Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра інформаційної та кібернетичної безпеки
імені професора Володимира Бурячка

«Допущено до захисту»

Завідувач кафедри інформаційної та
кібернетичної безпеки імені
професора Володимира Бурячка
кандидат технічних наук, доцент
Складанний П.М.

_____
(підпис)
« ___ » _____ 20__ р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття другого (магістерського)
рівня вищої освіти

Спеціальність 125 Кібербезпека та захист інформації

Тема роботи:

Розробка рекомендацій щодо застосування сучасних методів і засобів ідентифікації і автентифікації користувачів корпоративної мережі.

Виконав

студент групи КБм-1-__-1.4.д

Дубровський Іван Володимирович

(прізвище, ім’я, по батькові)

_____

(підпис)

Науковий керівник

_____

(науковий ступінь, наукове звання)

Аносов А.О.

(прізвище, ініціали)

_____

(підпис)

Київ – 2025

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра інформаційної та кібернетичної безпеки
імені професора Володимира Бурячка

Освітньо-кваліфікаційних рівень – магістр
Спеціальність 125 Кібербезпека та захист інформації
Освітня програма 125.00.01 Безпека інформаційних і комунікаційних систем

«Затверджую»
Завідувач кафедри
інформаційної та кібернетичної
безпеки імені професора
Володимира Бурячка кандидат
технічних наук, доцент
Складанний П.М.

_____
(підпис)
« ___ » _____ 20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Дубровському Івану Володимирову

1. Тема роботи: Розробка рекомендацій щодо застосування сучасних методів і засобів ідентифікації і автентифікації користувачів корпоративної мережі;
2. Керівник к.в.н., доцент Аносов А.О.

   затверджені наказом ректора від «___»____ 20___ року №__.
3. Термін подання студентом роботи «12» грудня 2025 р.
4. Вихідні дані до роботи:
   - 4.1 Теоретичні дані про вимоги законодавства України та документів щодо забезпечення безпеки кіберпростору
   - 4.2 Стандарти TOTP, HOTP, FIDO2, WebAuthn.
5. Зміст текстової частини роботи (перелік питань, які потрібно розробити):
   - 5.1 Провести аналіз сучасних методів та засобів ідентифікації і автентифікації користувачів у корпоративних мережах, дослідити їх переваги та недоліки.
   - 5.2 Виконати огляд існуючих технологій та протоколів багатофакторної автентифікації, включаючи стандарти TOTP, HOTP, FIDO2, WebAuthn.
   - 5.3 Дослідити можливості використання месенджерів як каналу додаткової автентифікації та доставки критичних повідомлень адміністраторам безпеки.
   - 5.4 Розробити вимоги та архітектуру системи багатофакторної автентифікації.
   - 5.5 Сформулювати рекомендації щодо реалізації окремих компонентів системи, включаючи парольну автентифікацію, мобільні аутентифікатори та месенджер-боти.
   - 5.6 Створити систему моніторингу та сповіщення адміністраторів про критичні події безпеки через месенджери.
   - 5.7 Здійснити практичну реалізацію запропонованих рішень у тестовому середовищі.
   - 5.8 Провести тестування функціональності, безпеки та продуктивності розробленої системи.

5.9 Виконати порівняльний аналіз розробленої системи з існуючими рішеннями на ринку.

6. Перелік графічного матеріалу:

6.1 Презентація доповіді, виконана в Microsoft PowerPoint.

7. Дата видачі завдання «___»_____ 20___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів підготовки роботи	Термін виконання	Примітка
1.	Уточнення постановки завдання	02.06.2025	
2.	Пошук літератури	03.06.2025 - 30.06.2025	
3.	Обґрунтування вибору рішення	01.07.2025	
4.	Збір даних	02.07.2025 – 31.07.2025	
5.	Виконання та оформлення розділу 1.	01.08.2025 – 29.08.2025	
6.	Виконання та оформлення розділу 2.	01.09.2025 – 30.09.2025	
7.	Виконання та оформлення розділу 3.	01.10.2025 – 31.10.2025	
8.	Вступ, висновки, реферат	03.11.2025 – 19.11.2025	
9.	Апробація роботи на науково-методичному семінарі та/або науково-технічній конференції	15.05.2025, 26.10.2025	
10.	Оформлення та друк текстової частини роботи	20.11.2025 – 04.12.2025	
11.	Оформлення презентацій	05.12.2025 – 09.12.2025	
12.	Отримання рецензій	10.12.2025	
13.	Попередній захист роботи		
14.	Захист в ЕК		

Студент _____ Дубровський Іван Володимирович
(підпис) (прізвище, ім’я, по батькові)

Науковий керівник _____ Аносов Андрій Олександрович
(підпис) (прізвище, ім’я, по батькові)

РЕФЕРАТ

Магістерська робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 140 сторінок. Список використаних джерел містить 50 найменувань.

Дослідження присвячене розв’язанню актуальної науково-практичної задачі — підвищенню рівня інформаційної безпеки корпоративних мереж шляхом удосконалення методів ідентифікації та автентифікації користувачів, оскільки понад 80% сучасних інцидентів пов’язані з компрометацією облікових даних. Метою роботи є розробка науково обґрунтованих рекомендацій щодо впровадження систем багатофакторної автентифікації (MFA) з максимальною кількістю факторів та інтеграцією з месенджерами для оперативного сповіщення про загрози. Об’єктом дослідження виступають процеси перевірки користувачів у корпоративному середовищі, а предметом — технології та протоколи MFA, такі як TOTP, HOTP, FIDO2 та WebAuthn. У ході дослідження було застосовано комплекс методів, зокрема системний аналіз архітектури систем доступу, криптографічний аналіз стійкості алгоритмів, математичне моделювання та об'єктно-орієнтоване програмування для створення програмного прототипу. Наукова новизна результатів полягає у розробці вперше запропонованої комплексної архітектури, що поєднує традиційні методи захисту з месенджер-ботами як додатковим фактором автентифікації та каналом доставки критичних подій безпеки адміністраторам у реальному часі. Практичне значення роботи підтверджується створенням дієвого інструментарію, який дозволяє знизити ризик несанкціонованого доступу на 99,9%, забезпечити відповідність міжнародним стандартам ISO/IEC 27001 та ISO/IEC 27002, а також мінімізувати час реагування на інциденти. Результати тестування реалізованої системи у тестовому середовищі довели її ефективність порівняно з існуючими ринковими аналогами, що робить запропоновані рішення придатними для впровадження в організаціях різного масштабу та використання у навчальному процесі за спеціальностями з кібербезпеки.

Ключові слова: КІБЕРБЕЗПЕКА, КОРПОРАТИВНІ МЕРЕЖІ, ІДЕНТИФІКАЦІЯ, БАГАТОФАКТОРНА АВТЕНТИФІКАЦІЯ (MFA), КОМПРОМЕТАЦІЯ ОБЛІКОВИХ ДАНИХ, ПРОТОКОЛИ АВТЕНТИФІКАЦІЇ, TOTP, FIDO2, WEBAUTHN, МЕСЕНДЖЕР-БОТИ, МОНІТОРИНГ БЕЗПЕКИ, ZERO TRUST, ІНЦИДЕНТИ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ, ISO/IEC 27001.

ЗМІСТ

ВСТУП

Актуальність теми дослідження

Стрімкий розвиток цифрових технологій та повсюдне впровадження інформаційних систем у всіх сферах людської діяльності призвели до критичної залежності сучасних організацій від корпоративних мереж. Водночас зростання кількості та складності кіберзагроз ставить перед фахівцями інформаційної безпеки все більш серйозні виклики щодо захисту корпоративної інфраструктури від несанкціонованого доступу [1, 2].

Згідно з дослідженням компанії Verizon за 2024 рік, понад 80 відсотків інцидентів безпеки пов'язані з компрометацією облікових записів користувачів [36]. Традиційні методи автентифікації, що базуються виключно на паролях, демонструють недостатню ефективність у протидії сучасним атакам. Це підтверджується даними IBM Security, згідно з якими середня вартість інциденту безпеки, пов'язаного з крадіжкою облікових даних, у 2024 році становила понад 4,5 мільйона доларів США [37].

Багатофакторна автентифікація визнана міжнародними стандартами та дослідниками як один із найефективніших методів підвищення рівня безпеки корпоративних систем. Численні дослідження підтверджують, що впровадження багатофакторної автентифікації знижує ризик компрометації облікових записів на 99,9 відсотка порівняно з використанням лише паролів [5, 8]. Проте існуючі рішення часто характеризуються недостатньою гнучкістю, складністю інтеграції з корпоративною інфраструктурою та обмеженими можливостями моніторингу подій безпеки.

Сучасні організації потребують комплексних рішень, які б поєднували максимальну кількість факторів автентифікації, забезпечували зручність використання для кінцевих користувачів та надавали адміністраторам ефективні інструменти контролю й моніторингу. Особливого значення набуває можливість отримання критичних повідомлень про події безпеки в режимі реального часу через

популярні месенджери, що дозволяє оперативно реагувати на потенційні загрози [18, 19].

Дослідження методів багатофакторної автентифікації активно проводяться науковцями з різних країн. Зокрема, китайські дослідники внесли значний вклад у розвиток теорії та практики захисту інформації в корпоративних мережах [29, 30, 31, 32, 33, 34, 35]. Водночас аналіз наукової літератури свідчить про відсутність комплексних досліджень, що охоплювали б розробку практичних рекомендацій щодо створення систем автентифікації з максимальною кількістю факторів та інтеграцією з сучасними каналами комунікації.

Таким чином, розробка науково обґрунтованих рекомендацій щодо застосування сучасних методів і засобів ідентифікації та автентифікації користувачів корпоративних мереж з акцентом на багатофакторність та інтеграцію з месенджерами є актуальною науковою задачею, що має важливе практичне значення для підвищення рівня інформаційної безпеки організацій.

Мета і задачі дослідження

Метою дослідження є розробка науково обґрунтованих рекомендацій щодо застосування сучасних методів та засобів ідентифікації і автентифікації користувачів корпоративної мережі з максимальною кількістю факторів автентифікації та інтеграцією з месенджерами для підвищення рівня інформаційної безпеки організацій.

Для досягнення поставленої мети необхідно вирішити такі задачі:

1. Провести аналіз сучасних методів та засобів ідентифікації і автентифікації користувачів у корпоративних мережах, дослідити їх переваги та недоліки.
2. Виконати огляд існуючих технологій та протоколів багатофакторної автентифікації, включаючи стандарти TOTP, HOTP, FIDO2, WebAuthn.
3. Дослідити можливості використання месенджерів як каналу додаткової автентифікації та доставки критичних повідомлень адміністраторам безпеки.

4. Розробити вимоги та архітектуру системи багатофакторної автентифікації.

5. Сформулювати рекомендації щодо реалізації окремих компонентів системи, включаючи парольну автентифікацію, мобільні аутентифікатори та месенджер-боти.

6. Створити систему моніторингу та сповіщення адміністраторів про критичні події безпеки через месенджери.

7. Здійснити практичну реалізацію запропонованих рішень у тестовому середовищі.

8. Провести тестування функціональності, безпеки та продуктивності розробленої системи.

9. Виконати порівняльний аналіз розробленої системи з існуючими рішеннями на ринку.

Об'єкт дослідження

Об'єктом дослідження є процеси ідентифікації та автентифікації користувачів у корпоративних мережах.

Предмет дослідження

Предметом дослідження є методи, засоби та технології багатофакторної автентифікації користувачів корпоративних мереж з використанням сучасних стандартів та каналів комунікації.

Методи дослідження

Для вирішення поставлених задач у роботі використовувалися такі методи дослідження:

- методи системного аналізу для дослідження архітектури та компонентів систем автентифікації;
- методи криптографічного аналізу для оцінки стійкості алгоритмів автентифікації;
- методи математичного моделювання для проєктування архітектури системи;

- методи об'єктно-орієнтованого програмування для практичної реалізації системи;
- експериментальні методи для тестування функціональності та продуктивності розробленої системи;
- методи порівняльного аналізу для оцінки ефективності запропонованих рішень.

