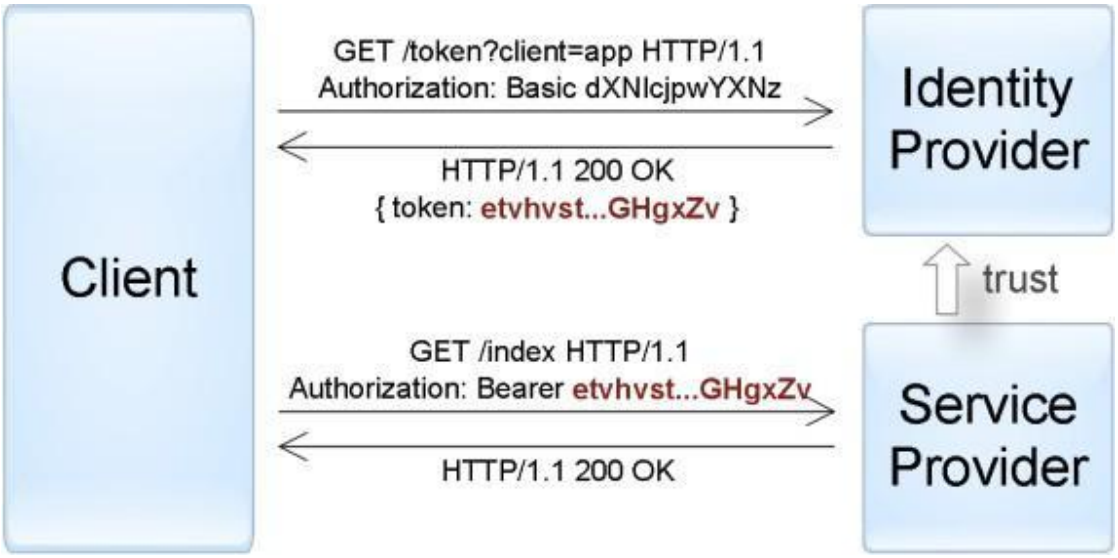


Рисунок 1.6 – Схема аутентифікації “активного” клієнта за допомогою токена

На загальному рівні, весь процес виглядає наступним чином:

- клієнт, який аутентифікує в identity provider одним із способів, специфічним для нього (пароль, ключ доступу, сертифікат, Kerberos, ітд.);

- клієнт просить identity provider надати йому токен для конкретного SPдодатки. Identity provider генерує токен і відправляє його клієнту;
- клієнт, який аутентифікує в SP-застосунку за допомогою цього токена.

Процес, описаний вище, відображає механізм аутентифікації активного клієнта, тобто такого, який може виконувати запрограмовану послідовність дій (наприклад, iOS / Android програми). Браузер ж — пасивний клієнт в тому сенсі, що він тільки може відображати сторінки, запитані користувачем. В цьому випадку аутентифікація досягається за допомогою автоматичного перенаправлення браузера між веб-додатками identity provider і service provider.

Існує кілька стандартів, в точності що визначають протокол взаємодії між клієнтами (активними і пасивними) і IP / SP-додатками і формат підтримуваних токенов. Серед найбільш популярних стандартів - OAuth, OpenID Connect, SAML, і WS-Federation. (рисунок 1.7)

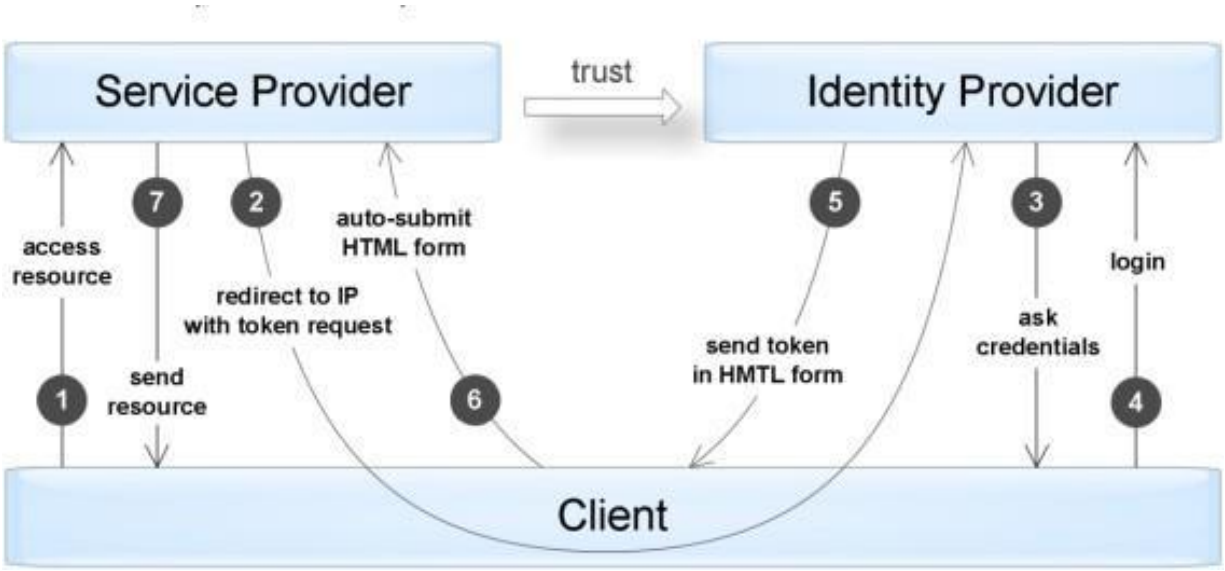


Рисунок 1.7 – Схема аутентифікації “пасивного” клієнта за допомогою токена

Сам токен зазвичай являє собою структуру 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 і гарантій автентичності.

1.6 Проблеми паролів у сучасних інформаційних системах

У сучасному цифровому середовищі паролі залишаються одним із найпоширеніших методів автентифікації, проте їх використання породжує низку значних проблем, які створюють ризики для безпеки даних як для користувачів, так і для організацій. Основні виклики, пов’язані з паролями, полягають у їхній вразливості до різних видів атак, незручності у використанні, а також необхідності управління великою кількістю облікових записів.

По-перше, вразливість до атак є однією з найбільших загроз для безпеки паролів. В сучасних умовах хакери використовують різні методи для отримання доступу до паролів, включаючи фішинг, атаки типу brute-force (груба сила), та перехоплення даних [6]. Фішинг, зокрема, є дуже поширеним методом, який дозволяє зловмисникам отримувати конфіденційну інформацію, вводячи користувачів в оману через підроблені веб-сайти або електронні листи. Це призводить до витоків даних, що може мати катастрофічні наслідки як для окремих осіб, так і для великих організацій.

Наступною проблемою є повторне використання паролів на різних інформаційних системах. Багато користувачів, згідно дослідження Google – більше половини, використовують однакові або подібні паролі для кількох облікових записів, що значно збільшує ризик компрометації [7]. Якщо один обліковий запис зламано, зловмисники можуть легко отримати доступ до інших сервісів, що використовують той самий пароль. Така практика є небезпечною, особливо в корпоративних середовищах, де доступ до важливих даних може бути здійснений через скомпрометований особистий обліковий запис працівника.

Окрім ризиків з безпеки, паролі створюють суттєві проблеми з точки зору зручності та користувацького досвіду. Користувачам часто доводиться запам'ятовувати

велику кількість складних паролів для різних облікових записів, що є незручним та призводить до використання слабких або легко передбачуваних паролів. Це також сприяє використанню таких небезпечних практик, як записування паролів або зберігання їх у незахищених місцях.

Регулярна зміна паролів, яка рекомендована багатьма організаціями, також додає незручностей для користувачів. Часті зміни паролів часто призводять до того, що користувачі використовують незначні варіації старих паролів або записують нові паролі, щоб уникнути їх забування, що знижує загальну безпеку системи.

З точки зору організацій, управління паролями стає ще однією проблемою. Для забезпечення належного рівня безпеки компанії повинні впроваджувати політики складності паролів, частого їх оновлення та багатофакторної автентифікації. Проте, навіть за цих умов, злом облікових записів через скомпрометовані паролі залишається поширеною загрозою.

Зрештою, економічний аспект також є важливим чинником. Витрати на відновлення після компрометації даних через втрачені або викрадені паролі можуть бути значними. Згідно зі звітами про інциденти кібербезпеки, паролі часто є слабкою ланкою, яка веде до масштабних витоків даних та втрат репутації організацій.

Таким чином, проблема паролів у сучасних інформаційних системах є важливою як з точки зору кібербезпеки, так і з позиції зручності користувачів.

Рішення, засновані на використанні безпарольної автентифікації, такі як WebAuthn та FIDO2, відкривають нові можливості для захисту систем від зловмисників, підвищуючи рівень безпеки та зменшуючи залежність від людського фактора. Впровадження цих технологій є кроком до побудови більш надійних і зручних для користувачів систем автентифікації, що робить цю тему надзвичайно актуальною для дослідження та розробки.

1.7 Висновки до першого розділу

У першому розділі було здійснено комплексний огляд сучасних методів автентифікації користувачів, розглянуто їх класифікацію, принципи роботи та ключові переваги й недоліки. Особливу увагу приділено традиційним способам автентифікації — зокрема, автентифікації за паролем, сертифікатами, ключами доступу, токенами та одноразовими паролями.

Проведений аналіз показав, що паролі, попри широке використання, залишаються найбільш вразливим елементом у системах автентифікації. Вони піддаються фішинговим атакам, зламу методом brute-force, повторно використовуються на різних ресурсах і вимагають складного управління як з боку користувачів, так і адміністраторів. Крім того, розробники часто припускаються типових помилок, що ще більше послаблює безпеку паролів. Навіть додаткові засоби захисту, як-от двофакторна автентифікація, лише частково вирішують проблему.

Інші методи, такі як автентифікація за сертифікатами або токенами, є більш безпечними, однак мають обмеження у застосуванні через складність налаштування, підтримки або інтеграції.

У контексті вищезгаданого, виявлена необхідність у переході до нових, безпечніших підходів, які виключають людський фактор і не вимагають збереження або передачі секретної інформації. Такий підхід реалізований у безпарольних технологіях автентифікації, серед яких технологія passkey (WebAuthn/FIDO2) вирізняється своєю стійкістю до атак, зручністю для кінцевого користувача та підтримкою з боку сучасних платформ.

Отже, дослідження сучасних механізмів автентифікації обґрунтовує актуальність впровадження безпарольних рішень у системи з контрольованим доступом, що і є предметом подальшого дослідження в наступних розділах роботи.

РОЗДІЛ 2
СУЧАСНІ ТЕХНОЛОГІЇ БЕЗПАРОЛЬНОЇ АВТЕНТИФІКАЦІЇ ТА ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Безпарольна авторизація

Безпарольна автентифікація перевіряє особу користувача без необхідності вводити ім'я користувача та пароль. Замість того, щоб вимагати пароль, сучасна або безпарольна автентифікація використовує більш безпечні альтернативи для перевірки ваших облікових даних, наприклад біометрія.

2.2 Стандарт FIDO

Стандарт FIDO (Fast Identity Online) є важливою технологією для безпарольної автентифікації, яка допомагає зменшити залежність від паролів та підвищити безпеку в інформаційних системах [22]. Створений у 2012 році, FIDO Alliance об'єднав ключових гравців в індустрії для розробки відкритих і сумісних технічних специфікацій, що дозволяють використовувати біометричні методи автентифікації та криптографію з відкритими ключами. Цей стандарт був заснований на основі ідеї, що паролі є небезпечними та незручними, і вимагає заміни на більш безпечні та зручні рішення. Принцип роботи ключів зображено на рисунку 2.1.

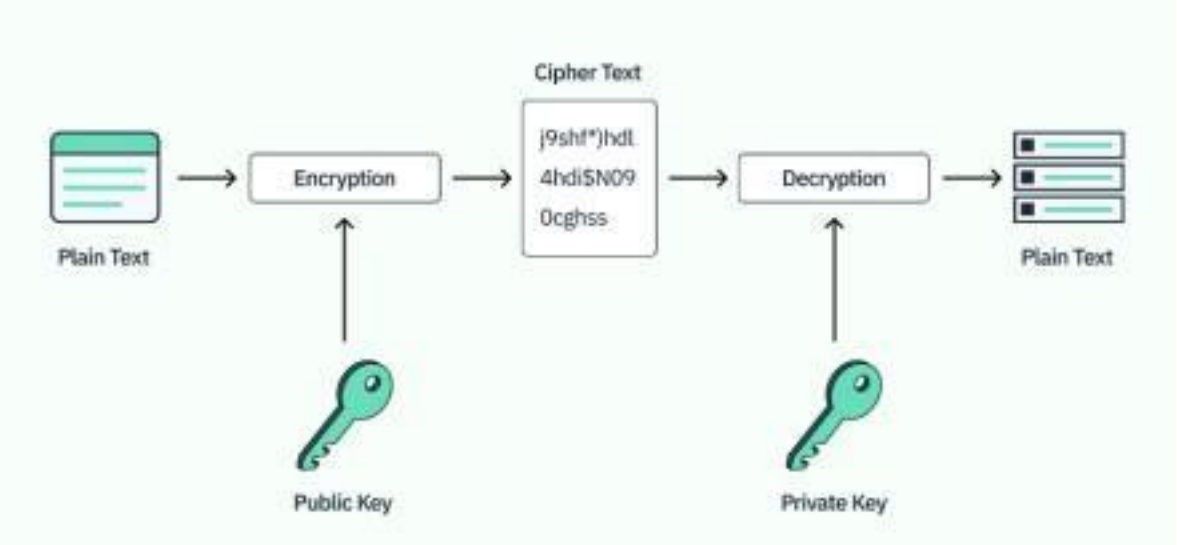


Рисунок 2.1 – Абстрактна діаграма роботи ключів в стандарті FIDO

За рисунком 2.1, автентифікація за стандартом FIDO використовує пару ключів — приватний ключ зберігається на пристрої користувача, а публічний ключ передається серверу. Приватний ключ використовується для підписання викликів (challenge), які сервер відправляє для перевірки автентичності користувача. Завдяки цьому механізму сервер може впевнитися, що користувач є тим, за кого себе видає, без необхідності збереження пароля на сервері.

Оскільки приватний ключ зберігається на пристрої користувача або в апаратному модулі безпеки, процес автентифікації стає локалізованим і більш захищеним від віддалених атак. Для підтвердження особи користувач може використовувати біометричні дані, такі як відбитки пальців або розпізнавання обличчя, що робить процес зручним і безпечним.

FIDO Alliance не лише розробляє технічні специфікації, а й сертифікує продукти, що використовують ці стандарти, забезпечуючи їх якість і відповідність вимогам безпеки. Усі специфікації FIDO також подаються для стандартизації через організації, що сприяє популяризації безпарольної автентифікації серед розробників та компаній.

2.3 Стандарт WebAuthn

WebAuthn (Web Authentication) – це стандарт, розроблений консорціумом W3C, який забезпечує безпечну автентифікацію користувачів без використання паролів.

WebAuthn дозволяє веб-застосункам автентифікувати користувачів через асиметричні криптографічні ключі замість традиційних паролів. Процес автентифікації відбувається через створення пари ключів: приватного та публічного. Приватний ключ зберігається на пристрої користувача (наприклад, на токені або апаратному ключі), а публічний ключ передається на сервер [8].

Процес реєстрації користувача передбачає створення нових облікових даних на основі наданого імені користувача. Це основа безпарольного входу, який підтримує WebAuthn. Його можна детальніше розглянути на рисунку 2.2.

