МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ КІБЕРБЕЗПЕКИ ТА ЗАХИСТУ ІНФОРМАЦІЇ
КАФЕДРА УПРАВЛІННЯ КІБЕРБЕЗПЕКОЮ ТА ЗАХИСТОМ ІНФОРМАЦІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: “ОЦІНКА ЗАХИЩЕНОСТІ ПРОТОКОЛІВ АВТЕНТИФІКАЦІЇ ТА АВТОРИЗАЦІЇ У МОБІЛЬНИХ ТА ВЕБЗАСТОСУНКАХ”

на здобуття освітнього ступеня магістра
зі спеціальності 125 Кібербезпека та захист інформації
освітньо-професійної програми Управління інформаційною та кібернетичною безпекою

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Максим БОЙКО

(підпис) *Ім’я, ПРІЗВИЩЕ здобувача*

Виконав: Здобувач вищої освіти гр. УБДМ-61
Максим БОЙКО

Керівник: Дмитро РАБЧУН

Рецензент:

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут кібербезпеки та захисту інформації

Кафедра Управління кібербезпекою та захистом інформації
Ступінь вищої освіти магістр
Спеціальність 125 Кібербезпека та захист інформації
Освітньо-професійна програма Управління інформаційною та кібернетичною безпекою

ЗАТВЕРДЖУЮ
Завідувач кафедрою УКБЗІ
_____ Світлана ЛЕГОМІНОВА
"____" _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студенту Бойку Максиму Андрійовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: "Оцінка захищеності протоколів автентифікації та авторизації у мобільних і вебзастосунках"

керівник кваліфікаційної роботи РАБЧУН Димтро, к.т.н.,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від "30" жовтня 2025 р. № 467.

2. Строк подання кваліфікаційної роботи "25" грудня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: протоколи автентифікації та авторизації, архітектура мобільних і веб-застосунків, методи аналізу захищеності програмного забезпечення, стандарти інформаційної безпеки, наукова та технічна література.
4. Перелік питань, які потрібно розробити:
   1. Проаналізувати сучасні протоколи автентифікації та авторизації у мобільних і веб-застосунках.
   2. Дослідити основні вразливості та загрози безпеки протоколів автентифікації та авторизації.
   3. Вивчити методи та інструменти оцінки захищеності протоколів автентифікації та авторизації в застосунках.

5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання "02" жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назви етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Визначення об’єкту, предмету, мети та завдань дослідження.	10.10.2025	
2.	Збір та аналіз літератури.	23.10.2025	
3.	Аналіз протоколів та технологій автентифікації й авторизації у веб- та мобільних застосунках.	27.10.2025	
4.	Дослідження вразливостей та загроз безпеці механізмів автентифікації та авторизації.	10.11.2025	
5.	Розроблення підходу до оцінки захищеності протоколів автентифікації та авторизації. Підготовка тестового середовища.Проведення практичної апробації запропонованого підходу. Тестування вразливостей у веб- та мобільних застосунках.	15.11.2025	
6.	Формулювання висновків за результатами дослідження.	22.11.2025	
7.	Оформлення роботи.	04.12.2025	
8.	Оформлення презентації.	14.12.2025	
9.	Отримання рецензії на роботу.	18.12.2025	
10.	Захист в ЕК.	__.01.2026	

Здобувач вищої освіти _____ Максим БОЙКО
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Дмитро РАБЧУН
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ КІБЕРБЕЗПЕКИ ТА ЗАХИСТУ ІНФОРМАЦІЇ

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня магістра

Направляється здобувач Бойко М.А. до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю 125 Кібербезпека та захист інформації
(код, найменування спеціальності)
Освітньо-професійної програми Управління інформаційною та кібернетичною безпекою
(назва)
на тему: “Оцінка захищеності протоколів автентифікації та авторизації у мобільних та вебзастосунках”
Кваліфікаційна робота і рецензія додаються.

Директор ННІКБЗІ _____ Євгенія ІВАНЧЕНКО
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач **БОЙКО Максим** у кваліфікаційній роботі проаналізував теоретичні основи протоколів автентифікації та авторизації, вивчив методи оцінки їх захищеності у мобільних та веб-застосунках, а також дослідив практичні аспекти впровадження безпечних механізмів автентифікації та авторизації в сучасних додатках.

БОЙКО Максим показав високу теоретичну і практичну підготовку, володіння науково-дослідницькими методами, вміння самостійно знаходити шляхи вирішення проблеми дослідження. Результати дослідження апробовані на Всеукраїнській науково-практичній конференції 28.02.2025.

Все це дозволяє оцінити кваліфікаційну роботу здобувача **БОЙКО Максим** на оцінку "добре" та присвоїти йому кваліфікацію “Магістр з кібербезпеки та захисту інформації за освітньо-професійною програмою Управління інформаційною та кібернетичною безпекою”.

Керівник кваліфікаційної роботи _____ Дмитро РАБЧУН
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

“___“_____2025 року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач Бойко М.А. допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедрою
Управління кібербезпекою та захистом
інформації _____ Світлана ЛЕГОМІНОВА
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну магістерську роботу

здобувача вищої освіти Бойка Максима Андрійовича
на тему "Оцінка захищеності протоколів автентифікації та авторизації у мобільних та вебзастосунках"

Актуальність У сучасному цифровому середовищі мобільні та веб-застосунки стали невід'ємною частиною повсякденного життя мільйонів користувачів і критично важливими інструментами для бізнесу. Водночас зростає кількість кіберінцидентів, пов'язаних з компрометацією облікових записів, несанкціонованим доступом до даних та порушенням конфіденційності користувачів. Протоколи автентифікації та авторизації є першою лінією захисту від таких загроз, тому оцінка їх захищеності набуває особливого значення. З огляду на зазначене дослідження проблем оцінки захищеності протоколів автентифікації та авторизації у мобільних та веб-застосунках є актуальним науковим завданням.

Позитивні сторони

1. У роботі досліджено теоретичні основи протоколів автентифікації та авторизації, проведено аналіз сучасних стандартів (OAuth 2.0, OpenID Connect, SAML, JWT) та методів їх реалізації у мобільних та веб-застосунках. Визначено типові вразливості протоколів автентифікації та авторизації, представлено класифікацію загроз безпеці та методів їх виявлення.
2. Кваліфікаційна робота оформлена відповідно до вимог. Виклад матеріалу здійснено відповідно до плану, зроблено логічні висновки. Ключові положення роботи представлено у вигляді рисунків, таблиць та схем. Автор опрацював значну джерельну базу: близько 70 публікацій та електронних джерел, в тому числі англомовних.
3. За результатами дослідження проведено практичну оцінку захищеності протоколів автентифікації та авторизації у тестовому середовищі та запропоновано рекомендації щодо підвищення рівня безпеки досліджуваних систем.

Недоліки

1. Доцільно було б приділити більше уваги порівняльному аналізу інструментів автоматизованого тестування безпеки протоколів автентифікації та авторизації, а також розглянути перспективні напрямки розвитку технологій безпарольної автентифікації.

Однак, вищезгадані зауваження не впливають на загальну позитивну оцінку кваліфікаційної роботи.

Висновок: Кваліфікаційна робота виконана на належному науково-методичному рівні і заслуговує позитивної оцінки, а здобувач Бойко Максим Андрійович заслуговує присвоєння кваліфікації "Магістр з кібербезпеки та захисту інформації за освітньо-професійною програмою Управління інформаційною та кібернетичною безпекою".

Рецензент: _____

підпис

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 122 стор., 20 рис., 4 табл., 78 джерел.

Метою роботи є аналіз та узагальнення даних про автентифікацію та авторизацію в мобільних та веб-застосунках, систематичний огляд вразливостей та інших проблем безпеки протоколів автентифікації та авторизації, а також розроблення підходу до оцінки їх захищеності.

Об'єктом дослідження є ідентичність користувача та права користувача в розрізі веб- та мобільних застосунків.

Предмет дослідження – процеси автентифікації та авторизації в мобільних та веб-застосунках та проблеми, пов'язані з незахищеністю протоколів, технологій та неправильною реалізацією цих процесів.

Методи дослідження. Для вирішення поставлених завдань використовуються методи системного аналізу для дослідження архітектури протоколів автентифікації та авторизації, методи моделювання загроз для побудови моделі вразливостей, методи статичного та динамічного аналізу коду для виявлення вразливостей у веб- та мобільних застосунках, а також практичні методи тестування на проникнення (penetration testing) для оцінки захищеності механізмів контролю доступу.

Короткий зміст роботи. У результаті проведеного дослідження здійснено аналіз сучасних протоколів та технологій автентифікації й авторизації, виконано їх класифікацію з точки зору архітектурних особливостей та потенційних загроз безпеці. Досліджено специфіку реалізації механізмів автентифікації та авторизації у веб- та мобільних застосунках, включаючи хмарні сервіси та корпоративні інформаційні системи. Сформовано узагальнену модель типових вразливостей, характерних для механізмів автентифікації та авторизації, включаючи атаки на облікові дані, компрометацію сесій, вразливості OAuth, порушення контролю доступу, ескалацію привілеїв та загрози, специфічні для мобільних застосунків. Розроблено підхід до оцінки захищеності механізмів

автентифікації та авторизації, що ґрунтується на поєднанні теоретичних моделей загроз, практичного аналізу векторів атак та використанні індустріальних стандартів тестування безпеки. Проведено практичну апробацію запропонованого підходу в тестовому середовищі з виявленням та класифікацією вразливостей у веб- та мобільних застосунках.

Галузь застосування. Розроблені підходи та результати дослідження можуть бути використані фахівцями з кібербезпеки, розробниками програмного забезпечення, адміністраторами інформаційних систем та аудиторами безпеки при оцінці захищеності існуючих механізмів автентифікації та авторизації, розробці захищених веб- та мобільних застосунків, проведенні аудиту безпеки хмарних та корпоративних інформаційних систем, а також при впровадженні більш захищених архітектур контролю доступу.

КЛЮЧОВІ СЛОВА : АВТЕНТИФІКАЦІЯ, АВТОРИЗАЦІЯ, ВЕБ-ЗАСТОСУНКИ, МОБІЛЬНІ ЗАСТОСУНКИ, БАГАТОФАКТОРНА АВТЕНТИФІКАЦІЯ, ВРАЗЛИВОСТІ БЕЗПЕКИ, КОНТРОЛЬ ДОСТУПУ, ТЕСТУВАННЯ НА ПРОНИКНЕННЯ.

ABSTRACT

The text part of the qualification work for obtaining a master's degree: 122 pages, 20 figures, 4 tables, 78 sources.

The purpose of the work is to analyze and systematize data on authentication and authorization in mobile and web applications, conduct a systematic review of vulnerabilities and other security issues of authentication and authorization protocols, as well as develop an approach to assess their security.

Object of research is user identity and user permissions in the context of web and mobile applications.

Subject of research is the processes of authentication and authorization in mobile and web applications and problems related to the insecurity of protocols, technologies and incorrect implementation of these processes.

Research methods To solve the set tasks, methods of system analysis are used to study the architecture of authentication and authorization protocols, threat modeling methods for building a vulnerability model, static and dynamic code analysis methods for detecting vulnerabilities in web and mobile applications, as well as practical penetration testing methods for assessing the security of access control mechanisms.

Brief content of research. As a result of the conducted research, an analysis of modern authentication and authorization protocols and technologies was carried out, and their classification was performed from the point of view of architectural features and potential security threats. The specifics of implementing authentication and authorization mechanisms in web and mobile applications, including cloud services and corporate information systems, were investigated. A generalized model of typical vulnerabilities characteristic of authentication and authorization mechanisms was formed, including attacks on credentials, session compromise, OAuth vulnerabilities, access control violations, privilege escalation, and threats specific to mobile applications. An approach to assessing the security of authentication and authorization mechanisms was developed, based on a combination of theoretical threat models, practical analysis of attack vectors, and the use of industry security testing standards.

Practical testing of the proposed approach was conducted in a test environment with identification and classification of vulnerabilities in web and mobile applications.

Field of research. The developed approaches and research results can be used by cybersecurity specialists, software developers, information system administrators, and security auditors when assessing the security of existing authentication and authorization mechanisms, developing secure web and mobile applications, conducting security audits of cloud and corporate information systems, as well as when implementing more secure access control architectures.

KEYWORDS: AUTHENTICATION, AUTHORIZATION, WEB APPLICATIONS, MOBILE APPLICATIONS, MULTI-FACTOR AUTHENTICATION, SECURITY VULNERABILITIES, ACCESS CONTROL, PENETRATION TESTING.

ЗМІСТ

ВСТУП

Сучасна людина, використовуючи будь-який комп’ютерний пристрій, фактично у більшості випадків застосовує його не як автономний обчислювальний засіб, а як інструмент для встановлення з’єднання з веб-застосунками та мережею Інтернет. Особливо це характерно для найпоширенішого на сьогодні типу комп’ютерних пристроїв — мобільних (смартфонів, планшетів, ноутбуків), у межах яких основна взаємодія користувача з цифровими сервісами відбувається саме через мобільні застосунки. Переважна частина світового інтернет-трафіку генерується запитами до веб-сторінок або серверних API мобільних застосунків.

Програми для спілкування, розваг, обміну файлами, професійної діяльності, наукових досліджень та навчання функціонують на основі використання захищених шифруванням протоколів передавання даних. Базовим стандартом безпечної взаємодії у веб-середовищі є протокол , який реалізує захищений обмін даними між клієнтом і сервером із застосуванням криптографічних механізмів протоколів та сертифікатів. Саме ці технології забезпечують конфіденційність, цілісність та захист інформації від перехоплення або модифікації під час передавання мережею.

Разом із тим, сам факт наявності захищеного каналу зв’язку не гарантує повної безпеки цифрового ресурсу. Веб-сайти та мобільні застосунки додатково повинні містити вбудовані програмні механізми для ідентифікації користувачів, перевірки їхніх облікових даних та коректного розмежування повноважень доступу та відповідних ролей користувачів. Ці функції реалізуються за допомогою протоколів автентифікації та авторизації, які дозволяють встановити особу користувача, надати йому доступ до облікового запису та визначити перелік дозволених дій для конкретного користувача або особи в межах інформаційної системи.

У сучасних веб- та мобільних застосунках механізми автентифікації та авторизації реалізуються з використанням таких протоколів і технологій, як

OAuth 2.0, OpenID Connect, SAML 2.0, JWT, session-based authentication, а також технологій багатофакторної та безпарольної автентифікації (MFA, TOTP, біометрія, passwordless). Захищений обмін даними при цьому забезпечується криптографічним протоколом TLS, який є обов'язковим елементом безпечної передачі облікових даних, токенів доступу та персональної інформації користувачів.

Стрімке зростання кількості мобільних і веб-застосунків у світі призвело до суттєвого підвищення обсягів персональних, фінансових та службових даних, які передаються та обробляються в цифровому середовищі. За даними відкритих аналітичних досліджень, понад 80 % світового інтернет-трафіку сьогодні припадає саме на мобільні пристрої, а кількість активних користувачів мобільних застосунків налічує мільярди осіб. Окрему категорію сучасних інформаційних систем становлять хмарні сервіси та корпоративні платформи, які не розгортаються локально в інфраструктурі організацій, а функціонують у середовищі віддалених центрів обробки даних. До таких сервісів належать хмарні платформи на кшталт Amazon Web Services, Microsoft Azure, Google Cloud Platform, а також бізнес-системи управління ресурсами підприємств, бухгалтерського обліку, електронного документообігу, CRM- та ERP-системи.

Для більшості компаній такі сервіси є критично важливими для безперервності бізнес-процесів, оскільки забезпечують фінансові операції, управління персоналом, логістику, аналітику та взаємодію з клієнтами. На відміну від локальних систем, доступ до хмарних платформ здійснюється виключно через механізми віддаленої автентифікації та авторизації, що значно підвищує вимоги до їхньої захищеності. Компрометація облікових записів у таких середовищах може призводити не лише до витоку конфіденційної інформації, а й до повного захоплення хмарної інфраструктури, підміни бізнес-логіки, знищення даних або зупинки критичних сервісів.

Найпоширенішими загрозами у сфері автентифікації та авторизації є компрометація облікових даних, фішингові атаки, викрадення сесійних cookie, перехоплення токенів доступу, атаки повторного відтворення (replay attack),

підміна redirect-адрес у OAuth-процесах, міжсайтові атаки (CSRF, XSS), а також експлуатація уразливостей в API серверної частини мобільних застосунків. Окрему небезпеку становить зберігання токенів у незахищеній пам’яті мобільних пристроїв і використання скомпрометованих середовищ (root, jailbreak).

Наразі питання оцінки захищеності механізмів автентифікації та авторизації переважно розглядаються у прикладному та технологічному вимірах, тоді як порівняно незначна частка наукових досліджень фокусується на системному аналізі безпеки протоколів автентифікації та авторизації у контексті сучасних веб- та мобільних застосунків, що характеризуються високим рівнем динамічності, масштабованості та залежності від хмарних сервісів. Дана магістерська робота спрямована насамперед на формування теоретичних і практичних засад оцінки захищеності протоколів ідентифікації, автентифікації та контролю доступу в клієнт–серверних інформаційних системах.

У результаті проведеного дослідження здійснено аналіз сучасних протоколів та технології, а також виконано їх класифікацію з точки зору архітектурних особливостей та потенційних загроз безпеці. На основі результатів аналізу було сформовано узагальнену модель типових вразливостей, характерних для механізмів автентифікації та авторизації у веб- і мобільних застосунках.

На основі створеної моделі у роботі розроблено підхід до оцінки захищеності механізмів автентифікації та авторизації, що враховує особливості функціонування клієнтських застосунків, серверних API, хмарних сервісів та моделей віддаленого доступу до критичних бізнес-систем. Запропонований підхід ґрунтується на поєднанні теоретичних моделей загроз, практичного аналізу векторів атак та використанні існуючих індустріальних стандартів тестування безпеки.

Запропонований підхід до оцінки захищеності механізмів автентифікації та авторизації є розширюваним універсальним рішенням, яке може застосовуватися як для аналізу безпеки окремих веб- та мобільних застосунків, так і для аудиту

захищеності хмарних та корпоративних інформаційних систем. Практична апробація запропонованих рішень може бути використана під час тестування безпеки, розробки захищених програмних продуктів, а також для підвищення рівня кіберстійкості критичних для бізнесу сервісів.

Запропоновані в роботі підходи та результати можуть використовуватися фахівцями з кібербезпеки, розробниками програмного забезпечення, адміністраторами інформаційних систем та аудиторами безпеки як для оцінки існуючих реалізацій механізмів автентифікації та авторизації, так і для впровадження більш захищених архітектур контролю доступу. Запропоновані рішення можуть застосовуватися у поданому вигляді або бути розширеними з урахуванням нових загроз, технологій та архітектурних підходів.

Метою роботи є аналіз та узагальнення даних про автентифікацію та авторизацію в мобільних та веб-застосунках, систематичний огляд вразливостей та інших проблем безпеки.

Об’єкт дослідження – ідентичність користувача та прав користувача в розрізі веб- та мобільних застосунків.

Предмет дослідження – процеси автентифікації та авторизації в мобільних та веб-застосунках та проблеми пов'язані з незахищеністю протоколів, технологій та неправильній реалізації цих процесів.

РОЗДІЛ 1 ПРОТОКОЛИ АВТЕНТИФІКАЦІЇ ТА АВТОРИЗАЦІЇ

1.1 Основні поняття

Автентифікація (authentication) та авторизація (authorization) є фундаментальними концепціями інформаційної безпеки, які часто плутають через схожість термінів, проте вони виконують різні функції в системі захисту інформації [1].

Автентифікація – це процес верифікації ідентичності користувача чи системи. Цей процес передбачає підтвердження того, що суб'єкт, який намагається отримати доступ до системи, справді є тим, за кого себе видає. Автентифікація здійснюється через надання облікових даних, таких як пароль, біометричні дані, цифровий сертифікат, одноразовий код або комбінацію цих факторів.

Авторизація – це процес визначення прав доступу автентифікованого користувача до ресурсів системи. Після успішної автентифікації система визначає, які дії користувач може виконувати, до яких даних має доступ і які операції йому дозволені. Авторизація базується на політиках контролю доступу та ролях користувачів у системі [2].

Критично важливо розуміти, що ці процеси є послідовними і взаємозалежними: спочатку відбувається автентифікація (встановлення ідентичності), а потім авторизація (надання відповідних прав доступу). Порушення будь-якого з цих процесів може призвести до серйозних наслідків для безпеки застосунку [3].

1.1.1 Роль у безпеці сучасних застосунків

У сучасному цифровому середовищі автентифікація та авторизація відіграють ключову роль у забезпеченні безпеки застосунків і захисті

конфіденційних даних користувачів. Ці механізми є першою лінією оборони проти несанкціонованого доступу та зловмисних дій [4]. Належно реалізовані протоколи автентифікації та авторизації забезпечують:

– Захист персональних даних користувачів від несанкціонованого доступу

– Забезпечення цілісності інформації шляхом запобігання неавторизованим змінам

– Можливість аудиту дій користувачів для виявлення аномальної поведінки

– Дотримання регуляторних вимог щодо захисту даних, включаючи GDPR, HIPAA, PCI DSS

– Побудову довіри користувачів до цифрових сервісів

Веб-застосунки та мобільні додатки оперують величезними обсягами чутливих даних, включаючи фінансову інформацію, медичні записи, особисту кореспонденцію та бізнес-дані. Компрометація системи автентифікації чи авторизації може призвести до витоку цих даних, фінансових збитків, репутаційних втрат та юридичних наслідків для організації [5].

1.1.2 Актуальність та статистика

Аналіз статистики інцидентів інформаційної безпеки за 2024-2025 роки демонструє критичну важливість надійних механізмів автентифікації та авторизації. Згідно з дослідженням Identity Theft Resource Center, у 2024 році було зафіксовано понад 3,158 компрометацій даних, що призвело до витоку інформації більш ніж 1.7 мільярда осіб. Особливо тривожним є той факт, що понад 94% цих інцидентів могли бути запобігти шляхом впровадження багатофакторної автентифікації [6].

Середня вартість витоку даних у 2025 році становила 4.44 мільйона доларів США, причому для організацій США цей показник сягав 10.22 мільйона доларів.

Фінансовий сектор зафіксував середню вартість інциденту на рівні 6.08 мільйона доларів, тоді як охорона здоров'я залишається найдорожчим сектором протягом 14 років поспіль із середньою вартістю 7.42 мільйона доларів за інцидент. Критично важливим є той факт, що 80% всіх порушень безпеки у 2024 році були спричинені кібератаками, причому використання скомпрометованих облікових даних залишається провідним вектором атак. Дослідження виявило, що порушення безпеки, пов'язані з викраденими обліковими даними, потребували найдовшого часу для усунення – 88 днів, що формує загальний цикл витоку даних у 292 дні [7].

OWASP Top 10 містить перелік найкритичніших ризиків безпеки веб-застосунків, який оновлюється організацією Open Web Application Security Project. У редакції OWASP Top 10:2025 RC1 проблеми автентифікації та авторизації посідають важливе місце: “A07 (Identification and Authentication Failures)”, категорія, яка охоплює слабкості у процесах ідентифікації та автентифікації, включаючи відсутність багатофакторної автентифікації, використання слабких паролів, неправильну інвалідацію сесій; “A01 (Broken Access Control)”, найпоширеніша категорія вразливостей, що стосується порушень механізмів авторизації, коли користувачі можуть діяти за межами своїх дозволених прав [8].

Для мобільних застосунків OWASP Mobile Top 10 2024 виділяє категорію “M3: Insecure Authentication/Authorization”, яка об'єднує проблеми автентифікації та авторизації, підкреслюючи комплексний підхід до безпеки користувацького доступу [9].

Окремо варто відзначити OWASP API Security Top 10 2023, де автентифікація та авторизація представлені у категоріях API2:2023 Broken Authentication та “API1:2023 Broken Object Level Authorization”, що свідчить про специфічні виклики безпеки в контексті API, які є основою сучасних веб-застосунків та мобільних додатків [10].

Ці дані однозначно демонструють, що питання захищеності протоколів автентифікації та авторизації залишаються критично важливими для безпеки

сучасних застосунків і потребують постійної уваги з боку розробників, архітекторів та фахівців з інформаційної безпеки [11].

1.1.3 Специфіка веб-застосунків та мобільних додатків

Веб-застосунки та мобільні додатки мають суттєві відмінності в архітектурі, моделях загроз та підходах до реалізації автентифікації й авторизації. Розуміння цих відмінностей є критично важливим для розробки адекватних механізмів захисту.

Таблиця 1.1

Порівняння веб- та мобільних застосунків за характеристиками

Характеристика	**Веб-застосунки**	**Мобільні додатки**
Середовище виконання	Браузер (Chrome, Firefox, Safari, Edge), платформонезалежне середовище	Нативні операційні системи (iOS, Android), більш контрольоване середовище
Зберігання токенів	Cookies (httpOnly, secure), LocalStorage, SessionStorage, IndexedDB	Keychain (iOS), Keystore (Android), Secure Enclave, захищене сховище з апаратним шифруванням
Специфічні загрози	XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery), Session hijacking, Clickjacking	Реверс-інжиніринг, витяг зашитих ключів, атаки на рівні OS, небезпечне зберігання даних, перехоплення мережевого трафіку
Контроль сесій	Сесійні cookies з коротким терміном дії, автоматична інвалідація при закритті браузера	Довготривалі токени доступу з механізмами оновлення, можливість фонової автентифікації

Продовження таблиці 1.1

Характеристика	Веб-застосунки	Мобільні додатки
Протоколи автентифікації	OAuth 2.0, OpenID Connect, SAML, JWT з cookies, WebAuthn	OAuth 2.0 з PKCE, Custom URL Schemes, Universal Links, JWT з Refresh Token, біометричні API
Біометрична автентифікація	Обмежена підтримка через WebAuthn API, залежність від апаратного забезпечення пристрою	Нативна підтримка (Face ID, Touch ID, BiometricPrompt), тісна інтеграція з апаратним забезпеченням
Модель безпеки	Same-Origin Policy, Content Security Policy, CORS (Cross-Origin Resource Sharing)	Sandbox ізоляція, App Transport Security (iOS), Network Security Config (Android), Code Signing
Оновлення та патчі	Швидке оновлення на сервері, користувачі автоматично отримують оновлення при перезавантаженні сторінки	Залежність від процесу оновлення через App Store/Google Play, користувачі повинні вручну оновлювати додаток
Видимість коду	Повна видимість клієнтського JavaScript коду через Developer Tools, необхідність обфускації	Скомпільований код складніший для аналізу, але вразливий до реверс-інжинірингу, необхідність обфускації та захисту бінарних файлів

Аналіз представлених відмінностей демонструє, що веб-застосунки та мобільні додатки потребують диференційованих підходів до реалізації автентифікації та авторизації. Веб-застосунки працюють у більш відкритому середовищі браузера, що вимагає особливої уваги до захисту від клієнтських атак, таких як XSS та CSRF. Водночас мобільні додатки, хоч і мають кращі можливості для захищеного зберігання даних через апаратні механізми

шифрування, стикаються з викликами реверс-інжинірингу та необхідності захисту бінарних файлів від аналізу [12].

Обидві платформи потребують застосування передових практик безпеки, включаючи використання сучасних протоколів автентифікації, багатофакторної автентифікації, захищеного зберігання токенів та регулярного аудиту безпеки для виявлення та усунення вразливостей [13].

1.2 Методи автентифікації

Автентифікація є критичним компонентом безпеки сучасних інформаційних систем, а вибір відповідного методу автентифікації безпосередньо впливає на баланс між безпекою, зручністю використання та операційними витратами. Протягом еволюції цифрових технологій було розроблено різноманітні підходи до верифікації ідентичності користувачів, кожен з яких має специфічні характеристики, переваги та обмеження. Сучасні методи автентифікації можна класифікувати за типом використовуваного фактора: знання (knowledge factors), володіння (possession factors) та властивості (inherence factors). Парольна автентифікація представляє категорію факторів знання і залишається найпоширенішим, хоча й найбільш вразливим методом. Сертифікатна автентифікація належить до категорії факторів володіння та знань одночасно, забезпечуючи значно вищий рівень безпеки завдяки використанню криптографії з відкритими ключами. Вибір методу автентифікації залежить від контексту застосування: для споживчих веб-застосунків критичною є простота використання, тоді як для корпоративних систем та критичної інфраструктури пріоритетом є максимальна безпека. Тенденція останніх років полягає у поєднанні множинних факторів автентифікації (багатофакторна автентифікація, MFA) для досягнення оптимального балансу між безпекою та зручністю. У цьому підрозділі розглядаються два фундаментальні методи автентифікації, що становлять основу більшості сучасних систем безпеки: парольна автентифікація

з її сучасними удосконаленнями та сертифікатна автентифікація на базі інфраструктури відкритих ключів.

1.2.1 Автентифікація на основі паролю

Парольна автентифікація залишається найпоширенішим методом верифікації ідентичності користувачів у сучасних веб-застосунках та мобільних додатках. Механізм базується на принципі “знання секрету” – користувач підтверджує свою ідентичність, надаючи унікальну комбінацію логіна (ідентифікатора) та пароля (секретної інформації), яка відома лише йому. Процес парольної автентифікації включає наступні етапи: користувач вводить свої облікові дані, система порівнює надану інформацію з хешованим значенням, збереженим у базі даних, і у разі збігу надає доступ до ресурсів. Критично важливим є той факт, що паролі ніколи не повинні зберігатися у відкритому вигляді – натомість використовуються криптографічні хеш-функції для створення незворотних відбитків паролів.

Незважаючи на широке розповсюдження, парольна автентифікація має низку суттєвих недоліків та вразливостей:

– Слабкі паролі – користувачі часто обирають легко запам'ятовувані, але передбачувані паролі (наприклад, “password123”, “qwerty”), що робить їх вразливими до атак методом перебору та словникових атак.

– Повторне використання паролів – практика використання одного пароля для множини сервісів створює ефект доміно, компрометація одного облікового запису автоматично ставить під загрозу всі інші акаунти користувача.

– Фішинг – соціальна інженерія залишається ефективним методом крадіжки облікових даних, коли користувачі добровільно надають свої паролі на підроблених сторінках.

– Атаки грубої сили (brute-force) – автоматизовані системи можуть здійснювати мільйони спроб входу за секунду, особливо якщо відсутні механізми обмеження кількості спроб.

– Витік баз даних – масштабні компрометації баз даних призводять до витоку мільйонів облікових записів, які потім використовуються в атаках credential stuffing.

– Людський фактор – користувачі схильні записувати паролі на паперових носіях, зберігати їх у незахищених текстових файлах або ділитися ними з іншими особами.

Дослідження Verizon Data Breach Investigations Report 2024 свідчить, що 81% порушень безпеки, пов'язаних з хакінгом, відбуваються через використання викрадених або слабких паролів. Це підкреслює критичну необхідність посилення парольної автентифікації додатковими механізмами захисту [14].

Для захисту паролів від компрометації у разі витоку бази даних використовуються криптографічні хеш-функції. Сучасні вимоги безпеки передбачають застосування спеціалізованих алгоритмів, розроблених саме для хешування паролів, які забезпечують стійкість до атак методом перебору [15].

Bcrypt – адаптивна хеш-функція, розроблена на базі шифру Blowfish, яка включає вбудований механізм “солі” (salt) та параметр “cost factor”, що визначає обчислювальну складність хешування. Основною перевагою bcrypt є можливість налаштування часу обчислення хешу, що дозволяє збільшувати стійкість до атак у міру зростання потужності обчислювальної техніки. Типовий час обчислення bcrypt становить 250-500 мілісекунд, що є достатнім для створення значної затримки при спробах масового перебору, але не впливає на користувацький досвід при легітимній автентифікації.

Argon2 – переможець конкурсу Password Hashing Competition 2015 року, представляє собою найсучасніший алгоритм хешування паролів. Argon2 існує у трьох варіантах: Argon2d (оптимізований для захисту від GPU-атак), Argon2i (оптимізований для захисту від атак по сторонніх каналах) та Argon2id (гібридна версія, рекомендована для більшості застосувань). Ключовою особливістю

Argon2 є можливість налаштування не лише обчислювальної складності, але й використання пам'яті, що робить атаки з використанням спеціалізованого обладнання (ASIC, GPU) економічно недоцільними. OWASP рекомендує використовувати Argon2id як пріоритетний вибір для нових систем, а bcrypt як надійну альтернативу для існуючих реалізацій. Критично важливим є використання достатньо високих параметрів складності, бо для bcrypt вартість становить мінімум 10 мегабайт, а для Argon2id – мінімум 19 мегабайт пам'яті та 2 ітерації [16].