Наукова новизна отриманих результатів

Наукова новизна результатів дослідження полягає в наступному:

1. Вперше запропоновано комплексну архітектуру системи багатофакторної автентифікації, що інтегрує традиційні методи автентифікації з сучасними месенджерами для забезпечення максимального рівня безпеки та оперативного моніторингу.
2. Запропоновано методику інтеграції месенджер-ботів у процес автентифікації користувачів корпоративної мережі, що дозволяє використовувати різні платформи обміну повідомленнями як додатковий фактор автентифікації.
3. Розроблено систему автоматичної доставки критичних подій безпеки адміністраторам через месенджери, що забезпечує зменшення часу реагування на інциденти інформаційної безпеки.
4. Запропоновано рекомендації щодо практичного впровадження системи багатофакторної автентифікації в корпоративних мережах різного масштабу з урахуванням вимог чинного законодавства України та міжнародних стандартів інформаційної безпеки.

Практичне значення отриманих результатів

Практичне значення результатів дослідження визначається можливістю їх безпосереднього застосування для підвищення рівня інформаційної безпеки корпоративних мереж. Розроблені рекомендації та програмні компоненти можуть бути використані організаціями різного профілю для створення або модернізації систем контролю доступу.

Впровадження запропонованих рішень дозволяє:

- суттєво знизити ризики несанкціонованого доступу до корпоративних ресурсів;
- забезпечити відповідність вимогам міжнародних стандартів інформаційної безпеки ISO/IEC 27001 та ISO/IEC 27002 [39, 40];
- підвищити ефективність моніторингу подій безпеки через автоматичну доставку критичних сповіщень;
- зменшити час реагування адміністраторів на інциденти інформаційної безпеки;
- оптимізувати витрати на впровадження та підтримку системи автентифікації за рахунок використання відкритих стандартів та технологій.

Результати роботи можуть бути використані у навчальному процесі при підготовці фахівців з кібербезпеки та інформаційних технологій.

Структура та обсяг роботи

Магістерська робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 140 сторінок. Список використаних джерел містить 50 найменувань.

У вступі обґрунтовано актуальність теми дослідження, сформульовано мету і задачі роботи, визначено об'єкт та предмет дослідження, описано методи дослідження, наукову новизну та практичне значення отриманих результатів.

У першому розділі проведено аналіз сучасних методів та засобів ідентифікації і автентифікації в корпоративних мережах, розглянуто основні технології багатофакторної автентифікації, досліджено існуючі рішення та можливості використання месенджерів як каналу автентифікації.

У другому розділі розроблено вимоги до системи багатофакторної автентифікації, запропоновано архітектуру системи, сформульовано детальні рекомендації щодо реалізації окремих компонентів, включаючи систему моніторингу та сповіщення адміністраторів.

У третьому розділі описано практичну реалізацію системи багатофакторної автентифікації у тестовому середовищі, наведено результати тестування

функціональності, безпеки та продуктивності системи, виконано порівняльний аналіз з існуючими рішеннями.

У висновках узагальнено результати дослідження, сформульовано основні наукові та практичні результати роботи, визначено перспективи подальших досліджень.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ЗАСОБІВ ІДЕНТИФІКАЦІЇ І АВТЕНТИФІКАЦІЇ В КОРПОРАТИВНИХ МЕРЕЖАХ

1.1. Основні поняття та принципи ідентифікації і автентифікації користувачів

1.1.1. Визначення базових термінів

Забезпечення інформаційної безпеки в корпоративних мережах спирається на послідовний ланцюг процесів контролю доступу: ідентифікацію, автентифікацію та авторизацію. Коректне розмежування цих етапів є необхідною передумовою побудови цілісної системи захисту.

Ідентифікація розглядається як момент, коли користувач заявляє системі свою особу, надаючи унікальний ідентифікатор: ім’я користувача, електронну адресу, номер облікового запису чи інший реквізит. На цьому етапі система лише фіксує заяву, не перевіряючи її достовірність; важливо лише, щоб ідентифікатор був унікальним у межах системи й однозначно відповідав конкретному суб’єкту доступу [16, 17].

Автентифікація є наступним кроком і полягає у перевірці достовірності заявленої ідентичності. Система вимагає від користувача пред’явити докази своєї особи, тобто автентифікаційні фактори (облікові дані), і саме на цьому етапі приймається рішення про легітимність спроби доступу [16, 17].

Після успішної автентифікації виконується авторизація, у ході якої користувачеві надаються або обмежуються права доступу до ресурсів відповідно до його ролі, політик безпеки та налаштованих привілеїв [39]. Таким чином, спочатку відбувається ідентифікація, потім перевірка особи, і лише потім – рішення про те, до яких саме ресурсів і в якому обсязі буде надано доступ.

1.1.2. Фактори автентифікації

У науковій літературі та міжнародних стандартах автентифікаційні фактори поділяють передусім на три основні категорії, що відображають різні види доказів особи: те, що користувач знає, чим володіє та чим є як фізична особа [17, 39].

До факторів знання належать відомості, які мають бути відомі лише легітимному користувачу: паролі й парольні фрази, персональні ідентифікаційні

номери, відповіді на контрольні запитання, графічні паролі чи послідовності жестів. Перевага такого підходу в простоті реалізації та відсутності потреби в додатковому обладнанні. Водночас ці фактори є найвразливішими до соціальної інженерії, фішингу та підбору паролів, а реальна практика показує, що компрометація паролів залишається домінуючою причиною успішних атак на корпоративні системи [36].

Фактори володіння пов'язані з фізичними або логічними об'єктами, які знаходяться у розпорядженні користувача. До них відносять апаратні токени безпеки (зокрема ключі FIDO2 і смарт-карти), мобільні пристрої з установленими засобами автентифікації, генератори одноразових паролів, SIM-карти, що використовуються для отримання кодів, а також цифрові сертифікати на захищених носіях. Такі фактори суттєво ускладнюють атаку, оскільки зловмиснику необхідно не лише знати секрет, а й заволодіти самим носієм. Однак існують ризики втрати або крадіжки пристрою, а в окремих випадках – можливість клонування чи копіювання носія [10].

Фактори властивості пов'язані з біометричними характеристиками людини. До них зараховують відбитки пальців, геометрію обличчя, параметри райдужної оболонки й сітківки ока, особливості голосу і мовлення, геометрію долоні, а також поведінкові ознаки, такі як динаміка набору тексту. Ці характеристики мають найвищий рівень прив'язки до конкретної особи, оскільки не можуть бути передані іншому користувачеві й не підлягають забуванню. Водночас застосування біометрії потребує спеціалізованого обладнання, точного налаштування алгоритмів, розв'язання питань приватності та врахування того, що скомпрометовані біометричні дані практично неможливо замінити [13, 14, 15].

У сучасних дослідженнях додатково розглядають фактор місцезнаходження та фактор часу [8, 11]. Місцезнаходження визначається за IP-адресою, координатами GPS, приналежністю до довіреної локальної мережі чи фізичною близькістю до захищеного об'єкта. Часовий фактор відображає дозволені часові вікна доступу, частоту й ритм спроб автентифікації, тривалість сесій. У комплексі ці

характеристики дозволяють системі точніше оцінювати ризик поточної спроби доступу.

У реальних корпоративних мережах дедалі частіше поєднуються кілька факторів із різних категорій, що дає змогу досягати прийнятного балансу між рівнем захищеності, зручністю для користувача і витратами на впровадження.

1.1.3. Вразливості традиційних методів автентифікації

Парольні механізми, незважаючи на їхню історичну роль і простоту, виявляються вразливими до багатьох типових загроз. Слабкі паролі є однією з найочевидніших проблем: користувачі часто обирають короткі, легко запам’ятовувані й передбачувані комбінації. Аналіз скомпрометованих баз показує, що однакові прості паролі використовуються мільйонами облікових записів, причому переліки найпоширеніших паролів майже не змінюються роками [36, 37].

Ще серйознішим чинником є повторне використання одних і тих самих паролів для різних сервісів. За результатами досліджень, значна частина користувачів застосовує ідентичні або майже ідентичні облікові дані для декількох незалежних систем [5, 3, 4]. У випадку витоку бази одного постачальника це створює ефект ланцюгової реакції, коли зловмисник може автоматизовано перевіряти ті самі комбінації на інших платформах.

Фішингові кампанії залишаються одним з найрезультативніших інструментів отримання облікових даних. Зловмисники імітують інтерфейси легітимних сервісів, надсилають переконливі електронні листи або повідомлення й змушують користувачів добровільно вводити свої паролі. За статистикою, цей вектор задіяний у значній частині успішних інцидентів безпеки [36].

Технічні методи зламу паролів базуються як на повному переборі можливих комбінацій, так і на словникових підходах із використанням списків найімовірніших паролів і їх варіацій. Сучасні обчислювальні ресурси, включно з графічними процесорами та спеціалізованим програмним забезпеченням, дозволяють перебирати мільйони комбінацій за секунду, що робить короткі й прості паролі фактично марними [48, 10].

Додаткову небезпеку становлять перехоплення мережевого трафіку та атаки з проміжною ланкою, коли зловмисник вбудовується між користувачем і сервером та отримує змогу читати або змінювати передавані дані. Якщо автентифікаційні відомості передаються незашифрованими або недостатньо захищеними каналами, навіть складний пароль може бути викрадений під час передавання [23].

Не менш небезпечними є шкідливі програми класу кейлоггерів, що приховано фіксують натискання клавіш на зараженому пристрої. У такому випадку злам не залежить ані від довжини, ані від складності пароля: кожне введення фактично дублюється зловмиснику [50]. Соціальна інженерія доповнює технічні засоби, експлуатуючи довіру, необізнаність або психологічний тиск. Зловмисник може представлятися співробітником технічної підтримки, керівником чи партнером і змушувати користувача розкрити секретну інформацію [6].

Сукупність цих факторів показує, що покладатися виключно на парольну автентифікацію вже неможливо. Навіть за умов правильної організації політик складності й періодичної зміни паролів людський фактор і технічні вразливості зберігають високий рівень ризику, тому перехід до багатофакторних схем стає необхідною вимогою для захисту корпоративних ресурсів.

1.1.4. Статистика інцидентів безпеки, пов’язаних з автентифікацією

Статистичні звіти у сфері інформаційної безпеки дають кількісне підтвердження того, що проблеми автентифікації залишаються одним із ключових чинників успішних атак. За даними досліджень про вартість витоку даних, середня сума прямих витрат на один інцидент обчислюється мільйонами доларів, причому саме випадки, пов’язані з крадіжкою або компрометацією облікових даних, характеризуються найбільшою тривалістю виявлення й усунення наслідків – сотні днів від моменту проникнення до повного усунення загрози [37].

Аналіз глобальної картини порушень безпеки веб-застосунків показує, що переважна частка підтверджених інцидентів пов’язана саме з викраденими або вкрай слабкими обліковими даними. Серед типовий векторів атак домінують фішинг, використання раніше викрадених даних та експлуатація вразливостей самих веб-застосунків [36].

Особливо небезпечними є атаки на облікові записи привілейованих користувачів. Зафіксовано, що у випадку успішної компрометації таких облікових записів зловмисники можуть тривалий час залишатися непоміченими в мережі, отримуючи доступ до критично важливих систем і даних. Середній час їхньої присутності до виявлення оцінюється в сотні днів [37].

Окремі галузі, насамперед охорона здоров'я, демонструють особливу чутливість до таких інцидентів. Значна частина медичних організацій повідомляє про багаторазові порушення безпеки впродовж коротких періодів, причому більшість атак починаються саме з компрометації облікових даних персоналу [5].

Статистика також ілюструє ефективність багатофакторної автентифікації. За даними провідних постачальників хмарних сервісів, впровадження багатофакторних рішень знижує ризик захоплення облікового запису майже на два порядки порівняно з моделлю, де використовується лише пароль [8, 11]. Навіть найпростіші варіанти, наприклад додаткове підтвердження через SMS, дають відчутний захисний ефект, попри відомі обмеження цього методу.