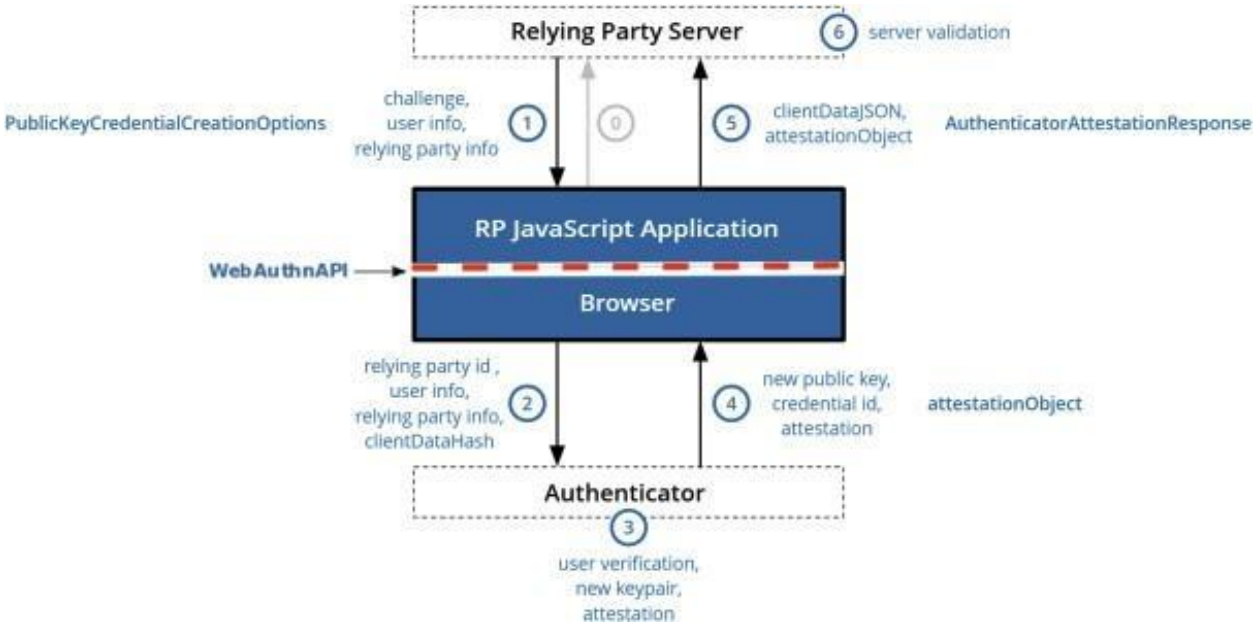


Рисунок 2.2 – Послідовна діаграма процесу реєстрації на основі WebAuthn

Розглянемо опис послідовності кроків на діаграмі:

1. Ініціація процесу реєстрації. Користувач заходить на веб-сайт, який підтримує WebAuthn, і натискає кнопку "Зареєструватися". Ця дія ініціює запит до

сервера автентифікації (Relying Party), який відповідає за обробку запитів на реєстрацію автентифікаторів.

2. Генерація виклику сервером. Сервер створює випадковий набір даних, відомий як "виклик" (challenge), який використовується для унікальності запиту. Разом із викликом сервер формує об'єкт PublicKeyCredentialCreationOptions, який містить інформацію про сервер (наприклад, доменне ім’я), ідентифікатор користувача, підтримувані криптографічні алгоритми, тайм-аут і налаштування автентифікатора. Цей об’єкт надсилається браузеру користувача.

3. Передача даних до браузера. Браузер отримує дані від сервера і передає їх локальному автентифікатору або зовнішньому апаратному пристрою (наприклад, YubiKey, Touch ID, Face ID).

4. Взаємодія з автентифікатором. Автентифікатор запитує підтвердження дій від користувача. Наприклад, користувач може ввести PIN-код, надати скан відбитка пальця або скористатися розпізнаванням обличчя. Цей крок гарантує, що операцію ініціює саме користувач.

5. Генерація ключів автентифікатором. Після успішного підтвердження автентифікатор створює пару ключів. Приватний ключ залишається збереженим на пристрої користувача. Публічний ключ і додаткові атестаційні дані передаються браузеру для подальшої обробки.

6. Формування атестації. Автентифікатор формує атестаційний об'єкт (attestationObject), який містить публічний ключ, інформацію про автентифікатор (тип, модель, виробника) і цифровий підпис. Цей об'єкт разом із даними браузера (clientDataJSON) передається на сервер.

7. Передача даних на сервер. Браузер відправляє дані автентифікації на сервер, включаючи атестаційний об'єкт, публічний ключ та унікальний ідентифікатор автентифікатора (id).

8. Перевірка атестації сервером. Сервер перевіряє відповідність виклику (challenge) отриманим даним, дійсність підпису, використовуючи атестаційні дані та

інформацію про автентифікатор, щоб переконатися у його легітимності. Після успішної перевірки сервер зберігає публічний ключ, ідентифікатор користувача і пов’язані метадані в базі даних.

9. Завершення реєстрації. Після успішної перевірки сервер повідомляє користувача, що процес реєстрації завершено. Тепер користувач може використовувати автентифікатор для майбутніх сеансів входу в систему.

Цей процес забезпечує безпеку, оскільки приватний ключ залишається на пристрої користувача і ніколи не передається через мережу, а сервер зберігає лише публічний ключ, необхідний для перевірки.

Процес автентифікації можна підсумувати, як зазначено на рисунку 2.3.

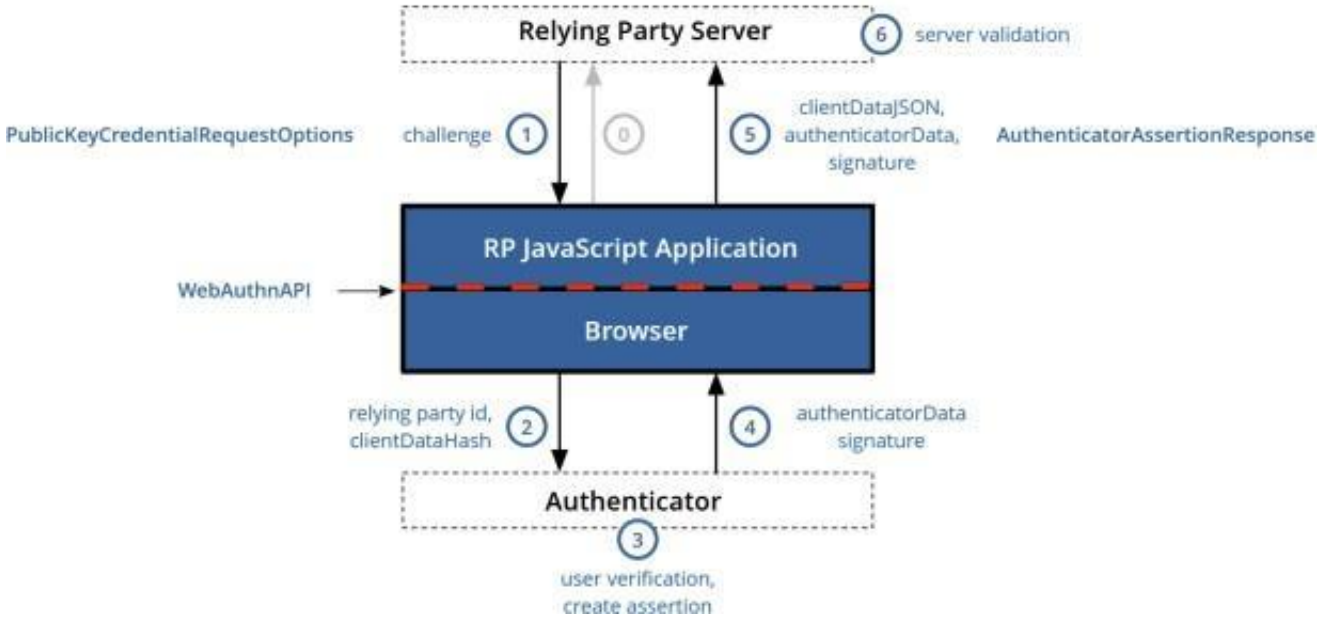


Рисунок 2.3 – Абстрактний опис процесу автентифікації на основі стандарту WebAuthn

Розглянемо опис послідовності кроків на діаграмі:

1. Ініціація процесу автентифікації. Користувач заходить на веб-сайт і натискає кнопку "Увійти". Ця дія створює запит до сервера автентифікації (Relying

Party), який перевіряє, чи існує обліковий запис користувача, і ініціює процес автентифікації.

2. Генерація виклику сервером. Сервер генерує випадковий набір даних, відомий як "виклик" (challenge), який забезпечує унікальність запиту та запобігає повторним атакам. Сервер також відправляє інформацію про обліковий запис користувача (наприклад, ідентифікатор зареєстрованог автентифікатора) разом із об’єктом PublicKeyCredentialRequestOptions, який включає: challenge – унікальний виклик для цього сеансу, rpId – ідентифікатор сервера (домен), allowCredentials – список ідентифікаторів автентифікаторів, які можна використовувати для автентифікації, timeout – час очікування відповіді, userVerification – вимога до перевірки користувача (наприклад, біометрія або PIN).

3. Передача даних клієнту. Сервер передає об’єкт PublicKeyCredentialRequestOptions до браузера або клієнтського додатка. Браузер ініціює виклик автентифікатора через WebAuthn API.

4. Запит до автентифікатора. Браузер надсилає отримані дані локальному автентифікатору або зовнішньому пристрою (наприклад, YubiKey, Touch ID, Face ID). Автентифікатор ідентифікує користувача та запитує підтвердження дії.

5. Перевірка користувача автентифікатором. Користувач підтверджує свою особу за допомогою біометричних даних (відбиток пальця, розпізнавання обличчя тощо), ведення PIN-коду чи використання фізичного пристрою (наприклад, торкання до YubiKey).

6. Генерація підпису автентифікатором. Автентифікатор використовує зареєстрований приватний ключ для створення цифрового підпису на основі виклику (challenge) від сервера, даних клієнта (clientDataJSON), інших специфічних параметрів сеансу.

7. Повернення відповіді клієнту. Автентифікатор передає браузеру підписані дані разом із додатковою інформацією: authenticatorData – дані про автентифікатор, signature – цифровий підпис та clientDataJSON – дані, отримані від браузера.

8. Передача відповіді на сервер. Браузер надсилає ці дані назад на сервер. Сервер отримує об'єкт PublicKeyCredential, що включає ідентифікатор автентифікатора (id), дані підпису (authenticatorData, signature) та JSON-дані клієнта (clientDataJSON).

9. Перевірка підпису сервером. Сервер порівнює отриманий виклик (challenge) із тим, що було згенеровано під час початку сеансу. Далі сервер перевіряє цифровий підпис, використовуючи публічний ключ, збережений під час реєстрації. В кінці аналізує authenticatorData для підтвердження коректності відповіді автентифікатора.

10. Завершення автентифікації: Якщо всі перевірки успішно пройдено, сервер підтверджує автентифікацію користувача. Користувач отримує доступ до системи або захищених ресурсів.

Після аналізу діаграм, що відображають структуру та процеси автентифікації за допомогою WebAuthn, варто звернути увагу на загальні переваги й недоліки цього стандарту.

WebAuthn пропонує значні переваги, особливо з точки зору безпеки та зручності. Найбільш помітною перевагою є використання асиметричної криптографії, яка забезпечує високий рівень захисту. Оскільки приватний ключ зберігається на апаратному автентифікаторі, а публічний ключ передається на сервер, доступ до облікового запису захищений навіть у разі компрометації сервера [9].

Крім того, WebAuthn значно підвищує стійкість до фішингових атак, оскільки доменне ім’я, необхідне для автентифікації, зберігається на апаратному токені, а не на віртуальному сервері, що унеможливлює підробку. Також WebAuthn зменшує ризики пов'язані з витоком конфіденційних даних, оскільки система використовує лише публічні ключі і не зберігає біометричну інформацію на сервері.

Додатковою перевагою є сумісність з багатьма браузерами і пристроями, що забезпечує гнучкість та уникнення залежності від одного постачальника. Користувачі можуть самі обирати засоби автентифікації, такі як відбитки пальців або апаратні токени, що підвищує зручність та контроль над процесом автентифікації. Крім того,

відсутність паролів робить взаємодію з системою менш стресовою для користувачів, оскільки їм не потрібно запам'ятовувати або керувати складними паролями.

Попри численні переваги, WebAuthn має і деякі суттєві недоліки. Обмежена крос-пристроєва сумісність є одним із головних викликів – користувачі не можуть легко переносити облікові дані з одного автентифікатора на інший. Це означає, що у випадку втрати або пошкодження автентифікатора може виникнути серйозна проблема з відновленням доступу до облікового запису.

Окрім того, високі витрати на впровадження можуть бути перешкодою для багатьох організацій, оскільки потрібні апаратні пристрої, а також технічний персонал для їх обслуговування [10]. Також цей стандарт вимагає високого рівня технічної компетенції як від користувачів, так і від організацій для забезпечення належного функціонування та безпеки [11].

У розділі розглянуто ключові аспекти проблеми використання паролів у сучасних інформаційних системах, їхні основні вразливості та ризики, які виникають через залежність від цього методу автентифікації. Паролі довели свою ненадійність через вразливість до атак, зокрема фішингу, brute-force, і повторного використання, що створює суттєві загрози для безпеки даних як для окремих користувачів, так і для організацій. Ці проблеми підкреслюють необхідність впровадження альтернативних методів автентифікації, які забезпечують як зручність, так і підвищений рівень захисту.

Аналіз сучасних стандартів, таких як FIDO2 і WebAuthn, демонструє їх потенціал у вирішенні цих проблем. Завдяки використанню асиметричної криптографії, ці технології дозволяють зберігати приватні ключі на пристроях користувачів, що усуває залежність від паролів і знижує ризики компрометації даних. Особлива увага приділена WebAuthn, який забезпечує сумісність з широким спектром браузерів і пристроїв, дозволяючи користувачам використовувати біометричні дані або апаратні токени для автентифікації.

Попри численні переваги, впровадження безпарольних методів вимагає вирішення певних викликів, зокрема щодо сумісності між пристроями, вартості реалізації та

необхідності високого рівня технічної компетенції. Тим не менш, переваги у вигляді стійкості до фішингових атак, підвищення зручності для користувачів і забезпечення конфіденційності даних значно переважають.

Таким чином, безпарольна автентифікація на основі промислових стандартів є важливим кроком у напрямку до створення безпечних і зручних для користувачів інформаційних систем. Подальші дослідження та моделювання цих систем дозволять вдосконалити їхню інтеграцію в сучасні платформи та сприяти їхньому широкому впровадженню.

2.4 Passkey

Основою будь-якої системи безпарольної автентифікації є асиметрична криптографія, яка забезпечує надійну автентифікацію без необхідності використання паролів. Ключі passkeys (або ключі доступу) — це метод, який застосовує пару криптографічних ключів: приватного, що зберігається на пристрої користувача, та публічного, що зберігається на сервері [12]. Під час автентифікації приватний ключ використовується для підписання запиту, а сервер перевіряє цей підпис за допомогою публічного ключа. Такий механізм гарантує, що дані користувача захищені на всіх етапах взаємодії .

Passkeys можуть бути двох типів:

1. Синхронізовані (synced) passkeys. Зберігаються на пристрої користувача та синхронізуються між його пристроями через захищені хмарні сервіси. Це дозволяє користувачу використовувати один passkey на різних пристроях, забезпечуючи зручність та безпеку [13].

2. Вбудовані (device-bound) passkeys. Зберігаються локально на конкретному пристрої та не синхронізуються з іншими пристроями. Вони забезпечують високий рівень безпеки, але обмежують використання лише на одному пристрої [14].

У контексті безпарольної автентифікації використовуються два типи автентикаторів:

1. Платформні автентикатори (platform authenticators). Вбудовані в пристрій користувача, такі як сканери відбитків пальців або розпізнавання обличчя на смартфонах чи ноутбуках [15]. Вони забезпечують зручну автентифікацію без додаткових пристроїв, але обмежені конкретною платформою.

2. Роумінгові автентикатори (roaming authenticators): зовнішні пристрої, такі як апаратні токени (наприклад, YubiKey), які можуть підключатися до різних пристроїв через USB, NFC або Bluetooth. Вони забезпечують високий рівень безпеки та універсальність, дозволяючи користувачу автентифікуватися на різних пристроях та платформах [16].