Багатофакторна автентифікація (Multi-Factor Authentication, MFA) є критичним засобом підвищення безпеки, що вимагає надання двох або більше незалежних факторів для підтвердження ідентичності. Фактори автентифікації поділяються на три категорії: щось, що ви знаєте (пароль, PIN-код), щось, чим ви володієте (фізичний токен, смартфон), та щось, чим ви є (біометричні дані). TOTP (Time-based One-Time Password) – алгоритм генерації одноразових паролів на основі поточного часу, стандартизований у RFC 6238. TOTP генерує шестизначний код, що змінюється кожні 30 секунд, використовуючи спільний секретний ключ між сервером та клієнтським пристроєм (зазвичай мобільним додатком-автентифікатором, таким як Google Authenticator, Microsoft Authenticator або Authy). Основною перевагою TOTP є робота без підключення до мережі та відсутність необхідності SMS-повідомлень, що робить його стійким до атак перехоплення комунікацій. Проте TOTP залишається вразливим до фішингу, оскільки користувач може ввести згенерований код на підробленій сторінці [17].

WebAuthn/FIDO2 – сучасний стандарт автентифікації, розроблений консорціумом FIDO Alliance та стандартизований W3C, який забезпечує стійку до фішингу криптографічну автентифікацію. WebAuthn використовує асиметричну криптографію: приватний ключ зберігається на пристрої користувача (в апаратному токені, смартфоні або вбудованому модулі TPM/Secure Enclave), тоді як публічний ключ реєструється на сервері [18]. Процес автентифікації включає криптографічний виклик від сервера, який

підписується приватним ключем користувача, що робить неможливим фішинг, оскільки підпис прив'язаний до конкретного домену [19]. WebAuthn підтримує як зовнішні апаратні ключі (YubiKey, Titan Security Key), так і вбудовані автентифікатори платформи (Windows Hello, Touch ID, Face ID) [20].

Біометрична автентифікація – метод верифікації ідентичності на основі унікальних фізіологічних або поведінкових характеристик людини [21]. Найпоширеніші біометричні методи включають:

– Сканування відбитків пальців – використовується в мобільних пристроях (Touch ID, датчики Android) та ноутбуках для швидкої автентифікації.

– Розпізнавання обличчя – технологія 3D-сканування (Face ID) забезпечує високу точність та стійкість до підробок, використовуючи інфрачервоні проєктори та сенсори глибини.

– Сканування райдужної оболонки – забезпечує найвищу точність ідентифікації, але вимагає спеціалізованого обладнання.

– Голосова автентифікація – аналіз голосових характеристик для верифікації особи, часто використовується в телефонних системах.

Критично важливим аспектом біометричної автентифікації є безпечне зберігання біометричних шаблонів. Сучасні реалізації використовують захищені апаратні модулі (Secure Enclave в iOS, StrongBox в Android), де біометричні дані зберігаються в зашифрованому вигляді та ніколи не передаються за межі пристрою [22]. Натомість біометрія розблоковує криптографічний ключ, який використовується для автентифікації. Дослідження Identity Theft Resource Center 2024 показало, що впровадження MFA може запобігти 94% успішних кібератак на облікові записи. Це робить багатофакторну автентифікацію обов'язковою вимогою для систем, що обробляють конфіденційні дані.

Сучасним трендом є Passkeys. Passkeys представляють собою революційний підхід до автентифікації, який має потенціал повністю замінити традиційні паролі. Технологія базується на стандарті WebAuthn та криптографічних протоколах FIDO2, але з удосконаленим користувацьким досвідом та можливістю синхронізації між пристроями.

На відміну від традиційних паролів, passkeys є парою криптографічних ключів (приватний та публічний), що генеруються автоматично для кожного веб-сайту чи застосунку. Приватний ключ надійно зберігається в менеджері паролів операційної системи (iCloud Keychain для Apple, Google Password Manager для Android, Windows Hello) та захищений біометричною автентифікацією або PIN-кодом пристрою. Публічний ключ зберігається на сервері сервісу.

Основні переваги passkeys:

– Стійкість до фішингу – криптографічні ключі прив'язані до конкретного домену, що робить неможливим використання на підробленій сторінці.

– Захист від витоків баз даних – навіть у разі компрометації серверу зловмисники отримають лише публічні ключі, які неможливо використати для автентифікації.

– Усунення слабких паролів – користувачі не обирають паролі, тому відсутні проблеми слабких або повторюваних паролів.

– Покращений користувацький досвід – автентифікація здійснюється за допомогою біометрії або PIN-коду пристрою без необхідності пам'ятати унікальні паролі для кожного сервісу.

– Синхронізація між пристроями – passkeys автоматично синхронізуються через хмарні екосистеми (iCloud, Google Account), забезпечуючи доступ з різних пристроїв.

Великі технологічні компанії активно впроваджують підтримку passkeys: Apple додала підтримку в iOS 16 та macOS Ventura, Google – в Android 9+, Microsoft – в Windows 10/11. Провідні веб-сервіси, включаючи Google, Microsoft, PayPal, Amazon, GitHub, вже підтримують автентифікацію через passkeys. FIDO Alliance прогнозує, що до 2025-2026 років passkeys стануть домінуючим методом автентифікації для споживчих сервісів, фактично здійснюючи перехід до “post-password” ери цифрової безпеки.

1.2.2 Автентифікація на основі сертифікату

X.509 та базові концепції Сертифікатна автентифікація базується на використанні цифрових сертифікатів стандарту X.509 – міжнародного стандарту, що визначає формат сертифікатів відкритих ключів у інфраструктурі відкритих ключів (Public Key Infrastructure, PKI). Цифровий сертифікат X.509 є електронним документом, що засвідчує приналежність публічного ключа конкретному суб'єкту (користувачу, серверу, організації) та підписаний довіреним центром сертифікації (Certificate Authority, CA) [23].

Структура сертифіката X.509 включає наступні ключові компоненти:

– Інформація про суб'єкта – ідентифікаційні дані власника сертифіката (Common Name, Organization, Country).

– Публічний ключ – криптографічний ключ, що використовується для шифрування та верифікації цифрових підписів.

– Інформація про емітента – дані центру сертифікації, що видав сертифікат.

– Термін дії – період валідності сертифіката (NotBefore та NotAfter).

– Цифровий підпис CA – криптографічний підпис центру сертифікації, що гарантує автентичність та цілісність сертифіката.

– Розширення – додаткові поля, що визначають призначення сертифіката, обмеження використання, альтернативні імена тощо.

Процес автентифікації з використанням сертифікатів базується на асиметричній криптографії. Клієнт, що володіє приватним ключем, відповідає на криптографічний виклик сервера, підписуючи його своїм приватним ключем. Сервер верифікує цифровий підпис, використовуючи публічний ключ з сертифіката, та перевіряє ланцюжок довіри до кореневого центру сертифікації. Успішна верифікація підтверджує, що клієнт дійсно володіє приватним ключем, пов'язаним з наданим сертифікатом [24].

Критичною перевагою сертифікатної автентифікації є можливість взаємної автентифікації (mutual authentication), коли обидві сторони комунікації підтверджують свою ідентичність за допомогою цифрових сертифікатів. Це забезпечує високий рівень довіри в обох напрямках комунікації.

Mutual TLS (mTLS) – розширення протоколу Transport Layer Security, що забезпечує взаємну автентифікацію між клієнтом та сервером з використанням сертифікатів X.509. На відміну від стандартного TLS, де лише сервер надає сертифікат для підтвердження своєї ідентичності, у mTLS обидві сторони обмінюються сертифікатами та взаємно верифікують один одного [25].

Процес встановлення mTLS-з'єднання включає наступні етапи:

– TLS Handshake – клієнт ініціює з'єднання та отримує сертифікат сервера.

– Верифікація сервера – клієнт перевіряє валідність сертифіката сервера, включаючи перевірку ланцюжка довіри до довіреного CA, терміну дії та відповідності домену.

– Запит клієнтського сертифіката – сервер запитує у клієнта надати його сертифікат.

– Верифікація клієнта – сервер перевіряє валідність клієнтського сертифіката аналогічно до верифікації серверного сертифіката.

– Встановлення захищеного каналу – після успішної взаємної автентифікації встановлюється шифроване з'єднання.

mTLS забезпечує найвищий рівень безпеки комунікації, оскільки гарантує автентичність обох сторін та шифрування всього трафіку. Протокол є стійким до атак типу “людина посередині” (man-in-the-middle), оскільки зловмисник не зможе надати валідний сертифікат, підписаний довіреним CA [26].

Сертифікатна автентифікація знаходить широке застосування в сценаріях, де критично важливою є висока надійність верифікації ідентичності та захист конфіденційної інформації:

– Корпоративні мережі та VPN – організації використовують клієнтські сертифікати для автентифікації співробітників при доступі до внутрішніх ресурсів через VPN. Це забезпечує строгий контроль доступу та можливість централізованого управління сертифікатами через корпоративний CA.

– Микросервисна архітектура та Service Mesh – mTLS є стандартом для забезпечення безпечної комунікації між мікросервісами в розподілених системах. Рішення типу Istio, Linkerd, Consul автоматизують управління сертифікатами та забезпечують mTLS для всього service-to-service трафіку.

– IoT та M2M комунікації – пристрої Інтернету речей використовують сертифікати для автентифікації при підключенні до хмарних платформ. Це особливо важливо для промислових IoT-систем, де безпека є критичною вимогою.

– API автентифікація – критичні API, особливо в фінансовому секторі та охороні здоров'я, використовують сертифікатну автентифікацію для верифікації клієнтів. Стандарт Open Banking, наприклад, вимагає використання кваліфікованих сертифікатів для доступу до банківських API.

– Цифровий підпис документів – сертифікати використовуються для створення кваліфікованих електронних підписів, що мають юридичну силу згідно з регуляціями eIDAS та законодавством України про електронний цифровий підпис.

– Підпис коду – розробники підписують програмне забезпечення та мобільні додатки сертифікатами для підтвердження їх автентичності та цілісності. Це запобігає поширенню шкідливого ПЗ, замаскованого під легітимні застосунки.

– Безпека пошти (S/MIME) – сертифікати X.509 використовуються для шифрування та підпису електронної пошти, забезпечуючи конфіденційність, автентичність та неспростовність повідомлень.

Незважаючи на високий рівень безпеки, сертифікатна автентифікація має певні операційні складнощі: необхідність управління життєвим циклом сертифікатів (видача, оновлення, відкликання), підтримка інфраструктури PKI, складність розгортання для великої кількості користувачів. Тому сертифікатна автентифікація найчастіше застосовується в корпоративному середовищі та для machine-to-machine комунікацій, тоді як для споживчих застосунків використовуються більш зручні методи, такі як OAuth 2.0 з багатофакторною автентифікацією або passkeys [27].

1.3 Протоколи авторизації

Авторизація визначає права доступу автентифікованих користувачів до ресурсів системи. Сучасні протоколи авторизації вирішують специфічні виклики цифрової екосистеми: делеговану авторизацію, федеративну ідентичність та безпечну комунікацію між різнорідними системами. OAuth 2.0 став де-факто стандартом для делегованої авторизації, OpenID Connect розширює його для автентифікації, а традиційні сесійні механізми продовжують використовуватися в монолітних застосунках [28].

1.3.1 OAuth 2.0

OAuth 2.0 являє собою протокол авторизації, специфікований у RFC 6749, який забезпечує можливість надання застосункам третіх сторін обмеженого доступу до ресурсів користувача без необхідності передачі облікових даних. Протокол вирішує фундаментальну проблему безпечного делегування доступу: надання застосунку можливості взаємодії з даними користувача без розкриття його автентифікаційної інформації.

Функціонування OAuth 2.0 базується на взаємодії чотирьох основних ролей. Resource Owner (власник ресурсу) представляє кінцевого користувача,

який володіє даними та має право надавати доступ до них. Client (клієнт) є застосунком, що ініціює запит на отримання доступу до захищених ресурсів. Authorization Server (сервер авторизації) відповідає за видачу токенів доступу після успішної автентифікації користувача. Resource Server (сервер ресурсів) зберігає захищені дані та обробляє запити на основі валідних токенів [29].

Протокол оперує двома категоріями токенів, які виконують різні функції у процесі авторизації. Access Token є короткостроковим токеном доступу з терміном дії від однієї до двадцяти чотирьох годин, що використовується для безпосереднього доступу до захищених ресурсів. Refresh Token представляє довгостроковий токен, призначений для отримання нових токенів доступу після закінчення терміну дії попередніх без повторної участі користувача.

Найбезпечнішим механізмом авторизації для всіх типів клієнтських застосунків є потік авторизаційного коду (Authorization Code Flow). Розширення PKCE (Proof Key for Code Exchange), визначене у RFC 7636, забезпечує додатковий рівень захисту для публічних клієнтів, таких як мобільні додатки та односторінкові веб-застосунки (SPA) [30].

Процес авторизації з використанням PKCE розпочинається з генерації криптографічних параметрів, коли клієнт створює Code Verifier (випадковий рядок довжиною 43-128 символів) та Code Challenge (SHA-256 хеш від Code Verifier). Далі формується запит на авторизацію з обов'язковими параметрами response_type=code, client_id, redirect_uri, scope, state, code_challenge та code_challenge_method=S256. Після автентифікації користувача та отримання його згоди на надання доступу через інтерфейс Authorization Server відбувається повернення авторизаційного коду та параметра стану через механізм перенаправлення. Наступним кроком є обмін авторизаційного коду на токени з обов'язковою верифікацією code_verifier, після чого клієнт отримує access_token, refresh_token та супутні метадані. Використання токену доступу здійснюється у HTTP-заголовку Authorization з схемою Bearer.

Механізм PKCE забезпечує захист від атак перехоплення коду: навіть у разі компрометації авторизаційного коду зловмисник не зможе обміняти його на токени без знання оригінального code_verifier.

Для сценаріїв міжсерверної взаємодії (machine-to-machine) без участі кінцевого користувача застосовується потік облікових даних клієнта (Client Credentials Flow). У цьому режимі застосунок автентифікується безпосередньо через пару client_id та client_secret, отримуючи токен доступу для взаємодії з власними ресурсами або ресурсами, до яких йому надано доступ. Типові сценарії використання включають backend-to-backend комунікацію, інструменти командного рядка, IoT-пристрої та пакетну обробку даних.

Сучасні стандарти безпеки не рекомендують використання деяких механізмів авторизації. Implicit Flow характеризується вразливістю до витоку токенів через URL та не забезпечує належного рівня захисту. Resource Owner Password Credentials Flow порушує базовий принцип OAuth 2.0 щодо нерозголошення облікових даних і створює додаткові ризики безпеки.

Валідація URI перенаправлення відіграє критичну роль у забезпеченні безпеки протоколу. Система підтримує білий список дозволених URI для кожного зареєстрованого клієнта з вимогою точної відповідності без використання масок підстановки для публічних клієнтів. Обов'язковим є використання протоколу HTTPS за винятком localhost у розробному середовищі, а також підтримка платформо-специфічних URI-схем для мобільних застосунків.

Параметр стану (State Parameter) забезпечує захист від атак підробки міжсайтових запитів (CSRF). Клієнт генерує криптографічно стійке випадкове значення розміром мінімум 128 біт, включає його до запиту авторизації та верифікує при отриманні відповіді від сервера.

Управління життєвим циклом токенів передбачає різні підходи для різних типів токенів. Access tokens мають термін дії 15-60 хвилин, зберігаються в оперативній пам'яті або httpOnly cookies та передаються виключно через захищене з'єднання HTTPS. Refresh tokens підлягають ротації при кожному використанні з автоматичним відкликанням у разі виявлення повторного

використання. Відкликання токенів здійснюється через спеціалізований endpoint згідно з RFC 7009.

Оновлена специфікація RFC 9700 встановлює ряд обов'язкових вимог для підвищення рівня безпеки. Використання PKCE стає обов'язковим для всіх типів клієнтських застосунків, а Implicit Flow повністю заборонений. Запроваджується обов'язкова ротація Refresh Token з механізмом виявлення повторного використання та строга відповідність URI перенаправлення (Strict Redirect URI Matching). Впроваджуються токени з прив'язкою до відправника (Sender-Constrained Access Tokens) через механізм DPoP (Demonstrating Proof-of-Possession). Ці вимоги спрямовані на усунення відомих вразливостей та забезпечення сучасного рівня захисту авторизаційних механізмів у розподілених системах.

1.3.2 OpenID Connect

OpenID Connect (OIDC) являє собою протокол автентифікації, побудований поверх OAuth 2.0, який доповнює базовий протокол стандартизованими механізмами верифікації ідентичності користувача. Принципова відмінність між протоколами полягає у їхньому призначенні – OAuth 2.0 забезпечує авторизацію та надання доступу до ресурсів, тоді як OpenID Connect фокусується на автентифікації та достовірному підтвердженні ідентичності користувача.

Центральним елементом OpenID Connect є ID Token, який реалізовано у форматі JWT (JSON Web Token) та складається з трьох частин, розділених крапками: Header, Payload та Signature. Заголовок (Header) містить метадані токену, зокрема алгоритм підпису (найчастіше RS256), тип токену (JWT) та ідентифікатор криптографічного ключа (kid), який використовується для верифікації підпису. Корисне навантаження (Payload) містить набір обов'язкових claims, що забезпечують базову функціональність автентифікації. Параметр iss (Провайдер) вказує URL-адресу OpenID Provider, який видав токен. Параметр sub

(Subject) містить унікальний ідентифікатор користувача в межах даного провайдера. Параметр aud (Audience) вказує Client ID застосунку, для якого призначено токен. Параметр exp (Expiration) визначає час закінчення дії токену у форматі Unix timestamp. Параметр iat (Issued At) фіксує момент видачі токену. Додатково токен може містити розширені claims, такі як auth_time (час останньої автентифікації користувача), nonce (одноразове значення для захисту від повторних атак), acr (рівень довіри автентифікації), amr (використані методи автентифікації), email (електронна адреса), name (повне ім'я) та picture (URL фотографії користувача).

Верифікація ID Token є критичним етапом забезпечення безпеки автентифікації. Процес включає кілька обов'язкових перевірок. Спочатку здійснюється валідація криптографічного підпису токену за допомогою публічного ключа, отриманого з JWKS endpoint провайдера. Далі виконується перевірка коректності параметрів iss (відповідність очікуваному провайдеру), aud (відповідність Client ID застосунку) та exp (актуальність токену). У разі використання параметра nonce під час запиту автентифікації обов'язково перевіряється його відповідність отриманому значенню в токені.

OpenID Connect визначає стандартизовані категорії claims для представлення інформації про користувача. Категорія Profile охоплює базові персональні дані, включаючи повне ім'я (name), прізвище (family_name), ім'я (given_name), фотографію (picture) та веб-сайт (website). Категорія Email містить електронну адресу (email) та статус її верифікації (email_verified). Категорія Address представляє структурований об'єкт з повною поштовою адресою користувача. Категорія Phone включає номер телефону (phone_number) та статус його верифікації (phone_number_verified).

Отримання інформації про користувача може здійснюватися двома способами: безпосередньо з ID Token, який містить обрані claims, або через окремий HTTP-запит до спеціалізованого UserInfo endpoint, що дозволяє отримати повний набір доступних даних про автентифікованого користувача.

UserInfo endpoint повертає JSON-об'єкт з claims відповідно до наданих під час автентифікації дозволів (scopes).

Фундаментальна різниця між протоколами проявляється у їхній функціональній спрямованості та механізмах реалізації. OAuth 2.0 призначений виключно для авторизації, надаючи застосункам обмежений доступ до ресурсів користувача через механізм Access Token. Протокол не визначає стандартизованих способів отримання інформації про ідентичність користувача, зосереджуючись на делегуванні прав доступу. OpenID Connect розширює можливості OAuth 2.0, додаючи рівень автентифікації через введення ID Token у форматі JWT. Протокол стандартизує процес верифікації ідентичності користувача, надає структуровані механізми отримання персональної інформації через claims та UserInfo endpoint, а також забезпечує єдиний підхід до Single Sign-On (SSO) у розподілених системах. Таким чином, OpenID Connect може розглядатися як комплексне рішення, що поєднує автентифікацію користувача з авторизаційними можливостями базового протоколу OAuth 2.0, забезпечуючи повний цикл управління доступом у сучасних веб-додатках та API [31].

1.3.3 Сесійна авторизація

Сесійна авторизація представляє традиційний підхід до управління автентифікованими з'єднаннями, заснований на серверному зберіганні стану сесії та ідентифікації користувача через механізм cookies. Цей метод характеризується збереженням інформації про активні сесії на сервері з використанням унікальних ідентифікаторів для зв'язку з клієнтськими запитами.

Процес сесійної авторизації розпочинається з автентифікації користувача через надання облікових даних. Після успішної верифікації сервер генерує унікальний ідентифікатор сесії (Session ID), який служить ключем для подальшої ідентифікації користувача. На сервері створюється запис сесії, що містить релевантну інформацію про автентифікованого користувача та контекст його

взаємодії з системою. Згенерований Session ID відправляється клієнту у вигляді HTTP cookie, який автоматично додається браузером до всіх наступних запитів до того самого домену. При отриманні запиту сервер здійснює валідацію Session ID через пошук відповідного запису сесії у сховищі та перевірку його актуальності.

Конфігурація cookies має критичне значення для забезпечення безпеки сесійної авторизації. Рекомендована конфігурація заголовку Set-Cookie включає наступні параметри: sessionId зі значенням ідентифікатора, атрибут HttpOnly для захисту від XSS-атак, атрибут Secure для забезпечення передачі виключно через HTTPS-з'єднання, атрибут SameSite=Strict для запобігання CSRF-атакам, параметр Max-Age для визначення тривалості життя cookie та Path для обмеження області дії. Атрибут HttpOnly забороняє доступ до cookie через JavaScript, що ефективно запобігає його компрометації у разі успішної XSS-атаки. Атрибут Secure гарантує, що cookie передається виключно через захищене HTTPS-з'єднання, унеможливлюючи перехоплення через незахищені канали. Атрибут SameSite регулює поведінку cookie у контексті міжсайтових запитів: значення Strict забезпечує найвищий рівень безпеки, повністю блокуючи передачу cookie у cross-site запитах; значення Lax забезпечує баланс між безпекою та функціональністю; значення None дозволяє cross-site використання за умови наявності атрибута Secure.

Вибір механізму зберігання сесій на сервері значно впливає на характеристики системи. Зберігання в оперативній пам'яті (In-Memory) забезпечує найшвидший доступ до даних сесій, проте характеризується втратою всієї інформації при перезапуску сервера та обмеженою масштабованістю у розподілених системах. Використання реляційних або NoSQL баз даних гарантує персистентність даних та можливість проведення аудиту, однак супроводжується нижчою швидкістю доступу порівняно з in-memory рішеннями. Спеціалізовані системи кешування, такі як Redis або Memcached, поєднують високу швидкість доступу з можливістю горизонтального масштабування, при цьому Redis додатково забезпечує опціональну персистентність даних.

Захист від атак типу Session Fixation досягається через обов'язкову регенерацію Session ID одразу після успішної автентифікації користувача, що унеможливлює використання попередньо відомих зловмиснику ідентифікаторів. Система таймаутів включає idle timeout тривалістю 15-30 хвилин для автоматичного завершення неактивних сесій та absolute timeout тривалістю 8-24 години для обмеження максимальної тривалості будь-якої сесії незалежно від активності користувача. У записі сесії зберігаються додаткові метадані для підвищення безпеки, зокрема IP-адреса клієнта, User-Agent браузера та CSRF token для захисту від атак підробки міжсайтових запитів.

Порівняння сесійної та токен-базованої авторизації

Фундаментальна різниця між підходами полягає у принципі зберігання стану автентифікації. Сесійна авторизація передбачає збереження стану на сервері з передачею клієнту лише ідентифікатора, що забезпечує централізований контроль та можливість миттєвого відкликання доступу, проте ускладнює горизонтальне масштабування та вимагає додаткових механізмів синхронізації у розподілених системах. Токен-базована авторизація реалізує stateless підхід, де вся необхідна інформація міститься безпосередньо в токені, що спрощує масштабування та міжсерверну взаємодію, однак ускладнює процес відкликання токенів до закінчення їхнього терміну дії. Backend-for-Frontend (BFF)Pattern представляє оптимальне рішення для односторінкових застосунків (SPA), поєднуючи переваги обох підходів. У цій архітектурі короткостроковий access token зі терміном дії 5-15 хвилин зберігається в оперативній пам'яті браузера, мінімізуючи ризики компрометації. Довгостроковий refresh token розміщується в HttpOnly Secure SameSite cookie, що унеможливлює його доступність для JavaScript-коду та забезпечує захист від XSS-атак. Backend-for-Frontend компонент виступає посередником, обробляючи процес оновлення access token через refresh token без участі frontend-коду. Такий підхід забезпечує оптимальний баланс між безпекою, зручністю розробки та користувацьким досвідом у сучасних веб-додатках [34].

1.4 JSON Web Tokens (JWT)

JSON Web Token (RFC 7519) є компактним URL-безпечним засобом передачі claims між сторонами в розподілених системах. JWT став де-факто стандартом для передачі ідентифікаційної інформації завдяки властивості самодостатності: токен містить всю необхідну інформацію без потреби звернення до бази даних, що є критичним для stateless архітектур та горизонтально масштабованих систем.

JWT складається з трьох частин, з'єднаних крапками у форматі Header.Payload.Signature. Заголовок (Header) містить метадані токена у форматі JSON, закодованому за допомогою Base64URL. Основні поля заголовка включають алгоритм підпису (alg), який може приймати значення HS256, RS256 або ES256, тип токена (typ) зі значенням JWT, а також ідентифікатор ключа (kid), що використовується для верифікації підпису. Корисне навантаження (Payload) містить claims про користувача у форматі JSON, також закодованому Base64URL. Claims поділяються на три категорії: зареєстровані (зареєстровані claims) включають стандартні поля iss, sub, aud, exp, iat, nbf та jti; публічні (public claims) містять загальноприйняті поля як-от name, email та roles; приватні (private claims) представляють кастомні поля, специфічні для конкретного застосунку. Критично важливо розуміти, що Payload не шифрується, а лише кодується Base64URL, тому категорично заборонено включати до нього конфіденційні дані, такі як паролі або фінансова інформація. Підпис (Signature) забезпечує криптографічну гарантію цілісності та автентичності токена. Підпис створюється шляхом підписування закодованих Header та Payload за допомогою обраного криптографічного алгоритму та секретного або приватного ключа [35].

Вибір алгоритму підпису має фундаментальне значення для архітектури безпеки системи. HS256 використовує симетричний алгоритм HMAC-SHA256 з єдиним секретним ключем мінімальною довжиною 256 біт. Цей підхід ідеально підходить для внутрішніх систем, де всі компоненти є довіреними та мають доступ до спільного секрету. Перевагою є висока продуктивність та простота

реалізації, недоліком виступає необхідність захищеного розповсюдження секретного ключа між всіма сервісами. RS256 використовує асиметричний алгоритм RSA з мінімальним розміром ключа 2048 біт та парою ключів: приватним та публічним. Приватний ключ зберігається виключно на Authorization Server та використовується для створення підпису, тоді як публічний ключ вільно розповсюджується для верифікації токенів будь-якими сервісами. Цей підхід ідеальний для розподілених систем, мікросервісної архітектури та публічних API, оскільки спрощує процес верифікації без необхідності захищеного зберігання секретів на кожному сервісі. Вибір алгоритму визначається архітектурними вимогами: RS256 рекомендується для мікросервісів, публічних API, систем з високими вимогами безпеки та при необхідності простої ротації ключів; HS256 доцільний для монолітних застосунків, внутрішніх токенів та систем з критичними вимогами до продуктивності. Важливою вразливістю є Algorithm Confusion Attack, коли деякі недосконалі бібліотеки дозволяють зміну значення alg з RS256 на HS256, що дозволяє зловмиснику підписати токен, використовуючи публічний ключ як HMAC-секрет. Захист реалізується через явну специфікацію дозволених алгоритмів у коді верифікації та категоричну заборону прийняття токенів з alg=none.

Зареєстровані claims визначені стандартом та мають чітку семантику. Поле iss (Провайдер) містить URL емітента токена, що дозволяє верифікувати джерело. Поле sub (Subject) представляє унікальний ідентифікатор користувача в системі емітента. Поле aud (Audience) визначає одержувача токена у форматі рядка або масиву рядків. Поле exp (Expiration) містить Unix timestamp, що визначає момент закінчення дії токена. Поле nbf (Not Before) вказує час, до якого токен вважається неактивним. Поле iat (Issued At) фіксує момент видачі токена. Поле jti (JWT ID) містить унікальний ідентифікатор токена, що використовується для реалізації механізмів blacklist.

Мінімальний термін життя токенів становить критичний аспект безпеки: access tokens повинні мати термін дії 5-15 хвилин, а для довгострокового доступу

використовуються refresh tokens. Критичні валідації включають обов'язкову верифікацію криптографічного підпису, перевірку exp для запобігання використанню застарілих токенів, перевірку iss для підтвердження довіреного емітента та перевірку aud для гарантії, що токен призначений саме для цього сервісу. Безпека payload вимагає категоричної заборони включення конфіденційних даних: паролів, номерів платіжних карток, номерів соціального страхування, медичної інформації та будь-яких персональних ідентифікаційних даних. Використання HTTPS є обов'язковим для всіх операцій з JWT, оскільки токени передаються виключно через захищені з'єднання. Механізми відкликання токенів (token revocation) реалізуються через ведення blacklist з використанням jti, встановлення короткого TTL та ротацію refresh tokens. Категорично забороняється приймати токени з alg="none", що представляють незахищені токени без підпису. Використання kid забезпечує підтримку ротації ключів без порушення роботи системи. Обмеження розміру payload до 2KB забезпечує оптимальну продуктивність, оскільки токени передаються з кожним HTTP-запитом. Регулярна ротація ключів підпису рекомендується кожні 6-12 місяців для HS256 та кожні 12-24 місяці для RS256. Використання перевірених бібліотек, таких як jsonwebtoken для Node.js, PyJWT для Python або jjwt для Java, мінімізує ризики вразливостей у реалізації.

Вибір методу зберігання JWT у браузері має критичне значення для безпеки застосунку. LocalStorage надає простий API, не передається автоматично з запитами (що виключає CSRF-атаки) та дозволяє зберігати до 5MB даних. Однак цей підхід характеризується критичною вразливістю до XSS-атак, оскільки токен доступний для будь-якого JavaScript-коду на сторінці. Вердикт: LocalStorage не рекомендується для застосунків, що обробляють чутливі дані. HttpOnly cookies забезпечують захист від XSS-атак, оскільки токен недоступний для JavaScript-коду. Атрибут SameSite захищає від CSRF-атак, а браузер автоматично управляє відправкою cookie. Обмеженнями є максимальний розмір 4KB та потреба в додатковому CSRF-захисті при відсутності SameSite. Рекомендована конфігурація включає атрибути HttpOnly, Secure, SameSite=Strict та Max-

Age=900. Вердикт: рекомендовано для традиційних веб-застосунків з серверним рендерингом. BFF Pattern (Backend-for-Frontend) представляє найбезпечніший підхід для односторінкових застосунків (SPA). Access token зберігається виключно в оперативній пам'яті JavaScript з терміном дії 5-15 хвилин, що мінімізує вікно компрометації. Refresh token розміщується в HttpOnly Secure SameSite cookie, унеможливлюючи доступ через XSS. Backend-for-Frontend компонент обробляє процес оновлення токенів, забезпечуючи максимальний захист від XSS та CSRF атак одночасно.

Для iOS-платформи рекомендується використання Keychain з апаратним шифруванням через Secure Enclave. Система забезпечує ізоляцію даних між застосунками та підтримує біометричний захист через Face ID або Touch ID. Рекомендований рівень доступу становить kSecAttrAccessibleWhenUnlocked, що дозволяє доступ до токенів лише при розблокованому пристрої. Категорично заборонено зберігати токени в UserDefaults або звичайних файлах.

Для Android-платформи оптимальним є поєднання Android Keystore та EncryptedSharedPreferences. Keystore забезпечує апаратну підтримку через Trusted Execution Environment (TEE) або StrongBox та використовується для генерації ключів шифрування, які ніколи не покидають захищене середовище. Токени зберігаються в зашифрованому вигляді в EncryptedSharedPreferences з підтримкою біометричного захисту через BiometricPrompt API. Категорично заборонено використовувати SharedPreferences без шифрування.