Експериментальні вимірювання демонструють, що простий восьмисимвольний пароль із малих літер може бути підібраний за лічені хвилини за допомогою сучасних апаратних засобів. Ускладнення структури пароля й збільшення довжини подовжують час повного перебору до років, але не вирішують проблеми фішингу і повторного використання паролів [48].

Фінансові наслідки інцидентів сильно залежать від галузі: особливо високі втрати спостерігаються у фінансовому секторі, сфері охорони здоров'я та фармацевтичній промисловості [37]. Додатково на статистику вплинув масовий перехід до віддаленої роботи, що різко збільшив кількість фішингових кампаній, спрямованих на працівників, які отримують доступ до корпоративних ресурсів ззовні [5]. Географічний аналіз підтверджує глобальний характер загрози, з високою концентрацією інцидентів у регіонах із найрозвиненішою цифровою інфраструктурою [36].

У сукупності ці дані демонструють не лише технічну, а й економічну доцільність переходу до більш надійних, насамперед багатофакторних систем

автентифікації, оскільки потенційні втрати від інцидентів суттєво перевищують витрати на впровадження додаткових рівнів захисту.

1.1.5. Принципи побудови надійних систем автентифікації

Проєктування системи автентифікації в корпоративному середовищі потребує не лише вибору окремих технологій, а й дотримання низки базових принципів інформаційної безпеки [51]. Центральним серед них є принцип багатошаровості захисту, згідно з яким жоден окремий механізм не повинен бути єдиною лінією оборони. Багатофакторна автентифікація безпосередньо реалізує цю ідею, вимагаючи одночасного використання різних типів факторів [39].

Принцип найменших привілеїв вимагає, щоб після успішної автентифікації користувач отримував лише ті права, які необхідні для виконання його посадових обов'язків, а процес ідентифікації і перевірки особи забезпечував достатній рівень упевненості для такого призначення ролей [16, 17]. Тісно пов'язаний із ним принцип розділення обов'язків передбачає, що ключові, особливо ризиковані операції не повинні виконуватися одноосібно, а система автентифікації має дозволяти однозначно встановити, хто і яку частину операції виконував [40].

З погляду користувача важливим є принцип прозорості: механізми безпеки мають інтегруватися у робочі процеси без створення зайвих перешкод. Надмірно складні процедури часто стимулюють користувачів шукати небезпечні обхідні шляхи, наприклад записувати паролі або передавати токени третім особам [22].

Принцип стійкості до атак означає, що система повинна зберігати працездатність і прийнятний рівень захисту навіть у разі часткової компрометації одного з компонентів: окремого фактора, каналу зв'язку чи проміжної ланки інфраструктури. Для цього кожен елемент автентифікаційної схеми проєктується з урахуванням можливих векторів атак та сценаріїв відмов [11].

Принцип неможливості відмови від дій передбачає, що після успішної автентифікації користувач не може достовірно заперечити факт виконання певних операцій. Це досягається завдяки надійному журналюванню процесів автентифікації й доступу, використанню криптографічних механізмів підтвердження дій та захищеному зберіганню відповідних записів [39].

Адаптивність системи стає критичною в умовах змінного ландшафту загроз. Вимоги до автентифікації мають можливість динамічно змінюватися залежно від контексту доступу, рівня ризику, типу операції, географічного розташування чи характеристик пристрою. Це дозволяє посилювати перевірку лише там, де це справді виправдано [8, 12].

Масштабованість є ще одним ключовим принципом: система повинна коректно працювати як для невеликих організацій, так і для великих інфраструктур із тисячами користувачів і пристроїв, не втрачаючи продуктивності та надійності при зростанні навантаження [11].

Не менш важливою є сумісність із різними платформами, клієнтськими пристроями та вже наявними корпоративними сервісами. Використання відкритих стандартів, таких як TOTP, FIDO2, OAuth 2.0, спрощує інтеграцію та забезпечує гнучкість при подальшій модернізації системи [1, 3, 25].

Принцип конфіденційності зобов'язує захищати автентифікаційні дані при зберіганні та передаванні. Паролі мають зберігатися лише у вигляді хешів із додатковим використанням «солі», а біометричні дані – оброблятися локально на пристрої користувача, без передачі первинних шаблонів на сервер і без можливості їх зворотного відтворення [3, 56].

Комплексне дотримання цих принципів дозволяє сформувати збалансовану систему автентифікації, яка одночасно забезпечує високий рівень захисту, відповідає вимогам регуляторів і не створює надмірного навантаження на користувачів.

1.2. Огляд сучасних методів автентифікації

1.2.1. Однофакторна автентифікація: парольні механізми

Пароль залишається базовим способом контролю доступу в корпоративних мережах, хоча саме його слабкі місця найчастіше стають причиною компрометації облікових записів. Механізм парольної автентифікації ґрунтується на порівнянні введеного користувачем значення з еталонним, яке зберігається у вигляді криптографічного хеша. Паролі у сучасних системах не зберігаються у відкритому

вигляді: до них застосовуються односторонні хеш-функції, що перетворюють рядок довільної довжини на фіксований дайджест [50].

Для підвищення стійкості до колізій, атак передобчислення та підбору використовують спеціалізовані алгоритми хешування паролів: bcrypt, scrypt, Argon2. Вони передбачають використання «солі» – випадкових даних, які додаються до пароля перед хешуванням, а також керовану обчислювальну складність. Це унеможливлює ефективне застосування попередньо обчислених таблиць і значно ускладнює масовий злам великих баз даних облікових записів [48].

Переваги парольної автентифікації пов’язані насамперед із простотою, низькою вартістю й відсутністю потреби в додатковому обладнанні. Такий підхід добре знайомий користувачам, легко реалізується практично в будь-якій інформаційній системі й дозволяє оперативно змінювати пароль у разі підозри на компрометацію. Водночас сукупність недоліків робить парольні механізми вразливими: поширеність соціальної інженерії та фішингу, схильність користувачів обирати прості або повторно використовувані паролі, ризики перехоплення в незахищених каналах, складність надійного відновлення забутих облікових даних і небезпека витоку хешованих баз, які все одно можуть бути зламані методом грубої сили [36, 50].

Дослідження підтверджують, що навіть жорсткі формальні вимоги до довжини й складності паролів не гарантують реальної стійкості. Користувачі часто дотримуються мінімальних вимог, але застосовують передбачувані шаблони – заміну літер на схожі цифри, додавання одного й того самого символу в кінці тощо. У результаті паролі легко піддаються модифікованим словниковим атакам [22].

Типові політики керування паролями в корпоративному середовищі регламентують мінімальну довжину, комбінацію різних типів символів, історію паролів, термін їх дії, а також блокування облікового запису після низки невдалих спроб. Для привілейованих облікових записів додатково вимагається використання другого фактора [17]. Однак сучасні рекомендації, зокрема NIST, ставлять під сумнів доцільність примусової частих змін паролів: користувачі в таких умовах

роблять мінімальні передбачувані модифікації або починають записувати складні паролі в незахищеному вигляді, що фактично знижує загальний рівень захищеності [17].

1.2.2. Двофакторна автентифікація

Двофакторна автентифікація стала наймасовішою формою багатофакторної перевірки, оскільки поєднує істотне зростання безпеки з відносно невеликим погіршенням зручності. Зазвичай вона реалізується як комбінація пароля (фактор знання) із додатковим кодом або іншим доказом володіння. Процес поділяється на два послідовні кроки: спочатку користувач вводить пароль, а потім підтверджує вхід одноразовим кодом, який може надходити через SMS, генеруватися токеном або мобільним застосунком [16, 22].

Найпростішим варіантом є SMS-автентифікація, коли система надсилає короткий числовий код на зареєстрований номер телефону користувача. Код діє обмежений час і має бути введений разом із паролем [9]. Такий підхід практично не вимагає додаткової інфраструктури, але має низку добре відомих вразливостей. Атаки з підміною SIM-карти дозволяють зловмиснику перереєструвати номер на себе й отримувати всі одноразові коди. Додатково SMS можуть бути перехоплені через недоліки сигнального протоколу SS7 або за допомогою спеціалізованих технічних засобів [50].

Надійнішою альтернативою виступають апаратні токени, що генерують одноразові паролі за алгоритмами TOTP або HOTP без підключення до мережі. Користувач зчитує код із дисплея токена й вводить його разом із паролем [1, 2]. У цьому випадку фактор володіння не залежить від операторів мобільного зв’язку, а криптографічний механізм побудований на спільному секреті між токеном і сервером. Висока безпека й автономність поєднуються з витратами на закупівлю обладнання, організацію логістики й ризиком фізичної втрати або пошкодження пристрою [48].

Програмні аутентифікатори для смартфонів фактично відтворюють функціональність апаратних токенів у вигляді застосунку. Такі рішення, як Google Authenticator, Microsoft Authenticator, Authy та інші, генерують коди TOTP на

основі секретного ключа, що зберігається в захищеному середовищі пристрою [27, 28]. Процес прив’язки зазвичай виконується через сканування QR-коду або ручне введення секрету, після чого застосунок може працювати офлайн, використовуючи лише поточний час та збережений ключ [1, 44]. Цей підхід поєднує високу стійкість алгоритму з низькими витратами, оскільки більшість користувачів уже мають необхідні пристрої.

Окремий клас становить двофакторна автентифікація на основі push-повідомлень. У такому випадку після введення пароля користувач отримує на смартфон запит із пропозицією підтвердити або відхилити спробу входу. Застосунок може відображати додаткову контекстну інформацію: приблизне географічне розташування, тип пристрою, час входу, що допомагає виявляти підозрілі спроби [18, 11]. Для підтвердження достатньо одного натискання на кнопці, тому користувацький досвід суттєво поліпшується, але метод вимагає постійного підключення до Інтернету й наявності сумісного мобільного пристрою.

Біометрична двофакторна автентифікація поєднує традиційний пароль із біометричними характеристиками, такими як відбиток пальця або розпізнавання обличчя. Сучасні смартфони та операційні системи надають вбудовані механізми біометричної перевірки, тому додання такого фактора у багатьох сценаріях є технічно нескладним і зручним для користувача [10, 13]. Перевагою є те, що біометричні ознаки не можна забути або передати іншій особі, однак виникають питання захисту персональних даних, можливості виготовлення високоякісних підробок та наслідків компрометації біометрії, яка, на відміну від пароля, не може бути змінена [56].

Результати досліджень і статистика великих постачальників хмарних сервісів свідчать, що застосування навіть базових схем двофакторної автентифікації знижує ймовірність успішного захоплення облікового запису більш ніж на 99 відсотків [8, 11]. Водночас складні фішингові атаки у режимі реального часу, а також сценарії «людина посередині» здатні обходити окремі реалізації, особливо якщо другий фактор передається у вигляді коду, який користувач вводить вручну. Відомі групи, зокрема APT20, демонстрували можливість обходу захисту через компрометацію

адміністративних консолей або поштових скриньок, використовуючи легітимні механізми відновлення доступу [16, 17]. Це підтверджує, що двофакторна автентифікація є важливим, але не єдиним компонентом комплексного захисту.

1.2.3. Багатофакторна автентифікація

Багатофакторна автентифікація розвиває ідею двофакторної, використовуючи три й більше незалежних фактори. Такий підхід застосовується для найбільш критичних систем, де необхідно забезпечити максимальний рівень упевненості в особі користувача. Типовою є комбінація пароля, одноразового коду з мобільного застосунку та біометричної характеристики, що дає змогу зберегти захист навіть у разі компрометації одного або двох компонентів [8, 11].

Сучасні системи дедалі частіше реалізують адаптивну багатофакторну автентифікацію. Вона аналізує контекст доступу: місцезнаходження, тип і стан пристрою, час доби, типові поведінкові шаблони, мережеве оточення. На основі оцінки ризику система динамічно вирішує, які й скільки факторів потрібно застосувати в конкретній ситуації [8, 12]. Наприклад, звичайний вхід із офісного комп’ютера в робочий час може вимагати лише пароля й одноразового коду, тоді як нестандартна спроба з невідомого пристрою вночі — додаткової біометричної перевірки або підтвердження через інший канал зв’язку [12].