Приватний ключ є таємним і ніколи не покидає пристрій користувача, що знижує ймовірність його компрометації. Наприклад, у системах на основі WebAuthn або FIDO2 цей ключ зберігається або в апаратному токені (наприклад, YubiKey), або в захищеній області пристрою, яка відповідає за зберігання ключів і інші конфіденційні дані (наприклад, Trusted Platform Module – TPM на комп’ютері або Secure Enclave на смартфонах) [17].

Публічний ключ передається серверу під час реєстрації користувача. Коли користувач ініціює автентифікацію, сервер генерує випадковий набір даних, який відправляється назад на пристрій користувача. Пристрій підписує цей набір даних приватним ключем і відправляє підпис назад на сервер. Сервер, використовуючи збережений публічний ключ, перевіряє підпис і, якщо він є коректним, надає доступ користувачу.

Такий підхід забезпечує надійний захист, оскільки навіть у випадку компрометації сервера, зловмисники не зможуть отримати доступ до приватних ключів користувачів. Це робить системи безпарольної автентифікації стійкими до багатьох атак, таких як фішинг або атаки типу "людина посередині" (Man-in-theMiddle) [18].

Крім того, в таких системах для автентифікації можуть використовуватись біометричні дані (відбитки пальців, розпізнавання обличчя тощо). Однак важливо зазначити, що ці дані також зберігаються лише на пристрої користувача і ніколи не передаються на сервер, що забезпечує додатковий рівень захисту [19].

У системах безпарольної автентифікації також використовується метод "магічних посилань" (Magic Links). Цей підхід передбачає, що після введення користувачем своєї електронної адреси або імені користувача на порталі входу, система надсилає на вказану адресу одноразове посилання [20]. Користувач, перейшовши за цим посиланням, автоматично входить до свого облікового запису без необхідності введення пароля. Такий метод спрощує процес автентифікації, усуваючи потребу в запам'ятовуванні паролів, і може використовуватися як основний спосіб входу або як додатковий фактор автентифікації [21].

Однак, варто зазначити, що безпека цього методу залежить від надійності електронної пошти користувача, оскільки компрометація поштової скриньки може призвести до несанкціонованого доступу.

Хоча магічні посилання пропонують зручний спосіб автентифікації без паролів, вони мають певні обмеження щодо безпеки та залежності від доступу до електронної пошти або телефону. Passkeys, завдяки використанню асиметричної криптографії та можливості інтеграції з біометричними методами, забезпечують вищий рівень безпеки та зручності для користувачів. Вони стійкі до фішингових атак і не потребують доступу до додаткових пристроїв або сервісів під час автентифікації. Крім того, passkeys можуть синхронізуватися між різними пристроями користувача, що підвищує зручність використання. Тому, для розробки сучасних систем безпарольної автентифікації, passkeys є більш надійним та перспективним вибором.

2.5 Огляд Keycloak

Keycloak - це програмне забезпечення з відкритим кодом, яке дозволяє єдиний вхід (IdP) за допомогою управління ідентифікацією та управління доступом для сучасних додатків та послуг. Це програмне забезпечення написане на Java та підтримує протоколи федерації ідентифікації за замовчуванням SAMLv2 та OpenID Connect (OIDC) / OAuth2. Він ліцензований Apache та підтримується Red Hat.

З концептуальної точки зору, мета інструменту - полегшити захист програм і служб із незначним шифруванням або взагалі без нього. IdP дозволяє програмі (яку часто називають Постачальником послуг або СП) делегувати свою автентифікацію.

Також має ряд інших переваг:

- Дозволяє розробникам зосередитись на бізнес-функціональності, не турбуючись про аспекти безпеки автентифікації, або безпосередньо інтегруючи бібліотеку, яка підтримує один із двох протоколів, або використовуючи модуль на веб-сервері.
- Вміє централізувати автентифікацію і, отже, увімкнути автентифікацію єдиного входу (SSO).
- Вміє уніфікувати методи автентифікації та змусити їх розвиватися, не змінюючи програми.

Також в межах своїх основних характеристик, виділяються наступні моменти:

- єдиний вхід;
- підтримка стандартних протоколів;
- захищені облікові записи та спрощене обслуговування;
- LDAP сумісний як зовнішнє сховище користувачів;
- висока продуктивність: кластер серверів, масштабований, висока доступність;
- повністю сумісний з контейнеризацією;
- прості теми для реалізації;

- потужна автентифікація за допомогою власного одноразового коду (OTP) за допомогою FreeOTP або Google Authenticator;
- автоматичне усунення несправностей, якщо ви забули пароль;
- автоматичне створення облікових записів (за формою або так званою соціальною автентифікацією).

Клієнтські адаптери Keycloak доступні для ряду платформ та мов програмування, наприклад Java, C#, Javascript, Python, Android, iOS та інших. Але якщо для обраної платформи немає жодного, Keycloak побудований на стандартних протоколах, тому можна використовувати будь-яку бібліотеку ресурсів OpenID Connect або бібліотеку постачальника послуг SAML 2.0.

Keycloak пропонує авторизацію на основі ролей, а також надає детальні послуги авторизації на основі атрибутів. Керувати дозволами для всіх своїх служб можна за допомогою консолі адміністратора, де надається можливість визначати потрібні політики доступу.

Keycloak підтримує чіткі політики авторизації та може поєднувати різні механізми контролю доступу, такі як:

- контроль доступу на основі атрибутів (ABAC);
- рольовий контроль доступу (RBAC);
- контроль доступу на основі списків доступу (ACL);
- контроль доступу на основі контексту (CBAC);
- контроль доступу на основі правил;
- використання JavaScript;
- контроль доступу за часом.

Перевага Keycloak в тому, що це опенсорс проект, має сумісність з більшістю мов програмування, але ця система розрахована здебільшого на досвідчених користувачів, тому що поріг входу досить високий.

2.6 Огляд pfSense

pfSense — це потужна та гнучка система з відкритим кодом, яка виконує функції мережевого шлюзу, брандмауера, VPN-сервера, а також системи контролю доступу до мережі. Вона базується на ядрі операційної системи FreeBSD, але завдяки інтуїтивно зрозумілому веб-інтерфейсу не вимагає від адміністратора глибоких знань самої ОС. Це робить pfSense зручною у використанні як для домашніх, так і для корпоративних мережевих інфраструктур.

Система має широкий функціонал, який охоплює майже всі аспекти мережевого адміністрування та безпеки. Основні можливості pfSense включають:

- Фільтрація трафіку за IP-адресами та портами джерела/призначення, з можливістю гнучкого налаштування правил на вхідний та вихідний трафік.
- Пасивне визначення операційних систем за допомогою модуля p0f, що дозволяє виявляти тип пристрою без активного сканування.
- Прозорий брандмауер другого рівня, який може працювати у режимі bridge без маршрутизації.
- Нормалізація мережевих пакетів, що дозволяє блокувати пакети з аномальними або потенційно шкідливими параметрами, захищаючи від деяких типів атак.
- Підтримка Stateful Firewall — збереження стану з'єднання з можливістю обробки складних сценаріїв NAT та фільтрації. Також підтримується SYN Proxy для захисту від атак типу SYN Flood.
- Гнучке налаштування NAT — включно з Port Forwarding, Outbound NAT і 1:1 NAT.
- Підтримка VPN-протоколів: OpenVPN, IPSec, L2TP, PPTP — що дає змогу будувати безпечні віддалені підключення.
- Моніторинг мережі в режимі реального часу за допомогою графіків (RRD, Status Monitoring) та журналів подій.

- Підтримка DDNS (динамічного DNS) для автоматичного оновлення публічної IP-адреси в DNS-сервісі.

Особливо актуальною в контексті цієї роботи є можливість налаштування Captive Portal – механізму, що дозволяє обмежити доступ до мережі до моменту проходження автентифікації. pfSense надає повний набір інструментів для створення сценаріїв Captive Portal з налаштуванням сторінок входу, підтримкою редіректів, білого списку адрес, інтеграції з зовнішніми сервісами автентифікації, а також скриптів для автоматичного додавання/видалення IP-адрес користувачів із контрольного списку.

Завдяки відкритому коду, великій спільноті користувачів, широким можливостям налаштування та сумісності з великою кількістю мережевих сценаріїв, pfSense є ефективною платформою для реалізації складних архітектур мережевого доступу, зокрема таких, які поєднують Captive Portal з зовнішніми Identity Providers (IdP), як-от Keycloak з підтримкою passkey.

2.7 Captive Portal у pfSense

Captive Portal у pfSense — це вбудований механізм обмеження доступу до мережі до моменту проходження користувачем процедури автентифікації. Такий підхід широко застосовується у публічних, гостьових або корпоративних Wi-Fi мережах, де необхідно контролювати, хто і на яких умовах може отримати доступ до інтернету чи внутрішніх ресурсів.

У pfSense Captive Portal реалізовано як окремий модуль, який дозволяє:

- перенаправляти користувача на кастомізовану сторінку входу після підключення до мережі;
- задавати правила авторизації через локальні облікові записи або зовнішні системи (Radius, LDAP, скрипти);
- формувати білі списки адрес, які будуть доступні навіть без авторизації;

- обмежувати швидкість підключення, тривалість сесії та кількість одночасних підключень;
- реєструвати IP- та MAC-адреси користувачів, що пройшли автентифікацію;
- запускати скрипти при вході/виході користувача — з можливістю динамічно додавати IP-адреси до правил фільтрації (наприклад, firewall alias або таблиці pf).

У рамках даної роботи Captive Portal виконує роль контролера доступу до Wi-Fi, забезпечуючи редірект на зовнішню систему автентифікації, якою є Keycloak. За допомогою проміжного додатку запити від користувача обробляються, і після успішної автентифікації IP-адреса клієнта вноситься до дозволеного списку, що відкриває йому повноцінний доступ до мережі.

Така модель дозволяє інтегрувати сучасну безпарольну автентифікацію (passkey) у класичну мережеву інфраструктуру без необхідності зміни клієнтських пристроїв або складної модернізації середовища.

2.8 Висновки до другого розділу

У другому розділі було всебічно розглянуто концепцію безпарольної автентифікації, її основні переваги та реалізацію за допомогою сучасних відкритих стандартів — FIDO2 і WebAuthn. Проведено глибокий аналіз механізмів автентифікації з використанням криптографічних ключів, що дозволяє уникнути традиційних вразливостей паролів, таких як фішинг, brute-force атаки чи крадіжка облікових даних.

Описані в роботі протоколи FIDO2 та WebAuthn забезпечують високий рівень безпеки та конфіденційності, оскільки приватні ключі не залишають пристрою користувача, а біометричні дані не передаються на сервер. Завдяки використанню асиметричної криптографії ці технології забезпечують стійкість до широкого спектра атак, у тому числі атак типу «людина посередині» (MITM).

Особливу увагу приділено ключам доступу (passkeys) — новому підходу, який дозволяє користувачам здійснювати автентифікацію без пароля з використанням

біометрії або апаратних автентикаторів. Розглянуто типи passkeys (синхронізовані та прив'язані до пристрою), а також два основних типи автентикаторів — платформні та роумінгові. Було обґрунтовано, що passkeys поєднують у собі зручність, безпеку та сумісність з великою кількістю пристроїв, що робить їх одним із найперспективніших напрямів розвитку автентифікаційних систем.

У цьому ж розділі здійснено аналіз двох ключових інструментів, що використовуються у практичній частині дослідження:

- Keycloak — як потужна система керування ідентичністю, що підтримує OpenID Connect, SAML, ролі, політики доступу та інтеграцію з сучасними методами автентифікації;
- pfSense — як мережевий шлюз із підтримкою Captive Portal, який дозволяє реалізувати контрольований доступ до Wi-Fi мережі з перенаправленням на зовнішній сервер автентифікації.

Розглянутий механізм Captive Portal у pfSense, що інтегрується з Keycloak через проміжну програму, дозволяє реалізувати сценарій безпарольної автентифікації у Wi-Fi мережі без потреби встановлення додаткового ПЗ на клієнтські пристрої. Такий підхід є ефективним, масштабованим і безпечним рішенням, що дозволяє гнучко контролювати доступ до мережі, використовуючи сучасні стандарти автентифікації.

Таким чином, результати аналізу цього розділу підтверджують як наукову обґрунтованість вибору технологій, так і їхню практичну доцільність для реалізації сучасних систем безпарольного доступу. Це створює міцне підґрунтя для практичної реалізації інтегрованого рішення з використанням pfSense, Keycloak і passkey, що буде описано в наступному розділі.

РОЗДІЛ 3
ПРОЄКТУВАННЯ ТА ВПРОВАДЖЕННЯ БЕЗПАРОЛЬНОЇ АВТЕНТИФІКАЦІЇ У WI-FI МЕРЕЖІ

3.1 Опис архітектури системи

У рамках практичної частини дипломної роботи було реалізовано експериментальне середовище, призначене для апробації та демонстрації функціонування безпарольної автентифікації користувачів у Wi-Fi мережі на основі технології Passkey. Мета даного рішення полягає в підвищенні рівня безпеки процесу контролю доступу до інформаційно-телекомунікаційних ресурсів завдяки усуненню факторів, властивих традиційним паролям, таких як вразливість до фішингу, повторне використання облікових даних і їх компрометація.

3.2 Загальний опис інфраструктури

Архітектура побудована на основі гіпервізора Proxmox Virtual Environment (VE), який забезпечує ефективне керування віртуальними машинами та контейнерами з використанням KVM та LXC технологій (рисунок 3.1).

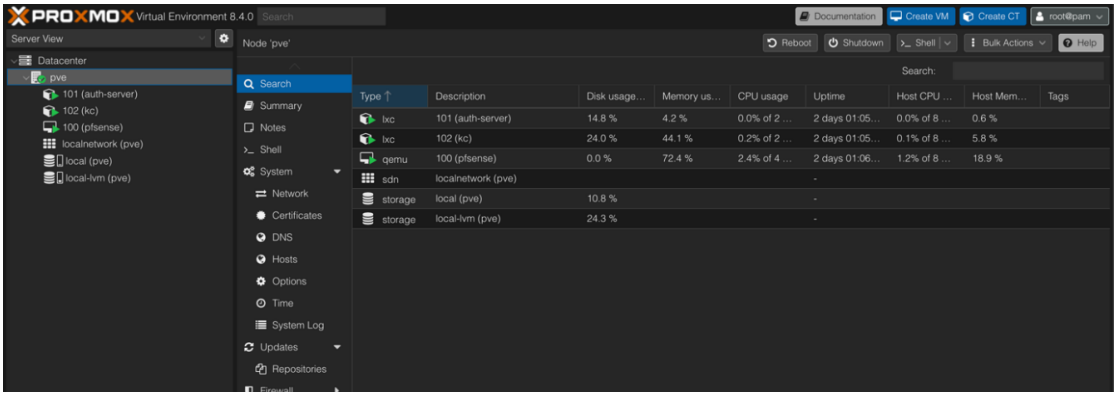


Рисунок 3.1 – Консоль Proxmox

У середовищі було розгорнуто три ключові віртуальні системи:

1. pfSense (версія 2.7.2-RELEASE) — система маршрутизації та міжмережевого екранування з увімкненим механізмом Captive Portal, яка виконує функції шлюзу, DHCP-сервера, DNS-форвардера та точки контролю доступу до мережі.

2. Keycloak (версія 23.0.0) — сервер ідентифікації та автентифікації користувачів, налаштований на використання протоколу OpenID Connect та підтримку WebAuthn/Passkey.

3. Авторизаційний застосунок — вебсервіс, реалізований з використанням Python (фреймворк Flask), який слугує проміжною ланкою між Captive Portal і системою авторизації Keycloak.

До мережевої інфраструктури інтегровано точку бездротового доступу TP-Link TL-WR841N/ND v9, на яку інстальовано прошивку OpenWrt 18.06.9, це зображено на рисунок 3.2 та рисунку 3.3, що дозволяє керувати маршрутизацією та реалізовувати режим *dumb access point* — тобто передача усіх даних у мережу без участі в маршрутизації чи фільтрації трафіку.