Device binding через fingerprint прив'язує токен до конкретного пристрою, ускладнюючи використання викрадених токенів. Certificate pinning для мобільних застосунків захищає від man-in-the-middle атак. Логування повинно здійснюватися без включення токенів у plain text форматі. Система alerts для виявлення аномальної активності дозволяє своєчасно реагувати на потенційні інциденти безпеки. Graceful degradation при втраті токенів забезпечує коректну поведінку застосунку та можливість повторної автентифікації користувача без критичних помилок [32].

1.5 Принципи безпечної реалізації

Безпечна реалізація механізмів автентифікації та авторизації вимагає не лише використання сучасних протоколів, але й дотримання фундаментальних принципів інформаційної безпеки. Навіть найнадійніші криптографічні протоколи можуть бути скомпрометовані через помилки імплементації або недотримання базових принципів безпеки.

Моделі контролю доступу використовуються для структурованого управління правами користувачів, фундаментальні принципи безпеки для побудови стійких систем, та механізми захисту комунікацій для запобігання перехопленню конфіденційних даних.

RBAC (Role-Based Access Control) є найпоширенішою моделлю контролю доступу, що базується на концепції ролей як проміжного рівня між користувачами та дозволами. Права групуються в ролі, які потім призначаються користувачам [36].

Основні компоненти:

– Користувачі (Users) – суб'єкти, що потребують доступу

– Ролі (Roles) – іменовані набори дозволів (Адміністратор, Редактор, Читач)

– Дозволи (Permissions) – права на виконання операцій над ресурсами

– Сесії (Sessions) – активація підмножини ролей користувача

Рівні RBAC згідно з NIST:

– Flat RBAC. Базова модель з користувачами, ролями та дозволами. Користувач може мати множину ролей, роль призначається множині користувачів.

– Hierarchical RBAC. Додає ієрархію ролей з успадкуванням дозволів. Старша роль успадковує всі дозволи молодших ролей (наприклад, менеджер успадковує дозволи підлеглого).

– Constrained RBAC. Додає обмеження Separation of Duty – користувач не може мати конфліктуючі ролі одночасно.

– Symmetric RBAC. Комбінує ієрархію ролей та обмеження для складних корпоративних систем.

Таблиця 1.2

Приклад матриці дозволів за ролями та операціями

Операція	**Admin**	**Editor**	**Reviewer**	**Viewer**
Створити статтю	Так	Так	Ні	Ні
Редагувати статтю	Так	Так (власні)	Ні	Ні
Видалити статтю	Так	Так (власні)	Ні	Ні
Опублікувати	Так	Ні	Ні	Ні
Коментувати	Так	Так	Так	Ні
Переглядати чернетки	Так	Так (власні)	Так	Ні
Переглядати опубліковане	Так	Так	Так	Так
Керувати користувачами	Так	Ні	Ні	Ні

Безпечна реалізація механізмів автентифікації та авторизації вимагає не лише використання сучасних протоколів, але й дотримання фундаментальних принципів інформаційної безпеки. Навіть найнадійніші криптографічні протоколи можуть бути скомпрометовані через помилки імплементації або недотримання базових принципів безпеки. Моделі контролю доступу використовуються для структурованого управління правами користувачів, фундаментальні принципи безпеки застосовуються для побудови стійких систем, а механізми захисту комунікацій запобігають перехопленню конфіденційних даних [37].

Role-Based Access Control (RBAC) є найпоширенішою моделлю контролю доступу, що базується на концепції ролей як проміжного рівня між користувачами та дозволами. Права групуються в ролі, які потім призначаються

користувачам. Основні компоненти моделі включають користувачів (Users) як суб'єктів, що потребують доступу, ролі (Roles) як іменовані набори дозволів (наприклад, Адміністратор, Редактор, Читач), дозволи (Permissions) як права на виконання операцій над ресурсами, та сесії (Sessions) для активації підмножини ролей користувача.

Стандарт NIST визначає чотири рівні RBAC. Flat RBAC представляє базову модель з користувачами, ролями та дозволами, де користувач може мати множину ролей, а роль призначається множині користувачів. Hierarchical RBAC додає ієрархію ролей з успадкуванням дозволів, де старша роль автоматично успадковує всі дозволи молодших ролей (наприклад, Менеджер успадковує дозволи Співробітника). Constrained RBAC додає обмеження Separation of Duty, згідно з якими користувач не може одночасно мати конфліктуючі ролі. Symmetric RBAC комбінує ієрархію ролей та обмеження для реалізації складних корпоративних систем [38].

Переваги RBAC включають спрощене управління через модифікацію ролей замість індивідуальних налаштувань, природну відповідність організаційній структурі підприємства, реалізацію принципу найменших привілеїв, простоту проведення аудиту та масштабованість при зростанні системи. Проте модель має певні виклики. Role Explosion виникає при надмірній деталізації, коли створюється занадто багато специфічних ролей. Система характеризується негнучкістю для контекстуальних умов, таких як час доступу або локація користувача. Складність підтримки великих ієрархій зростає при розширенні організаційної структури.

Найкращі практики впровадження RBAC передбачають мінімізацію кількості ролей на основі реальних посад в організації, надання кожній ролі тільки необхідного мінімуму дозволів, регулярний аудит актуальності ролей, детальну документацію призначення кожної ролі та автоматизацію процесів через інтеграцію з HR-системами для своєчасного оновлення прав доступу.

Attribute-Based Access Control (ABAC) приймає рішення на основі атрибутів користувача, ресурсу, дії та контексту. На відміну від RBAC, ABAC

дозволяє динамічно оцінювати політики доступу з урахуванням множини факторів. Компоненти моделі включають атрибути суб'єкта (відділ, посада, рівень допуску, локація), атрибути ресурсу (тип документа, рівень конфіденційності, власник), атрибути дії (читати, писати, видаляти, експортувати) та атрибути середовища (час, IP-адреса, тип пристрою, рівень загрози) [39].

Приклад політики ABAC демонструє складну логіку: дозволити читання документа, якщо користувач належить до відділу-власника АБО має рівень допуску більший або рівний конфіденційності документа АБО є власником документа, та якщо доступ здійснюється з корпоративної мережі, в робочі години від 9:00 до 18:00, з зареєстрованого пристрою. Переваги ABAC включають високу гнучкість у визначенні політик, контекстуальність прийняття рішень, природну масштабованість та детальний контроль доступу. Недоліки охоплюють складність початкового налаштування, повільнішу оцінку порівняно з RBAC та ускладнений процес аудиту. Гібридний підхід, який використовує більшість сучасних систем, передбачає застосування RBAC для базового управління доступом (приблизно 80% сценаріїв) та ABAC для специфічних випадків з додатковими контекстуальними умовами (решта 20%).

Defense in Depth (глибокоешелонований захист) являє собою стратегію багаторівневої безпеки з множиною незалежних рівнів захисту, де компрометація одного рівня не призводить до повного порушення безпеки системи, оскільки наступні рівні продовжують захищати ресурси. Архітектура включає сім послідовних рівнів. Мережевий периметр забезпечується через Firewall, Web Application Firewall (WAF), захист від DDoS-атак та IP «білі списки»ing. Транспортний рівень захищається через TLS/HTTPS, certificate pinning, HSTS headers та заборону застарілих версій TLS. Рівень застосунку включає rate limiting, CAPTCHA, валідацію вхідних даних та CSRF tokens. Автентифікація базується на MFA, сильних алгоритмах хешування паролів (Argon2id або bcrypt), Passkeys та біометрії. Авторізація реалізується через RBAC або ABAC з токенами короткого TTL та валідацією JWT claims. Рівень даних забезпечує шифрування

at-rest, безпечне зберігання токенів та маскування чутливої інформації в логах. Моніторинг включає логування спроб автентифікації, alerting при виявленні аномалій, інтеграцію з SIEM-системами та процедури incident response.

Практичний приклад демонструє послідовну роботу рівнів захисту: атака на endpoint логіну блокується WAF, який виявляє SQL ін’єкції; rate limiting обмежує кількість спроб автентифікації; CAPTCHA активується після трьох невдалих спроб; MFA вимагає другий фактор автентифікації; session monitoring виявляє аномальну поведінку; logging зберігає повні дані для подальшого аналізу та розслідування інциденту [41].

Принцип найменших привілеїв (Least Privilege) визначає, що кожен суб'єкт системи має мінімальний набір привілеїв, необхідний для виконання його функцій, без жодних додаткових прав. Застосування цього принципу для користувачів передбачає надання доступу тільки до необхідних ресурсів, призначення нових користувачів з мінімальними правами та надання додаткових прав виключно при обґрунтованій потребі. Токени в OAuth 2.0 обмежуються через scopes, JWT містить мінімум claims, встановлюється короткий TTL від 5 до 15 хвилин, а refresh tokens отримують обмежений scope. Сервіси оперують окремими service accounts з мінімальними правами, Client Credentials надаються з обмеженим scope згідно з принципом need-to-know. База даних конфігурується з окремими обліковими записами для застосунків, використанням read-only прав де це можливо та обмеженнями на рівні таблиць [42].

Just-In-Time Access забезпечує надання підвищених привілеїв тільки на час виконання конкретної задачі з автоматичним відкликанням після завершення, обов'язковим approval workflow та повним аудитом всіх операцій. Переваги застосування принципу найменших привілеїв включають мінімізацію впливу компрометації облікових записів, зменшення поверхні атаки системи, спрощення процесу аудиту та ефективний захист від внутрішніх загроз (інсайдерів).

Zero Trust ("Ніколи не довіряй, завжди перевіряй") представляє парадигму безпеки, що повністю відмовляється від концепції "довіреного периметра", визначаючи, що жоден користувач, пристрій або сервіс не є довіреним за

замовчуванням незалежно від розташування в мережі. Три фундаментальні принципи визначають архітектуру Zero Trust. Verify Explicitly вимагає автентифікації на основі всіх доступних даних, включаючи ідентичність, локацію, стан пристрою та виявлені аномалії, з підтримкою continuous authentication. Use Least Privilege Access реалізується через Just-In-Time та Just-Enough-Access підходи, адаптивні політики доступу та мікросегментацію мережі. Assume Breach передбачає, що компрометація вже відбулася, тому необхідна мінімізація blast radius через сегментацію, end-to-end шифрування та постійний моніторинг [40].

Компоненти Zero Trust включають Identity Verification через обов'язкову MFA, passwordless підходи (Passkeys, FIDO2), біометрію та continuous authentication через behavioral analytics. Device Trust забезпечується через перевірку device compliance, інтеграцію з MDM/UEM системами, health attestation та certificate-based authentication. Network Segmentation реалізується через мікросегментацію на основі ідентичності, Software-Defined Perimeter та ізоляцію ресурсів. Application Security досягається через mTLS для service-to-service комунікації, API Gateway з автентифікацією та context-aware policies. Data Protection включає класифікацію даних, шифрування at-rest та in-transit, системи DLP та права доступу на основі класифікації.

Adaptive Authentication динамічно змінює вимоги на основі оцінки ризику поточної операції. При низькому ризику, коли користувач звертається зі знайомої локації, у звичайний час та з відомого пристрою, достатньо пароля або passkey. Середній ризик при новій локації або новому пристрої вимагає пароль плюс TOTP або SMS. Високий ризик при поєднанні нової локації та нового пристрою або при доступі до чутливих даних потребує пароль, TOTP, біометрію, додаткову верифікацію та повідомлення адміністратора. Критичний ризик при множинних аномаліях призводить до блокування доступу, вимоги повної переавторизації та manual approval від відповідального адміністратора. Переваги Zero Trust включають захист від lateral movement при компрометації, ефективний контроль

remote workforce, відповідність вимогам compliance, захист від інсайдерів та повну visibility всіх операцій.

Transport Layer Security (TLS) є криптографічним протоколом для забезпечення безпечної комунікації в мережі, а HTTPS застосовує TLS для захисту веб-трафіку. Протокол виконує три основні функції. Конфіденційність забезпечується через шифрування всіх даних алгоритмами AES-GCM або ChaCha20-Poly1305, що запобігає перехопленню токенів та паролів. Цілісність досягається через HMAC або AEAD механізми для виявлення модифікації даних, захист від tampering та гарантію автентичності інформації. Автентифікація реалізується через верифікацію сервера за допомогою X.509 сертифіката, можливість mTLS для автентифікації обох сторін та перевірку ланцюжка довіри [43].

TLS Handshake включає послідовність етапів: ClientHello з переліком cipher suites та версій, ServerHello з вибором cipher suite та сертифікатом, Certificate Verification для перевірки валідності, Key Exchange через ECDHE, Finished для підтвердження та Encrypted Communication для всієї подальшої взаємодії.

Найкращі практики включають використання виключно TLS 1.2 або вище, переважно TLS 1.3, оскільки SSL 2.0/3.0 є критично вразливими, а TLS 1.0/1.1 офіційно застарілими. Cipher suites повинні надавати пріоритет AEAD алгоритмам (AES-GCM, ChaCha20-Poly1305) та забезпечувати Forward Secrecy через ECDHE з видаленням слабких алгоритмів (RC4, 3DES, CBC). Сертифікати мають отримуватися від довірених центрів сертифікації (Let's Encrypt, DigiCert), використовувати RSA 2048+ біт або ECDSA P-256+, покривати всі домени та автоматично оновлюватися. HSTS header з конфігурацією Strict-Transport-Security: max-age=31536000; includeSubDomains; preload забезпечує примусове використання HTTPS. Додатково рекомендується заборона mixed content, OCSP Stapling та регулярний аудит через SSL Labs Server Test.

Certificate Pinning представляє техніку додаткового захисту мобільних застосунків, що прив'язує застосунок до конкретного сертифіката або публічного ключа, запобігаючи MITM атакам навіть при компрометації центру сертифікації.

Certificate Pinning прив'язує до повного X.509 сертифіката, але вимагає оновлення застосунку при оновленні сертифіката. Public Key Pinning, який є рекомендованим підходом, прив'язується до публічного ключа з перевагою використання того самого ключа у різних сертифікатах [44].

Імплементація для iOS здійснюється через NSURLSession delegate з перевіркою SHA-256 хешу публічного ключа. Для Android використовується Network Security Config XML з вказанням SHA-256 pins, expiration date та backup pins. Найкращі практики включають конфігурацію множинних pins (primary та backup) для захисту від lock-out, встановлення expiration date для автоматичного fallback, pinning публічного ключа замість сертифіката, динамічне оновлення через remote configuration та ретельне тестування з правильними та неправильними сертифікатами [45].

Ризики certificate pinning включають lock-out користувачів при втраті контролю над сертифікатами, складність управління lifecycle та потенційне блокування корпоративних HTTPS-інспектуючих проксі. Рішення передбачають опцію вимкнення pinning для корпоративних пристроїв та динамічне оновлення pins через віддалену конфігурацію. Альтернативи для веб-застосунків включають HPKP, який є deprecated через ризик lock-out і не повинен використовуватися, та Certificate Transparency для моніторингу виданих сертифікатів, виявлення unauthorized сертифікатів через Expect-CT header.

Висновки щодо транспортної безпеки визначають, що TLS 1.2 або вище є обов'язковим для всіх комунікацій без винятків, HTTPS повинен використовуватися для всього трафіку, Certificate Pinning рекомендується для критичних мобільних застосунків у банкінгу, фінансах та охороні здоров'я, необхідний регулярний аудит TLS конфігурації та моніторинг Certificate Transparency logs для виявлення несанкціонованої видачі сертифікатів [46].

РОЗДІЛ 2 ВРАЗЛИВОСТІ ТА ЗАГРОЗИ

2.1 Модель загроз

Методологія STRIDE, розроблена компанією Microsoft, є однією з найпоширеніших та найбільш структурованих підходів до моделювання загроз. STRIDE представляє собою мнемонічну абревіатуру шести категорій загроз: підміна ідентичності (Spoofing), модифікація даних (Tampering), відмова від дій (Repudiation), розкриття інформації (Information Disclosure), відмова в обслуговуванні (Denial of Service) та підвищення привілеїв (Elevation of Privilege). Кожна категорія відображає специфічний тип атаки, що порушує один з фундаментальних принципів інформаційної безпеки [47].

Застосування методології STRIDE до систем автентифікації та авторизації дозволяє систематично проаналізувати всю поверхню атаки – сукупність точок, через які зловмисник може спробувати атакувати систему. Поверхня атаки включає всі компоненти системи, що взаємодіють з користувачами або зовнішніми системами: веб-форми автентифікації, кінцеві точки програмного інтерфейсу, токени в процесі передачі та зберігання, сесійні механізми, сервери авторизації та бази даних облікових записів.

Детальний розгляд застосування методології STRIDE до систем автентифікації та авторизації передбачає аналіз конкретних векторів атак для кожної категорії загроз, оцінку їх потенційного впливу на безпеку системи та визначення поверхні атаки сучасних веб та мобільних застосунків. Методологія STRIDE забезпечує структурований підхід для систематичної ідентифікації загроз через призму шести категорій. Кожна категорія STRIDE безпосередньо відповідає порушенню одного з ключових принципів безпеки: автентичності, цілісності, неспростовності, конфіденційності, доступності та авторизації [48].

Підміна ідентичності (Spoofing Identity) представляє найбільш пряму загрозу системам автентифікації, оскільки успішна підміна ідентичності надає

зловмиснику повний доступ до облікового запису жертви. Сучасні атаки автоматичного підбору облікових даних використовують бази викрадених паролів з попередніх порушень безпеки, які містять мільярди комбінацій логін-пароль, для автоматизованих спроб входу на різні сервіси, експлуатуючи проблему повторного використання паролів користувачами. Особливу небезпеку представляють фішингові атаки, які еволюціонували від простих підроблених електронних листів до витончених атак з ідеальними копіями легітимних веб-сайтів, включаючи валідні сертифікати безпечного з'єднання через атаки підміни символів або компрометовані піддомени. Єдиним надійним захистом від фішингу є використання методів автентифікації, стійких до фішингу, таких як WebAuthn/FIDO2 та Passkeys, де криптографічний підпис прив'язаний до конкретного домену.

Модифікація даних (Tampering) включає атаки на токени JWT, що є поширеною вразливістю при неправильній реалізації. Класична атака полягає у зміні алгоритму підпису з RS256 на значення "відсутній", що дозволяє створити начебто валідний токен без підпису. Інший вектор атаки передбачає зміну тверджень у корисному навантаженні, наприклад, зміну ролі з рівня користувача на рівень адміністратора, у токенах зі слабким алгоритмом підпису або при відсутності валідації підпису на стороні сервера. Модифікація параметрів у потоках OAuth, особливо зміна адреси перенаправлення, може призвести до атак перехоплення коду авторизації. Захист забезпечується через сувору валідацію адрес перенаправлення на сервері авторизації та використання параметра стану для виявлення спроб модифікації.

Відмова від дій (Repudiation) часто недооцінюється, однак відсутність належного аудиту може мати серйозні наслідки. У разі інциденту безпеки відсутність вичерпних журналів робить неможливим криміналістичний аналіз для визначення масштабу компрометації, постраждалих користувачів та векторів атаки. Це також створює проблеми відповідності нормативним вимогам, оскільки багато регуляцій вимагають детального журналування та аудиторських слідів. Критично важливим є журналування не тільки успішних автентифікацій,

але й невдалих спроб, джерела запиту (мережева адреса, ідентифікатор браузера, географічне розташування), часових міток, змін у правах доступу та подій життєвого циклу токенів (видача, оновлення, відкликання).

Розкриття інформації (Information Disclosure) є критичною проблемою, оскільки корисне навантаження JWT не є шифрованим – це просто кодування за стандартом Base64URL. Включення конфіденційних даних (електронна пошта, телефон, номер соціального страхування, медична інформація) у корисне навантаження JWT призводить до їх розкриття для будь-кого, хто отримає доступ до токена. Токени можуть потрапити в історію браузера (якщо передані в адресному рядку), журнали сервера, інструменти розробника браузера або бути викрадені через атаки міжсайтового скриптингу при зберіганні в локальному сховищі браузера. Порушення безпеки баз даних з незахищеними або слабко захешованими паролями залишаються однією з найпоширеніших причин масових компрометацій. Використання застарілих алгоритмів (MD5, SHA-1) або відсутність солі робить паролі вразливими до атак з використанням заздалегідь обчислених таблиць. Єдиний прийнятний підхід передбачає використання сучасних алгоритмів хешування паролів (Argon2id, bcrypt, scrypt) з належними параметрами.

Відмова в обслуговуванні (Denial of Service) має специфічну характеристику для систем автентифікації, навіть якщо головний застосунок залишається доступним, неможливість автентифікації робить його повністю нефункціональним для користувачів. Атаки грубої сили генерують величезне навантаження на службу хешування паролів, особливо при використанні правильно налаштованих ресурсомістких алгоритмів (Argon2 з 19 мегабайтами пам'яті, bcrypt з коефіцієнтом складності 12). Атаки блокування облікових записів представляють цікаву категорію, де зловмисник навмисно викликає блокування облікових записів легітимних користувачів через множинні невдалі спроби входу. Це вимагає балансу між безпекою (блокування після певної кількості невдалих спроб) та доступністю (недопущення можливості зловмиснику блокувати користувачів). Рішення включають прогресивні

затримки, системи верифікації людини та обмеження частоти запитів на рівні мережевої адреси.

Підвищення привілеїв (Elevation of Privilege) представляє критичну категорію загроз, оскільки успішна атака надає зловмиснику доступ до функцій та даних, що значно перевищують його легітимні права. Консорціум відкритої безпеки веб-застосунків стабільно ранжує порушений контроль доступу як найпоширенішу вразливість веб-застосунків. Незахищені прямі посилання на об'єкти залишаються поширеною вразливістю, коли застосунок не перевіряє, чи має поточний користувач право доступу до запитуваного ресурсу. Приклад: кінцева точка програмного інтерфейсу для отримання документів дозволяє отримати документи будь-якого користувача, просто змінюючи ідентифікатор в адресному рядку, якщо відсутня належна авторизаційна перевірка.

Горизонтальна ескалація привілеїв (доступ до ресурсів інших користувачів з тим самим рівнем привілеїв) та вертикальна ескалація привілеїв (отримання адміністративних або вищих прав) часто є результатом відсутності або неправильної реалізації авторізаційних перевірок на серверній частині. Обмеження на рівні клієнтського інтерфейсу (приховування елементів інтерфейсу) не є достатнім захистом, оскільки можуть бути легко обійдені через прямі запити до програмного інтерфейсу.

Поверхня атаки представляє сукупність усіх точок входу, через які зловмисник може взаємодіяти з системою автентифікації та авторизації. Систематичне картування поверхні атаки дозволяє визначити пріоритети для тестування безпеки та розподілу ресурсів захисту. Поверхня атаки охоплює декілька рівнів системи:

1. Мережевий периметр: міжмережеві екрани, застосункові міжмережеві екрани, захист від розподілених атак відмови в обслуговуванні, білі списки мережевих адрес.

2. Транспортний рівень: захищені протоколи передачі даних, прив'язка сертифікатів, заголовки примусового використання захищеного з'єднання, заборона застарілих версій протоколів.

3. Рівень застосунку: обмеження частоти запитів, системи верифікації людини, валідація вхідних даних, токени захисту від підробки міжсайтових запитів.

4. Автентифікація: багатофакторна автентифікація, сильні алгоритми хешування паролів, ключі доступу, біометричні методи.

5. Авторізація: рольовий контроль доступу, контроль доступу на основі атрибутів, токени з коротким терміном життя, валідація тверджень токенів.

6. Дані: шифрування даних у стані спокою, безпечне зберігання токенів, маскування чутливої інформації в журналах.

7. Моніторинг: журналування спроб автентифікації, сповіщення при виявленні аномалій, інтеграція з системами управління інформацією та подіями безпеки, процедури реагування на інциденти.

Клієнтська сторона представляє значну поверхню атаки через обмежений контроль над середовищем виконання. Атаки міжсайтового скриптингу дозволяють викрасти токени з локального сховища браузера через просте виконання відповідного коду. Мобільні застосунки вразливі до зворотної інженерії, особливо на пристроях з розширеними правами, де зловмисник може витягти токени із захищеного сховища або перехопити виклики програмного інтерфейсу через інструменти налагодження [49].

Мережевий рівень залишається актуальною точкою атаки, особливо в публічних бездротових мережах. Навіть при використанні захищеного протоколу передачі даних, відсутність прив'язки сертифіката в мобільних застосунках дозволяє зловмиснику встановити власний сертифікат через спеціалізовані інструменти та перехоплювати весь трафік, включаючи токени та облікові дані. Серверні кінцеві точки є основною метою для автоматизованих атак. Точки входу в систему зазнають мільйонів спроб грубої сили щодня. Процедури скидання пароля часто мають вразливості типу передбачуваних токенів, відсутності обмеження частоти запитів або можливості перевірки існування облікового запису через різний час відповіді або повідомлення для існуючих та неіснуючих користувачів.

Серверні компоненти, хоча й менш доступні безпосередньо, представляють цілі найвищої цінності. Компрометація бази даних користувачів надає доступ до всіх облікових записів. Незахищене управління секретами (жорстко закодовані ключі, секрети в системі контролю версій, слабке шифрування) є поширеною причиною порушень безпеки.

Кожен компонент поверхні атаки піддається специфічним категоріям загроз за методологією STRIDE. Форма входу в систему піддається підміні ідентичності через автоматичний підбір облікових даних, відмові в обслуговуванні через атаки грубої сили та розкриттю інформації через атаки часового аналізу для перевірки існування облікових записів. Токени JWT вразливі до підміни через підробку підпису, модифікації через зміну тверджень та розкриття інформації при включенні чутливих даних у корисне навантаження. Потоки OAuth піддаються підміні через викрадення коду авторизації, модифікації через маніпуляцію адресою перенаправлення та підвищенню привілеїв через розширення області дозволів. Авторізаційні перевірки вразливі до підвищення привілеїв через обхід захисту та незахищені прямі посилання на об'єкти, а також модифікації через маніпуляцію параметрами. База даних користувачів піддається розкриттю інформації через порушення безпеки з паролями, модифікації через ін'єкції структурованих запитів та відмові від дій через відсутність аудиту змін.

Систематичне застосування методології STRIDE до кожного компонента поверхні атаки дозволяє створити вичерпну модель загроз, що охоплює всі реалістичні вектори атак та забезпечує основу для пріоритизації засобів контролю безпеки та діяльності з тестування.

2.2 Вразливості автентифікації

Вразливості автентифікації представляють одну з найкритичніших категорій безпекових проблем у сучасних веб та мобільних застосунках. За

даними OWASP Top 10 2021, ідентифікація та автентифікаційні помилки (A07:2021 Identification and Authentication Failures) залишаються серед найпоширеніших та найнебезпечніших вразливостей. Успішна компрометація автентифікації надає зловмиснику прямий доступ до облікових записів користувачів, що може призвести до масштабних витоків даних, фінансових збитків та репутаційних втрат.

2.2.1 Атаки на основі облікових даних

Атаки на основі облікових даних експлуатують слабкості в механізмах валідації та зберігання паролів для отримання несанкціонованого доступу до системи. Ці атаки еволюціонували від простих методів грубої сили до складних автоматизованих кампаній, що використовують мільярди викрадених облікових даних з попередніх порушень безпеки [50].

Атаки методом грубої сили представляють систематичний перебір можливних комбінацій паролів до знаходження правильного варіанту. Традиційний підхід передбачає повний перебір всіх можливих комбінацій символів, проте його ефективність залежить від довжини пароля та набору використовуваних символів. Для пароля довжиною вісім символів з дев'яноста чотирма можливими символами кількість комбінацій становить приблизно шість цілих одна десята помножити на десять у п'ятнадцятому степені, що робить повний перебір практично нездійсненним для онлайн-атак при належних захисних механізмах.

Словникові атаки використовують попередньо складені словники поширених паролів, які базуються на реальних витоках даних. Найвідоміший з таких списків, rockyou.txt, містить чотирнадцять мільйонів реальних паролів, отриманих внаслідок порушення безпеки однойменного сайту. Дослідження демонструють, що приблизно десять відсотків користувачів обирають паролі з

переліку тисячі найпоширеніших варіантів, що робить словникові атаки надзвичайно ефективними проти систем без належного захисту.

Гібридні атаки комбінують словникові підходи з правилами мутацій, застосовуючи типові модифікації до базових слів. Користувачі часто створюють паролі шляхом додавання цифр до кінця слова, заміни літер на схожі за виглядом символи або капіталізації першої літери. Автоматизовані інструменти атак систематично застосовують ці трансформації, значно розширюючи ефективність словникових атак без необхідності повного перебору всіх можливих комбінацій. Критична вразливість CVE-2023-22515 в Atlassian Confluence демонструє серйозність загроз автентифікації. Ця вразливість дозволяла віддаленим зловмисникам створювати облікові записи адміністраторів без будь-якої автентифікації, отримуючи повний контроль над системою. Вразливість отримала максимальний бал CVSS 10.0 та була активно експлуатована в реальних умовах протягом тижнів до публічного розкриття, що призвело до компрометації численних корпоративних систем та витоку конфіденційних даних [51].

Credential stuffing представляє еволюційований підхід до атак на облікові дані, використовуючи автоматизацію та масштаб для експлуатації проблеми повторного використання паролів користувачами. За даними Akamai State of the Internet Report 2023, протягом року було зафіксовано понад сто дев'яносто три мільярди таких атак, що представляє зростання на сімдесят вісім відсотків порівняно з попереднім роком. Середній коефіцієнт успішності становить від нуля цілих одна десята до двох відсотків, що при таких обсягах призводить до мільйонів успішних компрометацій облікових записів. Джерелами облікових даних для credential stuffing служать масивні бази даних, створені з численних витоків інформації. Колекція під назвою “Collection #1-5”, опублікована у дві тисячі дев'ятнадцятому році, містить понад два цілих дві десятих мільярда унікальних комбінацій електронної пошти та паролів. База RockYou2021 включає вісім цілих чотири десятих мільярда паролів, тоді як Compilation of Many Breaches охоплює три цілих дві десятих мільярда унікальних пар облікових даних. Ці масиви даних вільно доступні в мережі та активно

використовуються зловмисниками. Механізм атаки credential stuffing включає використання спеціалізованих інструментів автоматизації, таких як Sentry MBA, SNIPR або OpenBullet, які здатні обробляти мільйони спроб входу. Для обходу захисних механізмів зловмисники застосовують розподілені атаки через ботнети, що включають тисячі різних IP-адрес, використовують повільний темп запитів з кожної адреси, застосовують резидентські проксі-сервери для створення легітимного вигляду трафіку та постійно ротують User-Agent strings для імітації різних браузерів та пристроїв.

Password spraying представляє альтернативний підхід до атак грубої сили, де зловмисник використовує невелику кількість найпоширеніших паролів проти великої кількості облікових записів користувачів. Цей метод дозволяє уникнути механізмів блокування облікових записів, оскільки жоден окремий акаунт не зазнає множинних невдалих спроб входу. Стратегія атаки включає фазу розвідки для збору списку валідних імен користувачів через enumeration електронної пошти, корпоративні каталоги або соціальні мережі типу LinkedIn. Зловмисники обирають паролі з високою ймовірністю успіху, включаючи загальні корпоративні варіанти на зразок “Password123!” або “Welcome1!”, сезонні комбінації типу “Summer2024!” або “Winter2023!”, та варіації з назвою компанії. Дослідження Microsoft демонструє, що password spraying атаки досягають коефіцієнта успішності близько одного відсотка при використанні десяти найпоширеніших паролів, що означає потенційну компрометацію приблизно тисячі облікових записів в організації зі ста тисячами користувачів.

SQL Ін’єкції залишається критичною вразливістю, що дозволяє повністю обійти механізми автентифікації через маніпулювання SQL запитами. За даними OWASP, ін'єкційні атаки посідають третє місце в “Top 10 2021”. Класичний сценарій обходу автентифікації виникає, коли вразливий код безпосередньо вставляє користувацький ввід у SQL запит без належної санітизації. Атакуючий може ввести спеціально сформований рядок у поле імені користувача, що змінює логіку SQL запиту, дозволяючи вхід без знання пароля. Критична вразливість CVE-2023-28432 в MinIO object storage демонструє

актуальність SQL ін’єкції для сучасних систем. Вразливість дозволяла обхід автентифікації через маніпулювання LDAP username inputs, надаючи зловмисникам повний віддалений доступ до сховища об'єктів без будь-якої автентифікації. Вразливість отримала бал CVSS 8.8 та могла призвести до масштабного витоку конфіденційних даних. NoSQL ін'єкції представляють аналогічну загрозу для систем, що використовують NoSQL бази даних, такі як MongoDB або CouchDB. Зловмисники можуть маніпулювати запитами, надсилаючи спеціально сформовані JSON об'єкти замість простих рядків. Наприклад, передача об'єкта з оператором not equal null замість пароля може обійти автентифікацію, оскільки умова стає завжди істинною для існуючих користувачів.