Багатоканальна автентифікація підсилює захист завдяки використанню різних каналів комунікації: SMS, електронної пошти, push-повідомлень, месенджерів. Зловмиснику в такому разі доводиться одночасно компрометувати кілька незалежних каналів, що значно підвищує складність атаки [18, 19]. Контекстна автентифікація, у свою чергу, використовує пасивні фактори, які не потребують активних дій від користувача: IP-адресу, геолокацію, параметри пристрою, характерні особливості взаємодії з інтерфейсом, часові шаблони доступу та належність до корпоративної мережі [8, 12]. Такі ознаки аналізуються у фоні, формуючи профіль «нормальної» поведінки, а відхилення від нього викликають підвищення рівня перевірки.

У корпоративних середовищах важливе місце займає ступінчаста автентифікація, коли вимоги залежать від критичності ресурсу. Для базових

сервісів достатньо двох факторів, а для фінансових систем, сховищ конфіденційних даних або адміністративних консолей може вимагатися повноцінна багатофакторна схема з додатковими перевірками й біометрією [39, 40]. Окремим напрямом розвитку є безперервна автентифікація: система не обмежується перевіркою на момент входу, а постійно контролює поведінку користувача протягом усієї сесії й може вимагати повторне підтвердження або завершувати сесію у разі виявлення нетипових дій [8].

Багатофакторна автентифікація на рівні окремих застосунків дозволяє кожній системі визначати власний набір вимог. Критично важливі сервіси можуть наполягати на максимально жорсткому наборі факторів, тоді як менш чутливі застосунки обмежуються спрощеними схемами, зберігаючи прийнятний рівень зручності для користувачів [11, 21].

Серед переваг багатофакторних рішень — можливість досягти найвищого рівня захисту від несанкціонованого доступу, гнучке налаштування політик під різні категорії користувачів і сценарії, відповідність вимогам регуляторів та мінімізація наслідків компрометації окремих факторів [8, 11, 39]. Однак впровадження таких систем пов’язане зі складністю керування великою кількістю факторів, потенційним зниженням зручності, необхідністю додаткового навчання користувачів і витратами на інтеграцію з наявною інфраструктурою [22]. Досвід показує, що ключ до успіху полягає в адаптивності: додаткові фактори повинні вимагатися лише тоді, коли об’єктивні ознаки вказують на підвищений ризик [12, 21].

1.2.4. Адаптивна автентифікація

Адаптивна, або ризик-орієнтована, автентифікація є логічним продовженням ідеї багатофакторного підходу: вона не просто перевіряє певний набір факторів, а змінює його залежно від оціненого ризику поточної спроби доступу. Система в реальному часі збирає відомості про контекст входу, поведінку користувача, характеристики пристрою та мережі й на основі цих даних формує числовий показник ризику [8, 12]. Від цього показника залежить, чи буде достатньо базової схеми автентифікації, чи знадобляться додаткові кроки.

До географічних факторів належить відстань між поточним і попереднім місцезнаходженням, країна та регіон доступу, виявлення «неможливих» сценаріїв подорожі (коли між двома входами у різних точках світу минуло замало часу), а також доступ із територій, які мають погану репутацію з точки зору кіберзлочинності [12]. Часові характеристики враховують відповідність спроби звичному графіку роботи користувача, частоту й ритм входів, незвично велику кількість спроб за короткий проміжок. Аналізуються і характеристики пристрою: відбиток браузера, тип і версія операційної системи, історія використання, належність до списку довірених, рівень оновленості й наявність засобів захисту [11, 12]. Мережеві ознаки включають IP-адресу та її репутацію, тип підключення, використання проксі або анонімних мереж, історію IP для конкретного користувача [8]. Додатковий шар становлять поведінкові патерни: швидкість і ритм набору тексту, характер руху миші, типовий маршрут дій після входу, звичні ресурси, до яких звертається користувач [12].

Архітектура адаптивної системи зазвичай складається з модуля збору даних, аналітичного ядра, модуля прийняття рішень і механізму адаптації інтерфейсу. Модуль збору акумулює вхідну інформацію з логів, мережевого обладнання, зовнішніх сервісів геолокації та систем моніторингу безпеки, працюючи у режимі реального часу [8]. Аналітичний компонент застосовує набір правил та алгоритми машинного навчання, щоб обчислити рівень ризику. Використовуються як методи керованого навчання для класифікації спроб на легітимні й підозрілі, так і некеровані моделі для виявлення аномалій, а також підходи з навчанням з підкріпленням, які дозволяють системі адаптуватися до нових типів загроз [12, 38].

Модуль прийняття рішень зіставляє отриманий показник ризику із заздалегідь визначеними порогами. Для низького ризику допускається стандартний вхід із мінімальною кількістю додаткових перевірок і тривалими сесіями. За середнього ризику система автоматично вмикає двофакторну автентифікацію, скорочує час дії сесії, активніше журналює дії користувача й може повідомляти його про незвичні спроби доступу [8]. Високий ризик означає необхідність повної багатофакторної перевірки, залучення кількох каналів підтвердження, обмеження доступу до

критичних ресурсів і сповіщення служби безпеки [11, 12]. За критичного ризику спроба доступу блокується, запускаються процедури реагування на інцидент і вимагається пряма верифікація особи користувача [8].

Головна перевага адаптивної автентифікації полягає в можливості одночасно підвищити безпеку й зменшити навантаження на легітимних користувачів у типових сценаріях. Система автоматично посилює вимоги лише тоді, коли поведінка або контекст виходять за межі звичайного профілю [8, 11, 12]. Разом із тим упровадження таких рішень вимагає значних ресурсів для збору й обробки даних, ретельного налаштування, зниження кількості хибнопозитивних спрацювань і врахування вимог до захисту приватності [22]. Успішні впровадження демонструють відчутне зменшення кількості невиправданих відмов легітимним користувачам і водночас помітне зростання виявлення шахрайських спроб у порівнянні зі статичними схемами [12].

1.2.5. Безпарольна автентифікація

Безпарольна автентифікація, або passwordless-підхід, спрямована на повне усунення паролів як фактора автентифікації. Вона базується на використанні криптографії з відкритим ключем: користувач генерує пару ключів, приватний ключ надійно зберігається на його пристрої, а відкритий реєструється на сервері. Під час входу система не запитує секрет, а перевіряє правильність криптографічного підпису виклику, створеного приватним ключем [3, 45, 46].

Стандарти FIDO2 та WebAuthn визначають, як браузери й веб-застосунки мають взаємодіяти з такими автентифікаторами. Під час реєстрації сервер формує випадковий виклик, який через WebAuthn передається автентифікатору. Після локального підтвердження користувачем (PIN, біометрія) пристрій створює нову пару ключів, підписує виклик і надсилає відкритий ключ разом із атестаційними даними. Сервер зберігає відкритий ключ і надалі використовує його для перевірки підписів під час автентифікації [47, 48, 49]. Процес входу побудований на такій самій схемі «виклик–підпис», але без створення нових ключів [3, 5, 6].

Автентифікатори FIDO2 можуть бути платформними, тобто вбудованими в операційну систему чи апаратну платформу (Windows Hello, Touch ID, Face ID,

механізми на Android із використанням захищених модулів TPM), або зовнішніми, у вигляді USB-, NFC- чи Bluetooth-пристроїв, смарт-карт тощо [52, 54, 55, 57]. Окремий напрям розвитку становлять ключі доступу (passkeys), коли пари FIDO-ключів синхронізуються між власними пристроями користувача через зашифроване хмарне сховище. Це усуває прив'язку до одного конкретного пристрою й дозволяє виконувати безпарольний вхід із будь-якого особистого пристрою в межах екосистеми постачальника.

Інші варіанти безпарольної автентифікації, наприклад «магічні посилання», ґрунтуються на надсиланні одноразового посилання на електронну пошту, перехід за яким автоматично підтверджує вхід. Такий підхід простий і знайомий користувачам, але його безпека повністю залежить від захищеності поштової скриньки, тоді як FIDO2 зберігає секретний ключ виключно на стороні автентифікатора. Біометрична безпарольна автентифікація використовує фізіологічні характеристики як єдиний фактор, причому самі біометричні дані зберігаються у вигляді неможливих до відновлення шаблонів на пристрої й не передаються на сервер [13].

Основні переваги безпарольного підходу — повна відсутність паролів у процесі, а отже, імунітет до фішингу, спрямованого на їх викрадення, та до атак на бази даних паролів. Сервер зберігає лише відкриті ключі, а приватні залишаються у захищеному середовищі користувача, тому навіть у разі витоку облікових записів зловмисник не отримує засобу для автентифікації [3]. Користувачі позбавляються необхідності запам'ятовувати складні паролі, а час проходження процедури входу помітно скорочується.

Водночас існують і обмеження. Безпарольна автентифікація потребує підтримки з боку операційних систем, браузерів і апаратних платформ, а також надійних механізмів резервування й відновлення у випадку втрати всіх пристроїв. Залишається питання сумісності зі старими системами та необхідність наявності резервних методів доступу на випадок відмови або недоступності основного автентифікатора. На практиці часто застосовуються гібридні схеми, коли безпарольний метод є основним, але користувачеві доступний і традиційний

пароль як запасний варіант. Досвід експлуатації показує різке скорочення звернень до служби підтримки щодо скидання паролів і суттєве зменшення часу автентифікації.

1.2.6. Біометрична автентифікація

Біометрична автентифікація використовує унікальні фізичні або поведінкові характеристики людини для підтвердження її особи. На відміну від паролів чи токенів, біометричні ознаки неможливо забути або передати іншій особі, що робить їх привабливим фактором для систем високого рівня безпеки.

Фізіологічна біометрія охоплює відбитки пальців, геометрію обличчя, райдужну оболонку, сітківку, форму долоні та інші стабільні властивості тіла. Відбитки пальців є найпоширенішим методом завдяки компактним і дешевим сенсорам, які додатково можуть перевіряти ознаки «живості» тканин. Системи розпізнавання обличчя аналізують просторову структуру рис, а сучасні реалізації використовують тривимірне сканування, що знижує ризик обману за допомогою статичних зображень [13]. Сканування райдужки й сітківки забезпечує ще вищу точність, але потребує спеціалізованого обладнання і має меньшу прийнятність для користувачів [13].

Поведінкова біометрія базується на динамічних шаблонах: стилі набору тексту, характері руху миші, властивостях голосу, особливостях ходи. Ці ознаки можуть застосовуватися не тільки для разової перевірки на вході, а й для безперервного контролю протягом сесії, коли система у фоні відстежує відповідність поточної поведінки звичному профілю користувача [13].

Типова система біометричної автентифікації складається з кількох послідовних модулів. Спочатку сенсор захоплює біометричні дані, після чого вони очищуються від шуму й нормалізуються. Далі спеціальні алгоритми виділяють ключові ознаки й формують компактний математичний шаблон, який зберігається в системі замість «сирих» даних . Під час автентифікації новий шаблон порівнюється з еталонним, а рішення приймається на основі порогового значення подібності. Окремий модуль виявлення підробок оцінює ознаки живості, щоб відрізнити живу людину від накладок, фотографій чи інших штучних об’єктів.

Якість біометричних систем характеризується показниками FAR (частота хибного прийняття), FRR (частота хибного відхилення) та EER (рівень, у якому обидві помилки збалансовані) [13]. Для критичних застосунків прагнуть мінімізувати FAR навіть за рахунок деякого збільшення FRR, тоді як у масових сервісах більше уваги приділяється зручності.

Біометрична автентифікація має очевидні переваги: фактор не можна забути або втратити, рівень точності сучасних систем дуже високий, контроль живості ускладнює підробку, а поведінкові методи дозволяють реалізувати безперервний моніторинг [13]. Разом із тим існують істотні ризики. Біометричні дані належать до особливих категорій персональної інформації, і їх компрометація має значно серйозніші наслідки, ніж витік паролів, оскільки змінити відбитки пальців чи райдужну оболонку неможливо. Вартість і складність деяких типів сенсорів, можливі збої через травми або фізіологічні зміни, а також загроза складних атак із використанням високоякісних підробок потребують особливо уважного підходу [13].