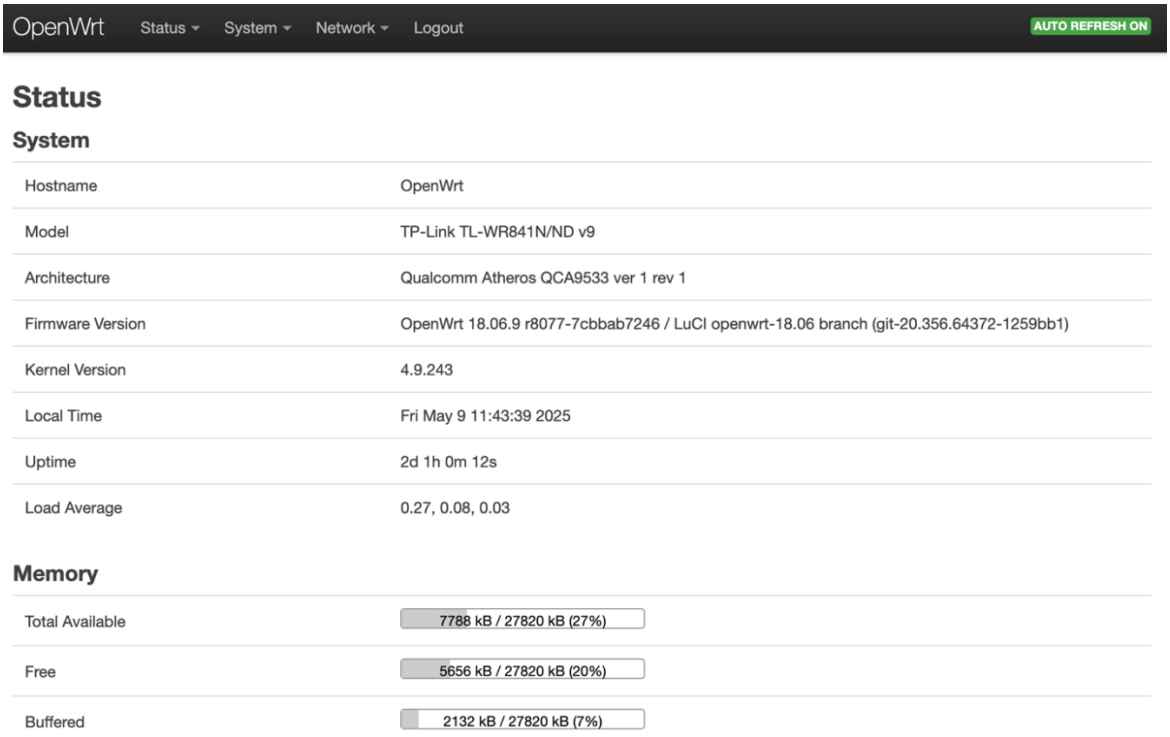


Рисунок 3.2 – Консоль OpenWRT

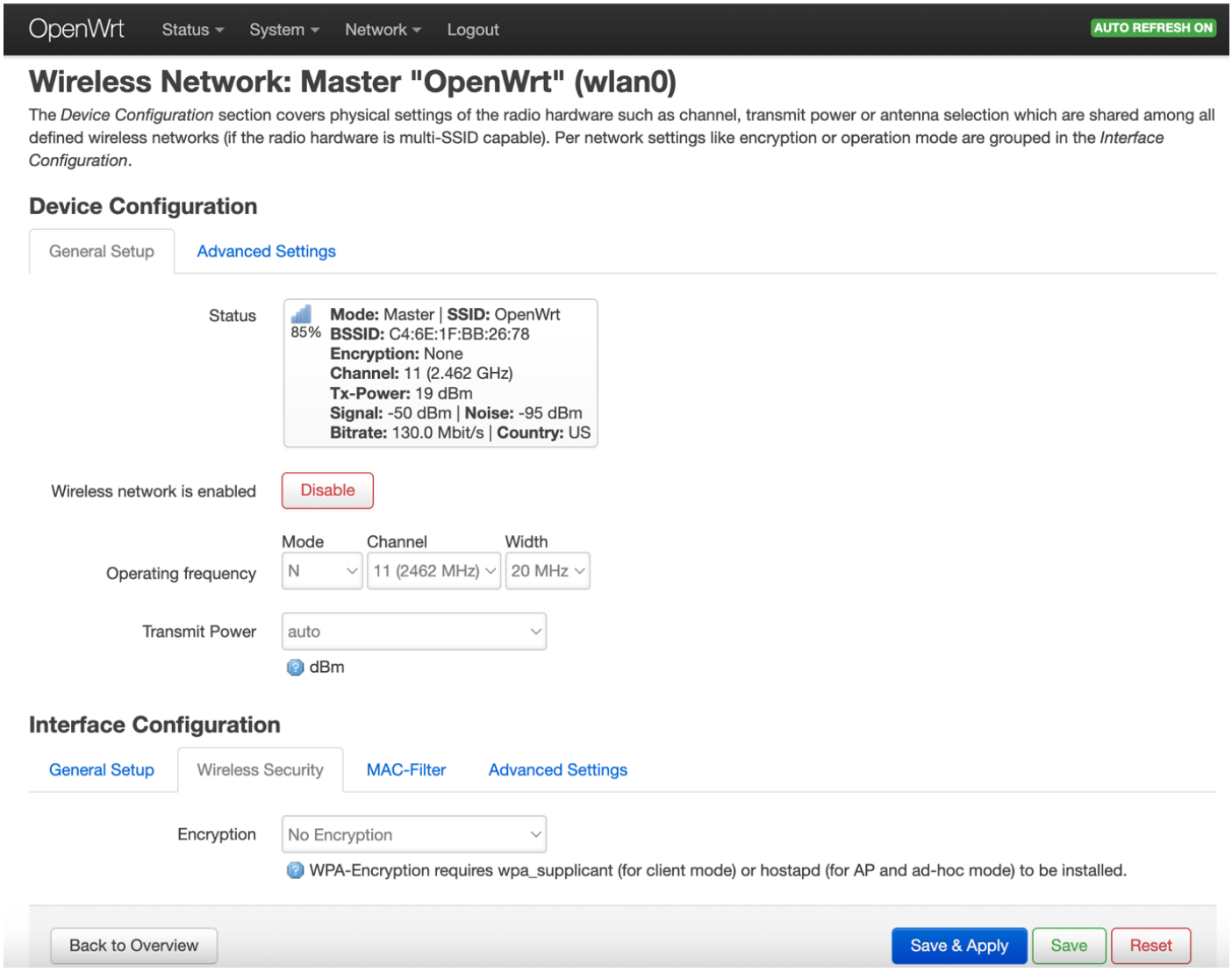


Рисунок 3.3 – Налаштування відкритої точки бездротової доступу без шифрування

Уся взаємодія між компонентами відбувається у межах єдиної ізольованої підмережі з адресним простором 192.168.11.0/24, що дозволяє зменшити ризик зовнішнього втручання та спростити налаштування фільтрації трафіку.

Нижче наведено таблицю, що ілюструє призначення IP-адрес кожного з компонентів:

Таблиця 3.1 – Призначення IP-адрес компонентів

Компонент	**IP-адреса**	**Призначення**
pfSense (LAN інтерфейс)	192.168.11.1	DHCP-сервер, шлюз за замовчуванням, Captive Portal, DNS
Keycloak (LXC на Ubuntu 22.04)	192.168.11.108	Система ідентифікації користувачів
Python-застосунок (LXC)	192.168.11.109	Вебзастосунок авторизації
OpenWRT (MGMT інтерфейс)	192.168.11.10	Dumb Access Point (Wi-Fi точка)

Сервери, вебзастосунок і Wi-Fi точка доступу об’єднані в локальну віртуальну мережу, що забезпечує гарантовану маршрутизацію без потреби в NAT або додатковій маршрутизуючій логіці. Віртуальні машини не виходять напряму у зовнішній інтернет, що також мінімізує ризики витоку даних чи впливу ззовні.

3.3 Логіка взаємодії компонентів

Функціональна логіка роботи системи є такою:

1. Користувач підключається до відкритої Wi-Fi мережі, створеної на базі OpenWRT. Оскільки точка доступу працює в режимі dumb AP, усі запити перенаправляються через pfSense.

2. pfSense автоматично блокує вихід у зовнішню мережу і спрямовує HTTP-запити користувача на локальний Captive Portal, сторінку якого модифіковано таким чином, щоб вона негайно здійснювала редірект на Python-застосунок авторизації.

3. Вебзастосунок ініціює авторизацію через OpenID Connect на Keycloak, передаючи параметри автентифікації та callback URL.

4. У процесі автентифікації користувачеві пропонується використати Passkey — криптографічний ключ, що зберігається на захищеному модулі пристрою (TPM, Secure Enclave, тощо). Підтвердження автентифікації здійснюється за допомогою біометрії (відбиток пальця, Face ID) або PIN-коду.

5. Після підтвердження особи Keycloak перенаправляє користувача назад до вебзастосунку з авторизаційним токеном.

6. Авторизаційний застосунок, у свою чергу, надсилає POST-запит до pfSense (на інтерфейс авторизації Captive Portal), що активує доступ користувача до мережі на основі його MAC-адреси.

7. Користувач отримує повноцінний доступ до мережі Інтернет без необхідності введення паролю.

Цей підхід забезпечує високий рівень захищеності, оскільки автентифікація реалізується на основі відкритих стандартів FIDO2/WebAuthn і не залежить від паролів, які можуть бути скомпрометовані або викрадені внаслідок соціальної інженерії.

3.4 Захист інформації та криптографічні аспекти

Технологія Passkey є заміною паролів і реалізована за допомогою публічно-приватної криптографії. При першій реєстрації на пристрої створюється унікальна пара ключів — приватний ключ зберігається лише на пристрої, а публічний передається до Keycloak.

При подальшій автентифікації користувач підтверджує свою особу, активуючи підпис запиту приватним ключем. Платформа Keycloak, у свою чергу, перевіряє автентичність підпису за допомогою збереженого публічного ключа, що дозволяє здійснити авторизацію без збереження будь-яких паролів.

З’єднання між клієнтськими пристроями, Python-застосунком та Keycloak здійснюється через зашифровані TLS-з’єднання, що відповідає вимогам до захисту конфіденційної інформації згідно з сучасними стандартами (TLS 1.2/1.3).

3.5 Апаратна база та загальна структура середовища

Гіпервізор розгорнуто на фізичному сервері з такими параметрами:

- Процесор: 8 ядер (архітектура x86_64), що дозволяє паралельну роботу декількох віртуальних компонентів;
- Оперативна пам’ять: 8 ГБ DDR4, розподілена динамічно між контейнерами та віртуальною машиною;
- Постійна пам’ять: SSD обсягом 120 ГБ, що забезпечує високу швидкодію при операціях читання/запису.

Обґрунтуванням такого вибору є потреба в стабільному, швидкому та ізольованому середовищі для моделювання взаємодії між компонентами системи без впливу зовнішніх чинників.

3.6 Розгортання та налаштування Keycloak

Keycloak було обрано як систему керування ідентичністю завдяки її повноцінній підтримці відкритих стандартів WebAuthn, FIDO2, OAuth 2.0 та OpenID Connect. У межах цього проєкту сервер Keycloak було розгорнуто у Docker-контейнері, що працює в межах LXC на базі Ubuntu 22.04.

Перевага використання Docker полягає у можливості декларативного розгортання конфігурацій через docker-compose або системні юніти, швидкого відновлення після збоїв, легкості оновлення версій та ізоляції залежностей. Розгорнутий Keycloak зображено на рисунку 3.4.

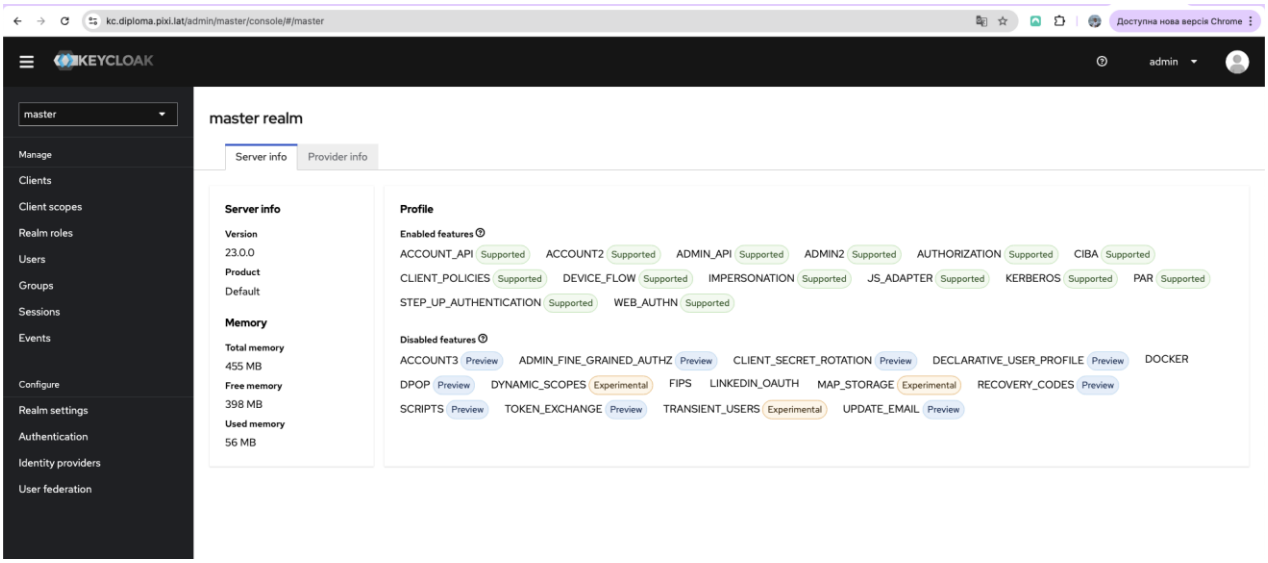


Рисунок 3.4 – Розгорнутий Keycloak

У Keycloak було створено спеціалізований realm, зображено на рисунку 3.5, у межах якого зареєстровано клієнта captive-portal-auth. Для забезпечення автентифікації без використання пароля було активовано підтримку WebAuthn, що дозволяє користувачеві ідентифікуватися на основі криптографічного ключа, захищеного апаратно.

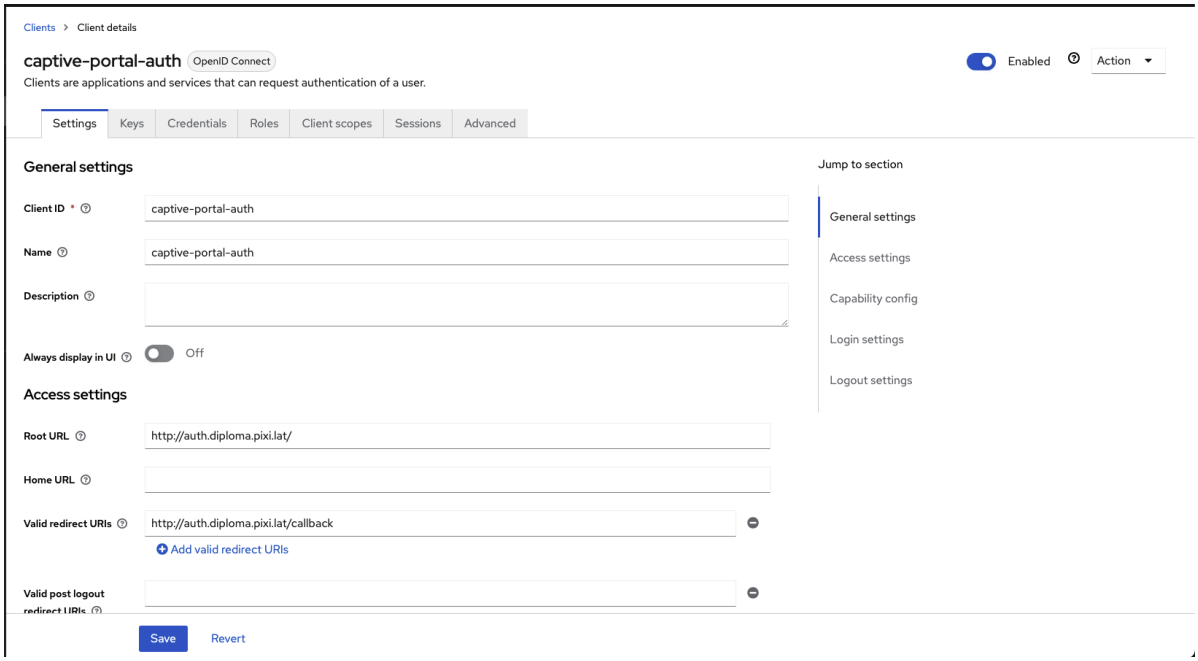


Рисунок 3.5 – Створення необхідного клієнта

Сервер Keycloak був прив’язаний до публічного доменного імені kc.diploma.pixi.lat, що дозволило уникнути проблем з автентифікацією WebAuthn (яка вимагає наявності довіреного origin). Сертифікат SSL було отримано з використанням Let’s Encrypt, що гарантує відповідність сучасним вимогам до TLS 1.2+.