Захисні механізми проти атак на основі облікових даних включають багаторівневий підхід. Rate limiting та account lockout механізми обмежують кількість спроб автентифікації, застосовуючи прогресивні затримки після кожної невдалої спроби або тимчасове блокування облікового запису після визначеної кількості невдач. Важливо балансувати безпеку з доступністю, оскільки надто агресивні політики блокування можуть призвести до атак типу denial of service проти легітимних користувачів.

CAPTCHA механізми активуються після декількох невдалих спроб, вимагаючи від користувача підтвердження людської природи запиту. Сучасні реалізації типу reCAPTCHA v3 працюють невидимо для користувача, аналізуючи поведінкові патерни без необхідності розв'язування візуальних головоломок. Перевірка паролів проти баз відомих порушень через сервіси типу HaveIBeenPwned API дозволяє виявляти та забороняти використання скомпрометованих паролів ще на етапі створення або зміни облікового запису.

Параметризовані запити та використання ORM фреймворків з вбудованим захистом усувають можливість SQL ін'єкцій на архітектурному рівні. Строга валідація всіх користувацьких вводів з використанням підходу білого списку, а не чорного, забезпечує додатковий рівень захисту. Моніторинг та виявлення аномалій дозволяє ідентифікувати незвичні патерни входу, такі як спроби з нових

географічних локацій, пристроїв або в нетипові години, генеруючи сповіщення для команди безпеки та потенційно вимагаючи додаткової верифікації від користувача [52].

2.2.2 Атаки на основі сесії

Атаки на сесії експлуатують вразливості в механізмах управління сесіями користувачів після успішної автентифікації, дозволяючи зловмиснику діяти від імені легітимного користувача без необхідності знання його облікових даних. Компрометація сесії часто є більш привабливою для зловмисників, оскільки обходить багато захисних механізмів, зосереджених на процесі автентифікації. Session hijacking представляє атаку перехоплення або передбачення валідного ідентифікатора сесії для отримання несанкціонованого доступу. Методи перехоплення варіюються від пасивного прослуховування мережевого трафіку до активної експлуатації вразливостей веб-застосунків. При відсутності шифрування транспортного рівня сесійні cookies передаються у відкритому вигляді та можуть бути перехоплені через прослуховування пакетів у незахищених WiFi мережах громадських місць.

Cross-Site Scripting вразливості дозволяють виконати зловмисний JavaScript код в контексті цільового веб-сайту, отримуючи доступ до сесійних cookies користувача. Якщо cookie не має атрибуту HttpOnly, JavaScript може безпосередньо отримати його вміст через властивість document.cookie та передати зловмиснику. Застарілий підхід передачі ідентифікаторів сесії через URL параметри створює множинні вектори витоку, включаючи збереження в історії браузера, передачу через Referer header при переходах на зовнішні сайти та логування на серверах. Передбачувані ідентифікатори сесій виникають при використанні генераторів з недостатньою ентропією або передбачуваними патернами. Послідовні ідентифікатори, базовані на timestamp значення або слабкі псевдовипадкові генератори дозволяють зловмисникам передбачати валідні

ідентифікатори сесій інших користувачів. Криптографічно безпечні генератори випадкових чисел є абсолютною вимогою для створення ідентифікаторів сесій. Критична вразливість CVE-2021-21972 у VMware vCenter Server демонструє серйозність загроз, пов'язаних з сесіями. Вразливість дозволяла неавтентифікованим зловмисникам виконувати довільні команди через експлуатацію Server-Side Request Forgery для перехоплення валідних токенів сесії. Вразливість отримала максимальний бал CVSS 9.8 та була масово експлуатована протягом днів після публічного розкриття, призводячи до компрометації критичної інфраструктури віртуалізації численних організацій.

Session fixation представляє атаку, при якій зловмисник встановлює ідентифікатор сесії жертви на заздалегідь відомий зловмиснику value, а потім очікує автентифікації жертви з цим ідентифікатором. Механізм атаки починається з отримання зловмисником валідного ідентифікатора сесії від цільового веб-сайту. Зловмисник потім встановлює цей ідентифікатор для жертви через різні методи, включаючи надсилання посилання з ідентифікатором у URL параметрі через фішинг, експлуатацію вразливостей для встановлення cookies між доменами або фізичний доступ до спільно використовуваних комп'ютерів. Коли жертва автентифікується, вразлива система не регенерує ідентифікатор сесії, залишаючи його незмінним. Зловмисник може тепер використовувати цей тепер автентифікований ідентифікатор сесії для доступу до облікового запису жертви без знання пароля. Варіації атаки включають експлуатацію субдоменів, коли cookie встановлений для всього домену, CRLF ін'єкції для вставки заголовків Set-Cookie, та session adoption, коли система приймає ідентифікатори сесій, надані клієнтом, замість генерації власних.

Захист від атак на сесії вимагає комплексного підходу. Регенерація ідентифікатора сесії після успішної автентифікації є критично важливою практикою, що усуває можливість session fixation атак. Система повинна генерувати повністю новий ідентифікатор сесії при переході користувача зі стану неавтентифікованого до автентифікованого, інвалідуючи попередній ідентифікатор. Сесійні ідентифікатори повинні прийматися виключно з cookies,

встановлених сервером, ігноруючи будь-які спроби надати ідентифікатори через URL або POST параметри. Строга валідація сесій включає перевірку додаткових параметрів, таких як IP адреса та User-Agent, для виявлення можливого перехоплення. Хоча ці параметри можуть змінюватися за легітимних обставин, раптові зміни можуть сигналізувати про компрометацію. Механізми timeout забезпечують автоматичне завершення сесій після періоду неактивності або абсолютного максимального часу життя, обмежуючи вікно можливої експлуатації. Безпечні атрибути cookies включають HttpOnly для запобігання доступу через JavaScript, Secure для забезпечення передачі тільки через HTTPS з'єднання, та SameSite для захисту від міжсайтових атак. Комбінація цих атрибутів створює багаторівневий захист проти різних векторів атак на сесії.

2.2.3 Обхід багатофакторної автентифікації

Багатофакторна автентифікація розглядається як золотий стандарт захисту облікових записів, значно підвищуючи складність несанкціонованого доступу. За даними Microsoft, дев'яносто дев'ять цілих дев'ять десятих відсотка скомпрометованих облікових записів не мали увімкненої багатофакторної автентифікації. Проте навіть увімкнена багатофакторна автентифікація не гарантує абсолютного захисту, оскільки зловмисники розробили складні методи її обходу.

MFA fatigue attacks експлуатують людський фактор через social engineering та психологічний тиск. Механізм атаки передбачає, що зловмисник, маючи валідні облікові дані користувача, ініціює множинні спроби входу, кожна з яких генерує push notification на мобільний пристрій жертви. Користувач отримує десятки або навіть сотні сповіщень протягом короткого періоду часу, що створює значний дискомфорт та бажання припинити потік сповіщень. Реальний випадок компрометації Uber у вересні дві тисячі двадцять другого року демонструє ефективність цього підходу. Зловмисник придбав викрадені облікові дані

підрядника Uber на темному ринку та ініціював атаку втоми багатофакторної автентифікації, відправляючи безперервні push notifications. Одночасно зловмисник надіслав повідомлення через WhatsApp, видаючи себе за представника IT підтримки Uber, та переконав жертву підтвердити запит автентифікації. Отримавши доступ, зловмисник скомпрометував внутрішні системи компанії, включаючи репозиторії вихідного коду, конфіденційні дані в AWS та Google Cloud, систему управління привілейованим доступом, та публічно оголосив про порушення в корпоративному Slack. Аналогічна атака на Cisco у серпні дві тисячі двадцять другого року демонструє поширеність цієї техніки. Група ransomware Yanluowang отримала початковий доступ через скомпрометовані облікові дані особистого Gmail акаунта співробітника та застосувала атаку втоми багатофакторної автентифікації. Після безперервного потоку push notifications співробітник підтвердив запит, надавши зловмисникам доступ до внутрішніх мереж через VPN. Атакуючі здійснили lateral movement, скомпрометували середовище Citrix та Active Directory, та ексфільтрували приблизно два цілих сімдесят п'ять сотих гігабайта даних.

Витік секретів TOTP представляє іншу категорію вразливостей багатофакторної автентифікації. Механізм Time-based One-Time Password базується на спільному секреті між сервером та клієнтом, компрометація якого дозволяє зловмиснику генерувати валідні одноразові коди. Під час початкового налаштування TOTP користувачам показується QR код з закодованим секретом. Користувачі часто роблять скріншоти цих кодів та зберігають їх у незахищених хмарних сховищах, надсилають іншим користувачам для допомоги, або втрачають контроль над ними через компрометацію резервних копій пристроїв.

Коди відновлення, надані при налаштуванні багатофакторної автентифікації, часто зберігаються небезпечно у текстових файлах на робочому столі, незашифрованих нотатках у хмарних сервісах, або залишаються доступними в електронній пошті після компрометації облікового запису. Деякі додатки автентифікації до недавнього часу не шифрували резервні копії, що означало, що компрометація основного облікового запису, наприклад Google

Account, автоматично надавала доступ до всіх секретів TOTP користувача. Складні фішингові сайти можуть здійснювати real-time relay облікових даних та одноразових паролів до легітимного сайту. Коли жертва вводить свої облікові дані на фішинговому сайті, зловмисний сервер негайно пересилає їх на справжній сайт. Коли справжній сайт запитує одноразовий пароль, фішинговий сайт відображає цей запит жертві, отримує код та пересилає його для завершення автентифікації. Інструменти типу Evilginx2 та Modlishka автоматизують цей процес, дозволяючи обхід навіть добре імплементованої багатофакторної автентифікації.

SIM swapping представляє атаку, при якій зловмисник переконує оператора мобільного зв'язку передати номер телефону жертви на SIM-карту, контрольовану зловмисником. Механізм починається з фази розвідки, де зловмисник збирає інформацію про жертву через соціальні мережі та витоки даних, визначає оператора мобільного зв'язку та отримує достатньо персональної інформації для проходження верифікації. Зловмисник може зателефонувати в службу підтримки оператора, стверджуючи, що втратив телефон та потребує перенесення номера на нову SIM-карту, надаючи викрадену персональну інформацію для верифікації. Альтернативно, зловмисник може підкупити співробітника оператора для виконання перенесення без належної верифікації, або відвідати офіс оператора з підробленим посвідченням особи. Після успішного перенесення зловмисник отримує всі SMS повідомлення та дзвінки, призначені для номера жертви, включаючи коди скидання паролів, коди двофакторної автентифікації, верифікаційні коди банківських операцій та повідомлення відновлення облікових записів. Високопрофільні випадки SIM swapping включають компрометацію облікового запису Twitter генерального директора Jack Dorsey у дві тисячі дев'ятнадцятому році, коли расистські твіти були опубліковані від його імені, демонструючи вразливість навіть для високопоставлених осіб. Крадіжки криптовалюти через SIM swapping оцінюються в понад сто мільйонів доларів, з визначним випадком крадіжки

двадцяти чотирьох мільйонів доларів у Michael Terpin у дві тисячі вісімнадцятому році.

Протидія обходу багатофакторної автентифікації вимагає багаторівневого підходу. Механізми проти втоми включають number matching, де замість простого підтвердження або відхилення користувач повинен ввести номер, показаний на екрані входу, в push notification, що запобігає випадковому підтвердженню через втому. Перевірки швидкості обмежують кількість push notifications за період часу, автоматично блокуючи після визначеної кількості запитів за короткий проміжок та сповіщаючи команду безпеки при підозрілих патернах.

Стійка до фішингу багатофакторна автентифікація через WebAuthn та FIDO2 забезпечує криптографічну верифікацію, прив'язану до конкретного домену, що робить фішинг технічно неможливим, оскільки приватний ключ ніколи не залишає пристрій, а підпис не може бути повторно використаний на фішинговому сайті. Passkeys представляють сучасну імплементацію WebAuthn з покращеним користувацьким досвідом та вбудованим захистом від фішингу. Захист від SIM swapping включає встановлення port freeze або port-out PIN з оператором мобільного зв'язку, додаткові вимоги верифікації для будь-яких змін SIM-карти, та сповіщення при будь-яких модифікаціях облікового запису. Критичні облікові записи повинні уникати SMS-базованої двофакторної автентифікації, використовуючи натомість додатки автентифікації або апаратні токени, залишаючи SMS тільки як резервний варіант, ніколи як основний метод.

2.2.4 Вразливості OAuth

OAuth 2.0, незважаючи на статус індустріального стандарту та широке використання, має численні підводні камені імплементації, що призводять до серйозних вразливостей безпеки. За даними OWASP API Security Top 10, порушена автентифікація включає неправильну конфігурацію OAuth як основну проблему.

Маніпулювання redirect URI представляє критичну вразливість, оскільки валідація redirect URI є ключовим механізмом безпеки в OAuth flows. Неправильна валідація дозволяє атаки перехоплення authorization code. Вразливі патерни включають використання prefix matching замість точної відповідності, коли Authorization Server приймає будь-який URL, що починається з зареєстрованого redirect URI, дозволяючи зловмисникам додавати параметри або модифікувати шлях для перенаправлення authorization code на контрольований ними endpoint.

Наявність open redirect вразливості на клієнтському застосунку створює додатковий вектор атаки. Навіть якщо redirect URI правильно зареєстрований, клієнтський застосунок може мати вразливість, що дозволяє перенаправлення на довільні URL. Authorization code спочатку доставляється на легітимний callback endpoint, проте клієнт потім перенаправляє на зловмисний сайт з кодом у URL, дозволяючи його перехоплення. Прийняття wildcard субдоменів створює можливість для атак, якщо зловмисник може контролювати будь-який субдомен зареєстрованого домену. Сценарій експлуатації починається з того, що зловмисник створює зловмисний authorization URL з redirect_uri, що вказує на контрольований ним сайт. Жертва переходить за посиланням через фішинг або social engineering, автентифікується та надає дозвіл. Authorization Server перенаправляє на зловмисний сайт з authorization code у параметрах URL. Зловмисник перехоплює код та обмінює його на access token, потенційно використовуючи викрадений або не обов'язковий client secret, отримуючи можливість видавати себе за жертву.

Вразливість CVE-2020-7695 у платформі OAuth.io демонструє реальні наслідки неправильної валідації redirect URI. Вразливість дозволяла обхід валідації через неналежну перевірку, призводячи до компрометації облікових записів користувачів на будь-якому сервісі, що використовує OAuth.io для автентифікації.

Перехоплення authorization code може відбуватися через різні вектори атак. На мобільних платформах множинні застосунки можуть зареєструвати однакову

custom URL scheme, що означає, що коли операційна система отримує OAuth redirect, вона може доставити authorization code до зловмисного застосунку замість легітимного. Android App Links та iOS Universal Links частково вирішують цю проблему через верифікацію Digital Asset Links та Apple App Site Association файлів відповідно, проте вимагають правильної конфігурації.

Витік через історію браузера виникає, коли authorization code передається в URL, де він зберігається в історії браузера. Компрометація браузера автоматично призводить до exposure authorization codes. Витік через Referer header відбувається, коли callback сторінка містить посилання на зовнішні сайти, передаючи authorization code через HTTP Referer header при завантаженні зовнішніх ресурсів.

Proof Key for Code Exchange був спеціально розроблений для захисту від перехоплення authorization code. Механізм передбачає, що клієнт генерує випадковий рядок code_verifier та обчислює code_challenge як SHA-256 хеш цього значення. Authorization request включає code_challenge, який зберігається Authorization Server. Навіть якщо authorization code перехоплений, зловмисник не має оригінального code_verifier. При обміні коду на токен вимагається надання code_verifier, який сервер верифікує, обчислюючи його хеш та порівнюючи зі збереженим code_challenge. Без правильного verifier обмін токенів зазнає невдачі. RFC 9700 від дві тисячі двадцять п'ятого року робить PKCE обов'язковим для всіх типів OAuth клієнтів.

CSRF атаки без state parameter експлуатують відсутність або неправильну валідацію механізму захисту від міжсайтових підроблень запитів. Сценарій атаки починається з того, що зловмисник автентифікується до OAuth Provider зі своїм власним обліковим записом та захоплює authorization callback URL з authorization code свого облікового запису. Зловмисник створює зловмисну веб-сторінку, що автоматично відправляє запит до callback endpoint з перехопленим кодом. Коли жертва відвідує цю сторінку, її браузер відправляє запит з cookies сесії жертви.

Клієнтський застосунок обмінює код зловмисника на access token та пов'язує обліковий запис зловмисника OAuth з сесією жертви.

Реальний вплив включає атаки прив'язування облікових записів, де жертва входить у застосунок з email та паролем, зловмисник примусово прив'язує свій обліковий запис Google до облікового запису жертви, після чого зловмисник може входити в обліковий запис жертви через функцію "Увійти через Google". Ексфільтрація даних відбувається, коли застосунок автоматично імпортує дані з прив'язаного OAuth облікового запису, і обліковий запис жертви імпортує зловмисні дані зловмисника, або зловмисник експортує дані жертви через прив'язаний обліковий запис. Правильне використання state parameter передбачає генерацію криптографічно випадкового значення з достатньою ентропією, зберігання цього значення в сесії користувача, включення в authorization URL, та валідацію при обробці callback запиту шляхом порівняння отриманого значення зі збереженим. Невідповідність сигналізує про CSRF атаку та повинна призводити до відхилення запиту.

Додаткові вразливості OAuth включають tampering scope, де зловмисники намагаються модифікувати або розширити запитувані права доступу, що вимагає серверної перевірки та ніколи не довіряє scope, наданим клієнтом. Exposure client secret виникає при жорсткому кодуванні секретів у мобільних застосунках або single-page applications, що вимагає використання PKCE для публічних клієнтів та належного управління секретами для конфіденційних клієнтів. Витік токенів через логи, URLs або клієнтське сховище вимагає зберігання токенів виключно в пам'яті або захищеному сховищі, короткого часу життя та належних практик логування.

CVE-2020-26945 у програмному забезпеченні форуму MyBB демонструвала неправильну валідацію OAuth state, що дозволяло CSRF атаки для перехоплення облікових записів.

CVE-2023-45826 у SharePoint Online виявила витік OAuth токенів через неправильно налаштовану валідацію redirect URI, дозволяючи несанкціонований доступ до SharePoint ресурсів.

CVE-2019-11479 у GitLab показала обхід валідації redirect URI OAuth застосунків через недостатню перевірку, що призводило до крадіжки authorization codes.

Глибокоешелонований захист для OAuth включає строгу валідацію redirect URI через точну відповідність зареєстрованим URI без wildcards для продакшн середовищ, виключно HTTPS та підхід білого списку. Обов'язкове PKCE для всіх клієнтів з методом challenge S256 та відхиленням flows без PKCE. State parameter повинен бути криптографічно випадковим з принаймні двома сотнями п'ятдесятьма шістьма бітами ентропії, прив'язаним до сесії, одноразовим та з обмеженням терміну дії в 5-10 хвилин. Авторизаційні коди повинні мати короткий термін життя тридцять-шістдесят секунд, примусове одноразове використання та негайну інвалідацію після обміну. Токени доступу повинні жити п'ять-п'ятнадцять хвилин, оновлені токени повинні ротуватися, зберігання має бути безпечним через HttpOnly cookies або Keychain/Keystore, та токени ніколи не повинні передаватися в URL. Ліміт частоти повинен застосовуватися до кінцевої точки токену, уінцевої точки авторизації та включати квоти відповідно до клієнту. Засобами моніторингу повині відстежуватись невдалі обміни токенів, незвичні патерни авторизації, географічні аномалії та спроби ескалації. Security headers повинні включати X-Frame-Options DENY для запобігання clickjacking, Content-Security-Policy та Referrer-Policy no-referrer.

2.3 Вразливості авторизації

Вразливості авторизації представляють критичну категорію безпекових проблем, що виникають після успішної автентифікації користувача та стосуються контролю доступу до ресурсів системи. На відміну від вразливостей автентифікації, що фокусуються на верифікації ідентичності користувача, вразливості авторізації експлуатують слабкості в механізмах визначення та примусового застосування прав доступу. За даними OWASP Top 10 2021,

порушення контролю доступу (A01:2021 Broken Access Control) посідає перше місце серед найпоширеніших та найнебезпечніших вразливостей веб-застосунків, зустрічаючись у понад тридцяти чотирьох відсотках протестованих застосунків.

Фундаментальна проблема вразливостей авторизації полягає в тому, що навіть за наявності надійних механізмів автентифікації система може надавати користувачам доступ до ресурсів, на які вони не повинні мати прав. Це може призводити до несанкціонованого перегляду конфіденційних даних, модифікації критичної інформації, виконання привілейованих операцій або повної компрометації системи. Складність сучасних застосунків з множиною ролей користувачів, динамічними правами доступу та розподіленими архітектурами значно ускладнює правильну імплементацію та підтримку механізмів авторизації.

Економічний вплив вразливостей авторизації є значним. Дослідження IBM Cost of a Data Breach Report 2024 вказує, що середня вартість порушення безпеки даних становить чотири цілих чотири десятих мільйона доларів США глобально та десять цілих двадцять дві сотих мільйона доларів у Сполучених Штатах. Значна частина цих порушень пов'язана з експлуатацією вразливостей контролю доступу, що дозволяють зловмисникам отримувати доступ до даних поза межами їхніх легітимних повноважень.

2.3.1 Порушення контролю доступу

Порушення контролю доступу виникають, коли застосунок не належним чином перевіряє та примусово застосовує обмеження на те, які дії або дані користувач може отримати або модифікувати. Ці вразливості часто є результатом помилкового припущення розробників, що користувачі будуть взаємодіяти з застосунком тільки через передбачений користувацький інтерфейс, ігноруючи

можливість прямих запитів до backend API або маніпулювання параметрами запитів.

Insecure Direct Object References представляють одну з найпоширеніших форм порушення контролю доступу. IDOR виникає, коли застосунок надає прямий доступ до об'єктів на основі користувацького вводу без належної перевірки авторизації. Типово це проявляється у використанні передбачуваних ідентифікаторів у URL або параметрах API запитів, де зміна значення ідентифікатора дозволяє отримати доступ до ресурсів інших користувачів. Класичний приклад IDOR вразливості можна спостерігати у системі управління документами, де endpoint для отримання документа має структуру GET /api/documents/12345. Якщо застосунок не перевіряє, чи має поточний автентифікований користувач право доступу до документа з ідентифікатором 12345, зловмисник може просто змінювати числове значення ідентифікатора, систематично перебираючи значення 12346, 12347, 12348 та отримуючи доступ до документів, що належать іншим користувачам або організаціям. Реальний випадок IDOR вразливості високого профілю трапився у системі бронювання авіакомпанії, де дослідники виявили, що booking reference codes були послідовними та передбачуваними. Змінюючи код бронювання в URL, зловмисник міг отримати доступ до повної інформації про бронювання інших пасажирів, включаючи персональні дані, деталі рейсу та платіжну інформацію. Вразливість існувала через відсутність серверної перевірки того, чи належить запитуване бронювання поточному автентифікованому користувачу.

Складніші варіації IDOR включають використання GUID або UUID замість послідовних чисел для ідентифікаторів ресурсів. Незважаючи на те, що GUID є непередбачуваними, їх використання не усуває IDOR вразливість, якщо відсутня належна авторизаційна перевірка. Зловмисник може отримати валідний GUID через витік інформації, social engineering або перехоплення легітимного запиту, після чого використовувати його для несанкціонованого доступу. Захист від IDOR вимагає не непередбачуваності ідентифікаторів, а обов'язкової серверної перевірки авторизації при кожному запиті до ресурсу.

Варіації IDOR можуть проявлятися не тільки в отриманні даних, але й у модифікації або видаленні ресурсів. Endpoint PUT “/api/users/789/profile” для оновлення профілю користувача може дозволяти зловмиснику модифікувати профілі інших користувачів, змінюючи ідентифікатор у шляху запиту. Запит DELETE “/api/orders/456” може дозволяти видалення замовлень інших користувачів. Запит POST “/api/accounts/123/transactions” може дозволяти ініціювання фінансових транзакцій з чужих облікових записів.

Missing function-level access control представляє вразливість, коли застосунок не перевіряє, чи має користувач право виконувати конкретну функцію або операцію. Цей тип вразливості часто виникає, коли розробники покладаються на приховування елементів користувацького інтерфейсу як на механізм контролю доступу, припускаючи, що якщо кнопка або посилання не відображаються для користувача, він не зможе викликати відповідну функцію. Типовий сценарій включає адміністративні функції, доступ до яких контролюється тільки на рівні інтерфейсу. Звичайний користувач не бачить посилання на адміністративну панель у навігаційному меню, проте якщо він безпосередньо звертається до URL “/admin/dashboard” або викликає кінцеву точку API запитом POST “/api/admin/users/delete”, система може виконати запит без перевірки ролі користувача. Зловмисник може виявити ці приховані кінцеві точки через аналіз JavaScript коду клієнтського застосунку, перехоплення мережевого трафіку, або використання автоматизованих сканерів для виявлення неавторизованих шляхів. Складніші випадки missing function-level access control включають ситуації, коли перевірка авторизації існує, але є неповною або непослідовною. Застосунок може перевіряти права доступу для GET запитів, але пропускати перевірку для POST, PUT або DELETE запитів до того самого ресурсу. Альтернативно, перевірка може здійснюватися на одному рівні архітектури, наприклад у веб-контролері, але бути відсутньою в безпосередньому доступі до API або мікросервісу. Проблема ускладнюється в мікросервісних архітектурах, де авторизаційна логіка може бути розподілена між множиною сервісів. Якщо один мікросервіс покладається на інший для перевірки

авторизації, проте ця перевірка може бути обійдена через прямий доступ до внутрішнього API, виникає критична вразливість. API шлюз може здійснювати авторизаційні перевірки, проте якщо мікросервіси доступні напряму в межах внутрішньої мережі без додаткової перевірки, компрометація будь-якого компонента всередині периметру дозволяє обійти всі контролі доступу [53].

Реальний приклад критичної вразливості контролю доступу демонструє CVE-2023-20198, виявлена у Cisco IOS XE Web UI. Вразливість дозволяла віддаленому неавтентифікованому зловмиснику створювати обліковий запис з рівнем привілеїв fifteen, що є максимальним рівнем доступу в Cisco IOS систем, та отримувати повний контроль над пристроєм. Вразливість виникла через відсутність належної автентифікації та авторизації в певних endpoints веб-інтерфейсу, дозволяючи зловмисникам обходити всі механізми контролю доступу.

CVE-2023-20198 отримала максимальний бал CVSS 10 та була активно експлуатована зловмисниками in-the-wild, призводячи до компрометації тисяч пристроїв Cisco по всьому світу. Експлуатація вразливості дозволяла зловмисникам не тільки створювати привілейовані облікові записи, але й встановлювати backdoors, модифікувати конфігурації маршрутизаторів та потенційно перехоплювати мережевий трафік. Інцидент підкреслює критичну важливість належної імплементації контролю доступу навіть у enterprise-grade обладнанні від провідних виробників [54].

Захист від порушень контролю доступу вимагає багаторівневого підходу. Фундаментальним принципом є примусове застосування авторизаційних перевірок на серверній стороні для кожного запиту до ресурсу або функції, незалежно від того, чи передбачається доступ через користувацький інтерфейс. Клієнтські обмеження повинні розглядатися виключно як покращення користувацького досвіду, а не як механізм безпеки. Імплементація контролю доступу повинна слідувати принципу “deny by default”, де доступ явно надається тільки після верифікації належних прав, замість підходу “allow by default” з виключеннями. Кожен endpoint повинен починатися з перевірки, чи

автентифікований користувач, чи має він належну роль або повноваження для виконання запитуваної операції, та чи має він право доступу до конкретного ресурсу, ідентифікованого в запиті. Використання непрямих посилань на об'єкти замість прямих ідентифікаторів може знизити ризик IDOR, проте не усуває необхідність авторизаційних перевірок. Замість використання ідентифікаторів з бази даних безпосередньо в URL, застосунок може використовувати співставлення між специфічними для користувача токенами та внутрішніми ідентифікаторами. Проте навіть з цим підходом серверна перевірка залишається обов'язковою. Централізація авторизаційної логіки через використання фреймворків контролю доступу або middleware компонентів забезпечує консистентність та знижує ймовірність пропущених перевірок. Декларативні моделі авторизації, де права доступу визначаються в конфігурації або анотаціях, а не розкидані по коду застосунку, полегшують аудит та підтримку. Автоматизоване тестування авторизаційних контролів через інтеграційні тести та безпекові тести допомагає виявляти регресії при розробці нових функцій [55].

2.3.2 Ескалація привілеїв

Ескалація привілеїв представляє категорію атак, при яких зловмисник отримує вищий рівень доступу або permissions, ніж було йому надано системою. Ці атаки експлуатують вразливості в механізмах управління привілеями та часто є наслідком неналежної валідації користувацьких вводів, логічних помилок у коді авторизації або недостатньої ізоляції між рівнями привілеїв.

Розрізняють дві основні категорії ескалації привілеїв: горизонтальну та вертикальну. Горизонтальна ескалація привілеїв виникає, коли зловмисник отримує доступ до ресурсів або функцій іншого користувача з тим самим рівнем привілеїв. Користувач А, автентифікований як звичайний користувач, отримує доступ до даних користувача Б, який також є звичайним користувачем. Хоча

рівень привілеїв залишається незмінним, порушується принцип ізоляції даних між користувачами одного рівня.

Класичний приклад горизонтальної ескалації можна спостерігати у банківському застосунку, де користувач може переглядати свої транзакції через GET запит до кінцевої точки “/api/accounts/USER_A_ID/transactions”. Якщо застосунок не перевіряє, що поточний автентифікований користувач дійсно є власником запитуваного облікового запису, зловмисник може замінити “USER_A_ID” на “USER_B_ID” та отримати доступ до фінансових транзакцій іншого користувача. Це є варіацією IDOR вразливості, специфічно класифікованою як горизонтальна ескалація через доступ до еквівалентного за рівнем, але не належного зловмиснику ресурсу.

Вертикальна ескалація привілеїв є більш серйозною категорією, коли зловмисник з нижчим рівнем доступу отримує привілеї вищого рівня, часто адміністративні права. Звичайний користувач здобуває можливості адміністратора, дозволяючи виконувати критичні операції типу створення або видалення облікових записів, модифікації конфігурацій системи, доступу до всіх даних або виконання довільного коду на сервері. Механізми вертикальної ескалації часто включають експлуатацію слабкостей у перевірці ролей користувачів. Застосунок може зберігати роль користувача в JWT token або session cookie, проте якщо ця інформація може бути модифікована клієнтом без належної верифікації серверної сторони, зловмисник може змінити своє значення ролі з user на admin. JWT токени з алгоритмом підпису none або слабким секретним ключем особливо вразливі до таких маніпуляцій.

Parameter tampering представляє техніку, що часто використовується для ескалації привілеїв. Зловмисник перехоплює легітимний запит та модифікує параметри для зміни поведінки застосунку у спосіб, що надає додаткові привілеї. Це може включати зміну прихованих полів форми, модифікацію cookies, зміну HTTP заголовків або маніпулювання параметрами API запитів. Приклад parameter tampering для вертикальної ескалації може виглядати наступним чином. Легітимний запит для оновлення профілю користувача може мати форму

запиту POST “/api/users/update” з тілом в форматі JSON, що містить поля username, email та profile_picture. Якщо та сама кодова база обробляє оновлення адміністративних полів без належної сегрегації, зловмисник може додати поле role зі значенням admin або is_admin зі значенням “true” до запиту. Якщо серверна логіка не фільтрує або не валідує, які поля може модифікувати користувач на основі його поточної ролі, система може прийняти модифікацію та надати адміністративні привілеї. Складніші сценарії parameter tampering включають експлуатацію mass assignment вразливостей у веб-фреймворках. Багато сучасних фреймворків дозволяють автоматичне прив'язування параметрів HTTP запиту до полів моделі даних. Якщо розробник не явно визначає, які поля можуть бути масово присвоєні, зловмисник може включити додаткові параметри у запит, що будуть автоматично встановлені в моделі, потенційно модифікуючи критичні поля типу ролі, повноважень або статусу облікового запису.

Реальний приклад критичної вразливості ескалації привілеїв демонструє CVE-2023-27532 у Veeam Backup and Replication. Вразливість дозволяла локальному користувачу з низькими привілеями отримувати облікові дані з кофігураційної БД Veeam, включаючи облікові дані для компонентів інфраструктури резервного копіювання.