Правові рамки, у тому числі українське законодавство про захист персональних даних і міжнародні регуляції на кшталт GDPR, вимагають підвищеного рівня захисту для біометричної інформації, прозорих процедур отримання згоди та обмеження цілей її обробки [41, 39]. У відповідь на ці вимоги набирають популярності мультимодальні системи, які поєднують кілька біометричних ознак, а також використовують біометрію не як єдиний фактор, а як частину ширшої багатофакторної схеми. Комбінація традиційного секрету із біометричною перевіркою зазвичай забезпечує набагато вищий рівень безпеки, ніж будь-який із цих методів окремо [10, 13].

1.3. Технології та протоколи багатофакторної автентифікації

1.3.1. Протоколи OAuth 2.0, OpenID Connect, SAML 2.0

Сучасні системи багатофакторної автентифікації спираються на узгоджені протоколи, які дають змогу розділити функції автентифікації, авторизації та зберігання даних між різними компонентами корпоративної інфраструктури. Протоколи OAuth 2.0, OpenID Connect та SAML 2.0 забезпечують узгоджений

обмін службовою інформацією, що дозволяє поєднувати локальні системи з хмарними сервісами, мобільними застосунками й веб-платформами [25, 26, 11].

OAuth 2.0 є протоколом авторизації, який дає змогу додаткам отримувати обмежений доступ до ресурсів без передачі паролів користувача. У цій моделі розрізняють власника ресурсу, сервер ресурсів, клієнт і сервер авторизації, між якими циркулюють токени доступу та оновлення [25]. Після автентифікації користувача сервер авторизації видає токен доступу, що діє обмежений час, а за потреби — токен оновлення для отримання нового токена без повторного введення пароля [25]. Різні «потоки» авторизації дають змогу адаптувати протокол до веб-застосунків, односторінкових інтерфейсів, сценаріїв взаємодії між серверами чи служб, де користувач безпосередньо не залучений. Для систем багатофакторної автентифікації OAuth 2.0 доповнюють параметрами, які описують вимоги до способу автентифікації та потрібний рівень довіри, що дозволяє підвищити захист доступу до критичних ресурсів [25, 26].

OpenID Connect будується поверх OAuth 2.0 і додає шар ідентифікації користувача. Якщо OAuth зосереджений на доступі до ресурсу, то OpenID Connect дає змогу однозначно встановити особу користувача та отримати базовий профіль у стандартизованому вигляді [26]. Для цього застосовується ID Token у форматі JSON Web Token, який містить низку тверджень про користувача та саму операцію автентифікації: унікальний ідентифікатор суб’єкта, відомості про видавця токена, аудиторію, час видачі й закінчення дії, а також параметр nonce як захист від повторного відтворення запиту [26]. OpenID Connect визначає фіксований набір областей доступу для запиту інформації про профіль, а також механізми керування сесіями, що дає можливість централізовано контролювати життєвий цикл входу користувача у розподіленому середовищі [26].

SAML 2.0 є стандартом на основі XML, що історично отримав широке застосування в корпоративних середовищах для реалізації єдиного входу. У цій моделі провайдер ідентичності відповідає за автентифікацію користувача, формує твердження у вигляді SAML Assertions і передає їх сервіс-провайдеру, який, довіряючи цьому джерелу, приймає рішення про надання доступу [11]. Твердження

можуть описувати факт автентифікації, набір атрибутів (роль, підрозділ, адресу електронної пошти) або рішення щодо дозволу чи заборони певної дії [11]. Процес SSO у SAML базується на перенаправленні користувача до провайдера ідентичності, автентифікації (у тому числі з використанням багатофакторних механізмів), формуванні підписаного твердження та поверненні його до сервісу, який перевіряє підпис і контекст автентифікації [11].

Як OAuth 2.0 з OpenID Connect, так і SAML 2.0 дають змогу фіксувати тип застосованого методу автентифікації й вимагати підвищений рівень безпеки, наприклад обов'язкове використання декількох факторів або апаратного токена [11, 26]. Підхід на основі OAuth 2.0 / OpenID Connect вважається більш природним для сучасних веб та мобільних застосунків, де використовуються інтерфейси на основі REST і JSON, тоді як SAML 2.0 залишається типовим рішенням для інтеграції з наявною корпоративною інфраструктурою та системами єдиного входу, що орієнтуються на XML і класичні каталоги на кшталт Active Directory чи LDAP [25, 26, 11]. На практиці великі організації часто підтримують обидва підходи, щоб забезпечити сумісність із ширшим колом сервісів [11].

1.3.2. Стандарти FIDO2, WebAuthn, U2F

Стандарти, розроблені в рамках FIDO Alliance, покликані зменшити залежність від паролів і запропонувати відмовостійкі механізми автентифікації на основі криптографії з відкритим ключем. Першим масовим стандартом цього сімейства став FIDO U2F, який запропонував фізичний другий фактор у вигляді невеликого токена, що активується вручну, зазвичай натисканням кнопки [5, 7]. Під час реєстрації на кожному веб-сайті такий токен створює окрему пару ключів, а під час входу виконує криптографічний підпис виклику з прив'язкою до домену, що робить фішинг малоефективним, адже токен не підпише запит від іншого сайту. Приватні ключі ніколи не покидають пристрій, а взаємодія може відбуватися через USB, NFC або Bluetooth, що забезпечує гнучкість використання на різних платформах.

Подальшим розвитком концепції став стандарт WebAuthn, який визначає єдиний програмний інтерфейс для взаємодії веб-застосунків із автентифікаторами.

У моделі WebAuthn розрізняють сторону, що покладається на автентифікацію (Relying Party), браузер або платформу, яка реалізує API, а також власне автентифікатор, що створює й зберігає ключі [3, 5, 6]. Під час реєстрації сервер генерує криптографічний виклик і параметри політики, браузер передає їх до автентифікатора, який після локальної верифікації користувача (наприклад, через PIN або біометрію) створює пару ключів та повертає відкритий ключ разом із атестаційними даними. У процесі автентифікації той самий відкритий ключ використовується для перевірки підпису нового виклику. Таким чином, сервер зберігає лише відкриті ключі, а всі секретні операції виконуються на пристрої користувача.

Стандарт FIDO2 поєднує WebAuthn із протоколом CTAP, який описує взаємодію між клієнтом і автентифікатором, у тому числі зовнішнім. CTAP1 забезпечує зворотну сумісність із токенами U2F, тоді як CTAP2 розширює можливості до безпарольної автентифікації, підтримки резидентних ключів, зберігання кількох облікових записів на одному пристрої, а також різних способів верифікації користувача. Резидентні ключі дозволяють виконувати вхід без явного введення логіна: автентифікатор зберігає облікові записи й за запитом пропонує ті, що відповідають конкретному сервісу. Атестація автентифікатора дає змогу серверу переконатися, що використовується справжній пристрій, який відповідає вимогам до безпеки; залежно від політики можуть застосовуватися різні типи атестації — від повної із сертифікатом до спрощеної або взагалі відсутньої.

Перевага FIDO2 і WebAuthn полягає в тому, що криптографічні ключі прив’язуються до конкретного домену, а секретна частина зберігається в захищеному середовищі, нерідко у спеціальному компоненті з апаратним захистом від фізичного та побічного аналіз. Сервер не має у своєму розпорядженні паролів, а лише відкриті ключі, тож витік бази облікових даних не дає зловмиснику безпосереднього засобу для входу. Водночас упровадження таких рішень вимагає сумісних браузерів, платформ і провайдерів ідентичності, а також продуманої процедури відновлення доступу у випадку втрати апаратного токена. Однак дослідження показують, що організації, які переходять на FIDO2-автентифікацію,

суттєво знижують частку інцидентів, пов’язаних із компрометацією облікових даних, і зменшують навантаження на служби підтримки, пов’язане зі скиданням паролів.

1.3.3. Апаратні токени безпеки

Апаратні токени безпеки можна розглядати як окремий клас засобів автентифікації, в яких криптографічні секрети фізично ізольовані від загальної обчислювальної системи. Такі пристрої або генерують одноразові коди, або зберігають пари ключів для схем із відкритим ключем, виконуючи всі операції всередині себе. Завдяки цьому компрометація робочої станції або мобільного пристрою не дає прямого доступу до секретів [48].

Класичний приклад — апаратні OTP-токени, які реалізують алгоритми TOTP або HOTP. У випадку TOTP код обчислюється на основі спільного секретного ключа та поточного часу, а сервер, маючи той самий секрет і орієнтуючись на власний годинник, незалежно отримує такий самий результат [1]. Параметри на кшталт довжини коду, інтервалу його дії й допустимого вікна для компенсації зміщення годинника задаються політикою безпеки [1, 44, 48]. Алгоритм HOTP замість часу використовує лічильник, що збільшується після кожної успішної автентифікації, завдяки чому код залишається дійсним доти, доки не буде використаний, але вимагає механізмів повторної синхронізації лічильників у разі розбіжностей [2, 48].

До іншого класу належать FIDO2/U2F-токени, USB-ключі, смарт-карти та пристрої з NFC або Bluetooth, які здебільшого побудовані навколо захищеного елемента. У такому елементі зберігаються ключі, реалізовані алгоритми RSA чи ECC, симетричне шифрування, хешування й генерація випадкових чисел [55]. Багато моделей сертифікуються за міжнародними стандартами безпеки й підтримують різні протоколи одночасно, поєднуючи OTP, FIDO2, PIV та інші механізми на одному фізичному носії. Керування доступом до токена зазвичай здійснюється через PIN, а після кількох невдалих спроб пристрій може блокуватися, що знижує ризик зловживання у разі фізичної крадіжки.

Основна перевага апаратних токенів полягає в тому, що вони забезпечують високий рівень захисту за умови коректного адміністрування. Проте їх використання супроводжується витратами на закупівлю, розподіл і облік пристроїв, необхідністю підтримувати резервні методи на випадок втрати чи пошкодження, а також потенційними труднощами сумісності з застарілими системами [48, 55]. Попри це, дослідження демонструють, що організації, які системно застосовують апаратні токени, мають набагато нижчий ризик компрометації облікових записів порівняно з тими, хто покладається на SMS чи суто програмні рішення [48].

1.3.4. Програмні токени (TOTP, HOTP)

Програмні токени відтворюють функціональність апаратних пристроїв у вигляді застосунків для смартфонів або робочих станцій. Вони також використовують алгоритми TOTP і HOTP, однак секретний ключ зберігається у захищеному сховищі операційної системи, а одноразовий код обчислюється локально на пристрої [1, 2, 44]. У випадку TOTP основою є поточний час, поділений на фіксовані інтервали, тоді як у HOTP використовується лічильник, що збільшується з кожним використанням [1, 2, 48, 50]. Практичні параметри, такі як тривалість інтервалу, довжина коду та допустимі відхилення, налаштовуються на стороні сервера й клієнта для досягнення балансу між безпекою та зручністю [1, 44, 48].

На ринку доступна низка поширених програмних аутентифікаторів. Частина з них реалізує лише базовий механізм TOTP, інші поєднують одноразові коди з push-повідомленнями, безпарольним входом, резервним копіюванням и синхронізацією між пристроями [27, 28]. Деякі рішення інтегровані з екосистемами великих постачальників, пропонують зберігання облікових записів у зашифрованому вигляді в хмарі, захист застосунку PIN-кодом або біометрією та додаткові перевірки на наявність модифікацій операційної системи [27, 28].

Налаштування програмного токена зазвичай відбувається через сканування QR-коду на екрані сервісу або вручну, шляхом введення секретного ключа у відповідне поле. Обидва способи використовують єдиний формат кодування

параметрів, який включає секрет, тип алгоритму, період дії й назву сервісу [1, 28, 44]. Після первинної прив’язки користувач підтверджує правильність налаштування, вводячи перший згенерований код.