3.7 Розгортання авторизаційного застосунку

Допоміжний вебзастосунок для обробки авторизації Captive Portal та редиректу на Keycloak реалізовано за допомогою мови Python та фреймворку Flask. Він виконує роль проміжного елемента між користувачем, який підключається до мережі, і системою автентифікації.

У застосунку реалізовано такі функції:

- Приймання HTTP-запиту від pfSense Captive Portal;
- Ініціація OpenID Connect авторизації через Keycloak;
- Приймання callback після успішної автентифікації;
- Формування POST-запиту до pfSense для відкриття мережевого доступу.

Запуск вебзастосунку здійснювався у режимі розробника за допомогою app.run(), що є припустимим у межах тестового стенду, але не рекомендовано для промислового середовища через відсутність підтримки багатопоточності та автоматичного масштабування.

3.8 Мережеві налаштування та топологія

Мережеве з’єднання між усіма компонентами забезпечується через віртуальний мережевий міст vmbr2, який відповідає за локальну ізольовану підмережу 192.168.11.0/24. У цьому середовищі:

- pfSense має IP-адресу 192.168.11.1 та виконує роль шлюзу та DNS-форвардера;

- keycloak працює на 192.168.11.108;
- авторизаційний застосунок доступний за адресою 192.168.11.109;
- wi-Fi точка (OpenWRT) підключена до цього ж сегменту з адресою 192.168.11.10.

Усі мережеві з’єднання реалізовано без використання VLAN, ізоляції рівня 2 або брандмауерних обмежень у межах віртуальної мережі. Такий підхід дозволяє моделювати поведінку корпоративної мережі при мінімальному рівні складності, а також полегшує налагодження інтеграції між компонентами.

3.9 Структура та принцип роботи

Captive Portal – це один із класичних підходів до контролю доступу в комп’ютерних мережах, який особливо ефективний у середовищах із динамічним підключенням користувачів (наприклад, Wi-Fi). У межах цього дослідження Captive Portal виступає точкою входу до процесу автентифікації користувачів без використання паролів, що реалізовано шляхом перенаправлення HTTP-трафіку до зовнішнього сервісу авторизації з підтримкою OpenID Connect і WebAuthn. Налаштований Captive Portal зображено на рисунку 3.6 та рисунку 3.7.

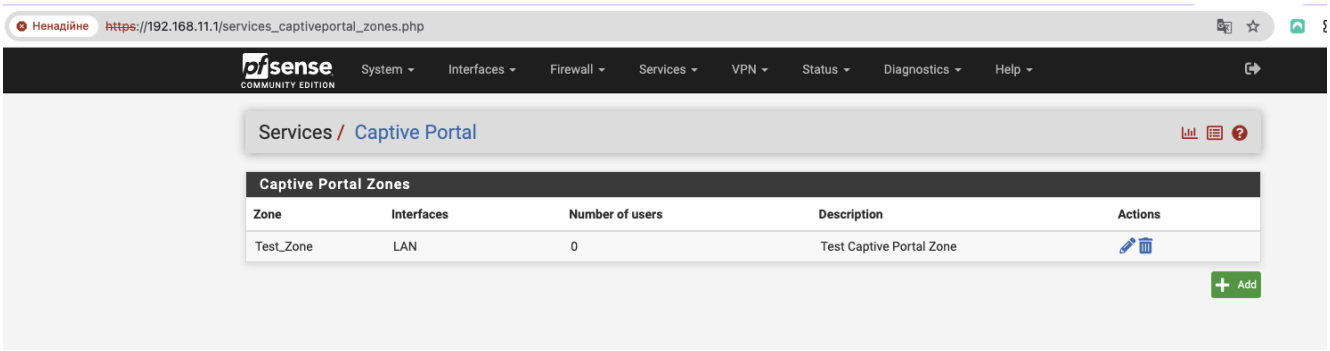


Рисунок 3.6 – Налаштований Captive Portal

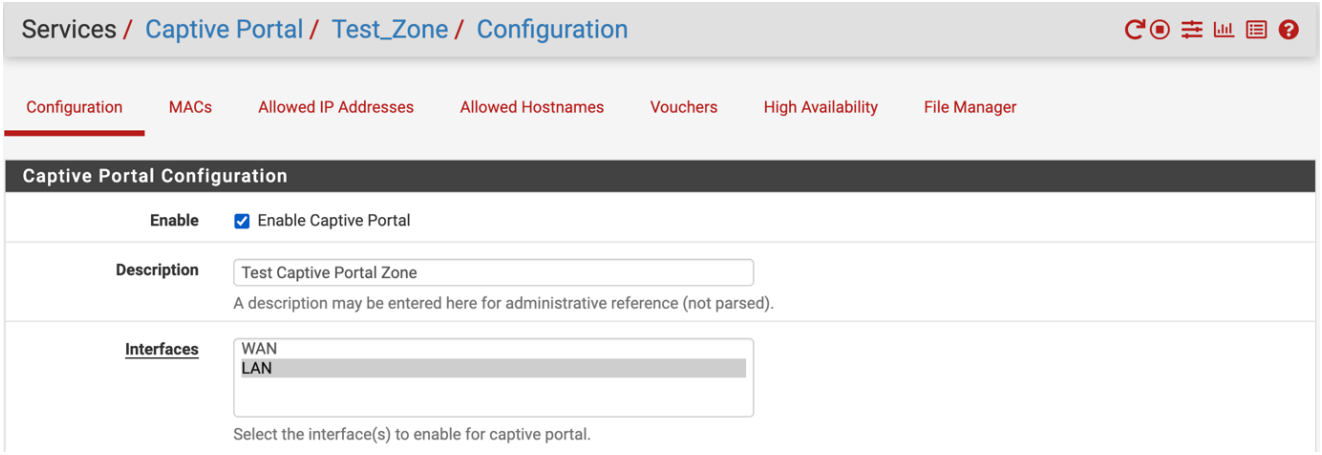


Рисунок 3.7 – Налаштування зони Captive Portal

У середовищі pfSense було створено окрему зону Captive Portal під назвою Test_Zone, прив'язану до інтерфейсу LAN, через який проходить увесь трафік з бездротової мережі. У цій конфігурації всі підключення Wi-Fi користувачів спрямовуються через pfSense, що дозволяє ефективно блокувати вихідний трафік до моменту проходження авторизації, зображено на рисунку 3.8.

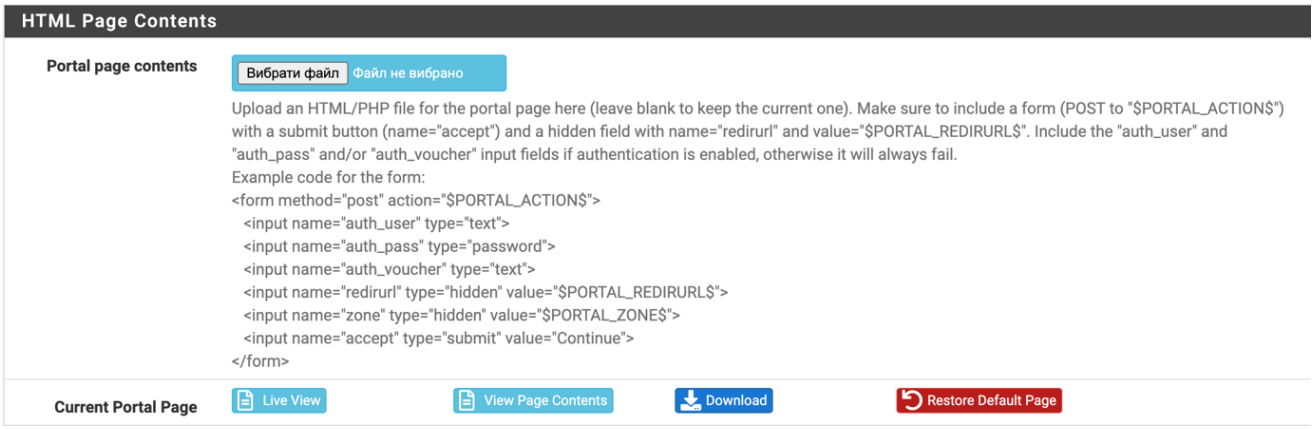


Рисунок 3.8 – Завантажена кастомна сторінка Captive Portal

На відміну від традиційної автентифікації Captive Portal, яка зазвичай передбачає ручове введення логіна/пароля у форму HTML, у запропонованій реалізації такий підхід був замінений на автоматизований редирект до зовнішнього Python-застосунку, який працює як OpenID клієнт.

Алгоритм взаємодії виглядає наступним чином:

1. Користувач підключається до відкритої Wi-Fi мережі.
2. При спробі доступу до будь-якого ресурсу браузер користувача отримує редирект на внутрішню HTML-сторінку Captive Portal.
3. Замість форми входу ця сторінка одразу здійснює редирект на зовнішній ресурс http://auth.diploma.pixi.lat/login.
4. Авторизаційний застосунок запускає OpenID-авторизацію через Keycloak.
5. Після автентифікації користувача за допомогою Passkey (біометрія/TPM) Keycloak повертає токен і переадресовує на /callback.
6. Python-застосунок формує POST-запит до http://192.168.11.1:8002/index.php?zone=test_zone з параметрами, які симулюють натискання кнопки “Continue” в оригінальній формі Captive Portal.
7. Captive Portal розпізнає запит як легітимний і надає користувачеві доступ до інтернету.

3.10 Технічна реалізація кастомної сторінки Captive Portal

Сторінка входу була переписана вручну і зведена до HTML-фрагмента з мета-редиректом,зображено на рисунку 3.9.

```html
<html>
  <head>
    <meta http-equiv="refresh" content="0; url=http://auth.diploma.pixi.lat/login">
  </head>
  <body>
    Redirecting to login...
  </body>
</html>
```

Рисунок 3.9 – Сторінка HTML-фрагмента з мета-редиректом

Це рішення дозволяє мінімізувати будь-яку взаємодію з боку користувача. Він не вводить паролі вручну, не заповнює форм — усі дії виконуються автоматично, з дотриманням стандарту Zero UI.

Такий підхід особливо ефективний для мобільних пристроїв та користувачів без технічних навичок.

Важливим етапом інтеграції є емулювання дій користувача після авторизації. pfSense очікує POST-запит до index.php, що імітує натискання кнопки «Продовжити». У типовій реалізації Captive Portal це виглядає як HTML-форма, у якій передаються параметри:

- zone — ідентифікатор зони Captive Portal (у нашому випадку test_zone);
- redirurl — адреса, на яку користувача буде перенаправлено після авторизації (наприклад, http://neverssl.com);
- accept — логічний прапорець, що імітує згоду з умовами користування.

Python-застосунок генерує HTML-документ з формою, яка автоматично відправляє POST-запит до Captive Portal, що дозволяє обійти стандартну HTML-форму pfSense без зміни його внутрішньої логіки.

Це рішення дозволяє ефективно поєднати два світи:

- Legacy Captive Portal, який працює з мінімальним API.
- сучасні методи автентифікації, реалізовані через OIDC та WebAuthn.

3.11 Архітектурні засади і вибір технології розгортання

Для розгортання Keycloak було прийнято рішення використовувати контейнеризовану архітектуру, що забезпечує високу гнучкість, портативність і швидке масштабування. Платформа розгорнута за допомогою Docker Compose, що дозволяє декларативно описати всі залежності та конфігурації. Даний процес зображено на рисунку 3.10.

```
root@kc:~/keycloak# docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED       STATUS       PORTS
                                NAMES
155bdf907cc1   nginx:alpine                         "/docker-entrypoint.…"   11 days ago   Up 2 days    80/tcp, 0
.0.0.0:443->443/tcp, :::443->443/tcp   keycloak_nginx_1
cf050d852a2f   quay.io/keycloak/keycloak:23.0.0     "/opt/keycloak/bin/k…"   11 days ago   Up 2 days    8080/tcp,
 8443/tcp                            keycloak_keycloak_1
root@kc:~/keycloak#
```

Рисунок 3.10 – Keycloak та реверс-проксі nginx в docker

Використання офіційного контейнерного образу гарантує сумісність із останніми оновленнями безпеки та підтримку функціоналу WebAuthn, який був суттєво покращений у версіях 21.0+. Запуск відбувався у стандартному режимі з командою start, без специфічного --optimized, який використовується для production-оточення (із увімкненням JPA-оптимізацій, кешування шаблонів і полегшеного CI/CD).

Усі зовнішні HTTP(S)-запити до Keycloak обробляються через NGINX як реверс-проксі, що термінує TLS-з’єднання. Отримання сертифікатів відбувалося за допомогою Certbot (Let’s Encrypt) безпосередньо на хості, що дозволило уникнути складнощів, пов’язаних із генерацією сертифікатів всередині контейнерів, зображено на рисунку 3.11.

```
root@kc:~/keycloak/nginx# less kc.conf
server {
    listen 443 ssl;
    server_name kc.diploma.pixi.lat;

    ssl_certificate /etc/nginx/ssl/kc.crt;
    ssl_certificate_key /etc/nginx/ssl/kc.key;

    location / {
        proxy_pass http://keycloak:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;
    }
}
kc.conf (END)
```

Рисунок 3.11 – Налаштування revers-proxy nginx

Таке розмежування між внутрішнім сервісом (Keycloak) і зовнішнім доступом (через NGINX) дозволяє масштабувати архітектуру — наприклад, додати балансування навантаження або спільний TLS-контроль для кількох сервісів.

3.12 Конфігурація realm, клієнта та механізму редиректу

У Keycloak було створено realm з назвою diploma, зображено на рисунку 3.12, який ізольовано зберігає користувачів, клієнтів і політики безпеки. Така сегментація дозволяє відокремити реалізацію конкретного проєкту від можливих майбутніх сценаріїв розширення системи (наприклад, додавання іншої автентифікації для внутрішніх адміністраторів).