CVE-2023-27532 отримала бал CVSS 7.5 та була особливо небезпечною в контексті атак з вимаганням, оскільки компрометація системи резервного копіювання дозволяє зловмисникам видаляти або шифрувати ресруси перед розгортанням вірусів-виагачів на основних системах, усуваючи можливість відновлення без оплати викупу. Вразливість виникла через неналежний контроль доступу до конфіденційних даних конфігурації, дозволяючи локальним користувачам читати інформацію, яка повинна бути доступна тільки адміністраторам системи.

Захист від ескалації привілеїв вимагає ретельної імплементації принципу найменших привілеїв на всіх рівнях системи. Користувачі повинні отримувати мінімальний набір прав, необхідний для виконання їхніх легітимних функцій, без надання додаткових можливостей на випадок майбутніх потреб. Ролі повинні

бути чітко визначені з явно специфікованими permissions для кожної ролі, уникаючи широких або нечітко визначених категорій доступу. Серверна валідація всіх операцій повинна включати не тільки перевірку автентифікації, але й детальну перевірку авторизації на рівні конкретної операції та конкретного ресурсу. Недостатньо перевірити, що користувач має роль admin, необхідно також верифікувати, що конкретна запитувана операція дозволена для цієї ролі в даному контексті. Розділення обов'язків між різними ролями запобігає ситуаціям, коли один користувач або роль має надмірну кількість критичних привілеїв. Імплементація строгої валідації вводів на серверній стороні є критичною для запобігання parameter tampering. Застосунок повинен явно визначати, які параметри допустимі для кожного endpoint та ролі користувача, відхиляючи запити з неочікуваними або несанкціонованими параметрами. Використання Data Transfer Objects або подібних патернів дозволяє контролювати, які поля можуть бути встановлені через користувацький ввід, запобігаючи mass assignment вразливостям.

Регулярний аудит та тестування механізмів авторізації через тестування на проникнення та авоматичне сканування вразливостей допомагає виявляти вразливості ескалації привілеїв до їх експлуатації зловмисниками. Статичний аналіз коду може ідентифікувати кінцеві точки або функції, що пропускають авторизаційні перевірки. Динамічне тестування з різними рівнями користувацьких привілеїв може виявити ситуації, коли нижчі ролі можуть виконувати операції, призначені для вищих рівнів доступу.

Логування та моніторинг спроб ескалації привілеїв забезпечує механізм виявлення та реагування на активні атаки. Система повинна реєструвати всі випадки, коли користувач намагається виконати операцію поза межами його авторизованих прав, генеруючи сповіщення для команди безпеки при виявленні підозрілих патернів. Аналіз цих логів може виявити систематичні спроби дослідження вразливостей або успішні експлуатації до настання значних наслідків.

Впровадження багаторівневої архітектури безпеки гарантує, що навіть при компрометації одного рівня захисту інші рівні продовжують забезпечувати обмеження доступу. Комбінація автентифікації, авторизації на рівні ролей, авторизації на рівні ресурсів, валідації вводів та моніторингу створює defense in depth підхід, що значно ускладнює успішну ескалацію привілеїв [56].

2.4 Вразливості JSON Web Tokens

JSON Web Tokens представляють широко використовуваний стандарт для представлення claims між двома сторонами у веб-застосунках та API. Незважаючи на елегантність архітектури та зручність використання, JWT піддаються численним категоріям атак, що експлуатують помилки імплементації, слабкості криптографічних механізмів або недостатню валідацію на стороні споживача токенів. За даними аналізу вразливостей Auth0, понад сорок відсотків застосунків, що використовують JWT, мають принаймні одну критичну помилку імплементації, що може призвести до повної компрометації автентифікації.

Фундаментальна проблема безпеки JWT полягає в тому, що токен є самодостатнім носієм інформації про автентифікацію та авторизацію, який передається та зберігається на стороні клієнта. На відміну від серверних сесій, де критична інформація залишається під контролем сервера, JWT делегує довіру криптографічним механізмам для забезпечення цілісності та автентичності claims. Будь-яка вразливість у перевірці підпису, валідації claims або управлінні криптографічними ключами може призвести до можливості підробки токенів та обходу всієї системи автентифікації.

Структура JWT, що складається з header, payload та signature, створює множинні вектори атак. Header містить метадані про алгоритм підпису та тип токена, проте ці метадані контролюються клієнтом та можуть бути маніпульовані. Payload містить claims у форматі JSON, що є Base64URL закодованим, але не зашифрованим, створюючи ризики розкриття інформації та

можливості модифікації. Signature має забезпечувати цілісність та автентичність, проте залежить від правильної імплементації криптографічних алгоритмів та належного управління ключами [57].

2.4.1 Атаки на підпис

Атаки на підпис JWT експлуатують вразливості в криптографічних механізмах, що забезпечують цілісність та автентичність токена. Ці атаки часто є результатом надмірної довіри до інформації, наданої в header токена, або неправильної імплементації валідації підпису на стороні сервера.

Атака none algorithm представляє одну з найбільш тривіальних, проте надзвичайно ефективних вразливостей JWT. Специфікація JWT (RFC 7519) визначає алгоритм none як валідний варіант для випадків, коли цілісність токена забезпечується іншими механізмами. Проте деякі бібліотеки JWT інтерпретують наявність алгоритму none в header як інструкцію повністю пропустити валідацію підпису, навіть якщо застосунок очікує підписаний токен.

Механізм атаки є тривіальним у виконанні. Зловмисник отримує легітимний JWT токен через звичайний процес автентифікації або перехоплення. Токен декодується для доступу до header та payload компонентів. В header змінюється значення поля alg з оригінального алгоритму (наприклад, HS256 або RS256) на none. Payload модифікується відповідно до цілей атаки, наприклад, зміна claim user_id для імперсонації іншого користувача або зміна role з user на admin для ескалації привілеїв. Signature компонент повністю видаляється або замінюється порожнім рядком. Модифікований токен відправляється серверу, де вразлива бібліотека JWT приймає алгоритм none з header та пропускає валідацію підпису, приймаючи підроблений токен як валідний. Приклад структури вразливого токена демонструє простоту атаки. Оригінальний легітимний токен може мати header з алгоритмом HS256 та типом JWT, payload з ідентифікатором користувача та роллю user, та валідну signature. Атакуючий модифікує header,

змінюючи алгоритм на none, модифікує payload, змінюючи роль на admin, та видаляє signature повністю. Вразливий сервер приймає цей токен без валідації, надаючи зловмиснику адміністративні привілеї.

Вразливість none algorithm була виявлена в численних JWT бібліотеках та фреймворках. Критична вразливість у node-jsonwebtoken до версії 4.2.2 дозволяла обхід валідації підпису через алгоритм none. Аналогічні вразливості були знайдені в бібліотеках для Python, Ruby та інших мовах програмування. Незважаючи на те, що більшість сучасних бібліотек виправили цю вразливість, спадковий код та застарілі залежності продовжують представляти ризик.

Algorithm confusion атаки експлуатують різницю між симетричними та асиметричними алгоритмами підпису JWT. RS256 (RSA Signature з SHA-256) є асиметричним алгоритмом, що використовує пару ключів: приватний ключ для підпису токенів на стороні сервера та публічний ключ для верифікації підпису. HS256 (HMAC з SHA-256) є симетричним алгоритмом, що використовує один секретний ключ як для підпису, так і для верифікації. Вразливість виникає, коли сервер динамічно визначає алгоритм верифікації на основі поля alg в header токена, що контролюється клієнтом, замість примусового використання очікуваного алгоритму. Якщо сервер налаштований для використання RS256 та зберігає публічний ключ для верифікації, зловмисник може змінити header токена, вказавши алгоритм HS256, та підписати токен, використовуючи публічний ключ RS256 як HMAC секрет для HS256. Механізм algorithm confusion атаки включає кілька етапів. Зловмисник отримує публічний ключ сервера, який часто доступний через JWKS endpoint або включений у документацію API. Створюється новий JWT з бажаними модифікованими claims в payload. Header токена встановлюється з алгоритмом HS256 замість оригінального RS256. Токен підписується з використанням публічного ключа RS256 як секретного ключа для HMAC-SHA256 алгоритму. Підроблений токен відправляється серверу, де вразлива імплементація зчитує alg: HS256 з header та використовує публічний ключ RS256 для верифікації HMAC підпису, що призводить до успішної валідації підробленого токена. Атака є можливою через те, що публічний ключ RSA, який

призначений бути публічним та широко доступним, використовується як секретний ключ для симетричного алгоритму. Оскільки зловмисник має доступ до того самого публічного ключа, що й сервер, він може створити валідний HMAC підпис, який сервер прийме, помилково інтерпретуючи алгоритм.

Реальний приклад algorithm confusion продемонстрований у CVE-2015-9235, що вплинула на множину JWT бібліотек. Вразливість дозволяла зловмисникам обходити автентифікацію шляхом зміни алгоритму підпису з RS256 на HS256 та використання публічного ключа як HMAC секрету. Вразливість була особливо поширеною через те, що багато розробників не усвідомлювали фундаментальну різницю між симетричними та асиметричними алгоритмами в контексті JWT.

Більш свіжий приклад представляє CVE-2024-54150, виявлена у OpenMetadata Platform. Вразливість дозволяла обходити автентифікацію через маніпулювання JWT токенами, включаючи algorithm confusion атаки. Зловмисники могли отримувати несанкціонований адміністративний доступ до OpenMetadata instances, компрометуючи метадані та конфігурації data infrastructure організацій. Вразливість отримала бал CVSS 9.4 та вимагала негайного оновлення програмного забезпечення всіх вразливих ресурсів.

Захист від атак на підпис JWT вимагає строгої імплементації валідації на серверній стороні. Абсолютною вимогою є явна специфікація дозволених алгоритмів підпису при конфігурації JWT валідації, категорично відхиляючи алгоритм none незалежно від його присутності в заголовку токена. Застосунок повинен визначати очікуваний алгоритм підпису в конфігурації та ігнорувати будь-які спроби клієнта диктувати алгоритм через заголовок. Для запобігання algorithm confusion атакам імплементація повинна явно перевіряти, що алгоритм, вказаний в header токена, точно відповідає очікуваному алгоритму, налаштованому для конкретного провайдер або audience. Використання підходу “білих списків”, де тільки конкретно дозволені алгоритми приймаються, значно знижує attack surface. Сучасні JWT бібліотеки надають параметри конфігурації для примусового використання специфічних алгоритмів, і ці параметри повинні

завжди використовуватися замість покладання на динамічне визначення алгоритму.

Регулярне оновлення JWT бібліотек до останніх версій забезпечує отримання безпекових оновлень для відомих вразливостей. Статичний аналіз коду та безпекові огляди коду повинні специфічно перевіряти конфігурацію JWT валідації, гарантуючи, що алгоритми явно специфіковані та не довіряють header токена. Тестування на проникнення повинно включати спроби algorithm confusion та none algorithm атак для верифікації належного захисту [58].

2.4.2 Слабкі секрети

Симетричні алгоритми підпису JWT, такі як HS256, HS384 та HS512, покладаються на спільний секретний ключ між провайдер та verifier токенів. Безпека цих токенів повністю залежить від складності та конфіденційності секретного ключа. Використання слабких, легко передбачуваних або загальновідомих секретів створює критичну вразливість, що дозволяє зловмисникам підробляти валідні токени.

Brute-force атаки на JWT секрети стали практично здійсненними завдяки спеціалізованим інструментам та зростанню обчислювальної потужності. Hashcat, широко використовуваний інструмент для відновлення паролів, включає спеціалізовані режими для атак методом “грубої сили” на JWT токени з HMAC підписами. Маючи перехоплений JWT токен, зловмисник може використовувати hashcat для систематичного тестування можливих секретних ключів до знаходження того, що генерує валідний підпис для токена.

Ефективність атаки “грубою силою” безпосередньо залежить від ентропії секретного ключа. Короткі секрети або секрети, що складаються тільки з малих літер або цифр, можуть бути зламані за хвилини або години на сучасному обладнанні. Дослідження показують, що секрети довжиною менше дванадцяти символів є вразливими до атак перебору з використанням GPU-прискореного

hashcat. Секрети, що є словами зі словника або поширеними фразами, можуть бути зламані практично миттєво через використанні словників таких слів та фраз.

Типова hashcat команда для атаки на JWT токен включає специфікацію hash mode для JWT (16500 для HS256), цільовий токен та wordlist або brute-force mask. Сучасні GPU можуть тестувати мільярди HMAC-SHA256 хешів за секунду, роблячи навіть відносно складні паролі вразливими при достатньому часі. Rainbow tables для поширених секретів та patterns додатково прискорюють процес відновлення слабких ключів.

Використання загальновідомих або default секретів представляє особливо серйозну вразливість. Численні tutorials, приклади коду та starter templates використовують тривіальні секрети типу secret, your-256-bit-secret, mysecretkey або навіть порожні рядки для демонстраційних цілей. Розробники часто копіюють ці приклади без зміни секрету перед deployment у production, створюючи критичну вразливість.

Публічні репозиторії коду на GitHub та подібних платформах часто містять залишені JWT секрети, не помічені розробниками. Автоматизовані сканери постійно моніторять ці репозиторії, збираючи секрети та роблячи їх доступними для зловмисників. Навіть після видалення секрету з поточної версії коду, він залишається в історії Git, дозволяючи його відновлення.

Списки поширених JWT секретів циркулюють у спільноті та серед зловмисників. Ці списки компілюються з публічних витоків, аналізу відкритого коду та документації. Зложії регулярно тестують перехоплені JWT токени проти цих списків, часто успішно компрометуючи токени за секунди без необхідності комплексного перебору.

Приклад реального випадку демонструє GitLab вразливість 2021 року, де JWT секрет за замовчуванням не було належним чином змінено в деяких інсталяціях, дозволяючи зловмисникам генерувати валідні токени та отримувати несанкціонований доступ. Інцидент підкреслив критичну важливість належної конфігурації серветів при розгортанні застосунків. Захист від атак на слабкі секрети вимагає генерації криптографічно стійких ключів з достатньою

ентропією. Секретні ключі для HS256 повинні бути мінімум 32 байти (256 біт) випадкових даних, згенерованих криптографічно безпечним генератором випадкових чисел. Для HS384 та HS512 рекомендовані розміри становлять 48 та 64 байти відповідно. Використання інструментів типу openssl для генерації випадкових ключів забезпечує належну ентропію. Команда “openssl rand” дозволяє генерувати криптографічно безпечні випадкові байти, які можуть бути закодовані у форматі base64 або hex для використання як JWT секрет. Автоматизація генерації секретів як частини процесу розгортання гарантує, що кожна інсталяція застосунку має унікальний, непередбачуваний секрет.

Управління секретами через спеціалізовані сервіси типу AWS Secrets Manager, Azure Key Vault, HashiCorp Vault або Kubernetes Secrets забезпечує централізоване, безпечне зберігання та зміну секретів. Ці системи дозволяють уникнути “залишків” секретів у коді застосунку або конфігураційних файлах, натомість динамічно отримуючи їх під час runtime з належним контролем доступу та аудитом.

Регулярна зміна JWT секретів обмежує вікно можливої експлуатації у випадку компрометації ключа. Стратегія зміни повинна включати підтримку множинних активних ключів одночасно для забезпечення плавного переходу, де нові токени підписуються новим ключем, проте старі токени продовжують верифікуватися старим ключем до закінчення їх терміну дії. Автоматизація процесу зміни через планувальники задач знижує операційне навантаження та забезпечує консистентність.

Моніторинг спроб використання невалідних або експірованих токенів може сигналізувати про активні атаки. Високий рівень валідаційних помилок JWT з різних IP адрес може вказувати на brute-force спроби або використання скомпрометованих токенів. Integration з SIEM системами дозволяє корелювати JWT-специфічні події з іншими security events для комплексного threat detection [59].

2.4.3 Маніпуляції твердженнями

JWT payload містить claims – твердження просутність, типового користувача, та додаткові метадані. Неналежна валідація тверджень створює множинні вектори атак, що можуть призвести до обходу автентифікації, ескалації привілеїв або витоку інформації.

Missing validation критичних зареєстрованих тверджень представляє поширену вразливість. Специфікація JWT визначає кілька стандартних тверджень з специфічною семантикою, проте їх використання є optional, і застосунок повинен явно імплементувати валідацію. Claim exp (expiration time) визначає час, після якого токен не повинен прийматися для обробки. Відсутність валідації exp дозволяє використання токенів необмежено довго після їх видачі, навіть якщо користувач деавтентифікувався, змінив пароль або був деактивований адміністратором.

Зловмисник, який отримав токен через перехоплення або крадіжку, може використовувати його необмежено довго у системах без exp валідації. Навіть якщо організація виявляє компрометацію та намагається відкликати доступ користувача, відсутність перевірки expiration робить відкликання неефективним. Токен залишається технічно валідним з точки зору підпису, і система продовжує приймати його.

Claim aud (audience) визначає призначених отримувачів токена. Токен, виданий для одного застосунку або API, не повинен прийматися іншим застосунком навіть в межах тієї самої організації. Відсутність валідації aud створює ризик cross-service атак, де токен, легітимно отриманий для одного сервісу з обмеженими правами, використовується для доступу до іншого сервісу з більш чутливими функціями.

Claim iss (провайдер) ідентифікує сторону, що видала токен. У системах з множинними authorization servers або федеративною автентифікацією валідація iss гарантує, що токен був виданий довіреним джерелом. Відсутність перевірки

дозволяє зловмисникам створювати токени з підробленим провайдер та потенційно обманювати систему, якщо інші валідації (наприклад, підпис) можуть бути обійдені або якщо існують інші вразливості.

Claim nbf (not before) визначає час, до якого токен не повинен прийматися. Хоча менш критична, ніж exp, валідація nbf може запобігати використанню токенів, виданих для майбутнього використання, в сценаріях з повноваженнями за графіком або пре-авторизацією.

JWT ін’єкції через маніпулювання загловком твердження представляє складнішу категорію атак. Claim kid (key ID) в заголовку токена вказує, який ключ використовувався для підпису токена, дозволяючи підтримку множинних ключів одночасно для зміни або сенарію використання в різних середовищах. Вразливі імплементації можуть використовувати kid для динамічного завантаження ключів з файлової системи, бази даних або зовнішніх кінцквих точок без належної валідації.

Зловмисник може модифікувати kid для виконання path traversal атак, вказуючи шляхи типу “../../etc/passwd” або подібні, щоб змусити систему використовувати довільний файл як ключ верифікації. Якщо зловмисник має контроль над вмістом файлу або знає його вміст, він може створити токен, підписаний з використанням цього вмісту як ключа, який система прийме при валідації. Інша варіація включає SQL ін’єкції через kid, якщо значення використовується в запиті до бази даних без належної санітизації для отримання ключа. Зловмисник може ввести SQL запит, що модифікує кінцеву відповідь для повернення контрольованого значення, як ключа верифікації.

Claim jku (JWK Set URL) вказує URL, де можна знайти набір JSON Web Keys для верифікації токена. Вразливі імплементації можуть довіряти jku твердженням та автоматично завантажувати ключі з вказаного URL без валідації джерела. Зловмисник може встановити jku на контрольований ним сервер, що повертає публічний ключ та підписаний токен відповідним приватним ключем. Система завантажить публічний ключ зловмисника з jku URL та використає його для верифікації токена, помилково приймаючи підроблений токен як валідний.

Аналогічна атака можлива через claim x5u (X.509 URL), що вказує URL для X.509 сертифіката або ланцюжка сертифікатів. Зловмисник може вказати URL на контрольований сервер зі згенерованим сертифікатом та підписати токен відповідним приватним ключем, обманюючи систему завантажити та довіряти сертифікату зловмисника.

Custom claims без належної валідації також представляють ризик. Застосунки часто додають власні claims до JWT payload для передачі application-specific інформації типу user roles, permissions, tenant IDs або feature flags. Якщо система довіряє цим claims без перевірки їх відповідності backend state або без re-validation при критичних операціях, зловмисник може модифікувати claims для ескалації привілеїв або обходу access controls.

Приклад може включати твердження “isAdmin: false” у токені звичайного користувача. Якщо система перевіряє тільки підпис токена, проте довіряє значенню “isAdmin” в пейлоаді при прийнятті авторизаційних рішень, зловмисник з доступом до валідного токена може спробувати змінити claim на isAdmin: true. Якщо існує інша вразливість, що дозволяє обхід валідації підпису (наприклад, none algorithm або algorithm confusion), модифікований claim буде прийнятий системою.

Захист від маніпуляцій claims вимагає comprehensive validation на стороні сервера. Імплементація повинна обов'язково валідувати всі зареєстровані claims, що мають значення для безпеки застосунку. Claim exp повинен завжди перевірятися, відхиляючи токени з минулим expiration timestamp з урахуванням clock skew tolerance. Claim aud повинен точно відповідати очікуваному audience для поточного сервісу або API. Claim iss повинен бути з «білі списки» довірених провайдерs.

Для запобігання JWT ін’єкції атакам система не повинна динамічно завантажувати ключі верифікації на основі claims, що контролюються клієнтом, таких як kid, jku або x5u, без строгої валідації. Якщо kid використовується, його значення повинно бути «білі списки» з дозволених key IDs, що відповідають попередньо сконфігурованим ключам. Claims jku та x5u повинні або повністю

ігноруватися, або обмежуватися строгим «білі списки» довірених URLs з перевіркою SSL/TLS сертифікатів при завантаженні.

Custom claims повинні розглядатися як untrusted input навіть після валідації підпису токена. Критичні авторизаційні рішення не повинні покладатися виключно на claims в JWT, натомість re-querying authoritative backend systems для верифікації permissions при доступі до чутливих ресурсів або функцій. JWT може використовуватися для початкової ідентифікації та caching базової інформації про користувача, проте critical access controls повинні базуватися на server-authoritative data.

Використання статично сконфігурованих ключів верифікації замість динамічного discovery усуває цілий клас вразливостей, пов'язаних з header claims маніпуляцією. Якщо система знає точно, які ключі використовуються для підпису токенів від конкретного провайдер, немає необхідності довіряти kid або jku claims для знаходження ключів.

Regular security audits JWT implementation повинні включати тестування відсутності або неправильної валідації кожного security-relevant claim. Automated testing може генерувати токени з модифікованими або відсутніми claims та верифікувати, що система належним чином відхиляє їх. Penetration testing повинно специфічно включати спроби JWT ін’єкції через kid, jku та x5u claims для виявлення вразливостей до deployment.

Документація та developer education є критичними, оскільки багато вразливостей JWT виникають з непорозуміння security implications різних claims та header fields. Розробники повинні бути навчені тому, що валідація підпису є необхідною, проте недостатньою умовою безпеки JWT, та що comprehensive validation всіх security-relevant claims є обов'язковою для належного захисту [60].

2.5 Загрози мобільним застосункам

Мобільні застосунки представляють унікальний набір безпекових викликів, що відрізняються від традиційних веб-застосунків через фундаментальні особливості мобільних платформ. На відміну від веб-застосунків, де код виконується на контрольованому сервері, мобільні застосунки повністю завантажуються та виконуються на пристроях користувачів, що перебувають поза контролем розробників. Це створює притаманну проблему довіри, де потенційно ворожий актор має повний фізичний та логічний доступ до всього коду застосунку, даних у локальному сховищі та мережевого трафіку.

За даними OWASP Mobile Security Project, понад сімдесят відсотків мобільних застосунків мають принаймні одну середньої тяжкості або критичну вразливість безпеки. Специфіка мобільної екосистеми, включаючи необхідність автономної функціональності, обмеження продуктивності, різноманітність пристроїв та операційних систем, а також розширені можливості доступу до апаратних компонентів, створює розширену поверхню атаки порівняно з веб-застосунками.

Компрометація мобільного застосунку може мати особливо серйозні наслідки через персональний характер мобільних пристроїв. Смартфони зберігають величезні обсяги конфіденційної інформації, включаючи особисті повідомлення, фінансові дані, геолокаційну історію, фотографії та доступ до корпоративних ресурсів. Крім того, мобільні пристрої часто залишаються автентифікованими до критичних сервісів протягом тривалих періодів, роблячи їх привабливими цілями для зловмисників.

У цьому підрозділі розглядаються специфічні загрози безпеці мобільних застосунків у контексті автентифікації та авторизації, включаючи вразливості локального зберігання конфіденційних даних, проблеми мережевої безпеки та ризики, пов'язані з механізмами глибоких посилань [61].

2.5.1 Проблеми зберігання

Локальне зберігання конфіденційних даних на мобільних пристроях представляє критичний виклик безпеки. Мобільні застосунки часто потребують зберігання токенів автентифікації, ключів програмного інтерфейсу, користувацьких облікових даних або інших чутливих даних для забезпечення автономної функціональності або покращення користувацького досвіду через персистентну автентифікацію. Проте неналежне зберігання цих даних створює можливість для їх витоку через зворотну розробку застосунку, доступ до файлової системи на скомпрометованих пристроях або експлуатацію вразливостей операційної системи.

Жорстко закодовані облікові дані представляють одну з найпоширеніших та найнебезпечніших помилок у мобільних застосунках. Розробники інколи вбудовують ключі програмного інтерфейсу, секретні токени, паролі до серверних систем або криптографічні ключі безпосередньо в код застосунку, помилково вважаючи, що компіляція забезпечує достатній захист від виявлення. Проте сучасні інструменти зворотної розробки дозволяють тривіально декомпілювати мобільні застосунки та екстрагувати жорстко закодовані секрети з коду або ресурсів.

Інструмент jadx представляє широко використовуваний декомпілятор для застосунків Android, що конвертує байт-код DEX назад у читабельний код Java. Дослідник безпеки або зловмисник може завантажити файл APK застосунку з Google Play Store або іншого джерела, використовувати jadx для декомпіляції та систематично шукати жорстко закодовані секрети через пошук або ручний перегляд коду. Рядки з патернами ключів програмного інтерфейсу, адреси з вбудованими обліковими даними або криптографічні ключі часто легко ідентифікуються через характерні формати або контекст використання.

Реальні випадки демонструють поширеність проблеми. Дослідження мобільних застосунків першої тисячі в Google Play Store виявило, що приблизно двадцять відсотків містять жорстко закодовані ключі програмного інтерфейсу або

облікові дані. У деяких випадках ці облікові дані надають доступ до продакшн баз даних, сховищ AWS S3 з конфіденційними даними користувачів або адміністративних панелей серверних систем. Компрометація навіть одного застосунку може призвести до масштабного витоку даних, що впливає на мільйони користувачів.

Проблема ускладнюється для застосунків iOS, де бінарний код також може бути деобфускований та проаналізований через інструменти типу Hopper або IDA Pro. Незважаючи на те, що iOS використовує скомпільований нативний код замість байт-коду, рядки та структури даних залишаються доступними для екстракції. Навіть з обфускацією, яка ускладнює аналіз, жорстко закодовані секрети можуть бути виявлені через динамічний аналіз або дампи пам'яті під час виконання застосунку.

Незахищені спільні налаштування на платформі Android представляють іншу критичну вразливість зберігання. Механізм SharedPreferences є стандартним способом Android для збереження простих пар ключ-значення, що зберігаються між запусками застосунку. За замовчуванням, ці дані зберігаються у текстових файлах XML у приватній директорії застосунку. Хоча ці файли теоретично доступні тільки застосунку-власнику завдяки пісочниці Android, множинні сценарії дозволяють несанкціонований доступ.

На пристроях зroot-доступом обмеження пісочниці можуть бути обійдені, дозволяючи будь-якому процесу з привілеями суперкористувача читати спільні налаштування будь-якого застосунку. Зловмисне програмне забезпечення з доступом суперкористувача може систематично сканувати спільні налаштування всіх встановлених застосунків, шукаючи токени автентифікації, паролі або інші конфіденційні дані. Навіть на звичайних пристроях, налагоджувальний міст Android з увімкненим налагодженням через USB дозволяє екстрагувати дані застосунку через механізм резервного копіювання або команди отримання файлів, якщо застосунок не явно відключив функціональність резервного копіювання.

Застосунки часто зберігають у спільних налаштуваннях токени доступу, токени оновлення, ідентифікатори сесій або навіть паролі користувачів у незашифрованому вигляді. Дослідження показує, що близько тридцяти відсотків застосунків Android зберігають чутливі автентифікаційні дані у незашифрованих спільних налаштуваннях. Компрометація пристрою через зловмисне програмне забезпечення або фізичний доступ дозволяє зловмиснику екстрагувати ці дані та використовувати для несанкціонованого доступу до облікових записів користувачів.

Аналогічна проблема існує з користувацькими налаштуваннями iOS, де дані також зберігаються у незашифрованих файлах списку властивостей. Хоча пісочниця iOS є більш обмежуючою, пристрої з jailbreak або складне зловмисне програмне забезпечення можуть обходити захист. Крім того, резервні копії iTunes та iCloud за замовчуванням включають дані користувацьких налаштувань, створюючи додатковий вектор витоку якщо резервні копії не належним чином зашифровані або якщо зловмисник отримує доступ до файлів резервних копій.
Проблема незахищеного зберігання часто поглиблюється неправильним використанням інтерфейсів шифрування. Деякі розробники намагаються шифрувати дані перед зберіганням у спільних налаштуваннях, проте використовують жорстко закодовані ключі шифрування, роблячи шифрування марним, оскільки ключ може бути екстраговано з того самого декомпільованого коду. Інші використовують слабкі алгоритми шифрування або неправильні режими роботи, що можуть бути зламані або призводять до витоку відкритого тексту через атаки на заповнення або інші криптографічні вразливості.

Захист від проблем зберігання вимагає використання специфічних для платформи механізмів безпечного зберігання. Android надає зашифровані спільні налаштування як частину бібліотеки безпеки AndroidX, що автоматично шифрує ключі та значення перед зберіганням, використовуючи ключі з сховища ключів Android. Сховище ключів забезпечує апаратне шифрування на пристроях з підтримкою довіреного середовища виконання або захищеного сховища, роблячи екстракцію ключів практично неможливою навіть на пристроях з root-доступом.

iOS надає програмний інтерфейс служб ланцюжка ключів для безпечного зберігання облікових даних та криптографічних ключів. Дані ланцюжка ключів шифруються з використанням ключів, похідних від коду доступу пристрою та унікального ідентифікатора обладнання, забезпечуючи сильний захист навіть якщо файлова система скомпрометована. Елементи ланцюжка ключів можуть бути налаштовані з різними рівнями доступності, контролюючи коли дані доступні (тільки коли пристрій розблокований, завжди тощо) та чи вони включаються у резервні копії.

Критично важливим є ніколи не закодовувати жорстко секрети у код застосунку. Ключі програмного інтерфейсу та облікові дані серверної частини повинні зберігатися на сервері та доступ до них повинен контролюватися через належні потоки автентифікації. Якщо застосунок потребує локального зберігання токенів автентифікації, вони повинні бути отримані динамічно після автентифікації користувача та зберігатися у безпечному сховищі з обмеженим часом життя.

Обфускація коду через ProGuard на Android або аналогічні інструменти на iOS може ускладнити зворотну розробку, проте не повинна розглядатися як основний контроль безпеки. Обфускація є додатковим заходом захисту, що підвищує складність аналізу, проте не запобігає йому повністю. Автоматизовані інструменти сканування секретів повинні використовуватися у конвеєрі безперервної інтеграції для виявлення випадкового внесення секретів до системи контролю версій або включення у пакети застосунків [62].

2.5.2 Мережева безпека

Мережева комунікація між мобільними застосунками та серверами представляє критичну поверхню атаки, особливо у контексті передачі автентифікаційних облікових даних та токенів. Незважаючи на широке використання протоколів захищеного транспортного рівня для шифрування

трафіку, мобільні платформи вразливі до складних атак посередника, що експлуатують слабкості у перевірці сертифікатів або довіру до встановлених користувачем сертифікатів.

Відсутність закріплення сертифікатів представляє одну з найпоширеніших вразливостей мобільної мережевої безпеки. За замовчуванням, мобільні операційні системи довіряють будь-якому сертифікату захищеного з'єднання, підписаному центром сертифікації з їхнього системного сховища довіри. Це означає, що якщо зловмисник може змусити пристрій довіряти сертифікату центру сертифікації, контрольованого зловмисником, він може виконувати атаки посередника, перехоплюючи та декодуючи весь трафік захищених з'єднань застосунку.