Безпека програмних токенів значною мірою залежить від захищеності кінцевого пристрою. Операційні системи сучасних смартфонів надають засоби ізоляції ключів, апаратне шифрування та інтерфейси, що обмежують доступ до секретів лише самим застосункам-аутентифікаторам [28]. Додатково можуть застосовуватися блокування доступу до програми, автоматичне блокування після періоду неактивності та, за бажанням, заборона створення знімків екрана [27, 28]. Водночас залишається ризик зараження пристрою шкідливим програмним забезпеченням, яке здатне зчитувати коди з екрана або перехоплювати екранну активність, а також небезпека прямого фішингу в режимі реального часу, коли користувач вводить код на підробленому сайті, а зловмисник миттєво використовує його на справжньому ресурсі [50]. Критичним є також ризик втрати пристрою, особливо якщо користувач не має резервних кодів або альтернативних факторів для відновлення доступу [28].

Порівняно з апаратними токенами програмні рішення мають суттєві переваги у вартості та простоті розгортання: користувачі переважно вже володіють сумісними пристроями, не потрібно організовувати фізичну логістику, набагато легше додавати нові облікові записи [28, 48]. Натомість апаратні токени забезпечують вищу фізичну ізоляцію й менш залежні від стану основного пристрою, а в поєднанні з FIDO2 можуть забезпечувати найвищий рівень захисту для привілейованих облікових записів [55, 57]. На практиці програмні TOTP-токени часто обирають як основний метод для широкого кола користувачів, а апаратні ключі залишають для адміністраторів і доступу до найкритичніших систем [28, 48].

1.3.5. Push-повідомлення для автентифікації

Підхід, заснований на push-повідомленнях, використовує мобільний пристрій як фактор володіння і дає змогу максимально спростити взаємодію користувача з системою. Замість ручного введення коду користувач отримує запит на

підтвердження входу у спеціальний застосунок та приймає рішення одним дотиком, при цьому має змогу переглянути контекст спроби [11].

Типова схема роботи включає етап введення логіна і пароля, формування сервером унікального запиту з контекстними даними, доставку цього запиту через службу push-повідомлень операційної системи на смартфон і відображення у застосунку детальної інформації про спробу входу: приблизне місцезнаходження, IP-адресу, тип клієнта, час та назву сервісу [11, 27]. Якщо користувач підтверджує дію, застосунок формує відповідь, яка криптографічно підписується приватним ключем, збереженим у захищеному сховищі пристрою, і передається на сервер по захищеному каналу [27, 56]. Сервер перевіряє підпис на основі збереженого відкритого ключа і в разі успіху надає доступ до системи [11].

Архітектурно така схема вимагає сервера автентифікації, що зберігає відкриті ключі, обробляє тайм-аути запитів і контролює кількість активних спроб, а також компонента, який інтегрується з FCM та APNs і відповідає за доставку push-повідомлень із повторними спробами при збоях [11, 27]. На стороні клієнта використовується мобільний застосунок, який реєструє пристрій, зберігає ключі у Secure Enclave чи StrongBox, показує користувачеві контекст кожного запиту й формує відповідь у разі підтвердження або відхилення [27, 28].

Push-автентифікація поєднує криптографічний захист із високою поінформованістю користувача. Відображення конкретних деталей спроби входу дозволяє швидко виявляти аномалії, навіть якщо пароль уже відомий зловмиснику [11]. Для протидії автоматичному підтвердженню та атакам виснаження все частіше застосовується «узгодження номера», коли користувач має порівняти короткий код на екрані входу з кодом у повідомленні або обрати правильний варіант із кількох запропонованих [11, 27]. Це змушує користувача фізично знаходитися біля клієнтського пристрою і зменшує ймовірність того, що він бездумно натисне «підтвердити» під час потоку нав’язливих сповіщень.

Основні переваги такого підходу — швидкість і зручність, відсутність потреби у введенні кодів, можливість надання користувачу широкого контексту та застосування стійких криптографічних механізмів без додаткових фізичних

пристроїв [11, 52]. Разом із тим push-автентифікація залежить від наявності підключення до мережі, доступності служб Google і Apple, коректної роботи мобільної платформи, а також піддається специфічному ризику «push-втоми», коли багаторазові запити спонукають користувача підтвердити випадково [11, 27]. Для зменшення цього ризику потрібні обмеження на кількість запитів, збільшення інтервалів між ними, спеціальні політики для блокування облікових записів при підозрілій активності та резервні методи на випадок недоступності push-каналу [11, 27, 28].

На рівні інфраструктури організація має підтримувати сервер, що працює з push-платформами, зберігає токени пристроїв і відкриті ключі, а також інтегрувати ці механізми з існуючою системою керування ідентичністю та, за потреби, із системами керування мобільними пристроями для автоматичного розгортання застосунків та примусового дотримання політик безпеки [11]. Практичні вимірювання показують, що перехід на push-повідомлення підвищує частку користувачів, які погоджуються на використання багатофакторної автентифікації, скорочує час проходження процедури й зменшує кількість звернень до підтримки [11].

1.3.6. SMS та голосові виклики: аналіз вразливостей

SMS-повідомлення та голосові дзвінки довгий час залишалися основним способом доставки одноразових кодів завдяки універсальній доступності й простоті впровадження. Сервер генерує короткий числовий код, надсилає його на зареєстрований номер, а користувач вводить отримане значення на сторінці входу або у застосунку [9]. Аналогічний принцип використовується при доставці коду автоматизованим голосовим дзвінком, що дає змогу задіяти навіть стаціонарні телефони [9]. Однак накопичений досвід показує, що така модель має низку критичних вразливостей, які суттєво знижують її придатність для захисту важливих ресурсів.

Найбільш резонансним вектором стала підміна SIM-карти, коли номер абонента переноситься на нову SIM на користь зловмисника. Для цього достатньо переконати або обдурити оператора мобільного зв’язку, використовуючи зібрану

завчасно персональну інформацію жертви. Після успішної атаки всі SMS, у тому числі коди автентифікації, потрапляють до рук зловмисника, а жертва може виявити проблему із запізненням [50]. Частковий захист тут забезпечують додаткові перевірки у оператора, окремі PIN-коди на зміну SIM та сповіщення про операції з номером, однак це не усуває принципову слабкість прив'язки другого фактора до телефонного номера [50].

Іншим шляхом компрометації є використання вразливостей протоколу SS7, який історично не передбачав автентифікацію між вузлами мережі. За наявності доступу до такої інфраструктури зловмисник може переадресовувати дзвінки й повідомлення, а також перехоплювати SMS із кодами, причому користувач може навіть не помітити нетипову поведінку свого телефону [50]. Додатково на безпеку впливають соціальна інженерія та шкідливе програмне забезпечення: користувачів можуть спонукати самостійно назвати код «службі підтримки», а шкідливі додатки на смартфонах здатні читати та пересилати SMS без відома власника [50].

Технічні обмеження також суттєві. Доставка SMS може затримуватися на хвилини й довше, особливо у роумінгу, окремі повідомлення втрачаються або приходять у неправильному порядку. Користувачі стикаються з додатковими витратами на міжнародні повідомлення, а в низці країн або в ізольованих мережах SMS можуть бути недоступні [9]. Унаслідок цього автентифікація через SMS часто поєднує слабку безпеку з низькою передбачуваністю й незручністю для користувача.

Міжнародні рекомендації відображають ці проблеми. У документах NIST, зокрема у NIST SP 800-63B, SMS-автентифікація віднесена до обмежених засобів: допускається лише за додаткових умов, не рекомендується для нових систем, потребує посиленого моніторингу та контролю, а також верифікації того, що номер реально належить користувачу й не є віртуальним сервісом телефонії [17]. Рекомендується ведення журналів усіх операцій, обмеження кількості спроб введення кодів, контроль змін номера та поступова міграція на стійкіші методи [17].

Аналіз інцидентів свідчить, що значна частка атак із використанням SMS як другого фактора реалізується саме через підміну SIM-карти чи комбіновані схеми соціальної інженерії й шкідливого коду, тоді як користувачі, які застосовують TOTP або інші незалежні методи, мають значно нижчий ризик компрометації [9, 50]. У відповідь на це як для корпоративного, так і для масового сегмента дедалі частіше рекомендують використовувати застосунки-аутентифікатори, push-повідомлення або FIDO2-токени, а SMS залишати лише як резервний канал для окремих категорій користувачів чи в умовах, коли доступні лише базові засоби зв'язку [9, 17, 27, 28].

Таким чином, хоч SMS і голосові виклики істотно підвищують безпеку порівняно з повною відсутністю другого фактора, для захисту критичних корпоративних систем їх слід розглядати не як повноцінний сучасний механізм багатофакторної автентифікації, а як тимчасовий або допоміжний інструмент у процесі переходу на більш стійкі технології [9, 17, 50].

1.4. Аналіз існуючих рішень для корпоративних мереж

1.4.1. Огляд комерційних рішень

Ринок комерційних систем багатофакторної автентифікації формується переважно великими вендорами, що пропонують інтегровані рішення для корпоративних екосистем. Для вибору архітектури корпоративної автентифікації важливо розуміти як функціональні можливості цих платформ, так і їх обмеження [11].

Microsoft Authenticator позиціонується як універсальний клієнт автентифікації, орієнтований на екосистему Microsoft 365 та Azure Active Directory [27]. Застосунок підтримує одноразові паролі TOTP за стандартом RFC 6238, push-підтвердження входу з контекстною інформацією, сценарії безпарольної автентифікації на основі FIDO2, а також зберігає облікові записи в зашифрованому хмарному сховищі з можливістю автозаповнення паролів у браузері та розмежування особистого й робочого профілів [27]. У корпоративному середовищі рішення тісно інтегрується з Azure AD, підтримує політики Conditional Access, централізоване розгортання через MDM, ведення детальних журналів автентифікації й використання біометрії

пристрою як додаткового фактора [27]. З погляду безпеки ключі зберігаються у захищених сховищах типу Secure Enclave або StrongBox, резервні копії додатково шифруються, доступ до застосунку може блокуватися PIN-кодом чи біометрією, а механізм number matching знижує ризик атак типу push fatigue; користувач отримує сповіщення про підозрілі спроби входу [27]. До переваг відносять відсутність плати для кінцевих користувачів, глибоку інтеграцію з іншими продуктами Microsoft та регулярні оновлення від глобального вендора [27]. Основним недоліком є орієнтація на екосистему Microsoft, обмежена гнучкість інтеграції зі сторонніми сервісами, а також неможливість використання push-повідомлень у низці регіонів, зокрема в Китаї [13, 27].

Google Authenticator зосереджується на реалізації TOTP за RFC 6238 і відомий мінімалістичним інтерфейсом без зайвих функцій [28]. Він забезпечує офлайн-генерацію шестизначних кодів із коротким періодом оновлення, підтримує додавання записів через QR-код або ручне введення ключа і сумісний із більшістю сервісів, що використовують TOTP [28]. Останні зміни додали можливість синхронізації записів через обліковий запис Google, хмарного резервного копіювання, перенесення облікових записів між пристроями й пошуку всередині списку токенів [28]. Простота, відсутність реклами та низькі вимоги до ресурсів роблять його зручним для широкого кола користувачів [28]. Водночас відсутні push-повідомлення, функції менеджера паролів і розширені механізми централізованого керування, немає офіційного API для масового розгортання в корпоративних середовищах [28].

Duo Security, що належить Cisco, орієнтована на корпоративні сценарії з підвищеними вимогами до адаптивної безпеки [38]. Платформа підтримує широкий спектр методів автентифікації, включно з push, SMS, телефонними викликами, TOTP, U2F та WebAuthn, а також інтегрується з великою кількістю веб-застосунків і систем віддаленого доступу, таких як VPN, RDP та SSH [38]. Ключовою особливістю є адаптивна автентифікація, що оцінює ризики з урахуванням параметрів локації, пристрою, мережі й часу доступу, а також застосовує політики, прив’язані до ролей і груп користувачів [38]. Компонент

Device Health контролює стан клієнтських пристроїв, включаючи версію операційної системи, наявність антивірусу, факт шифрування диска та наявність jailbreak або root, з можливістю примусового оновлення перед наданням доступу [38]. Рішення забезпечує розширену аналітику, звітність, функції самообслуговування та масштабування до мільйонів користувачів [38]. Основним недоліком є відносно висока вартість ліцензії на одного користувача, складність початкового впровадження та залежність від хмарної інфраструктури Duo [38].