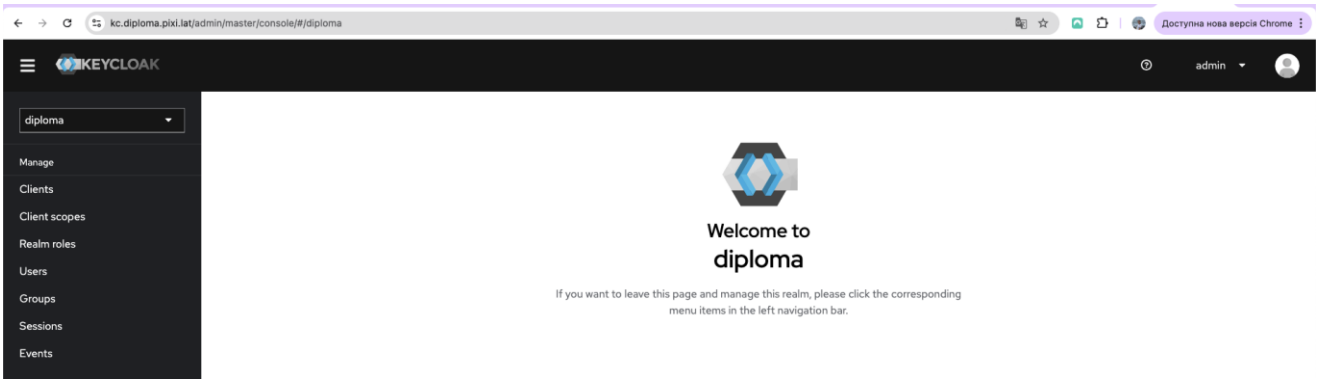


Рисунок 3.12 – Створений окремий realm

У межах realm створено OpenID Connect client (рисунок 3.13) під назвою captive-portal-auth. Його основні параметри:

- Тип доступу: *ймовірно confidential*, однак не перевірено вручну.
- Callback URI: http://auth.diploma.pixi.lat/callback.
- Увімкнено стандартні флоу авторизації Authorization Code Flow.

- Ймовірно не включено PKCE — що є потенційною вразливістю у відкритому середовищі, оскільки PKCE запобігає перехопленню authorization code через повторне використання (рисунок 3.13).

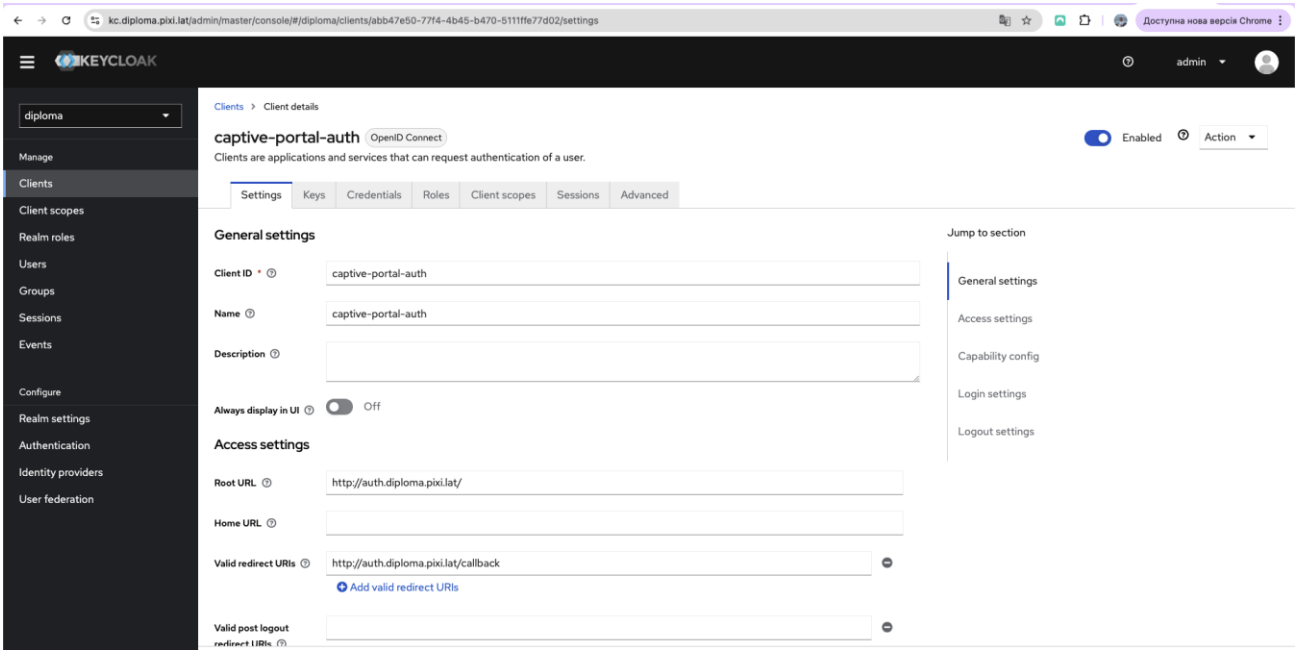


Рисунок 3.13 – Створений необхідний клієнт

При запиті авторизації застосунок переадресовує користувача на веб-сайт https://kc.diploma.pixi.lat/realms/diploma/protocol/openid-connect/auth, де Keycloak ініціює повноцінний браузерний флоу.

Однією з ключових новацій реалізованої системи є підтримка автентифікації за допомогою Passkey, яка реалізована в Keycloak через протокол WebAuthn, що базується на специфікаціях FIDO2.

Для цього було:

- Увімкнено WebAuthn в authentication flow.
- Створено новий authentication flow, у якому стандартну автентифікацію по паролю замінено на WebAuthn Register + WebAuthn Authenticate.
- Користувачам дозволено реєструвати нові WebAuthn credential через графічний інтерфейс при першому вході (рисунок 3.14).

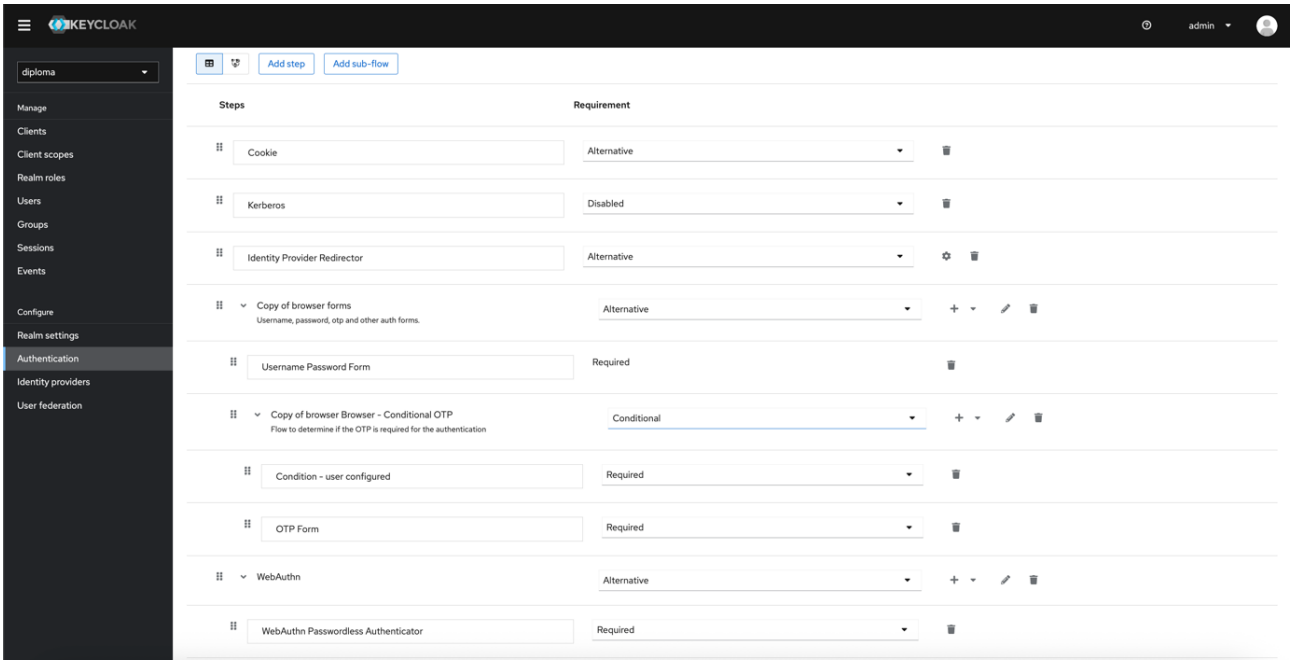


Рисунок 3.14 – Налаштування Authentication Flow з використання WebAuthn

Passkey, на відміну від апаратного ключа FIDO (наприклад, YubiKey), може зберігатися в TPM або Secure Enclave пристрою користувача (ноутбук, смартфон) і синхронізується між пристроями (наприклад, через iCloud Keychain або Google Password Manager).

Таким чином, Passkey створює асиметричну пару ключів:

- Приватний ключ залишається на пристрої користувача;
- Публічний ключ зберігається у Keycloak.

Під час автентифікації Keycloak надсилає challenge, який підписується приватним ключем, а перевірка підпису здійснюється за публічним ключем — без передачі будь-яких секретів у мережу.

Цей підхід:

- Нейтралізує фішинг;
- Повністю усуває залежність від паролів;
- Є незалежним від конкретного браузера (працює в Chrome, Safari, Firefox).

3.13 Робота з користувачами та створення credential

Створення облікових записів користувачів виконувалось вручну через адмін-панель Keycloak. На момент дослідження не було активовано self-registration, що повністю виключає ризик спаму чи самостійної генерації облікових записів.

Користувачі, під час першої автентифікації, ініціюють прив'язку WebAuthn credential вручну – браузер активує платформенний діалог, де система запитує дозвіл на створення ключа, наприклад, через Touch ID. Цей процес є прозорим для користувача та не потребує знань про криптографію, зображено на рисунку 3.15.

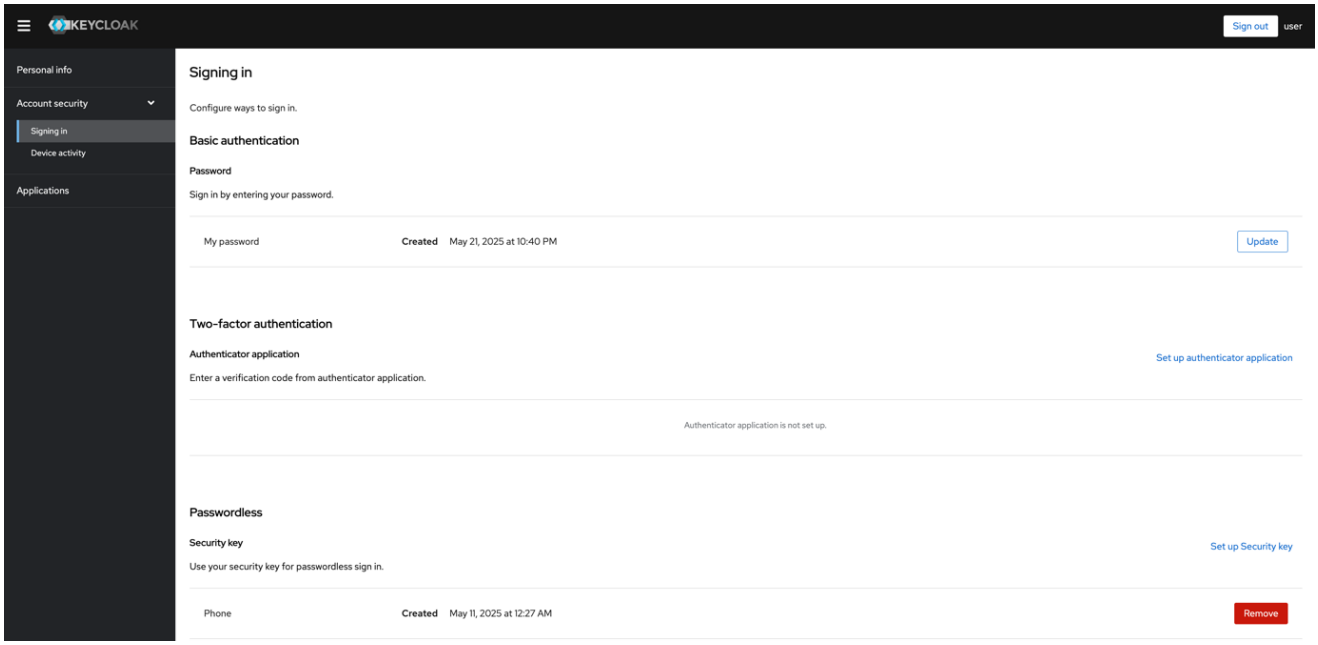


Рисунок 3.15 – Налаштований Security Key у користувача

У процесі реєстрації credential Keycloak зберігає публічну частину ключа в розділі credential користувача, дозволяючи надалі здійснювати автентифікацію без додаткових запитів на стороні сервера.

3.14 Реалізація авторизаційного застосунку (Python)

Ключовим елементом реалізованої системи, що пов’язує механізм автентифікації (Keycloak) з механізмом контролю доступу (pfSense), є власний вебзастосунок, реалізований мовою Python з використанням фреймворку Flask. Цей застосунок виконує роль OpenID Connect клієнта, що забезпечує взаємодію з Keycloak у межах протоколу авторизації та обробляє подальший автоматичний допуск користувача до мережі шляхом взаємодії з Captive Portal.

Застосунок реалізовано як легкий Flask-сервер із мінімальною кількістю зовнішніх залежностей. Основна бібліотека, яка відповідає за інтеграцію з Keycloak через протокол OAuth 2.0 / OpenID Connect, — це Authlib, яка забезпечує високорівневу абстракцію для реалізації клієнта.

```python
from flask import Flask, redirect, url_for, render_template_string
from authlib.integrations.flask_client import OAuth
```

Рисунок 3.16 – Код застосунку Flask

Сервер запускається у режимі розробки (app.run()), що дозволено в умовах локального тестового середовища. У майбутньому для продуктивного середовища рекомендовано використовувати продакшн-сервери (Gunicorn, uWSGI) з реверс-проксі (NGINX) для масштабування, безпеки й підтримки TLS.

На рівні ініціалізації Flask-застосунок реєструє OIDC-клієнт із такими параметрами, зображено на рисунку 3.17:

```python
oauth.register(
    name='keycloak',
    client_id='captive-portal-auth',
    client_secret='...',
    access_token_url='https://kc.diploma.pixi.lat/realms/diploma/protocol/openid-connect/token',
    authorize_url='https://kc.diploma.pixi.lat/realms/diploma/protocol/openid-connect/auth',
    api_base_url='https://kc.diploma.pixi.lat/realms/diploma/protocol/openid-connect',
    client_kwargs={'scope': 'openid profile email'}
)
```

Рисунок 3.17 - Flask-застосунок OIDC-клієнт з параметрами

Ці параметри повністю відповідають протоколу OpenID Connect та дозволяють клієнтові:

- Ініціювати авторизацію через Keycloak.
- Приймати токен доступу та ідентифікаційний токен.
- Отримати основну інформацію про користувача.

Застосунок має два основні HTTP-ендпоінти:

1. /login

Цей маршрут відповідає за генерацію та переадресацію користувача до Keycloak:

```python
@app.route("/login")
def login():
    redirect_uri = url_for('callback', _external=True)
    return oauth.keycloak.authorize_redirect(redirect_uri)
```

Redirect URI, наданий Keycloak'у, — це /callback, на якому обробляється відповідь після автентифікації.

2. /callback

Після проходження автентифікації користувач повертається на /callback, де зазвичай здійснюється обробка токена (в даній реалізації — лише генерація POST-запиту до pfSense).

```python
@app.route("/callback")
def callback():
    pf_url = "http://192.168.11.1:8002/index.php?zone=test_zone"
    html = f"""
    <html><body>
    <form id="authform" method="POST" action="{pf_url}">
        <input type="hidden" name="redirurl" value="http://neverssl.com">
        <input type="hidden" name="zone" value="test_zone">
        <input type="hidden" name="accept" value="Continue">
        <input type="submit" value="Continue">
    </form>
    <script>
      document.getElementById("authform").submit();
    </script>
    </body></html>
    """
    return render_template_string(html)
```

Ця HTML-форма автоматично імітує дію користувача на сторінці Captive Portal. Це — обхідне рішення, яке дозволяє здійснити авторизацію на pfSense без інтеграції через API (оскільки pfSense не надає повноцінного REST API для Captive Portal).

Хоча рішення є ефективним у лабораторному середовищі, воно має низку потенційних вразливостей:

- Відсутність обробки й перевірки ID токену: застосунок не перевіряє, хто авторизувався, не валідує токен, не перевіряє підпис чи термін дії;
- Відсутність захисту від CSRF: оскільки авторизаційний флоу завершується через автоматичний POST-запит, CSRF-токени не використовуються;
- Жорстко задані параметри (zone, redirurl) у шаблоні HTML;
- Відсутність перевірки відповідності IP/MAC-адреси користувача — pfSense вирішує це самостійно на рівні L2.