Сценарій атаки типово включає встановлення зловмисником проксі-сервера типу mitmproxy, Burp Suite або Charles Proxy в мережі між мобільним пристроєм та сервером. Це легко досяжимо у публічних мережах бездротового доступу, скомпрометованих домашніх маршрутизаторах або через перехоплення системи доменних імен. Проксі генерує самопідписаний сертифікат центру сертифікації та представляє його пристрою. У застосунках без закріплення користувач може бути обманутий або примушений встановити цей сертифікат у системне сховище довіри, після чого проксі може розшифровувати, інспектувати та модифікувати весь трафік захищених з'єднань.

Перехоплення автентифікаційного трафіку дозволяє зловмиснику екстрагувати імена користувачів, паролі, ключі програмного інтерфейсу, токени OAuth та інші конфіденційні дані, що передаються між застосунком та сервером. Навіть якщо паролі належним чином хешуються на сервері, зловмисник отримує пароль у відкритому вигляді до його передачі, дозволяючи атаки повторного використання облікових даних на інших сервісах, де користувач може використовувати той самий пароль [63].

Закріплення сертифікатів є технікою, що усуває цю вразливість шляхом жорсткого прив'язування застосунку до конкретного сертифіката або публічного ключа сервера, ігноруючи системне сховище довіри. Застосунки із закріпленням

відхиляють з'єднання до серверів, що представляють сертифікати, які не точно відповідають закріпленим значенням, навіть якщо ці сертифікати підписані довіреними центрами сертифікації. Це робить атаки посередника практично неможливими, оскільки зловмисник не може отримати приватний ключ, відповідний закріпленому публічному ключу сервера.

Проте навіть закріплення сертифікатів може бути обійдене через фреймворки маніпулювання під час виконання типу Frida. Frida є потужним інструментарієм динамічної інструментації, що дозволяє впроваджувати код JavaScript у запущені процеси для модифікації їх поведінки під час виконання. Досвідчені зловмисники використовують Frida для перехоплення функцій перевірки сертифікатів у мобільних застосунках та примусового їх завжди повертати успіх, ефективно відключаючи закріплення.

Атаки обходу через Frida вимагають технічної експертизи, проте автоматизовані сценарії та готові інструменти широко доступні. Зловмисник встановлює серверну частину Frida на пристрої з root-доступом або jailbreak, ідентифікує функції перевірки сертифікатів у цільовому застосунку через зворотну розробку, пише сценарій Frida для перехоплення цих функцій та модифікації їх повернутих значень для обходу перевірок закріплення. З запущеним сценарієм Frida застосунок приймає будь-які сертифікати, включаючи контрольовані зловмисником, дозволяючи атаку посередника.

Захист від обходу через Frida є складним викликом. Виявлення root-доступу або jailbreak може виявити скомпрометовані пристрої та відмовити у сервісі, проте складні методи отриманняroot-доступу можуть приховувати свою присутність. Техніки протидії втручанню типу перевірки цілісності бінарного файлу застосунку, виявлення налагоджувачів або моніторингу специфічних для Frida артефактів можуть ускладнити маніпулювання під час виконання, проте рішучі зловмисники часто можуть обходити ці захисти через послідовні рівні обфускації та протидії виявленню.

Підхід глибокоешелонованого захисту комбінує закріплення сертифікатів з додатковими рівнями безпеки. Закріплення сертифікатів повинно

використовувати закріплення публічних ключів замість закріплення сертифікатів, дозволяючи ротацію сертифікатів без необхідності оновлення застосунку, поки публічний ключ залишається незмінним. Резервні закріплення повинні бути включені для полегшення ротації закріплення та запобігання постійному блокуванню у випадку втрати контролю над закріпленими сертифікатами.

Техніки самозахисту застосунків під час виконання можуть виявляти та реагувати на спроби втручання або налагодження. Обфускація коду та перевірки протидії налагодженню підвищують складність зворотної розробки та маніпулювання під час виконання. Проте критично важливо розуміти, що жоден контроль безпеки на стороні клієнта не є абсолютним проти рішучого зловмисника з повним контролем над пристроєм.

Серверні захисти доповнюють клієнтські контролі. Серверна частина повинна імплементувати всебічну валідацію вхідних даних, оскільки весь трафік, що походить від клієнта, повинен розглядатися як потенційно зловмисний. Обмеження швидкості та виявлення аномалій можуть виявляти патерни, узгоджені з атаками посередника, такі як множинні запити з ідентичними цифровими відбитками або підозрілі географічні патерни. Багатофакторна автентифікація забезпечує додатковий рівень захисту навіть якщо облікові дані перехоплені через атаку посередника [64].

2.5.3 Викрадення глибоких посилань

Глибокі посилання дозволяють зовнішнім джерелам відкривати конкретний вміст або функціональність усередині мобільного застосунку через власні схеми адрес або універсальні посилання. У контексті автентифікації та авторизації глибокі посилання часто використовуються для зворотних викликів перенаправлення OAuth, де код авторизації або токени передаються назад до мобільного застосунку після автентифікації користувача на сервері авторизації.

Викрадення глибоких посилань виникає, коли зловмисний застосунок реєструє ту саму власну схему адреси або патерн універсального посилання, що й легітимний застосунок, створюючи можливість перехоплення конфіденційних даних, призначених для легітимного застосунку. Коли операційна система отримує глибоке посилання, вона може не мати однозначного способу визначити, який застосунок повинен його обробити, потенційно доставляючи його до зловмисного застосунку замість призначеного одержувача.

Крадіжка коду авторизації OAuth через викрадення глибоких посилань представляє серйозну загрозу. Типовий потік OAuth для мобільних застосунків використовує власну схему адреси для зворотного виклику перенаправлення. Застосунок реєструє схему типу myapp://oauth/callback та включає її як адресу перенаправлення у запиті авторизації OAuth. Після автентифікації користувача сервер авторизації перенаправляє браузер назад до “myapp://oauth/callback?code=КОД_АВТОРИЗАЦІЇ”. Операційна система повинна доставити цю адресу до легітимного застосунку.

Проте якщо зловмисний застосунок також реєструє схему myapp://, операційна система може доставити зворотний виклик до зловмисного застосунку замість легітимного. Зловмисний застосунок перехоплює код авторизації, який потім може бути обміняний на токен доступу на сервері авторизації, якщо автентифікація клієнта не належним чином реалізована або якщо використовується публічний клієнт без підтвердження ключа для обміну кодом.

На платформі Android до недавнього часу множинні застосунки могли реєструвати ідентичні фільтри намірів для тих самих схем адрес, та система показувала діалог вибору користувачу для вибору застосунку. Складні зловмисні застосунки могли використовувати техніки маніпулювання інтерфейсом для обману користувача у виборі зловмісного застосунку. Навіть без явного вибору користувача, залежно від версії Android та конфігурації, система могла доставляти намір до першого встановленого або нещодавно використаного застосунку.

iOS має аналогічні вразливості з власними схемами адрес, де множинні застосунки можуть реєструвати ту саму схему, та система не гарантує доставку до конкретного застосунку. Універсальні посилання на iOS та посилання застосунків на Android були введені для вирішення цієї проблеми через криптографічну верифікацію асоціації між доменом та застосунком, проте їх впровадження вимагає належної конфігурації сервера та можуть повертатися до власних схем адрес якщо верифікація не вдається.

Захист від викрадення глибоких посилань для потоків OAuth критично залежить від підтвердження ключа для обміну кодом. Цей механізм був спеціально розроблений для захисту мобільних потоків OAuth від перехоплення коду авторизації. Навіть якщо зловмісний застосунок перехоплює код авторизації через викрадення глибоких посилань, він не може обміняти його на токен доступу без відповідного верифікатора коду, який був згенерований легітимним застосунком та ніколи не передавався через перенаправлення [65].

Використання універсальних посилань на iOS та посилань застосунків на Android замість власних схем адрес значно знижує ризик. Ці механізми вимагають розміщення спеціального файлу верифікації (apple-app-site-association або assetlinks.json) з домену, вказаного в адресі перенаправлення, криптографічно асоціюючи домен з конкретним застосунком. Операційна система верифікує цю асоціацію перед доставкою посилання до застосунку, запобігаючи зловмісним застосункам викрадати посилання для доменів, які вони не контролюють.

Належна реалізація універсальних посилань або посилань застосунків вимагає розміщення файлів верифікації через захищене з'єднання, правильної конфігурації прав застосунку або маніфесту, та обробки потенційних сценаріїв відкату. Застосунки повинні валідувати джерело глибоких посилань перед обробкою конфіденційних даних, перевіряючи що посилання прийшло з очікуваного домену та містить очікувані параметри.

Серверна валідація додає додатковий рівень захисту. Сервер авторизації повинен імплементувати сувору валідацію адреси перенаправлення, вимагаючи точної відповідності зареєстрованим адресам та не приймаючи шаблонів або

зіставлення префіксів для мобільних адрес перенаправлення. Короткочасні коди авторизації з примусовим одноразовим використанням обмежують вікно експлуатації навіть якщо код перехоплено.

Клієнтський застосунок повинен імплементувати валідацію параметра стану у зворотних викликах OAuth для захисту від атак підробки міжсайтових запитів та верифікувати цілісність даних зворотного виклику. Всебічне логування подій обробки глибоких посилань може допомогти виявити спроби викрадення через аналіз неочікуваних патернів або множинних доставок того самого коду авторизації [66].

2.6 Проблеми імплементації

Вразливості автентифікації та авторизації часто виникають не через фундаментальні слабкості протоколів або алгоритмів, а через помилки імплементації розробників та експлуатацію людського фактору. Навіть найнадійніші криптографічні протоколи та найсучасніші фреймворки безпеки можуть бути скомпрометовані через тривіальні помилки у коді, неналежну конфігурацію або успішні атаки соціальної інженерії на кінцевих користувачів. За даними аналізу порушень безпеки Verizon Data Breach Investigations Report 2024, понад вісімдесят відсотків інцидентів безпеки включають людський елемент, включаючи соціальну інженерію, помилки конфігурації або зловживання привілеями.

Розрив між теоретичною безпекою протоколів та практичною безпекою реальних систем часто визначається якістю імплементації. Розробники можуть мати обмежену експертизу у криптографії та безпеці, покладатися на застарілі приклади коду або документацію, працювати під тиском строків, що призводить до скорочення шляхів у безпеці, або просто не усвідомлювати безпекові імплікації певних рішень дизайну. Ці фактори призводять до систематичних категорій помилок, що повторюються у різних застосунках та організаціях.

Людський фактор додає додатковий рівень складності до безпеки систем автентифікації. Навіть технічно досконалі системи можуть бути скомпрометовані через маніпулювання користувачами для розголошення облікових даних, вибір слабких паролів або необережну поведінку з конфіденційною інформацією. Психологічні принципи, що лежать в основі соціальної інженерії, експлуатують природні людські тенденції довіряти авторитету, поспішати у стресових ситуаціях або прагнути бути корисними.

У цьому підрозділі розглядаються найпоширеніші категорії помилок імплементації у системах автентифікації та авторизації, а також аналізується роль людського фактору як критичної вразливості безпеки [67].

2.6.1 Поширені помилки

Неналежна обробка помилок представляє фундаментальну категорію вразливостей, що виникає через розголошення надмірної інформації про внутрішню структуру системи або різні шляхи виконання для різних станів помилок. Застосунки часто генерують детальні повідомлення про помилки для полегшення налагодження під час розробки, проте ці повідомлення випадково залишаються увімкненими у продакшн середовищах, надаючи зловмисникам цінну інформацію для розвідки та експлуатації.

Класичний приклад неналежної обробки помилок виникає у формах автентифікації, де система надає різні повідомлення про помилки для невірного імені користувача порівняно з невірним паролем. Коли користувач намагається увійти з неіснуючим іменем користувача, система може відобразити повідомлення “Користувача не знайдено”, тоді як для дійсного користувача з невірним паролем повідомлення може бути “Невірний пароль”. Ця тонка різниця дозволяє зловмисникам здійснювати перерахування облікових записів, систематично тестуючи імена користувачів для визначення, які існують у системі, перш ніж намагатися вгадати паролі.

Детальні повідомлення про помилки можуть розголошувати технічні деталі реалізації, що допомагають зловмисникам у плануванні атак. Повідомлення типу "Помилка з'єднання з базою даних MySQL на “localhost:3306” розкриває тип бази даних, розташування та потенційні вразливості конфігурації. Повідомлення про помилки SQL ін'єкції, що включають фрагменти SQL запитів або повні трасування стеку, надають зловмисникам детальну інформацію про структуру бази даних та можливі вектори атак.

Різниця у часі відповіді між різними сценаріями помилок створює вектор атаки через аналіз часових характеристик. Якщо перевірка існування користувача відбувається перед перевіркою пароля, система може відповідати швидше для неіснуючих користувачів порівняно з дійсними користувачами з невірними паролями, де виконується дороговартісна операція хешування пароля. Зловмисники можуть вимірювати ці часові різниці для перерахування дійсних облікових записів навіть якщо повідомлення про помилки ідентичні.

Логування конфіденційних даних представляє критичну помилку безпеки, що виникає через занадто агресивне логування під час розробки та тестування, яке випадково залишається активним у продакшн середовищах. Системи логування призначені для запису діагностичної інформації для налагодження та аудиту, проте часто фіксують дані, які ніколи не повинні зберігатися у незашифрованому вигляді або бути доступними персоналу з обмеженими привілеями.

Паролі користувачів у відкритому вигляді є найбільш критичною категорією конфіденційних даних, що неналежним чином логується. Застосунки можуть логувати вхідні параметри запитів автентифікації для діагностичних цілей, випадково захоплюючи паролі у відкритому вигляді. Лог-запис типу "Спроба входу: користувач=john.doe@example.com, пароль=MySecretPass123" створює постійний запис облікових даних користувача, доступний будь-кому з доступом до лог-файлів, включаючи адміністраторів системи, персонал підтримки або зловмисників, що компрометували сервер.

Токени автентифікації та сесійні ідентифікатори також часто неналежним чином логуються. Лог-записи веб-серверів за замовчуванням можуть включати повні адреси запитів, що містять токени у параметрах запиту. Логування заголовків автентифікації для налагодження інтеграцій програмних інтерфейсів може захоплювати токени доступу або ключі програмних інтерфейсів. Детальне логування тіла запитів може записувати токени оновлення або інші конфіденційні облікові дані під час обміну токенів.

Персональні ідентифікаційні дані користувачів, що логуються надмірно, створюють ризики порушень регуляцій про захист даних. Загальний регламент захисту даних та подібні нормативні акти вимагають мінімізації обробки персональних даних. Логування повних профілів користувачів, включаючи електронні адреси, номери телефонів, фізичні адреси або фінансову інформацію, може порушувати ці вимоги та створювати значні зобов'язання у випадку компрометації лог-файлів.

Проблема посилюється через централізацію логів у системах управління інформацією та подіями безпеки або платформах агрегації логів. Хоча централізація покращує можливості моніторингу безпеки, вона також створює єдину точку концентрації конфіденційних даних. Компрометація системи управління логами потенційно розкриває роки накопичених конфіденційних даних з множини систем.

Слабкість випадковості у генерації криптографічних матеріалів представляє тонку, проте критичну вразливість. Багато компонентів безпеки покладаються на непередбачувані випадкові значення, включаючи ідентифікатори сесій, токени скидання пароля, криптографічні ключі, параметри стану OAuth та солі для хешування паролів. Використання слабких генераторів випадкових чисел робить ці значення передбачуваними, підриваючи безпеку всієї системи.

Програмування мовами часто надають множинні генератори випадкових чисел з різними властивостями безпеки. Функції типу Math.random у JavaScript, random.random у Python або java.util.Random у Java призначені для загальних

цілей, таких як випадковий вибір елементів або симуляції, проте не є криптографічно безпечними. Ці генератори використовують детерміновані алгоритми з обмеженими значеннями ініціалізації, роблячи їх послідовності передбачуваними для зловмисників, які можуть визначити або вгадати значення ініціалізації.

Реальний приклад демонструє вразливість у функції скидання пароля, що використовувала Math.random для генерації токенів скидання. Дослідники виявили, що токени слідували передбачуваному паттерну, дозволяючи зловмисникам генерувати валідні токени скидання для довільних облікових записів без доступу до електронної пошти користувачів. Вразливість виникла через те, що Math.random ініціалізується поточним часом з точністю до мілісекунди, надаючи тільки близько тридцяти біт ентропії замість необхідних ста двадцяти восьми або більше.

Криптографічно безпечні генератори випадкових чисел, такі як crypto.getRandomValues у JavaScript, secrets у Python або SecureRandom у Java, використовують джерела ентропії операційної системи та забезпечують непередбачувані значення навіть для зловмисників зі знанням внутрішнього стану генератора. Використання цих генераторів є абсолютною вимогою для будь-яких значень, пов'язаних з безпекою.

Застарілі залежності представляють постійну проблему безпеки у сучасній розробці програмного забезпечення. Застосунки покладаються на численні бібліотеки третіх сторін та фреймворки для реалізації функціональності автентифікації, шифрування та обробки токенів. Ці залежності регулярно отримують оновлення безпеки для виправлення виявлених вразливостей, проте застосунки часто продовжують використовувати застарілі версії через відсутність процесів управління залежностями або побоювання, що оновлення можуть порушити функціональність.

Дослідження екосистеми npm виявило, що понад вісімдесят відсотків застосунків використовують принаймні одну залежність з відомими вразливостями безпеки. Бібліотеки автентифікації та криптографії є особливо

критичними, оскільки вразливості у цих компонентах безпосередньо компрометують безпеку системи. Застаріла версія бібліотеки JWT з вразливістю обходу перевірки підпису дозволяє зловмисникам підробляти токени незалежно від того, наскільки правильно решта системи імплементована.

Проблема ускладюється транзитивними залежностями, де бібліотека, безпосередньо використовувана застосунком, сама покладається на інші бібліотеки. Вразливість у транзитивній залежності може не бути очевидною розробникам, які фокусуються тільки на прямих залежностях. Автоматизовані інструменти сканування залежностей, інтегровані у конвеєри безперервної інтеграції, є необхідними для виявлення та відстеження застарілих компонентів. Захист від помилок імплементації вимагає прийняття практик безпечної розробки. Обробка помилок повинна використовувати загальні повідомлення для зовнішніх користувачів, незалежно від внутрішньої причини помилки. Автентифікація повинна завжди повертати ідентичне повідомлення типу "Невірні облікові дані" незалежно від того, чи ім'я користувача не існує, чи пароль невірний. Часові характеристики повинні бути константними через штучні затримки або постійно-часові операції порівняння.

Логування повинно дотримуватися принципу мінімізації конфіденційних даних. Паролі ніколи не повинні логуватися у будь-якій формі. Токени повинні логуватися тільки у зрізаній або хешованій формі, якщо необхідно для аудиту. Персональна інформація повинна маскуватися або псевдонімізуватися у логах. Рівні логування повинні бути належним чином налаштовані для продакшн середовищ, відключаючи детальне логування налагодження.

Всі криптографічні операції повинні використовувати тільки криптографічно безпечні генератори випадкових чисел. Перегляд коду та автоматизовані інструменти статичного аналізу можуть виявляти неналежне використання слабких генераторів. Управління залежностями повинно бути автоматизованим через інструменти типу Dependabot, Snyk або OWASP Dependency-Check, що регулярно сканують вразливості та генерують сповіщення для застарілих компонентів [68].

2.6.2 Людський фактор

Фішинг представляє найпоширенішу та найефективнішу категорію атак соціальної інженерії, що націлена на отримання облікових даних автентифікації через обман користувачів. За даними звіту Anti-Phishing Working Group, протягом 2023 року було зареєстровано понад п'ять мільйонів фішингових атак, що представляє зростання на сорок відсотків порівняно з попереднім роком. Фішинг еволюціонував від простих масових електронних розсилок до високо цільових, персоналізованих атак, що використовують складну соціальну інженерію та технічні прийоми для обходу захисних механізмів.

Електронний фішинг залишається найпоширенішим вектором атаки, де зловмисники надсилають електронні листи, що видають себе за легітимні організації, просячи користувачів натиснути на посилання та ввести свої облікові дані на підробленому веб-сайті. Сучасні фішингові електронні листи часто є високоякісними імітаціями легітимної кореспонденції, використовуючи офіційні логотипи, форматування та мову. Вони створюють відчуття терміновості через повідомлення типу "Ваш обліковий запис буде заблоковано протягом двадцяти чотирьох годин якщо ви не підтвердите свою інформацію" для стимулювання поспішних дій без критичного аналізу.

Спіар-фішинг представляє більш цільову варіацію, де зловмисники досліджують конкретних жертв через соціальні мережі, публічні бази даних або попередні витоки даних для створення персоналізованих повідомлень. Електронний лист може посилатися на реальні проєкти, колег або події у житті жертви, значно підвищуючи правдоподібність та ймовірність успіху. Виконавчий директор може отримати електронний лист, що нібито від їхнього фінансового радника, посилаючись на недавню зустріч та просячи термінового підтвердження деталей облікового запису.

Технічна складність фішингових атак зросла з впровадженням досконалих технік обходу. Зловмисники реєструють доменні імена, візуально подібні до легітимних доменів через гомографічні атаки, використовуючи символи Unicode,

що виглядають ідентично латинським буквам. Домен apple.com може використовувати кириличний символ “і” замість латинського “i”, роблячи його візуально нерозрізненним у більшості шрифтів, проте технічно це повністю інший домен.

Фішингові сайти часто розміщуються на легітимних платформах хмарних сервісів або компрометованих легітимних веб-сайтах, успадковуючи валідні сертифікати захищених з'єднань та репутацію домену. Користувачі, навчені перевіряти наявність значка замка та протоколу захищеного з'єднання, можуть бути обмануті, оскільки фішинговий сайт демонструє ці індикатори безпеки. Деякі фішингові кампанії використовують проміжні сторінки перенаправлення на легітимних сервісах типу скорочувачів адрес або рекламних мереж для обходу фільтрів електронної пошти.

Соціальна інженерія охоплює ширший спектр психологічних маніпуляцій поза електронним фішингом. Телефонні атаки, де зловмисники дзвонять жертвам, видаючи себе за персонал технічної підтримки, представників банку або колег, залишаються ефективними. Зловмисник може стверджувати, що виявив підозрілу активність на обліковому записі жертви та потребує верифікації облікових даних для розслідування, експлуатуючи природне занепокоєння користувача про безпеку облікового запису.

Претекстинг включає створення складного вигаданого сценарію для отримання інформації. Зловмисник може провести тижні, встановлюючи довіру з жертвою через множинні взаємодії, поступово збираючи фрагменти інформації перед фінальним запитом критичних облікових даних. У корпоративному контексті зловмисник може видавати себе за нового співробітника, консультанта або аудитора, використовуючи соціальну динаміку бажання бути корисним або поважання авторитету.

Приманювання використовує обіцянки винагороди або цікавості для залучення жертв до компрометуючої поведінки. Залишені USB накопичувачі з етикетками типу “Конфіденційно - зарплати виконавчого персоналу” у паркінгу компанії експлуатують цікавість, спонукаючи співробітників підключати їх до

корпоративних комп'ютерів, де вони можуть захоплювати облікові дані або встановлювати зловмисне програмне забезпечення. Цифрові варіації включають привабливі завантаження, що вимагають автентифікації перед доступом до нібито цінного вмісту.

Слабкі паролі залишаються постійною вразливістю незважаючи на десятиліття підвищення обізнаності безпеки. Користувачі продовжують обирати легко запам'ятовувані паролі, які також є легко вгадуваними, через когнітивне навантаження запам'ятовування множини складних паролів для численних облікових записів. Аналіз мільярдів скомпрометованих паролів з витоків даних постійно демонструє, що найпоширеніші паролі включають тривіальні варіанти типу "password", “123456”, “qwerty” та “hello”.

Навіть коли користувачі намагаються створювати сильніші паролі, вони часто слідують передбачуваним паттернам. Вимоги політики паролів для включення великої літери, цифри та спеціального символу часто призводять до паттерну “Word1!” де користувачі просто капіталізують першу літеру та додають “1!” в кінець. Сезонні паролі, що включають поточний рік або сезон, є поширеними та легко вгадуються зловмисниками, які знають приблизний час створення пароля.

Повторне використання паролів між множинними сервісами створює каскадний ризик безпеки. Коли один сервіс зазнає витоку даних, скомпрометовані комбінації електронної пошти та пароля стають доступними зловмисникам для атак повторного використання облікових даних на інших платформах. Користувачі часто використовують ідентичний або незначно модифікований пароль для критичних облікових записів типу електронної пошти, банківських сервісів та соціальних мереж, дозволяючи одиничній компрометації поширюватися на всю їхню цифрову ідентичність.

Психологічні фактори сприяють слабким практикам паролів. Оптимістичне упередження призводить до того, що користувачі вважають себе малоймовірними цілями зловмисників, недооцінюючи ризики слабких паролів. Втома безпеки від постійних попереджень та вимог призводить до апатії, де

користувачі свідомо ігнорують рекомендації безпеки. Зручність переважає безпеку у щоденних рішеннях, особливо коли безпосередні наслідки слабких практик не є очевидними.

Захист від людського фактору вимагає багатогранного підходу. Навчання обізнаності безпеки повинно бути постійним та включати симульовані фішингові вправи для навчання співробітників розпізнавати та звітувати про підозрілі комунікації. Організації можуть відстежувати ефективність навчання через метрики типу рівня кліків на симульовані фішингові електронні листи та швидкості звітування.

Технічні контролі доповнюють навчання користувачів. Фільтри електронної пошти та системи захисту від фішингу можуть блокувати багато фішингових спроб до досягнення користувачів. Автентифікація електронної пошти через протоколи перевірки відправника допомагає виявляти підроблені електронні листи. Багатофакторна автентифікація значно знижує ризик компрометації навіть якщо паролі фішинговані, особливо стійкі до фішингу методи типу ключів доступу.

Менеджери паролів вирішують проблему слабких та повторюваних паролів шляхом генерації та зберігання унікальних, складних паролів для кожного сервісу. Користувачам потрібно запам'ятати тільки один майстер-пароль для доступу до всіх своїх облікових даних. Організації можуть заохочувати або вимагати використання менеджерів паролів через корпоративні політики та надання ліцензій.

Належна політика паролів повинна балансувати безпеку з зручністю використання. Вимоги мінімальної довжини принаймні дванадцять символів є більш ефективними ніж складні правила композиції. Перевірка паролів проти баз скомпрометованих паролів під час створення запобігає використанню відомо слабких варіантів. Періодична примусова зміна паролів часто є контрпродуктивною, оскільки призводить до передбачуваних інкрементних змін, натомість фокусуючись на виявленні компрометації та примусовій зміні при підозрі [69].

РОЗДІЛ 3 ОЦІНКА ЗАХИЩЕНОСТІ

3.1 Методологія та тестове середовище

3.1.1 Методологія оцінки

Дослідження базується на принципах системного підходу, що передбачає розгляд механізмів автентифікації та авторизації як цілісної системи взаємопов'язаних компонентів. Застосовано метод порівняльного аналізу для виявлення спільних та відмінних характеристик вразливостей веб та мобільних платформ.

Оцінка критичності виявлених вразливостей здійснювалась за стандартом CVSS v3.1, що враховує такі метрики: вектор атаки, складність експлуатації, необхідні привілеї, вплив на конфіденційність, цілісність та доступність.

Таблиця 3.1

Шкала оцінки CVSS v3.1

Числова оцінка	**Рівень серйозності**	**Характеристика**
0.1 – 3.9	Низький (Low)	Обмежений вплив на систему
4.0 – 6.9	Середній (Medium)	Помірний вплив, потребує уваги
7.0 – 8.9	Високий (High)	Значний вплив, пріоритетне усунення
9.0 – 10.0	Критичний (Critical)	Критичний вплив, негайне усунення

3.1.2 Тестове середовище

В якості основи для розгортання і використання, як інстурментів так і середовища тестування, було використано операційно систему Linux, дистрибутив Kali.

Задля охоплення найпоширеніших вразливостей протоколів автентифікації та авторизації, використовуватиметься комбінований підхід до середовища тестування. Використовую вразливі середовища для веб застосунків – OWASP Juice Shop (повноцінний веб-ресурс розгорнутий в контейнері), PortSwigger Web Security Academy (зовнішнє публічне лабораторне середовище) та середовища для мобільних застосунків – InsecureBankv2 (мобільний затосунок в форматі файлу .apk), DIVA (мобільний затосунок в форматі файлу .apk).

Також більш детально розглянемо обрані рішення.

1. OWASP Juice Shop є умисно вразливим веб-застосунок, розробленим Open Web Application Security Project (OWASP) для навчальних цілей та практичного тестування безпеки. Застосунок імітує сучасний інтернет-магазин із повноцінним функціоналом електронної комерції та містить понад 100 задокументованих вразливостей різного рівня складності. Архітектура Juice Shop базується на сучасних технологіях: інтерактивну частину додатку реалізовано на Angular, серверна на Node.js з Express.js, а для зберігання даних використовується база даних SQLite. Така архітектура репрезентує типовий сучасний веб-застосунок, що робить результати тестування релевантними для реальних систем. З точки зору механізмів безпеки, Juice Shop реалізує автентифікацію на основі JWT, сесійне управління, систему ролей користувачів та базовий контроль доступу. Вразливості охоплюють усі категорії OWASP Top 10, включаючи проблеми автентифікації та авторизації [71].

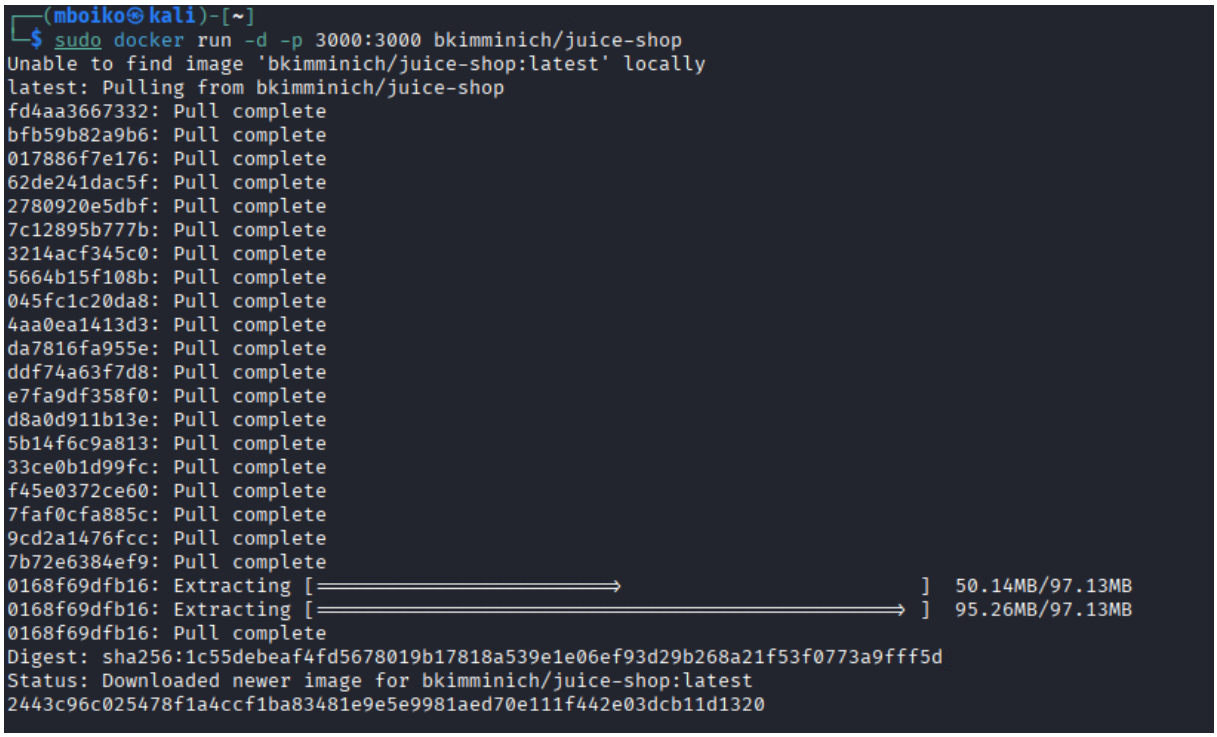

Рис. 3.1 Процес встановлення Juice Shop

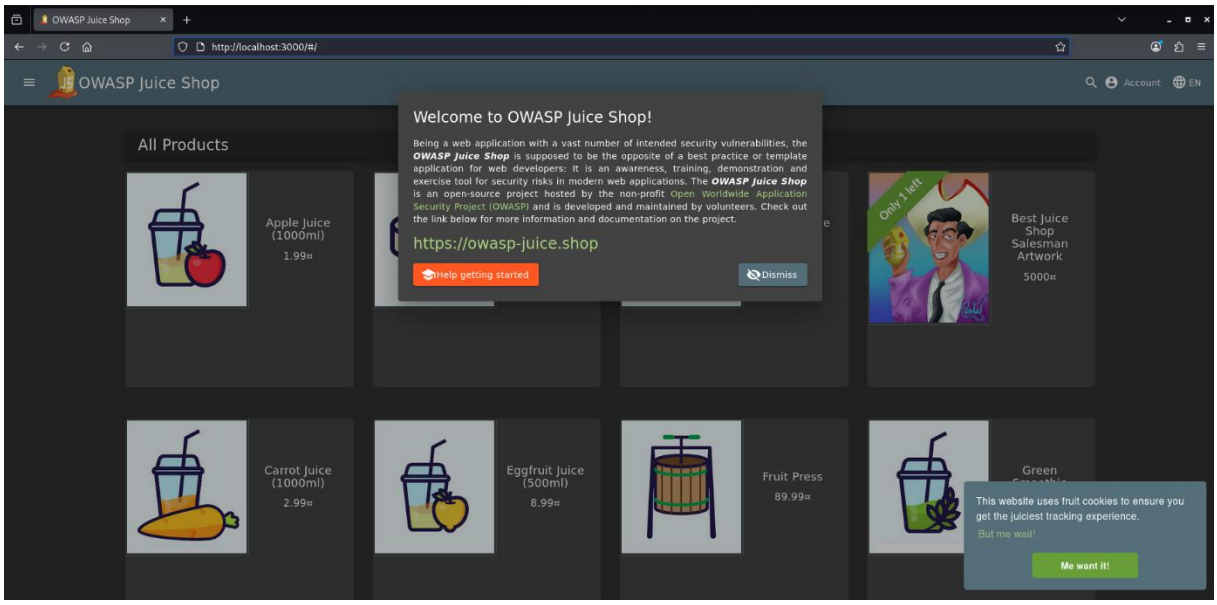

Рис. 3.2 Інтерфейс застосунку Juice Shop

2. PortSwigger Web Security Academy PortSwigger Web Security є безкоштовною онлайн-платформою для навчання веб-безпеці, створена компанією PortSwigger. Платформа містить понад 250 інтерактивних лабораторних робіт, згрупованих за категоріями вразливостей. Ключовою

перевагою PortSwigger Labs є ізольованість кожної лабораторії: кожне середовище створюється індивідуально для користувача, містить конкретну вразливість та має чіткі критерії успішного виконання. Це дозволяє зосередитися на конкретному типі атаки без впливу сторонніх факторів. Кожна лабораторія має рівень складності (Apprentice, Practitioner, Expert), що дозволяє систематизувати дослідження від базових до складних векторів атак [70, 74].