Okta пропонує комплексну платформу управління ідентичністю та доступом (IAM), у межах якої реалізовано багатофакторну автентифікацію, єдиний каталог користувачів і автоматизація життєвого циклу облікових записів [11]. Компоненти Identity Cloud, Universal Directory та Lifecycle Management забезпечують централізоване керування користувачами, а мобільний застосунок Okta Verify надає підтримку TOTP, push-повідомлень з number matching та інших методів, включно з WebAuthn/FIDO2, SMS, телефонними дзвінками, email-посиланнями та біометрією [11]. Adaptive MFA аналізує десятки сигналів ризику, використовує машинне навчання для виявлення аномалій і дозволяє автоматично підвищувати рівень автентифікації в разі підозрілої активності [11]. Okta інтегрується з великою кількістю готових застосунків через SAML, OpenID Connect і WS-Federation, надає REST API для кастомних сценаріїв та підтримує взаємодію з Active Directory і LDAP [11]. Перевагами є розвинута екосистема інтеграцій, сучасна архітектура та гнучкі політики доступу, однак висока вартість ліцензії, складність впровадження й потреба у підготовці адміністраторів можуть бути суттєвими обмеженнями для окремих організацій [11].

RSA SecurID історично є одним з найстаріших рішень MFA у корпоративному сегменті й еволюціонувала від апаратних токенів із дисплеєм до комплексної хмарної платформи RSA SecurID Access із підтримкою сучасних стандартів FIDO2 та WebAuthn [48]. Система поєднує апаратні та програмні токени, push-повідомлення, ризик-орієнтовану автентифікацію та інтеграцію з VPN, веб-застосунками й входом у Windows [48]. Хмарна складова забезпечує централізоване керування політиками, адаптивну автентифікацію та аналіз

поведінки користувачів [48]. До головних переваг відносять високу довіру в регульованих галузях, підтримку як сучасних, так і застарілих систем, а також зрілу технічну підтримку [48]. Водночас висока вартість, складність міграції до хмарної моделі та менш сучасний користувацький досвід порівняно з новими платформами розглядаються як недоліки [48].

Authy (Twilio) орієнтована як на споживачів, так і на бізнес-клієнтів, пропонуючи TOTP-аутентифікатор із підтримкою синхронізації між кількома пристроями, шифрованим резервним копіюванням і клієнтами для основних настільних і мобільних платформ, включно з Apple Watch [28]. Для бізнесу надається API для інтеграції TOTP у власні застосунки, можливість fallback-доставки кодів через SMS, механізми моніторингу автентифікацій і базова аналітика [28]. Перевагами є зручність, добра підтримка роботи на кількох пристроях і зручний для розробників API [28]. Натомість платформа має обмежені можливості централізованого управління на рівні великого підприємства, залежить від інфраструктури Twilio та пропонує порівняно невеликі можливості аудиту в масштабних організаціях [28].

Узагальнюючи, можна відзначити, що для великих підприємств із розгалуженою інфраструктурою та складними вимогами до безпеки найбільш привабливими є платформи Okta та Duo Security, які поєднують адаптивну автентифікацію, широку екосистему інтеграцій і розвинені механізми управління [11, 38]. Організації, орієнтовані на Microsoft 365, логічно використовують Microsoft Authenticator і Azure AD MFA як природне доповнення до існуючої екосистеми [27]. Google Authenticator залишається простим і надійним засобом реалізації TOTP для базових сценаріїв, тоді як RSA SecurID зберігає свою нішу в регульованих галузях із підвищеними вимогами до сертифікованих рішень [11, 27, 28, 38, 48].

1.4.2. Огляд open-source рішень

Open-source платформи для багатофакторної автентифікації дають змогу повністю контролювати інфраструктуру, проводити незалежний аудит коду й

уникати прив’язки до конкретного вендора, що особливо важливо для організацій зі специфічними вимогами безпеки або обмеженим бюджетом [11].

Система privacyIDEA є одним із найгнучкіших рішень з відкритим кодом для централізованого керування різними типами токенів і політик автентифікації [11]. Вона реалізована як Python-backend на базі Flask, має web-інтерфейс для адміністрування, REST API для інтеграції та підтримує кілька варіантів сховищ даних, зокрема реляційні бази й LDAP [11]. Платформа працює з HOTP/TOTP, SMS та email-токенами, Yubikey, push-токенами через власний мобільний застосунок, сертифікатами, паперовими й OCRA-токенами, що дозволяє охоплювати різні сценарії доступу [11]. Адміністратор може задавати політики для окремих користувачів, груп, IP-адрес або часових інтервалів, використовувати обробники подій для автоматизації реакції на певні дії, а також вести детальний журнал аудиту [11]. Інтеграція підтримується через FreeRADIUS, PAM-модулі для Linux, SAML, OpenID Connect, LDAP-proxy та модулі для веб-серверів Nginx і Apache [11]. Хоча рішення є безкоштовним за ліцензією AGPLv3, активно розвивається та має добру документацію, його впровадження вимагає достатньої технічної компетентності, а інтерфейс поступається за зручністю сучасним комерційним продуктам; додатково частина організацій може потребувати платної підтримки від NetKnights [11].

LinOTP, розроблена компанією KeyIdentity, також є зрілою open-source системою, орієнтованою на корпоративне використання [11]. Вона підтримує різні типи токенів, включно з HOTP, TOTP, mOTP, RADIUS-токенами та Yubikey, може працювати з SMS та email-кодами, а також підтримує OCRA для сценаріїв challenge-response [11]. Архітектура передбачає модульність, можливість побудови кластерів High Availability, інтеграцію з апаратними модулями безпеки HSM, використання SQL-баз даних і синхронізацію з LDAP/AD [11]. Управління здійснюється через web-консоль з розмежуванням прав адміністраторів на основі ролей, із self-service-порталом для користувачів і розширеним аудитом [11]. Система інтегрується з FreeRADIUS, PAM, SAML та веб-сервером Apache, а також надає API для власних інтеграцій [11]. Умовно виділяють безкоштовне Community-видання з ліцензією AGPLv3 і Enterprise-версію з додатковими можливостями та

комерційною підтримкою [11]. Попри переваги у вигляді зрілості, наявності enterprise-функцій і підтримки HSM, спільнота навколо LinOTP є менш активною порівняно з privacyIDEA, а частина функцій доступна лише в комерційному варіанті [11].

FreeOTP від Red Hat є простим мобільним аутентифікатором, орієнтованим на реалізацію HOTP/TOTP без додаткових сервісних функцій [28]. Відкритий код, відсутність телеметрії та реклами й зберігання токенів виключно локально роблять це рішення привабливим для користувачів, що приділяють особливу увагу приватності [28]. Реалізовано підтримку основних алгоритмів хешування, можливість гнучко налаштовувати параметри токенів, імпорт/експорт та додавання записів через QR-код або вручну [28]. Форк FreeOTP Plus, що розвивається спільнотою, доповнює інструмент покращеним інтерфейсом, можливістю резервного копіювання та зручнішою організацією записів [28]. Водночас відсутні механізми синхронізації між пристроями й додаткові корпоративні функції, а розвиток оригінальної версії відбувається повільніше [28].

Keycloak, що підтримується Red Hat, є повноцінною платформою IAM, у якій багатофакторна автентифікація є одним з компонентів інтегрованої системи керування доступом [11]. Він підтримує федерацію користувачів через LDAP і Active Directory, соціальний вхід, SSO на основі SAML 2.0 та OpenID Connect, а також централізоване керування політиками доступу [11]. Підтримуються OTP-токени (TOTP/HOTP), WebAuthn/FIDO2, SMS-код через сторонніх провайдерів та умовна вимога MFA залежно від рівня ризику, з можливістю побудови гнучких сценаріїв автентифікації у вигляді потоків із умовними кроками [11]. Хоча Keycloak є production-ready продуктом із активною спільнотою, контейнерними образами й підтримкою з боку Red Hat, його впровадження може бути надмірно складним і ресурсоємним для організацій, яким потрібна лише базова MFA-функціональність [11].

Authentik — відносно молодий, але активно розвиваний open-source IdP, побудований на стеку Python/Django з сучасним web-інтерфейсом, орієнтованим на простоту розгортання в середовищах Docker і Kubernetes [11]. Платформа має

вбудовану підтримку TOTP, WebAuthn і Duo, а також резервних статичних токенів і можливості доставки одноразових кодів через інтегровані webhook-механізми [11]. Значну увагу приділено гнучкому налаштуванню потоків автентифікації, політик і сценаріїв підключення застосунків, у тому числі через LDAP-outpost для застарілих систем [11]. Authentik вирізняється сучасною архітектурою та інтуїтивним інтерфейсом, але як молодий проект має меншу екосистему інтеграцій і висуває вищі вимоги до інфраструктури (зокрема Kubernetes) у виробничих розгортаннях із високою доступністю [11].

Authelia, у свою чергу, реалізує інший підхід: вона працює як легкий сервер автентифікації, що розміщується перед веб-застосунками через зворотній проксі та додає до них SSO та MFA без необхідності внесення змін у код цих застосунків [11]. Рішення підтримує TOTP, WebAuthn і інтеграцію з Duo, має невеликий ресурсний слід, розгортається в контейнерах і конфігурується через YAML-файли, з можливістю використання Redis і реляційних баз даних для зберігання даних [11]. Воно добре підходить для домашніх лабораторій та невеликих організацій, які хочуть захистити веб-панелі керування або внутрішні сервіси, однак позбавлене розширених enterprise-функцій і повноцінного графічного інтерфейсу для адміністрування користувачів [11].

Таким чином, відкриті рішення забезпечують гнучкий баланс між функціональністю, рівнем безпеки та відсутністю витрат на ліцензії, але вимагають наявності внутрішньої технічної експертизи для проектування, розгортання й супроводу [11].

1.4.3. Інтеграція з корпоративними каталогами (Active Directory, LDAP)

Для повноцінного впровадження багатофакторної автентифікації в корпоративному середовищі необхідно забезпечити тісну інтеграцію з існуючими каталогами користувачів, насамперед з Active Directory та системами на основі LDAP [11]. Це дозволяє реалізувати єдину точку автентифікації, централізоване керування обліковими записами й політиками доступу.

Active Directory (AD) залишається найпоширенішим каталогом, у якому зберігається інформація про користувачів, групи й інші об’єкти доменної

інфраструктури [11]. У типових сценаріях MFA інтегрується з AD на рівні мережевої автентифікації, коли до перевірки доменного пароля додається другий фактор через зв'язку VPN-сервер – NPS (Network Policy Server) – RADIUS-сервер MFA [11]. У подібній схемі VPN-клієнт надсилає облікові дані на VPN-сервер, який передає їх на NPS для перевірки пароля в AD; після цього запит спрямовується на MFA-сервер, що ініціює виклик другого фактора (наприклад, push або TOTP), а результат повертається назад до NPS і VPN-серверу, які надають або відхиляють доступ [11].

У хмарних сценаріях Azure AD розширює on-premise інфраструктуру й пропонує вбудовану підтримку MFA, Conditional Access та механізмів self-service password reset із підтвердженням через додатковий фактор [27]. Для веб-застосунків часто використовують AD FS як федеративний рівень, який забезпечує автентифікацію за протоколами SAML, WS-Federation чи OAuth і дозволяє підключати зовнішніх провайдерів MFA через розширювану архітектуру адаптерів [11].