3.15 Детальна послідовність інтеграції компонентів

Етап 1. Підключення до мережі та перехоплення Captive Portal.

Коли користувач приєднується до Wi-Fi мережі, перший HTTP-запит автоматично перенаправляється шлюзом pfSense на Captive Portal. Це виконується на рівні firewall із захопленням HTTP-пакетів, після чого браузер користувача отримує відповідь з HTML-документом із мета-редиректом:

```html
<meta http-equiv="refresh" content="0; url=http://auth.diploma.pixi.lat/login">
```

Виконання цього тегу у браузері призводить до автоматичного переходу на /login маршруту Python-застосунку.

Етап 2. Ініціація авторизації через OpenID Connect. Зображено на рисунку 3.18, 3.19, а також 3.20.

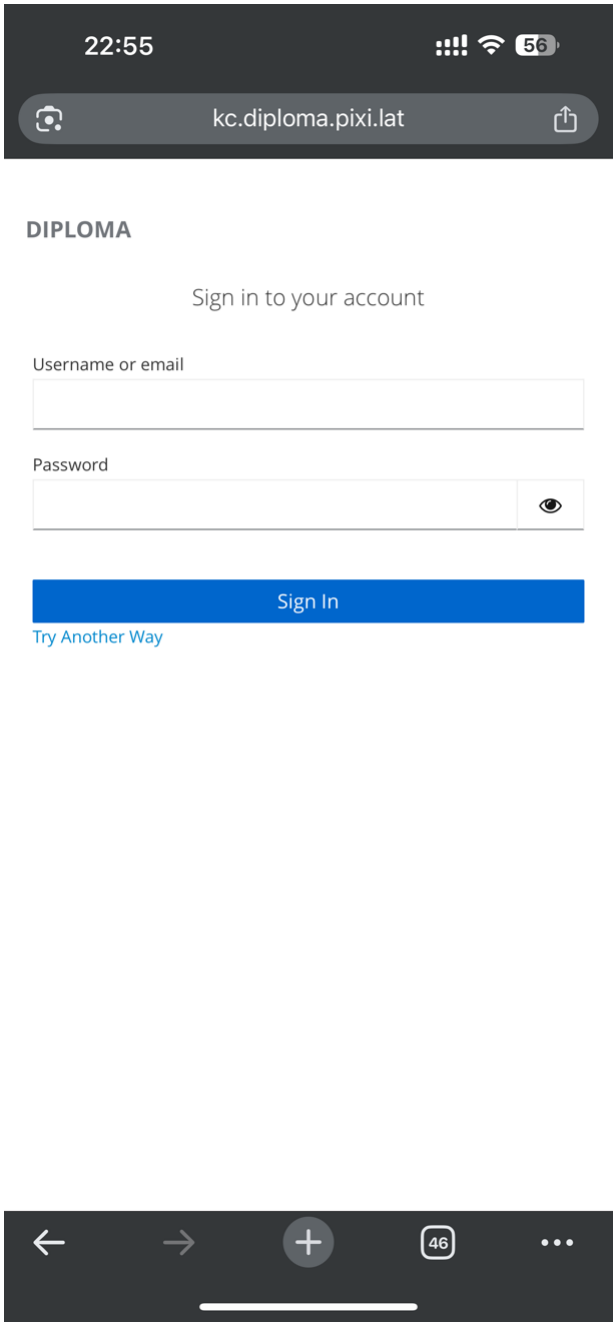


Рисунок 3.18 – Редірект Captive Portal на сторінку автентифікації Keycloak

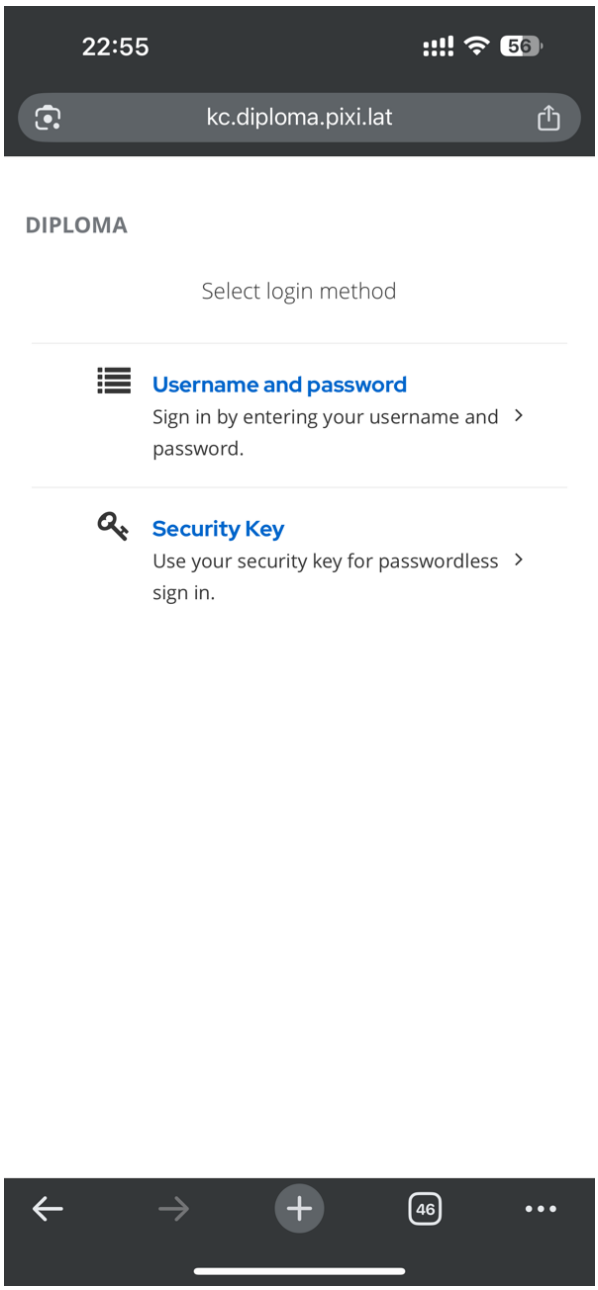


Рисунок 3.19 – Вибір варіанта безпарольної автентифікації

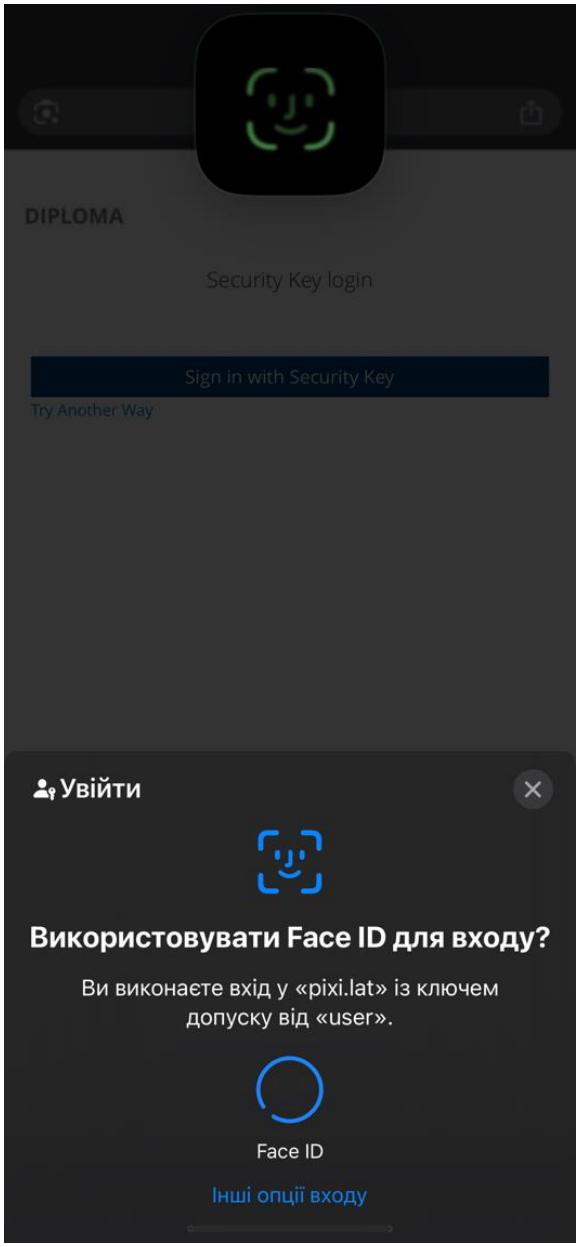
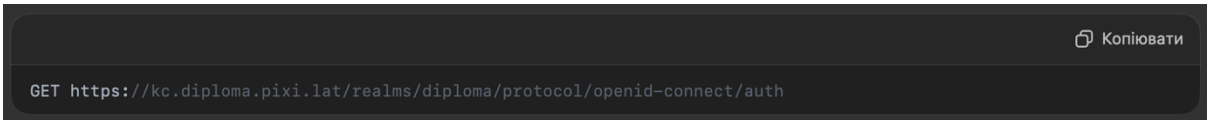


Рисунок 3.20 – Автентифікація за допомогою біометрії

Python-застосунок, написаний на Flask із використанням бібліотеки Authlib, реалізує клієнта OpenID Connect, даний процес зображено на рисунку 3.21, який виконує стандартний Authorization Code Flow. При зверненні до /login застосунок виконує запит:



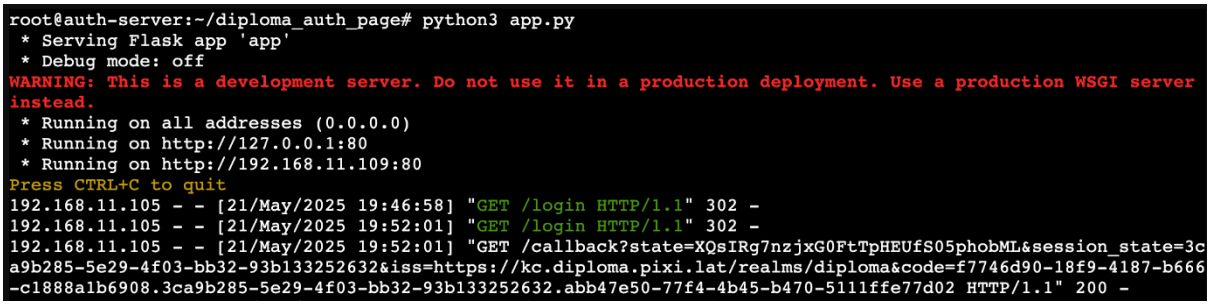


Рисунок 3.21 – Прийняття запиту скриптом

Етап 3. Взаємодія користувача з Keycloak.

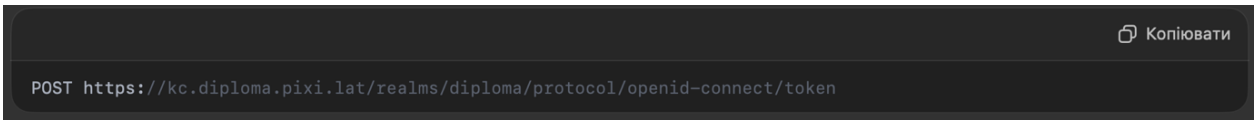
Користувач потрапляє на вебінтерфейс Keycloak, де проходить автентифікацію за допомогою Passkey/WebAuthn. Це процес, в якому браузер активує WebAuthn API:

1. Keycloak створює challenge (випадковий nonce) для підпису.
2. Користувач підтверджує свою особу за допомогою біометрії або PIN-коду.
3. Пристрій користувача підписує challenge своїм приватним ключем.
4. Браузер повертає підпис Keycloak, який перевіряє його за публічним ключем, збереженим при реєстрації Passkey.

У разі успішної перевірки Keycloak створює авторизаційний код і переадресовує користувача назад на http://auth.diploma.pixi.lat/callback.

Етап 4. Обмін авторизаційного коду на токени. Успішний варіант зоюражено на рисунку 3.22 та 3.23.

На /callback Flask-застосунок має (але поки не реалізовує) виконати обмін коду на токен через POST-запит:



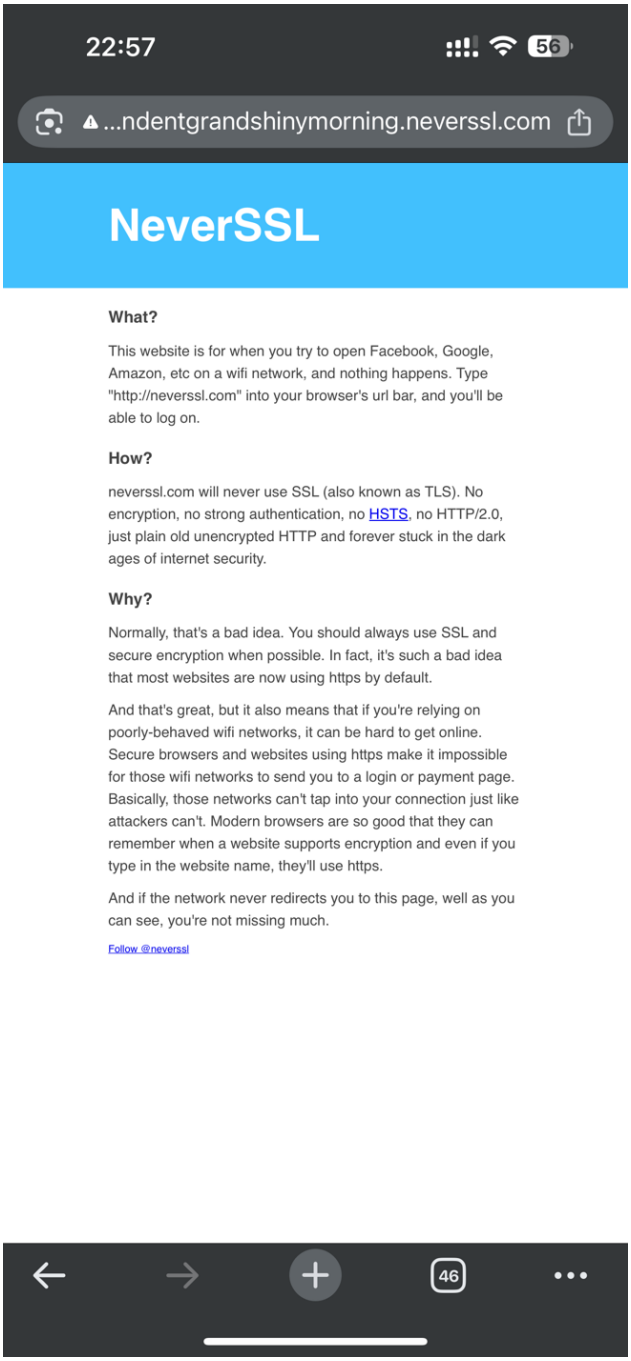


Рисунок 3.22 – Успішно наданий доступ до інтернету

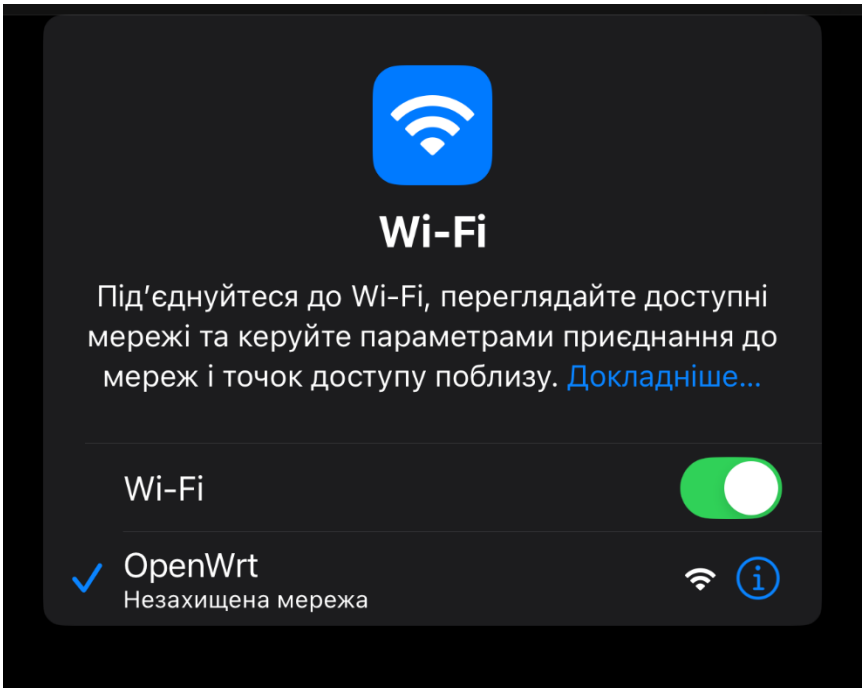


Рисунок 3.23 – Підключений девайс до точки доступу

У відповідь Keycloak повертає:

- ID Token (JWT) — містить інформацію про користувача (sub, email, name) і підписаний приватним ключем Keycloak.
- Access Token (JWT) — використовується для авторизації в API (у нашому випадку не використовується).
- Refresh Token — для подовження сесії (не використовується в даній архітектурі).
- Поточна реалізація: не виконує обмін коду на токен. Вона імітує завершення авторизації, одразу переходячи до наступного етапу.