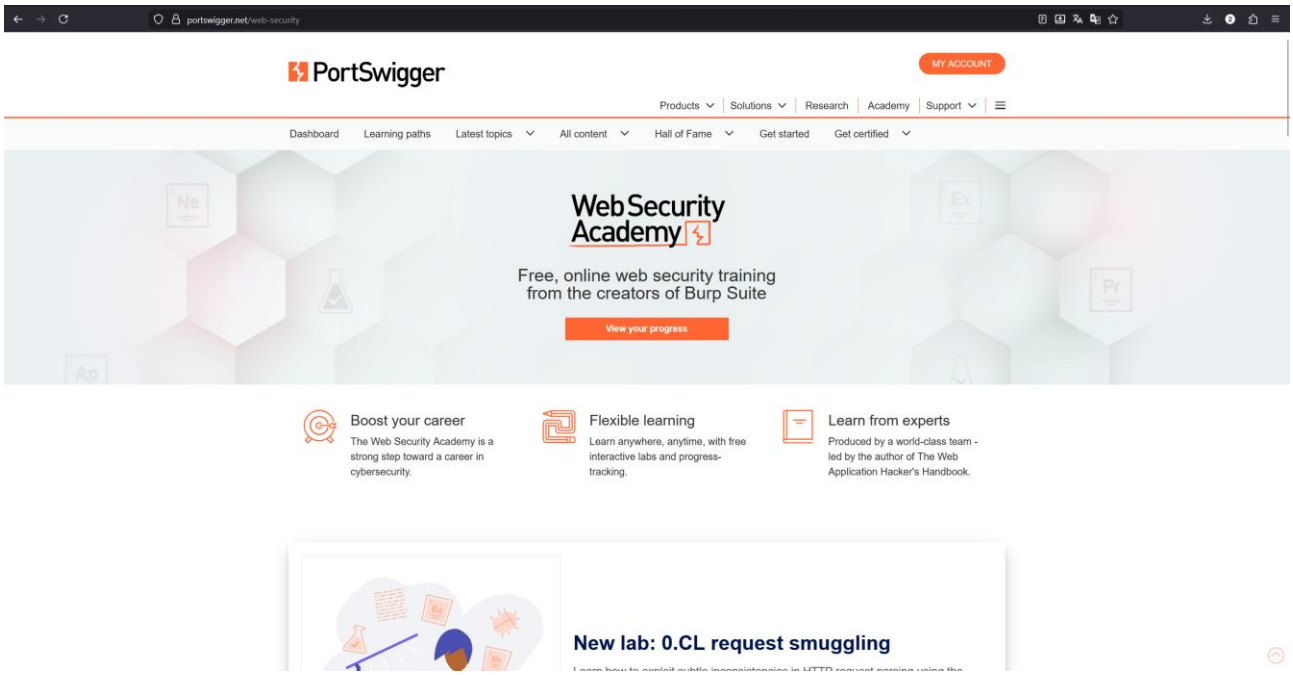

Рис. 3.3 Головна сторінка PortSwigger Web Academy

3. DIVA (Damn Isecure and Vulnerable App) є навмисно вразливим Android-застосунок, розробленим компанією Payatu для навчання мобільній безпеці. Застосунок містить 13 категорій вразливостей, серед яких релевантними для дослідження є проблеми зберігання облікових даних, записані в коді чутливі до розкриття дані та обхід локальної автентифікації. DIVA має просту архітектуру нативного Android-застосунку, що дає змогу виконати як статичний так і динамічний аналіз [72].

4. InsecureBankv2 є більш комплексним тестовим середовищем, яке імітує мобільний банківський застосунок. Проєкт включає як Android клієнтську частину, так і Python серверну частину. Застосунок реалізує повний

цикл автентифікації та авторизації - вхід за логіном та паролем, управління сесіями, переказ коштів з використанням авторизацією, а також інтеграцію з подібними до OAuth механізмами [73].

3.1.3 Інструменти тестування

Використання Kali Linux дозвололило отримати частину інструментів одразу предвстановленими в системі. Іншу частину встановлюю окремо [75, 76, 77]:

1. MobSF (Mobile Security Framework) – рішення для дослідження мобільних застосунків та проведенння широкого аналізу (приватності, безпеки. щкідливості).
2. Android Studio
3. Jwt_tool
4. Objection
5. Jadx

3.1.4 Критерії оцінки та класифікації вразливостей

Для систематизації результатів дослідження застосовано стандартизовані методології оцінки вразливостей.

1. CVSS. Common Vulnerability Scoring System – це стандарт для оцінки серйозності вразливості. Кожна виявлена вразливість оцінюється за шкалою CVSS v3.1, яка враховує базові метрики (вектор атаки, складність, необхідні привілеї, вплив на конфіденційність, цілісність та доступність) та надає числову оцінку критичності від 0 до 10.
2. Класифікація за OWASP. Вразливості також класифікуються відповідно до OWASP Top 10 (для веб-застосунків) та OWASP Mobile Top 10

(для мобільних додатків), що дозволяє співвіднести результати дослідження з галузевими стандартами. Релевантні категорії OWASP Top 10 2021:

– A01:2021 – Broken Access Control;
– A02:2021 – Cryptographic Failures;
– A07:2021 – Identification and Authentication Failures.

Релевантні категорії OWASP Mobile Top 10 2024:

– M3 – Insecure Authentication/Authorization;
– M5 – Insecure Communication;
– M9 – Insecure Data Storage.

3. Структура опису вразливості

Кожна досліджена вразливість описується за уніфікованою структурою:

1) Назва та ідентифікатор – унікальна назва вразливості та посилання на CWE (Common Weakness Enumeration).

2) Опис – технічний опис природи вразливості та причин її виникнення.

3) Тестове середовище – вказівка на конкретну лабораторію або застосунок, де проводилось тестування.

4) Методика експлуатації – покрокова інструкція відтворення атаки з прикладами запитів та відповідей.

5) Докази (Evidence) – скріншоти, фрагменти коду, HTTP-запити та відповіді, що підтверджують наявність вразливості.

6) Оцінка критичності – CVSS-оцінка та обґрунтування.

7) Рекомендації – заходи щодо усунення вразливості.

4. Методика проведення тестування.

Тестування проводиться за методологією OWASP Testing Guide v4.2 з адаптацією до специфіки дослідження протоколів автентифікації та авторизації. Загальний процес складається з етапів: розвідка (reconnaissance), аналіз механізмів безпеки, ідентифікація потенційних вразливостей, експлуатація та документування результатів. Для мобільних застосунків додатково

застосовується методологія OWASP MSTG (Mobile Security Testing Guide), яка визначає специфічні техніки статичного та динамічного аналізу мобільних додатків [77, 78].

3.2 Вразливості автентифікації у веб-застосунках

3.2.1 Атаки на парольну автентифікацію

User enumeration (перебір користувачі) – це вразливість, яка дозволяє зловмиснику визначити, чи існує обліковий запис з певним ім’ям в системі. Ця інформація спрощує атаки, оскільки дозволяє зосередити зусилля на відомих користувачах. В якості тестового середовища використовую PortSwigger Lab “Username enumeration via differen responses”. Тестове середовище містить сторінку входу.

Кроки експлутації вразливості:

1. Запускаю лабороторії на порталі PortSwigger Web Security Academy.
2. Запускаю Burp Suite в Kali Linux.
3. Налаштовую проксі в браузері на адресу 127.0.0.1:8080

4. Переходжу на сторінку входу в лабораторне середовище та ввожу тестові дані, наприклад ім’я користувача “test” та пароль “test”

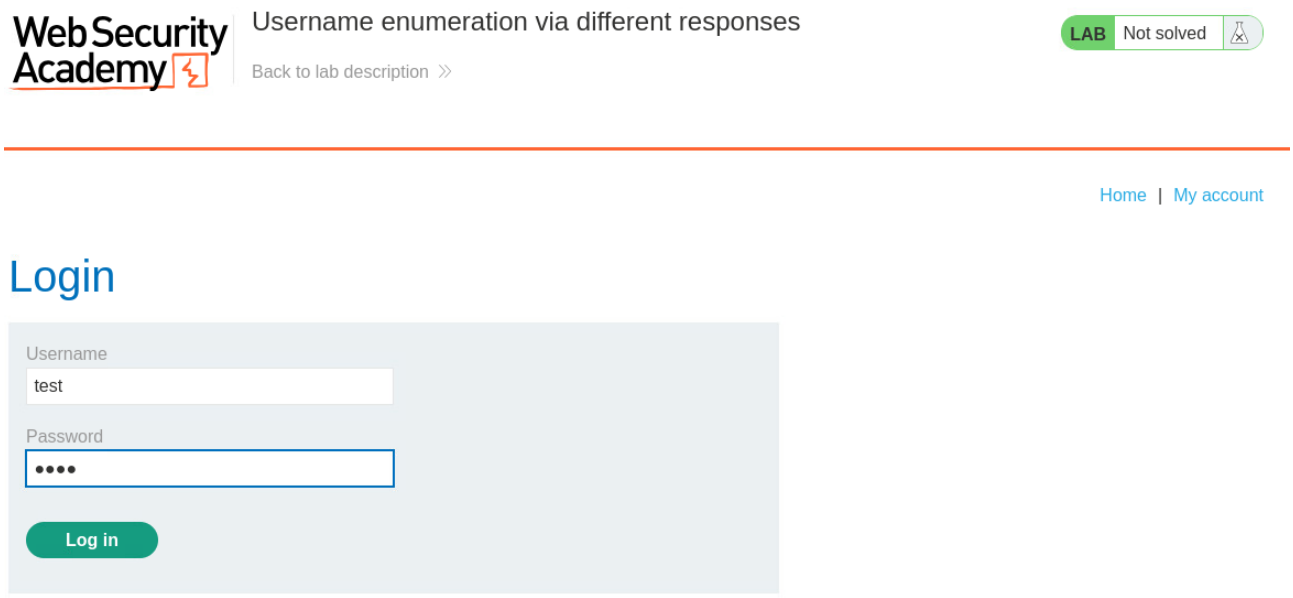

Рис. 3.4 Форма логіну веб-застосунку

5. Переходжу на вкладку Proxy в Burpsuite, а потім вкладку HTTP History та знаходжу POST-запит до кінцевої точки “/login” відправляю запит до частини Intruder. Отримую виконаний запит та замінюю значення в запиті на список користувачів.

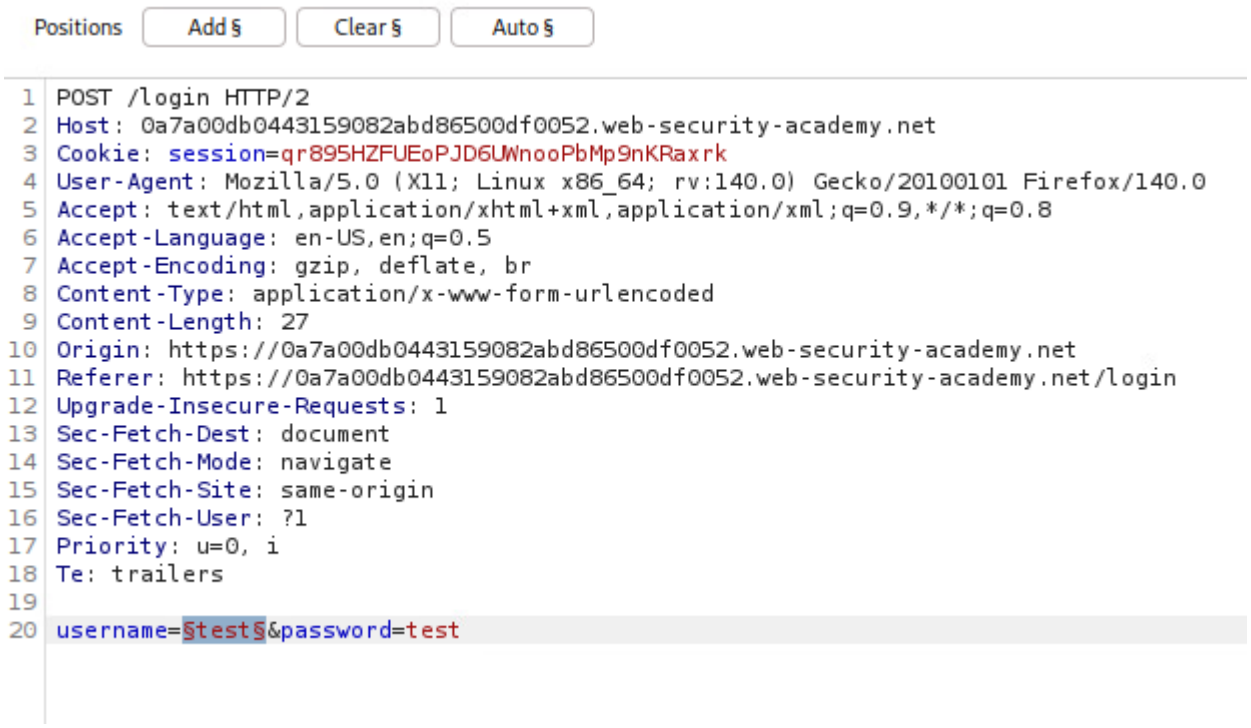

Рис. 3.5 Операції над запитом в Intruder Burp Suite

6. Натискаю кнопку “Start attack”. Burp Suite відправляє запит з кожним ім’ям зі словника.

7. За зміною значення в відповіді з “Invalid Username” на “Invalid password”, визначаю користувача , як існуючого в веб-ресурсі.

3.2.2 Тестування обходу багатофакторної автентифікації

Обхід 2FA через пряму навігацію є можливим при реалізації автентифікації з логічною вразливістю – після успішного введення логіна та пароля сесія вже частково автентифікована, і перевірку другого фактора можна обійти прямим переходом на захищену сторінку. В якості тестового середовища використовую PortSwigger Lab “2FA simple bypass”.

Кроки есплуатації вразливості:

1. Запустив лабораторію та виконав вхід обліковим записом “wiener” з паролем “peter” для перевірки нормального потоку автентифікації.

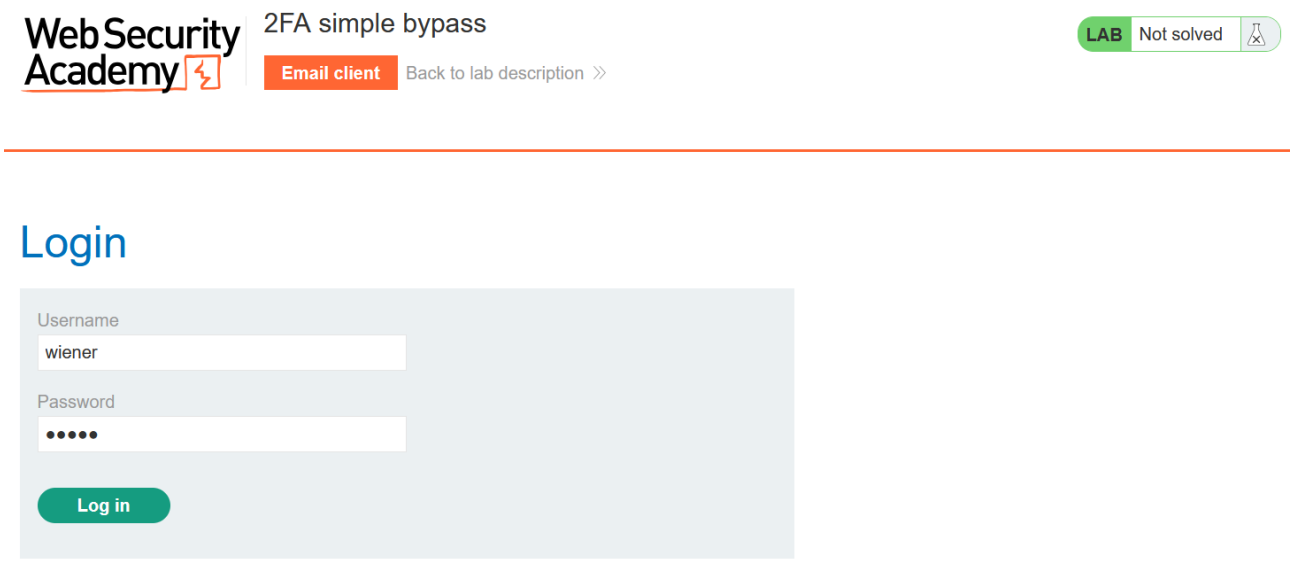

Рис. 3.6 Логін форма веб-застосунку з MFA

2. Після введення логіна та пароля система перенаправляє на сторінку “/login2” для введення 4-значного коду верифікації.

3. Відкрив поштовий клієнт та отримав код верифікації. Ввів код та успішно завершив автентифікацію. Звертаю увагу на URL після входу – "/my-account"

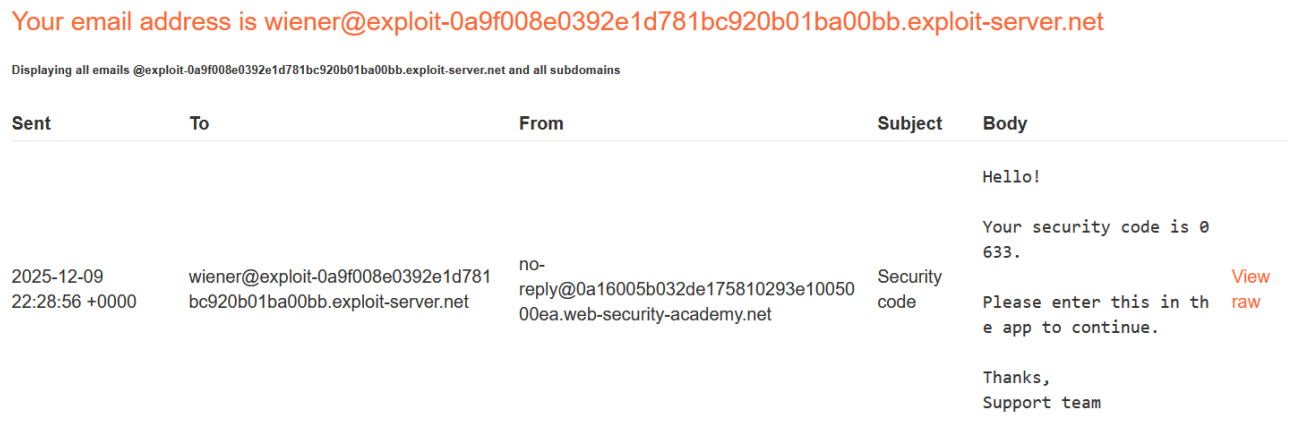

Рис. 3.7 Повідомлення другого фактору в поштовому клієнті

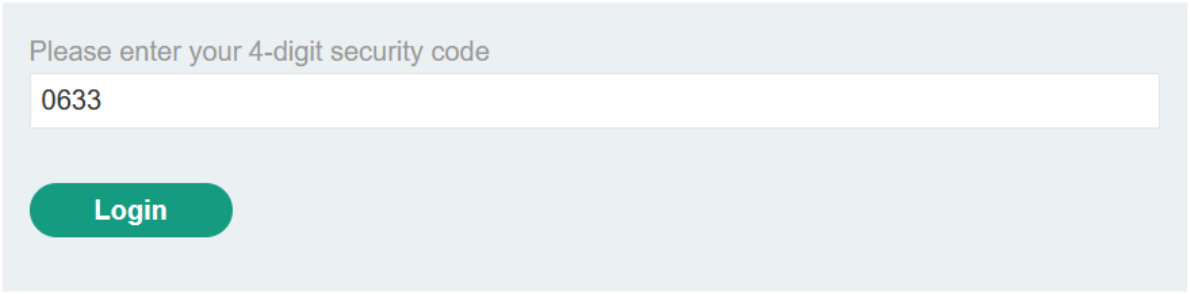

Рис. 3.8 Форма введення коду другого фактору

4. Виходжу з облікового запису та розпочинаю вхід під цільовим обліковим записом “carlos” з паролем “montoya”.

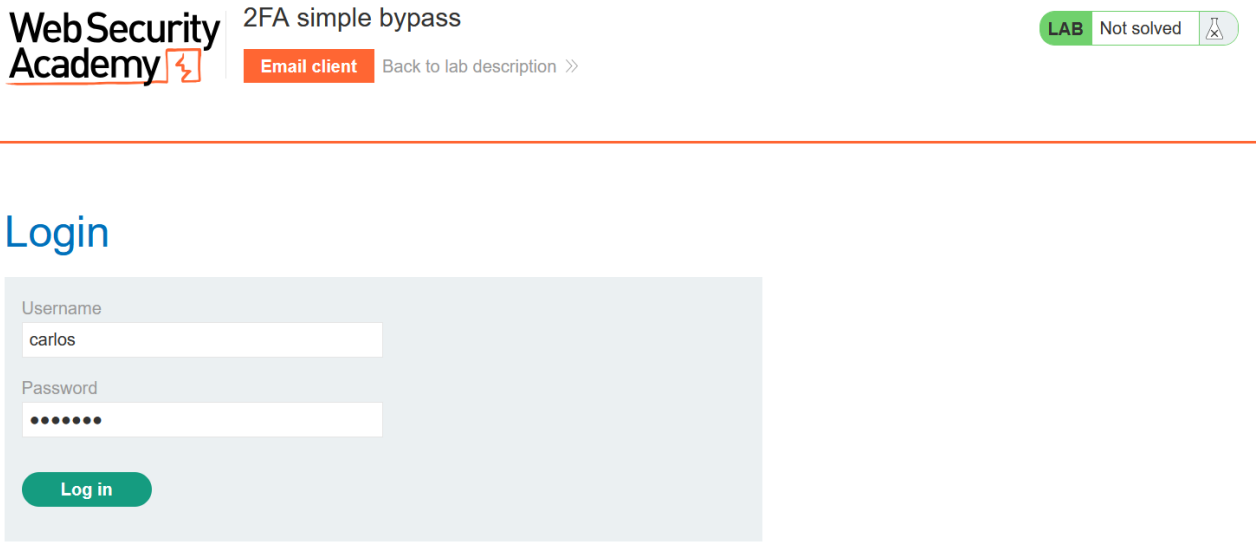

Рис. 3.9 Використання облікових даних

5. Після введення логіна та пароля система перенаправляє на сторінку “/login2”. Код верифікації не вводжу.

6. В адресному рядку браузера вручну змінюю URL з “/login2” на “/my-account” та натискаю Enter. Сервер повертає сторінку облікового запису “carlos” без верифікації другого фактора. Отримано повний доступ до облікового запису, обійшовши 2FA.

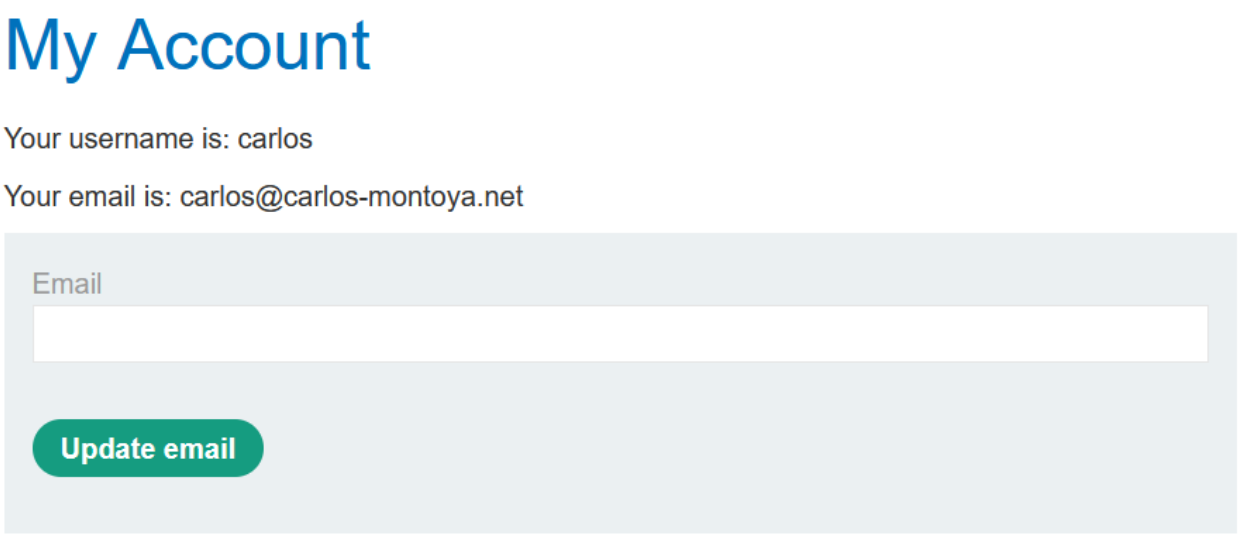

Рис. 3.10 Профіль користувача з MFA

3.2.3 Тестування атаки на JSON Web Token

Неправильна реалізація валідації JSON Web Tokens призводить до критичних вразливостей. Специфікація JWT передбачає алгоритм "none" для випадків, коли токен не потребує підпису. Якщо сервер приймає такі токени, зловмисник може створити довільний токен без знання секретного ключа. В якості тестового середовища використовую PortSwigger Lab “JWT authentication bypass via unverified signature”.

Кроки есплуатації вразливості:

1. Запускаю лабораторію та встановлюю розширення “JWT Editor” у Burp Suite через меню Extensions в “BApp Store”.
2. Виконую вхід під обліковим записом “wiener” з паролем “peter”. Після успішного входу у відповіді сервера встановлено JWT-токен у cookie session.
3. У Burp Suite переходжу до HTTP history та знаходжу запит до “/my-account”.
4. Відправляю його до Repeater.

5. У вкладці Repeater натискаю на вкладку “JSON Web Token” внизу вікна запиту. Розширення автоматично декодує токен та показує його структуру. Структура оригінального JWT-токена складається з трьох частин. Heder містить:

{
"kid": "1f7e708a-6dd4-49ac-b982-ac9633651f29",
"alg": "RS256"
}

Payload містить:

{
"iss": "portswigger",
"exp": 1733754897,
"sub": "wiener"
}

6. У вкладці JWT Editor змінюю значення “alg” в header з “RS256” на “none”.

7. Змінюю значення «sub» в payload з «wiener» на «administrator».

8. Внизу вкладки JWT Editor натиска.ю “Attack» → “Embed unsigned” для створення токена без підпису.

9. Змінюю URL запиту з “/my-account” на “/admin” та відправляю запит. Сервер прийняв токен без підпису та надав доступ до адміністративної панелі.

3.2.4 Тестування управління сесіями

Session fixation – це атака, при якій зловмисник фіксує ідентифікатор сесії жертви до автентифікації, а потім використовує цей відомий ідентифікатор для доступу до автентифікованої сесії після входу жертви.

В якості тестового середовища виступав Juice Shop. Було досліджено поведінку JWT-токенів при автентифікації. Хоча Juice Shop переважно використовує JWT,

концепція вразливості демонструється через механізм кошика, який прив'язаний до сесії. Дослідження показало, що при автентифікації Juice Shop генерує новий JWT-токен, що є правильною практикою. Однак деякі елементи стану користувача (наприклад, вміст кошика) зберігаються окремо та можуть бути вразливими до маніпуляцій.

3.3 Тестування вразливостей авторизації в веб-застосунках

3.3.1 Тестування захищеності контролю доступу

Порушення контролю доступу (Broken Access Control) охоплює широкий спектр вразливостей, де користувач може виконувати дії або отримувати доступ до ресурсів поза межами своїх повноважень. Розрізняють горизонтальну ескалацію (доступ до ресурсів інших користувачів того ж рівня) та вертикальну ескалацію (отримання привілеїв вищого рівня). Вразли вість незахищена адміністративної функціональності виникає, коли адміністративна панель доступна за передбачуваним URL без будь-якої автентифікації або авторизації. Зловмисник може знайти такі приховані сторінки через аналіз файлу "robots.txt", JavaScript-коду або шляхом простого перебору типових шляхів ресурсу. В якості тестового середовища використовую PortSwigger Lab “Unprotected admin functionality”.

Кроки есплуатації вразливості:

1. Переходжу на головну сторінку лабораторії.
2. Перевіряю наявність файлу robots.txt, який використовується для вказівок пошуковим системам. Додаю частину “/robots.txt” до базового URL.
3. Переглянути вміст файлу “robots.txt”. У ньому вказано директиву Disallow, яка показує шлях, який не повинен індексуватися пошуковими системами:

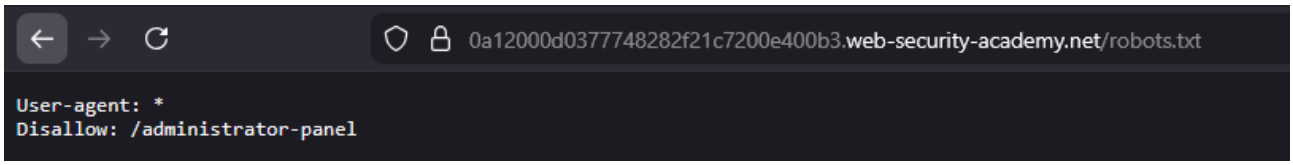

Рис. 3.11 Вміст файлу robots.txt

4. Переходжу за знайденим шляхом адміністративної панелі, додавши його до базового URL.