Етап 5. Імітація підтвердження авторизації для Captive Portal.

У маршруті /callback Python-застосунок формує HTML-сторінку з автозаповненою POST-формою, результат зображено на рисунку 3.24:

```html
<form method="POST" action="http://192.168.11.1:8002/index.php?zone=test_zone">
  <input type="hidden" name="redirurl" value="http://neverssl.com">
  <input type="hidden" name="zone" value="test_zone">
  <input type="hidden" name="accept" value="Continue">
</form>
<script> document.getElementById("authform").submit(); </script>
```

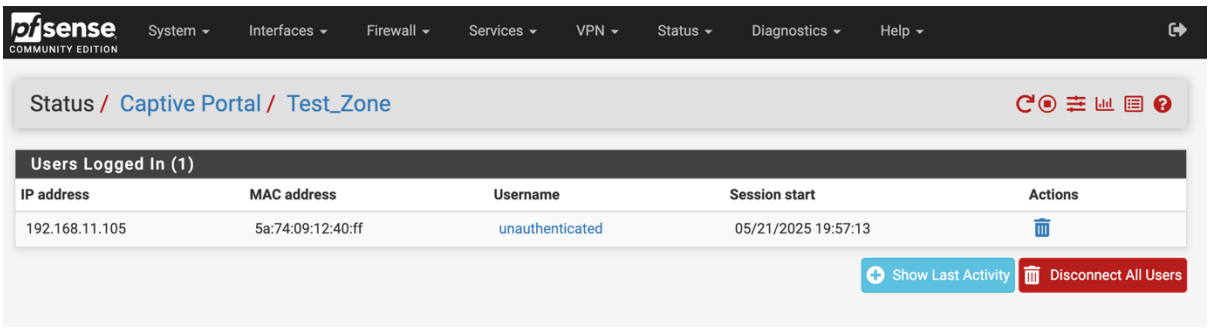


Рисунок 3.24 – Авторизований девайс на pfSense

Попри наявність спрощень, така модель є цілком робочою і демонструє архітектурну перспективу для розширення, інтеграції з SIEM, логуванням, багатофакторною автентифікацією та підключенням внутрішніх інформаційних ресурсів через єдиний ідентифікаційний провайдер.

3.16 Висновки до третього розділу

У процесі практичної реалізації дипломного проєкту було побудовано повноцінне тестове середовище, що демонструє можливість впровадження безпарольної автентифікації користувачів у Wi-Fi мережах із використанням технології Passkey, яка базується на відкритому стандарті WebAuthn.

Архітектура системи, реалізована на платформі Proxmox VE, включає інтеграцію наступних компонентів:

- pfSense як шлюз з функцією Captive Portal, що контролює доступ до мережі;
- Keycloak як ідентифікаційний провайдер, який забезпечує автентифікацію користувачів через протокол OpenID Connect із підтримкою Passkey;

- Flask-застосунок на Python, який виконує роль проміжного клієнта авторизації;
- бездротову точку доступу на базі OpenWRT, що надає підключення до мережі.

Система реалізує автоматизований процес авторизації користувача в мережі без необхідності введення паролів. Користувач проходить автентифікацію за допомогою біометрії або апаратного захищеного ключа (TPM, Secure Enclave), після чого отримує доступ до інтернету.

Запропоноване рішення забезпечує:

- сучасний рівень захисту автентифікації;
- зручність для користувачів;
- гнучкість у налаштуванні компонентів;
- відповідність міжнародним стандартам безпеки (FIDO2, WebAuthn, OpenID Connect).

Результати тестування підтвердили працездатність усіх компонентів, коректну послідовність обробки запитів, а також ефективність використання Passkey як альтернативи традиційним методам автентифікації. Архітектура показала себе стабільною, логічно узгодженою та придатною до масштабування і впровадження в більш складні сценарії (SSO, MFA, інтеграція з внутрішніми ресурсами тощо).

Таким чином, практична частина проєкту доводить технічну та прикладну доцільність впровадження безпарольної автентифікації в локальні мережі, а запропоноване рішення може виступати основою для побудови безпечних цифрових інфраструктур наступного покоління.

ВИСНОВКИ

У межах дипломної роботи було проведено комплексне дослідження традиційних і сучасних методів автентифікації користувачів у мережах з контрольованим доступом. Аналіз існуючих підходів до автентифікації показав численні недоліки паролів як основного механізму перевірки особи, включаючи їхню вразливість до фішингових атак, складність у використанні, низьку стійкість до компрометації та повторного використання. Це обґрунтувало доцільність переходу до безпарольних технологій, зокрема до використання passkeys на основі стандартів FIDO2/WebAuthn.

У теоретичній частині роботи детально розглянуто принципи функціонування безпарольної автентифікації, її основні стандарти — FIDO, WebAuthn та типи автентикаторів. Окрему увагу приділено аналізу платформи Keycloak як сучасного рішення для управління ідентичністю з підтримкою технологій безпарольної автентифікації.

Практичним результатом роботи стала розробка та впровадження архітектури для безпечної автентифікації у Wi-Fi мережі на основі взаємодії pfSense Captive Portal, Keycloak та passkey. Створена система забезпечує перенаправлення користувача з Captive Portal до сторінки авторизації Keycloak, проходження автентифікації за допомогою passkey та автоматичне додавання IP-адреси до списку дозволених після успішної перевірки. Такий підхід дозволяє досягти високого рівня безпеки та зручності, виключаючи необхідність зберігання паролів, і є стійким до сучасних атак, таких як phishing та man-in-the-middle.

Наукова новизна роботи полягає у поєднанні технології passkey з Captive Portal у контексті доступу до бездротових мереж, що дозволяє застосовувати сучасні механізми автентифікації у середовищах, де раніше домінували традиційні підходи.

Практична цінність полягає у можливості впровадження розробленого рішення у реальних умовах — корпоративних, навчальних чи публічних мережах — з мінімальними витратами та високим рівнем захисту.

Таким чином, результати цієї роботи не лише підтверджують ефективність безпарольної автентифікації на основі passkeys, але й демонструють реальну можливість її інтеграції в існуючу інфраструктуру для підвищення рівня інформаційної безпеки без зниження зручності користувачів. Розроблений підхід може стати основою для подальших досліджень та масштабованих рішень у сфері сучасної автентифікації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Thomas D. The Current State Of Authentication: We Have A Password Problem. Smashing Magazine 2016.

2. Сміт. Р. Аутентифікація: від паролей до відкритих ключів (Authentication: From Passwords to Public Keys First Edition.) / Річард Э. Сміт — М.: Вильямс, 2002. — 432 с.

3. Microsoft. Basic Authentication [Електронний ресурс] / Microsoft. – 2005. – Режим доступу: https://docs.microsoft.com/enus/windows/desktop/secauthn/basic-authentication-concepts Дата доступу: 27.10.2018

4. Офіційна документація Amazon Web Service [Електронний ресурс]/ Amazon — Режим доступу: https://docs.aws.amazon.com/en_us/cli/latest/userguide/cli-chap-gettingstarted.html Дата доступу: 27.10.2018

5. Duncan D. "Two-factor authentication" / de Borde Duncan., 2009.

6. America's Password Habits | Security.org. Security.org. URL: https://www.security.org/resources/online-password-strategies/ (дата звернення: 03.10.2024).

7. Das S., Phelan L. A., Hoyos-Rivera J. A. Password Managers Usage and Trust: A US Study of User Behavior, Preferences, and Perceptions. ACM Transactions on Privacy and Security. Vol. 24, no. 3, article 16, July 2021. URL: https://doi.org/10.1145/3447564 (date of access: 09.10.2024).

8. Saul Johnson, Jo˜ao F. Ferreira, Alexandra Mendes, and Julien Cordry. “Skeptic: Automatic, justified and privacy-preserving password composition policy selection”. In: Proceedings of the 15th ACM Asia Conf

9. What is WebAuthn? Authentication standard. Wallarm | Integrated App and API Security Platform. URL: https://www.wallarm.com/what/webauthn-webauthentication (дата звернення: 05.10.2024).

10. Gordin, A. Graur, S. Vlad and C. I. Adomniței, "Moving forward passwordless authentication: challenges and implementations for the private cloud," 20th RoEduNet Conference: Networking in Education and Research (RoEduNet), 2021, p. 1-5, doi: 10.1109/RoEduNet54112.2021.9638271 (date of access: 09.10.2024).

11. Decoding How WebAuthn Works. FusionAuth. URL: https://fusionauth.io/articles/authentication/webauthn (дата звернення: 05.10.2024).

12. Що таке ключ Passkey та як він працює? Просте пояснення | Hideez. Passwordless Workforce Identity Solutions | Hideez. URL: https://hideez.com/ukua/blogs/news/what-is-a-passkey (дата звернення: 25.11.2024).

13. Rolfe B. Synced vs Device-Bound Passkeys: How User Convenience and Authentication Experiences Vary. Authsignal - Drop-in Passkeys & Passwordless Authentication. URL: https://www.authsignal.com/blog/articles/synced-vs-devicebound-passkey s-convenience-and-authentication-experiences (date of access: 25.11.2024).

14. Device-Bound vs. Synced Passkeys (SCA & Passkeys I). Corbado - Add passkeys to any new or existing app. URL: https://corbado.com/blog/device-boundsynced-passkeys (date of access: 25.11.2024).

15. HYPR. What is a FIDO Platform Authenticator? | Security Encyclopedia. HYPR: Identity Security & Passwordless Authentication Solution. URL: https://www.hypr.com/security-encyclopedia/platform-authenticator (date of access: 25.11.2024).

16. Platform vs Cross-Platform. Yubico Developers. URL: https://developers.yubico.com/WebAuthn/WebAuthn_Developer_Guide/P latform_vs_C ross-Platform.html (date of access: 25.11.2024).

17. NIST SP 800-53 Rev. 5: Security and Privacy Controls for Information Systems and Organizations. Independently Published, 2022, 462 p.

18. Blokdyk G. Passwordless Authentication, Second Edition. 5STARCooks, 2021. 80-93 p

19. Методи аутентифікації без пароля - Wise IT Ukraine. Wise IT Ukraine. URL: https://wiseit.com.ua/metody-autentyfikacziyi-bez-parolya-vid-fudo/ (дата звернення: 30.09.2024).

20. A Study on Passwordless Authentication Technology and Its Effects. The International Journal of Reliable Information and Assurance. 2018. Vol. 6, no. 1. URL: https://doi.org/10.21742/ijria.2018.6.1.03 (date of access: 19.11.2024).

21. Passwordless Magic Links vs. Certificates for Secure Access. SecureW2. URL: https://www.securew2.com/blog/passwordless-magic-link-authenticatione xplained (date of access: 25.11.2024).

22. FIDO Authentication. *Fidoalliance*. URL: https://fidoalliance.org/fido2/.

23. Web Authentication. *W3*. URL: https://www.w3.org/TR/webauthn/

24. Web Authentication API. *Developer*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API.

25. Passkeys. *Apple*. URL: https://developer.apple.com/passkeys/.

26. Authorization services. *Keycloak*. URL: https://www.keycloak.org/docs/latest/authorization_services/index.html.

27. OpenID Connect Core 1.0. *Openid*. URL: https://openid.net/specs/openid-connect-core-1_0.html.

28. The OAuth 2.0 Authorization Framework. *Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc6749.

29. Шнайєр Б. Криптографія та безпека: практичне застосування. Діалектика, 2019.

30. Анастасов А., Стеценко В. Захист інформації в комп’ютерних системах : К. : Наукова думка, 2020.

31. Architecture. *Wazuh*. URL: https://documentation.wazuh.com/current/getting-started/architecture.html#architecture.

32. OAuth2.0 and OpenID Connect: The Professional Guide. Auth0. URL: https://auth0.com/resources/ebooks/oauth-openid-connect-professional-guide.

33. Passwordless sign-in with passkeys. Googleblog. URL: https://security.googleblog.com/.

34. WebAuthn Introduction. Yubico. URL: https://developers.yubico.com/WebAuthn/.

35. The Transport Layer Security (TLS) Protocol Version 1.3. Ietf. URL: https://datatracker.ietf.org/doc/html/rfc8446.

36. NIST Special Publication 800-63B. Nist. URL: https://pages.nist.gov/800-63-3/sp800-63b.html.

37. More than a Password. Cisa. URL: https://www.cisa.gov/MFA.

ДОДАТОК А
СПИСОК ОПУБЛІКОВАНИХ ПРАЦЬ ЗА ТЕМОЮ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Тези наукових доповідей:

Власюк Ю. Дослідження Сучасних Методів Аутентифікації та Технології Passkey / VIII МІЖНАРОДНА НАУКОВО-ПРАКТИЧНА КОНФЕРЕНЦІЯ 11 квітня 2025, КИЇВ, Україна, стр. 107-108.

ДОДАТОК Б

ЛІСТИНГ ПРОГРАМНОГО КОДУ ЗАСТОСУНКУ APP.PY

```
from flask import Flask, redirect, url_for
from authlib.integrations.flask_client import OAuth
from flask import render_template_string


app = Flask(name)
app.secret_key = "secret"  # Можеш змінити пізніше

# Налаштування OAuth для Keycloak
oauth = OAuth(app)
oauth.register(
    name='keycloak',
    client_id='captive-portal-auth',
    client_secret='zIv96KH10GAdf0JjGrr5jKhBR3VOclqO',
    access_token_url='https://kc.diploma.pixi.lat/realms/diploma/protocol/openid-
connect/token',
    authorize_url='https://kc.diploma.pixi.lat/realms/diploma/protocol/openid-
connect/auth',
    api_base_url='https://kc.diploma.pixi.lat/realms/diploma/protocol/openid-connect',
    client_kwargs={'scope': 'openid profile email'}
)

@app.route("/login")
def login():
    # редірект на Keycloak для авторизації
    redirect_uri = url_for('callback', _external=True)
    return oauth.keycloak.authorize_redirect(redirect_uri)


# Заглушка для наступного кроку
@app.route("/callback")
def callback():
#    return "Keycloak повернув користувача сюди"
    pf_url = "http://192.168.11.1:8002/index.php?zone=test_zone"

    html = f"""
    <html><body>
    <form id="authform" method="POST" action="{pf_url}">
        <input type="hidden" name="redirurl" value="http://neverssl.com">
        <input type="hidden" name="zone" value="test_zone">
        <input type="hidden" name="accept" value="Continue">
        <input type="submit" value="Continue">
    </form>
    <script>
      document.getElementById("authform").submit();
    </script>
    </body></html>
    """
    return render_template_string(html)

if name == "main":
    app.run(host="0.0.0.0", port=80)
```