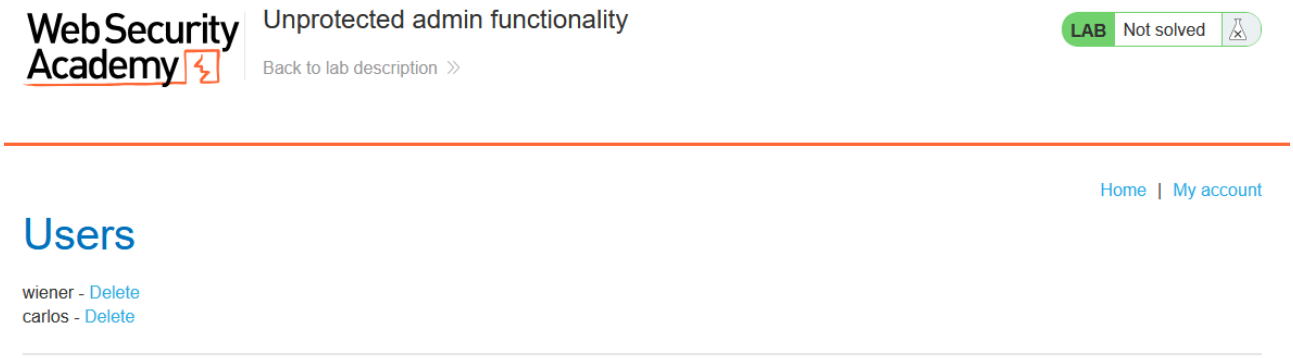

Рис. 3.12 Адміністративна панель ресурсу

5. Адміністративна панель відкриється без запиту автентифікації. На сторінці відображається список користувачів з можливістю їх видалення. Можливість видаляти користувачів підтверджує критичність вразливості.

3.3.2 Тестування захищеності OAuth2

Вразливість неявного потоку – неявний потік OAuth (Implicit Flow) передає токен доступу безпосередньо у фрагменті URL, що робить його вразливим до перехоплення. Додатково, якщо клієнтський застосунок неправильно валідує дані з токена, зловмисник може підмінити інформацію про користувача. Тестовим середовищем є PortSwigger Lab “Authentication bypass via OAuth implicit flow”.

Покрокова методика експлуатації:

1. Відкриваю лабораторію. Натискаю “My account” для переходу до сторінки входу.

2. Обираю опцію входу через соціальну мережу – “Login with social media”. Це ініціює OAuth-потік.

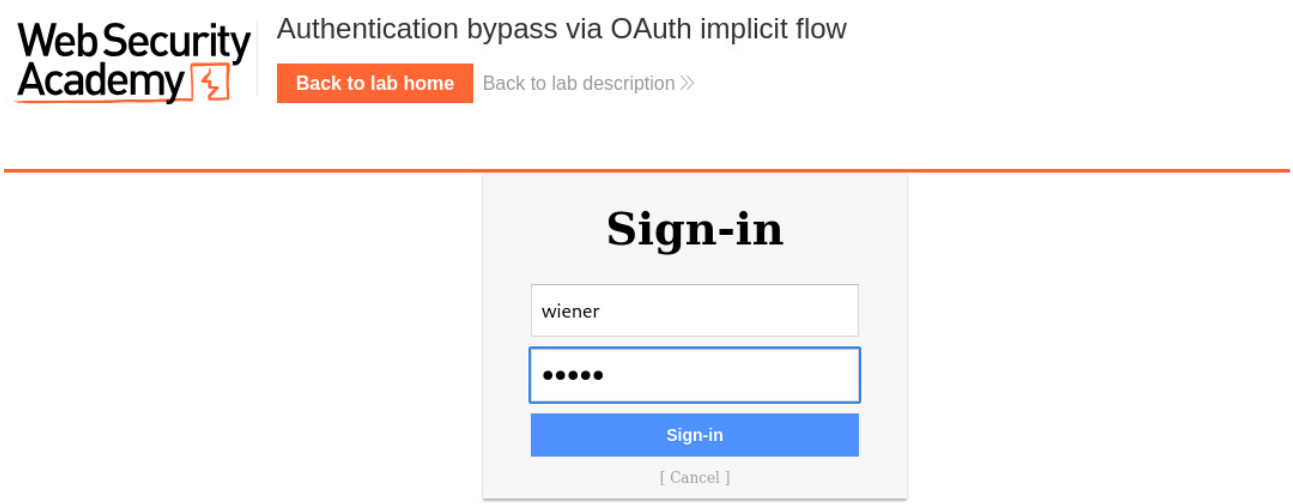

Рис. 3.13 Логін форма для автентифікації з використанням стороніх сервісів

3. На сторінці OAuth-провайдера заходжу під наданим ім'ям користувача wiener та паролем peter та підтверджую надання доступу застосунку.

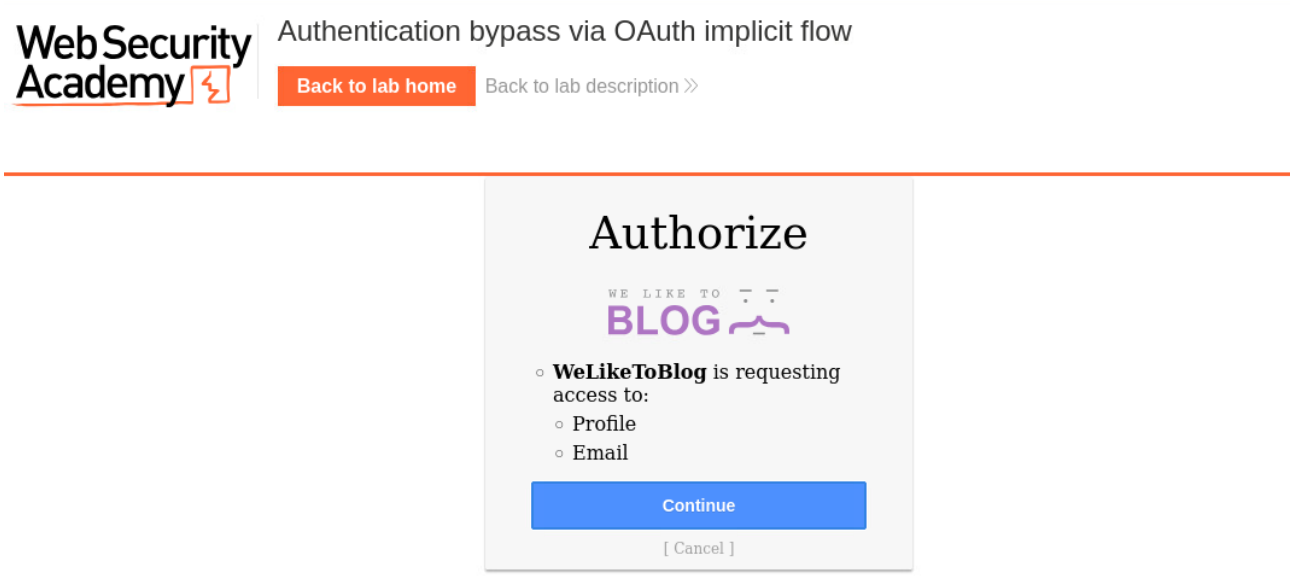

Рис. 3.14 Підтвердження автентифікації

4. Після успішної авторизації система перенаправляє назад до застосунку. У Burp Suite переходжу до вкладки Proxy та HTTP history та знаходжу POST-запит до кінцевої точки “/authenticate”:

POST /authenticate HTTP/2

Host: TARGET-LAB-ID.web-security-academy.net

Content-Type: application/json

{"email":"wiener@hotdog.com","username":"wiener","token":"..."}

5. Аналізую структуру запиту. Клієнт надсилає email, username та access_token серверу. Це означає, що сервер довіряє даним, які надходять від клієнта.

Рис. 3.15 Структура запиту

6. Натискаю правою кнопкою миші на запит з POST “/authenticate” та обираю “Send to Repeater”.

7. У вкладці Repeater змінюю значення email на “carlos@carlos-montoya.net”, а username на “carlos”. Токен залишаю без змін:

{"email":"carlos@carlos-montoya.net","username":"carlos","token":"..."}

8. Натискаю “Send” для відправки модифікованого запиту. Сервер повертає успішну відповідь з встановленням сесійної cookie.

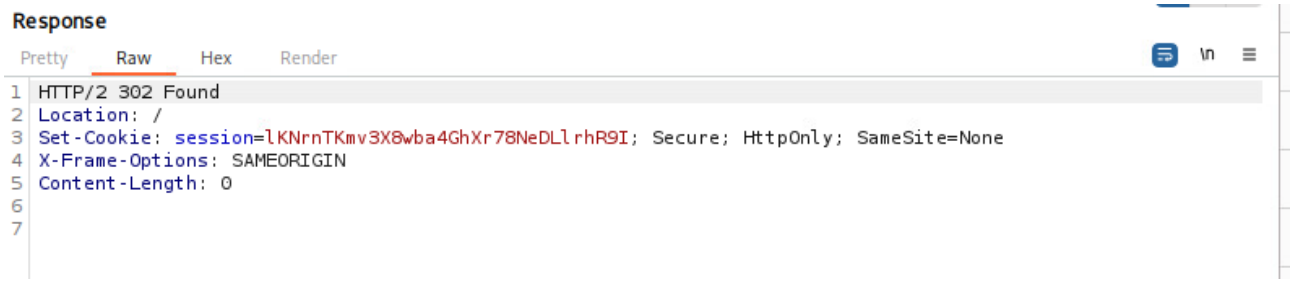

Рис. 3.16 Відповідь веб-ресурсу

9. Натискаю правою кнопкою миші на відповіді та обираю “Show response in browser”. Копіюю згенерований URL та відкриваю його в браузері.

10. Переходжу на сторінку “My account” – система відобразить обліковий запис carlos, підтверджуючи успішний вхід. Таким чином, отримано

доступ до облікового запису carlos без знання його пароля. Сервер не перевіряє, чи відповідають email та username інформації, закодованій в access_token.

3.4 Тестування вразливостей автентифікації в мобільних застосунках

Hardcoded credentials – це облікові дані, API-ключі або секрети, залишені розробниками безпосередньо в коді застосунку. Такі дані легко витягуються через декомпіляцію APK. Аналіз конфігурації зберігання даних здійснюю в тестовому середовищі DIVA за допомогою інструменту MobSF.

Кроки перевірки вразливості:

1. Завантажую APK-файл DIVA з репозиторію GitHub: https://github.com/0xArab/diva-apk-file/blob/main/DivaApplication.apk

2. Запускаю MobSF та відкриваю веб-інтерфейс за адресою http://localhost:8000. Перетягую APK-файл у вікно браузера або натиснути “Upload & Analyze”.

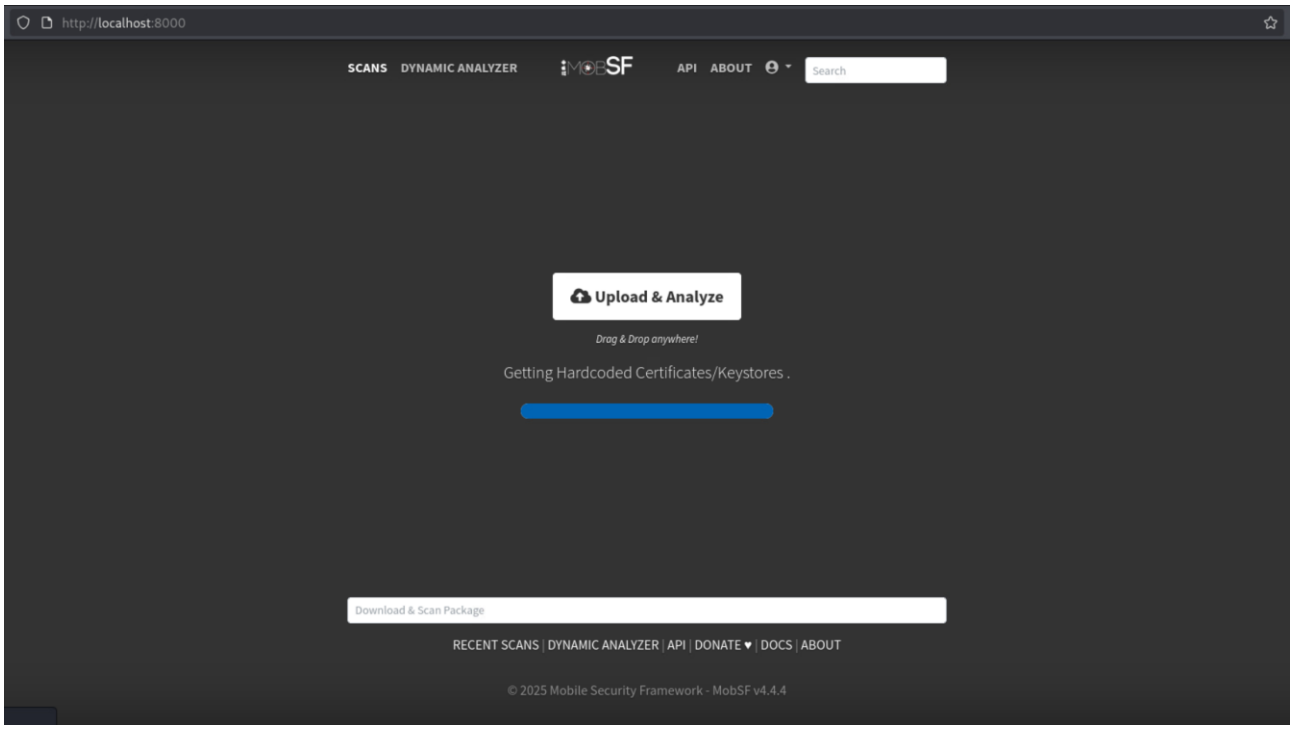

Рис. 3.17 Процес аналізу APK-файлу

3. Чекаю завершення аналізу. MobSF автоматично виконує декомпіляцію та аналіз коду.

4. У результатах аналізу MobSF переходжу до розділу “Reconnaissance” та “Hardcoded Secrets”.

5. Отримую список виявлених секретів. MobSF автоматично знаходить патерни, схожі на паролі, ключі та токени. Виявлено ключ доступу “vendorsecretkey” безпосередньо в коді застосунку.

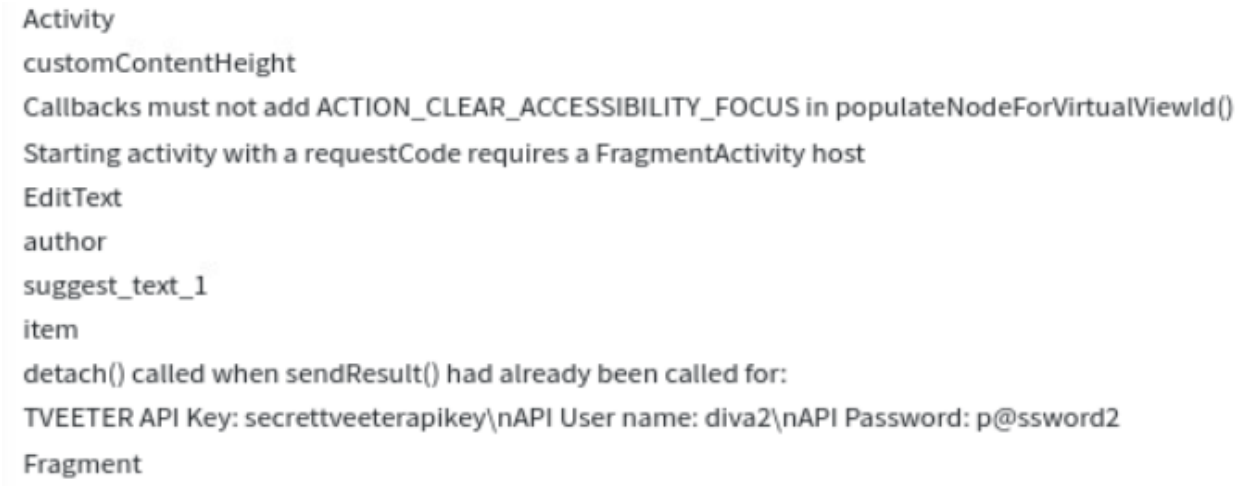

Рис. 3.18 Приклад конфіденційних даних API в коді

3.5 Тестування вразливостей авторизації в мобільних застосунках

Content Provider – компонент Android, що надає структурований доступ до даних застосунку (бази даних, файли) іншим застосункам. Аналіз здійснюю в тестовому середовищі InsecureBankv2 за допомогою інструменту MobSF

Кроки перевірки вразливості:

1. Завантажую InsecureBankv2 APK: https://github.com/dineshshetty/Android-InsecureBankv2/blob/master/InsecureBankv2.apk

2. Завантажую APK до MobSF для аналізу.

3. У розділі “Code Analysis” перевіряю підрозділ “Providers”. Вихідний код “TrackUserContentProvider.java” демонструє, що Content Provider надає доступ до бази даних з іменами користувачів без жодних повноважень,

тобто будь-який застосунок на пристрої Anroid може виконати запит та отримати список користувачів системи без авторизації.

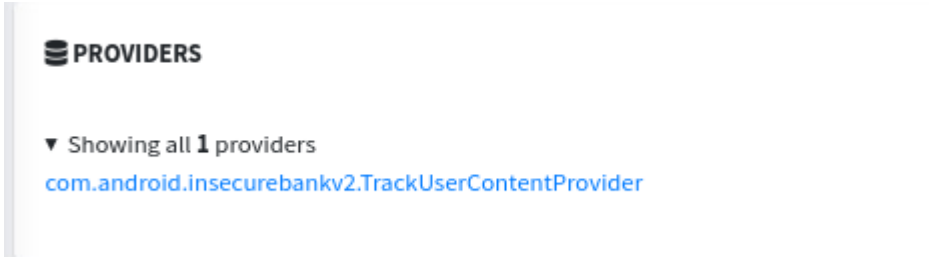

Рис. 3.19 Звіт MobSF

```
/* loaded from: classes.dex */
public class TrackUserContentProvider extends ContentProvider {
    static final String CREATE_DB_TABLE = " CREATE TABLE names (id INTEGER PRIMARY KEY AUTOINCREMENT,  name TEXT NOT NULL);";
    static final String DATABASE_NAME = "mydb";
    static final int DATABASE_VERSION = 1;
    static final String PROVIDER_NAME = "com.android.insecurebankv2.TrackUserContentProvider";
    static final String TABLE_NAME = "names";
    static final String name = "name";
    static final int uriCode = 1;
    private static HashMap<String, String> values;
    private SQLiteDatabase db;
    static final String URL = "content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers";
    static final Uri CONTENT_URI = Uri.parse(URL);
    static final UriMatcher uriMatcher = new UriMatcher(-1);

    static {
        uriMatcher.addURI(PROVIDER_NAME, "trackerusers", 1);
        uriMatcher.addURI(PROVIDER_NAME, "trackerusers/*", 1);
    }
```

Рис. 3.20 Вихідний код TrackUserContentProvider.java

3.6 Результати оцінки

У процесі дослідження проаналізовано та верифіковано вісім типових вразливостей протоколів автентифікації та авторизації. Вибірка охоплює різні категорії вразливостей та платформи (веб-застосунки, мобільні застосунки), що дозволяє сформувати цілісне уявлення про стан безпеки сучасних механізмів контролю доступу.

Таблиця 3.2

Систематизований зріз досліджених вразливостей

Вразливість	**Категорія**	**CVSS**	**Рівень**	**Платформа**
Username enumeration	Автентифікація	5.3	Medium	Веб

Продовження таблиці 3.2

Вразливість	**Категорія**	**CVSS**	**Рівень**	**Платформа**
Обхід 2FA	Автентифікація	8.1	High	Веб
JWT none algorithm	Автентифікація	9.8	Critical	Веб
Session fixation	Автентифікація	6.5	Medium	Веб
Незахищена адмін-панель	Авторизація	9.8	Critical	Веб
OAuth implicit flow	Авторизація	9.8	Critical	Веб
Hardcoded credentials	Автентифікація	7.5	High	Мобільні
Незахищений Content Provider	Авторизація	6.5	Medium	Мобільні

Проведений аналіз дозволяє виокремити характерні особливості вразливостей для веб та мобільних платформ. Попри спільну природу проблем автентифікації та авторизації, специфіка кожної платформи зумовлює відмінності у векторах атак та методах їх реалізації.

На підставі проведеного дослідження сформульовано рекомендації щодо підвищення рівня захищеності механізмів автентифікації та авторизації.

Рекомендації для веб-застосунків:

– забезпечити уніфікацію повідомлень про помилки автентифікації з метою унеможливлення перебору ідентифікаторів користувачів;

– реалізувати багатофакторну автентифікацію як обов'язковий етап з відповідним прапорцем валідації на рівні серверної сесії;

– імплементувати білий список дозволених алгоритмів JWT з явною забороною алгоритму «none»;

– забезпечити генерацію нового ідентифікатора сесії після успішної автентифікації;

– використовувати Authorization Code Flow з PKCE замість Implicit Flow при реалізації OAuth 2.0.

Рекомендації для мобільних застосунків:

– виключити зберігання секретних даних у вихідному коді застосунку;

– застосовувати механізми обфускації коду (ProGuard, R8);

– використовувати Android Keystore для зберігання криптографічних ключів;

– застосовувати EncryptedSharedPreferences для зберігання чутливих даних.

Проведене дослідження дозволило систематизувати знання про типові вразливості протоколів автентифікації та авторизації у веб та мобільних застосунках. Верифікація вразливостей на спеціалізованих навчальних платформах підтвердила їх практичну реалізованість та потенційний вплив на безпеку застосунків.

Встановлено, що більшість досліджених вразливостей (62,5%) належать до категорій високого та критичного рівня небезпеки за шкалою CVSS v3.1. Середньозважена оцінка критичності становить 7,8 бала, що підтверджує значущість проблематики безпеки механізмів контролю доступу.

Порівняльний аналіз веб та мобільних платформ виявив відмінності у характері типових вразливостей: для веб-застосунків характерні логічні помилки в реалізації протоколів, тоді як для мобільних — проблеми зберігання даних та захисту мережевого каналу.

Сформульовані рекомендації можуть бути використані розробниками та аудиторами безпеки для підвищення рівня захищеності застосунків на етапах проєктування, розробки та тестування.

ВИСНОВКИ

Проведено аналіз сучасного стану безпеки протоколів автентифікації та авторизації у веб та мобільних застосунках. Виявлено основні причини виникнення вразливостей у механізмах контролю доступу та визначено ключові ризики, що можуть виникати внаслідок їх експлуатації зловмисниками.

Здійснено систематизацію теоретичних засад автентифікації та авторизації. Проаналізовано базові концепції, принципи функціонування та технічні особливості основних механізмів: парольної автентифікації, багатофакторної автентифікації, JWT-токенів, протоколу OAuth 2.0 та управління сесіями. Визначено типові помилки реалізації кожного з механізмів.

Виконано категоризацію найбільш поширених вразливостей за типом механізму та платформою. Встановлено, що для веб-застосунків характерні логічні помилки в реалізації протоколів автентифікації та авторизації, тоді як для мобільних застосунків – проблеми небезпечного зберігання даних та недостатнього захисту мережевого каналу.

Визначено методи виявлення та верифікації типових вразливостей протоколів автентифікації та авторизації. На основі аналізу нормативних документів (OWASP Testing Guide) та практичної верифікації на навчальних платформах сформовано набір інструментів для оцінки захищеності – Burp Suite для веб-застосунків, MobSF та jadx для мобільних застосунків.

Проведено оглядово-аналітичне дослідження вісьмох типових вразливостей на спеціалізованих навчальних середовищах (PortSwigger Web Security Academy, OWASP Juice Shop, DIVA, InsecureBankv2). Серед досліджених вразливостей: перебір імен користувачів, обхід двофакторної автентифікації, експлуатація алгоритму “none” у JWT, фіксація сесії, незахищена адміністративна функціональність, вразливість неявного потоку OAuth, наявність облікових даних у вихідному коді та відсутність SSL Pinning.

Здійснено оцінку критичності досліджених вразливостей за стандартом CVSS v3.1. Встановлено, що 62,5% вразливостей належать до категорій високого

та критичного рівня небезпеки. Середньозважена оцінка становить 7,8 бала, що підтверджує значущість проблематики безпеки механізмів контролю доступу.

Сформульовано науково-практичні рекомендації щодо підвищення рівня захищеності протоколів автентифікації та авторизації. Для веб-застосунків рекомендовано: уніфікацію повідомлень про помилки, обов'язкову валідацію етапів багатофакторної автентифікації, заборону алгоритму “none” у JWT, регенерацію ідентифікаторів сесії та використання Authorization Code Flow з PKCE для OAuth 2.0. Для мобільних застосунків – виключення секретів із коду, застосування обфускації, реалізацію SSL Pinning та використання захищених сховищ.

Потенційними напрямками подальших досліджень є: апробація запропонованих методів оцінки захищеності на реальних корпоративних застосунках; розширення дослідження на платформу iOS; розробка автоматизованих інструментів для комплексної оцінки безпеки механізмів автентифікації та авторизації; дослідження новітніх методів автентифікації (passkeys, WebAuthn) та аналіз їх потенційних вразливостей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OWASP. Identification and Authentication Failures. URL: https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/.
2. Madhu. M3: Insecure Authentication & Authorization - OWASP Mobile Top 10 2024. URL: https://medium.com/@madhuhack01/m3-insecure-authentication-authorization-owasp-mobile-top-10-2024-510a2d0f419e.
3. Indusface. OWASP Mobile Top 10 - 2024. URL: https://www.indusface.com/blog/owasp-mobile-top-10-2024/.
4. OWASP. OWASP Project for API Security. URL: https://owasp.org/www-project-api-security/.
5. OWASP. Authentication Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html.
6. Identity Theft Resource Center. 2024 Annual Data Breach Report: Near Record Compromises. URL: https://www.idtheftcenter.org/post/2024-annual-data-breach-report-near-record-compromises/.
7. HIPAA Journal. 1.7 Billion Individuals' Data Compromised in 2024. URL: https://www.hipaajournal.com/1-7-billion-individuals-data-compromised-2024/.
8. Secureframe. Data Breach Statistics. URL: https://secureframe.com/blog/data-breach-statistics.
9. Varonis. Data Breach Statistics. URL: https://www.varonis.com/blog/data-breach-statistics.
10. Cyber Wyoming. Insights from the ITRC 2024 Data Breach Report: MFA. URL: https://cyberwyoming.org/insights-from-the-itrc-2024-data-breach-report-mfa/.

11. IBM. 2024 Roundup: Top Data Breach Stories and Industry Trends. URL: https://www.ibm.com/think/insights/2024-roundup-top-data-breach-stories-and-industry-trends.
12. Cloudflare. OWASP Top 10. URL: https://www.cloudflare.com/learning/security/threats/owasp-top-10/.
13. OWASP. Insecure Authentication Methods and Default Credentials. URL: https://owasp.org/www-project-top-10-infrastructure-security-risks/docs/2024/ISR07_2024-Insecure_Authentication_Methods_and_Default_Credentials.
14. Verizon. Data Breach Investigations Report. URL: https://www.verizon.com/business/resources/reports/dbir/.
15. OWASP. Password Storage Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html.
16. P-H-C. Argon2. URL: https://github.com/P-H-C/phc-winner-argon2.
17. IETF. RFC 6238: Algorithm for OTP Generation. URL: https://tools.ietf.org/html/rfc6238.
18. W3C. Web Authentication: An API for Accessing Public Key Credentials. URL: https://www.w3.org/TR/webauthn-2/.
19. FIDO Alliance. FIDO Specifications. URL: https://fidoalliance.org/specifications/.
20. FIDO Alliance. FIDO Passkeys. URL: https://fidoalliance.org/passkeys/.
21. Apple. Welcome to the Security Guide. URL: https://support.apple.com/guide/security/welcome/web.
22. Android Developers. Biometric Authentication. URL: https://developer.android.com/training/sign-in/biometric-auth.
23. ITU. Recommendation X.509. URL: https://www.itu.int/rec/T-REC-X.509.
24. IETF. RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. URL: https://tools.ietf.org/html/rfc5280.

25. IETF. RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3. URL: https://tools.ietf.org/html/rfc8446.
26. Cloudflare. What is Mutual TLS?. URL: https://www.cloudflare.com/learning/access-management/what-is-mutual-tls/.
27. NIST. NIST SP 800-63B: Digital Identity Guidelines. URL: https://pages.nist.gov/800-63-3/sp800-63b.html.
28. IETF. RFC 6749: The OAuth 2.0 Authorization Framework. URL: https://tools.ietf.org/html/rfc6749.
29. IETF. RFC 7636: Proof Key for Code Exchange by OAuth Public Clients. URL: https://tools.ietf.org/html/rfc7636.
30. IETF. RFC 9700: The OAuth 1.0 Protocol. URL: https://datatracker.ietf.org/doc/rfc9700/.
31. OpenID. OpenID Connect Core 1.0. URL: https://openid.net/specs/openid-connect-core-1_0.html.
32. IETF. RFC 7519: JSON Web Token (JWT). URL: https://tools.ietf.org/html/rfc7519.
33. IETF. RFC 8725: OAuth 2.1. URL: https://tools.ietf.org/html/rfc8725.
34. OWASP. Session Management Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html.
35. OWASP. JSON Web Token for Java Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html.
36. Android Developers. Android Keystore System. URL: https://developer.android.com/training/articles/keystore.
37. Apple. Keychain Services. URL: https://developer.apple.com/documentation/security/keychain_services.
38. NIST. Role-Based Access Control. URL: https://csrc.nist.gov/projects/role-based-access-control.

39. NIST. NIST SP 800-162: Guide to Attribute Based Access Control (ABAC) Definition and Considerations. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-162.pdf.
40. NIST. NIST SP 800-207: Zero Trust Architecture. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf.
41. OWASP. Defense in Depth. URL: https://owasp.org/www-community/Defense_in_Depth.
42. CISA. Principle of Least Privilege. URL: https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/least-privilege.
43. OWASP. Transport Layer Protection Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html.
44. Mobile Security. Mobile Security Testing Guide. URL: https://mobile-security.gitbook.io/mobile-security-testing-guide/.
45. Android Developers. Security Configuration. URL: https://developer.android.com/training/articles/security-config.
46. Apple. Preventing Insecure Network Connections. URL: https://developer.apple.com/documentation/security/preventing_insecure_network_connections.
47. Microsoft. Threat Modeling Tool Threats. URL: https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats.
48. MITRE. MITRE ATT&CK. URL: https://attack.mitre.org/.
49. OWASP. Attack Surface Analysis Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Attack_Surface_Analysis_Cheat_Sheet.html.
50. Akamai. State of the Internet. URL: https://www.akamai.com/resources/state-of-the-internet.
51. NVD. CVE-2023-22515. URL: https://nvd.nist.gov/vuln/detail/CVE-2023-22515.

52. OWASP. Credential Stuffing Prevention Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Credential_Stuffing_Prevention_Cheat_Sheet.html.
53. OWASP. Broken Access Control. URL: https://owasp.org/Top10/A01_2021-Broken_Access_Control/.
54. IBM. Data Breach. URL: https://www.ibm.com/security/data-breach.
55. OWASP API Security. Broken Object Level Authorization. URL: https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/.
56. OWASP. Access Control Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html.
57. Auth0. Critical Vulnerabilities in JSON Web Token Libraries. URL: https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/.
58. NVD. CVE-2015-9235. URL: https://nvd.nist.gov/vuln/detail/CVE-2015-9235.
59. PortSwigger. JSON Web Tokens (JWT). URL: https://portswigger.net/web-security/jwt.
60. HackTricks. Hacking JWT (JSON Web Tokens). URL: https://book.hacktricks.xyz/pentesting-web/hacking-jwt-json-web-tokens.
61. Mobile Security. Mobile Security Testing Guide. URL: https://mobile-security.gitbook.io/mobile-security-testing-guide/.
62. OWASP. OWASP Mobile Application Security Verification Standard (MASVS). URL: https://github.com/OWASP/owasp-masvs.
63. Android Developers. Best Practices for Android Security. URL: https://developer.android.com/topic/security/best-practices.
64. Frida. Frida. URL: https://frida.re/.
65. IETF. RFC 8252: OAuth 2.0 for Native Apps. URL: https://tools.ietf.org/html/rfc8252.

66. Android Developers. App Links. URL: https://developer.android.com/training/app-links.
67. OWASP. OWASP Code Review Guide. URL: https://owasp.org/www-project-code-review-guide/.
68. OWASP. Error Handling Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html.
69. OWASP. Logging Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html.
70. PortSwigger. Web Security. URL: https://portswigger.net/web-security.
71. OWASP. OWASP Juice Shop. URL: https://owasp.org/www-project-juice-shop/.
72. Payatu. DIVA for Android. URL: https://github.com/payatu/diva-android.
73. Shetty D. Android InsecureBankv2. URL: https://github.com/dineshshetty/Android-InsecureBankv2.
74. PortSwigger. Burp Suite. URL: https://portswigger.net/burp.
75. MobSF. Mobile Security Framework (MobSF). URL: https://github.com/MobSF/Mobile-Security-Framework-MobSF.
76. Jadx. jadx. URL: https://github.com/skylot/jadx.
77. FIRST. CVSS v3.1 Specification Document. URL: https://www.first.org/cvss/v3.1/specification-document.
78. MITRE. Common Weakness Enumeration. URL: https://cwe.mitre.org/.