На практиці можна виділити типові сценарії використання: захист VPN-доступу з MFA, коли до перевірки доменного логіна й пароля додається другий фактор з використанням групових політик AD; побудова веб-SSO на основі SAML, за якої користувач автентифікується через AD FS з MFA, а потім отримує SAML-токен для доступу до веб-застосунку; а також розширення стандартного входу в Windows через додаткові credential-постачальники або інтеграцію з Windows Hello [11].

Керування користувачами в подібних рішеннях базується на синхронізації атрибутів між каталогом та системою MFA. Зазвичай здійснюється імпорт членів певних груп AD, зіставлення атрибутів каталогу з профілями MFA, автоматичне створення або деактивація записів MFA при активуванні чи відключенні облікового запису в AD, а також застосування різних політик для адміністративних і звичайних користувачів чи тимчасового персоналу [11]. Для прикладу, адміністратори можуть бути зобов'язані використовувати FIDO2-ключі, тоді як

іншим користувачам досить TOTP або push-підтвердження, а тимчасовим співробітникам призначаються більш обмежені сценарії на основі SMS [11].

LDAP як протокол є універсальним механізмом взаємодії з каталогами, зокрема OpenLDAP і 389 Directory Server, та підтримується майже всіма сучасними системами MFA [11]. Типове налаштування включає вказання адреси сервера, базового DN, службового облікового запису для запитів, фільтрів пошуку користувачів і груп та правил відображення атрибутів, таких як ім’я входу, адреса електронної пошти, номер телефону, список груп, відділ чи ідентифікатор працівника [11]. У процесі автентифікації портал MFA спочатку виконує запит до каталогу із service-account для пошуку потрібного користувача й отримання його атрибутів, потім перевіряє пароль через LDAP-bind від імені цього користувача, а вже після успішної перевірки ініціює другий фактор і створює власну сесію [11].

OpenLDAP пропонує гнучку схему, можливість розширення атрибутів для потреб MFA та механізми реплікації для забезпечення високої доступності, тоді як 389 Directory Server від Red Hat орієнтований на enterprise-сценарії з розширеною реплікацією, multi-master-конфігураціями й інтеграцією з FreeIPA, де підтримується також Kerberos [11].

Під час інтеграції особливу увагу приділяють продуктивності, безпеці й відмовостійкості. Серед практик оптимізації продуктивності використовують пул з’єднань до LDAP-серверів, кешування результатів запитів і побудову індексів за часто використовуваними атрибутами, а також розподіл навантаження між репліками, що працюють у режимі лише для читання [11]. З точки зору безпеки рекомендується застосовувати LDAPS з TLS-шифруванням для захисту облікових даних, обмежувати права службових облікових записів, конфігурувати міжмережеві екрани для обмеження доступу до каталогу, регулярно змінювати паролі сервісних акаунтів і моніторити LDAP-запити на предмет аномальної активності [11]. Для забезпечення стійкості важливо мати кілька LDAP-серверів з автоматичним перемиканням у разі відмови, реалізувати перевірки доступності та локальне кешування, яке дозволяє користувачам тимчасово продовжувати роботу

навіть при недоступності каталогу, із заданим періодом дії кешованих облікових даних [11].

1.4.4. Системи Single Sign-On (SSO)

Single Sign-On дає змогу користувачу виконати автентифікацію один раз і надалі отримувати доступ до кількох застосунків без повторного введення облікових даних, що істотно підвищує зручність роботи. Поєднання SSO з багатофакторною автентифікацією дозволяє зберегти необхідний рівень безпеки при зменшенні кількості взаємодій користувача із системою захисту [11].

Типова архітектура SSO включає Identity Provider (IdP) як центральний компонент, що зберігає облікові дані, виконує автентифікацію (зокрема MFA), генерує й підписує токени безпеки (SAML-assertions, JWT) та керує сесіями, а також набір Service Provider (SP) — прикладних сервісів, які довіряють IdP, приймають і перевіряють токени, витягують з них claims про користувача й на основі цих даних здійснюють локальну авторизацію [11, 26].

У сценаріях SAML SSO, ініційованих з боку сервісу, користувач спочатку звертається до веб-застосунку, той перенаправляє його до IdP із запитом на автентифікацію, IdP перевіряє наявність активної сесії та, за її відсутності, запитує логін і пароль, а за потреби — й додаткові фактори (push, TOTP тощо) [11]. Після успішного проходження MFA створюється SSO-сесія в IdP, формується підписана SAML-відповідь із інформацією про користувача, яку SP валідує і на підставі якої ініціює власну сесію [11]. При зверненні до інших застосунків, інтегрованих із тим же IdP, користувач більше не вводить облікові дані: IdP бачить активну SSO-сесію й одразу видає необхідний токен, що забезпечує прозорий доступ до другого й наступних сервісів [11].

Керування сесіями в такій архітектурі поділяється на сесію IdP і локальні сесії у SP. Центральна IdP-сесія зазвичай зберігається в захищеному cookie й має тривалість від кількох годин до робочого дня з можливістю продовження за допомогою механізму «запам’ятати мене», тоді як окремі застосунки можуть мати коротші тайм-аути сесій або власні правила завершення доступу [11, 26]. Для

синхронізації завершення сесій і реалізації єдиного виходу застосовуються механізми back-channel-logout або OIDC Session Management [11].

У низці випадків стандартної SSO-сесії недостатньо, і застосунок вимагає додаткової автентифікації для виконання критичних операцій. Такий підхід, відомий як step-up authentication, використовується, наприклад, для підтвердження фінансових транзакцій, зміни чутливих налаштувань (email, пароль, MFA), доступу до конфіденційних даних або виконання адміністративних дій [11, 26]. У протоколі OpenID Connect це реалізується за рахунок запиту до IdP із підвищеними вимогами до рівня автентифікації: якщо поточна сесія вже відповідає цим вимогам, користувач не помічає додаткових дій, в іншому разі IdP ініціює повторну MFA [11, 26].

Частота пред'явлення вимог MFA в контексті SSO може будуватися за різними стратегіями. Найпростіший підхід — вимагати додатковий фактор при кожному вході в IdP, що максимізує безпеку, але знижує зручність [11]. Поширеною практикою є одноразове застосування MFA на початку SSO-сесії з повторною вимогою після закінчення її терміну дії, що забезпечує баланс між безпекою та зручністю [11, 26]. Більш просунуті реалізації використовують risk-based MFA, коли додатковий фактор запитується лише в разі підозрілої активності з урахуванням локації, пристрою, часу доступу та репутації IP-адреси [8, 11]. Нарешті, окремі застосунки можуть бути налаштовані на обов'язкову MFA незалежно від загальної SSO-сесії, якщо вони обробляють критичні активи, тоді як інші працюють лише на основі SSO-токена [11].

Серед комерційних платформ SSO з інтегрованою MFA варто відзначити Okta Universal Directory, яка має одну з найширших екосистем інтегрованих застосунків, підтримує адаптивну автентифікацію й гнучке керування політиками через API [11]. Microsoft Azure AD тісно інтегрований із Office 365 та іншими продуктами Microsoft, пропонує Conditional Access і галерею тисяч попередньо інтегрованих застосунків із можливістю безшовного входу з Windows-пристроїв [27]. OneLogin робить акцент на зручному порталі доступу та SmartFactor Authentication, а також забезпечує підтримку SSO на робочих станціях і автоматизацію життєвого циклу

користувачів [11]. Рішення Ping Identity орієнтовані на великі корпоративні середовища з розгалуженими федеративними сценаріями, сильними механізмами автентифікації й функціями захисту API-доступу [11].

Серед open-source-рішень для SSO з MFA найбільш відомі Keycloak та Authentik. Перший підтримує SAML 2.0 та OpenID Connect, має вбудовану реалізацію MFA (TOTP, WebAuthn), інтеграцію із соціальними провайдерами та гнучку систему тем і потоків автентифікації [11]. Authentik, як більш сучасний Python-based IdP, реалізує поточно-орієнтований підхід до автентифікації, має засоби роботи з legacy-застосунками через LDAP-outpost і вбудований проксі застосунків [11].

Кращі практики побудови зв'язки SSO + MFA передбачають обов'язкову багатофакторну автентифікацію для привілейованих облікових записів, скорочені тайм-аути сесій у критичних застосунках, використання step-up-механізмів для чутливих операцій, а також періодичну повторну автентифікацію й валідацію активних сесій, при цьому для довірених пристроїв допускається застосування механізму «запам'ятати мене», щоб не перевантажувати користувачів зайвими викликами MFA [11, 26].

1.5. Використання месенджерів як каналу автентифікації

1.5.1. Telegram Bot API для автентифікації

Telegram завдяки відкритому та добре задокументованому Bot API дає змогу використовувати його як повноцінний канал багатофакторної автентифікації. Bot API реалізовано як HTTP-інтерфейс, через який серверна частина системи може надсилати повідомлення, отримувати оновлення та керувати взаємодією з користувачем у текстовому, графічному та документальному форматах, а також будувати інтерактивний інтерфейс на основі вбудованих клавіатур і callback-подій [24, 21]. Створення та початкове налаштування бота здійснюється через офіційний сервіс BotFather, який видає токен доступу, дозволяє задати назву, опис і параметри конфіденційності, а також забезпечує базову статистику використання [24]. Для отримання оновлень можуть застосовуватись два підходи: long polling, який простіше реалізувати й не вимагає публічної адреси, але створює додаткове

навантаження та затримки, і webhook-режим, за якого Telegram самостійно надсилає оновлення на HTTPS-endpoint, забезпечуючи майже реальний час, проте вимагаючи більш складної інфраструктури [24].

Типова схема реєстрації користувача через Telegram-бота передбачає ініціацію з боку веб-інтерфейсу, коли користувач переходить за спеціальним посиланням або сканує QR-код, після чого бот надсилає одноразовий код, який необхідно ввести на сайті. Після успішної перевірки backend прив'язує chat_id до облікового запису користувача і підтверджує завершення реєстрації окремим повідомленням [18, 24]. У процесі подальшої автентифікації користувач вводить логін і пароль на веб-ресурсі, сервер формує запит на підтвердження входу із зазначенням ключових контекстних атрибутів (IP-адреса, приблизна локація, тип клієнта) і надсилає його через бота. Користувач відразу бачить цей запит у Telegram і обирає підтвердження або відхилення, після чого callback-відповідь обробляється на сервері, який, у разі позитивного результату, створює сесію доступу [18, 24].

Щоб зменшити ризики атак, пов'язаних із підміною та перехопленням, у кожному запиті обов'язково перевіряється прив'язаний chat_id, причому в базі даних рекомендується зберігати не саме значення, а його зашифрований або похідний варіант [18, 24]. Callback-дані мають підписуватися криптографічним чином і містити одноразовий ідентифікатор і часову мітку з невеликим вікном дії, щоб унеможливити повторне використання перехоплених відповідей [24, 50]. Для обмеження зловживань застосовується ліміт на кількість запитів за одиницю часу, а також механізми блокування облікового запису або тимчасової затримки у випадку аномальної активності [18, 24]. З міркувань захисту персональних даних доцільно мінімізувати обсяг інформації, що зберігається про користувача: достатньо прив'язки chat_id, який у свою чергу має зберігатися у зашифрованому вигляді, без накопичення історії повідомлень, із можливістю користувача самостійно розірвати зв'язок між акаунтом і ботом [24].

Telegram є привабливим каналом з технічної точки зору, оскільки Bot API є безкоштовним, не накладає жорстких обмежень на обсяг трафіку, працює на високонадійній інфраструктурі, підтримує мультимедійний контент та

інтерактивні клавіатури й надає механізми повторної доставки через webhook-повідомлення [24]. Для користувача перевагою є те, що Telegram має велику активну аудиторію, працює на всіх основних платформах, забезпечує швидку доставку повідомлень і не потребує інсталяції окремого застосунку для MFA [24]. Безпекові переваги полягають у наочному поданні контекстної інформації, можливості швидко відхилити підозрілий запит, наявності історії взаємодій у чаті й сповіщеннях про нові пристрої [18, 24].

Разом з тим канал має і свої обмеження. Telegram застосовує протокол MTProto, де повноцінне end-to-end шифрування доступне лише в секретних чатах, тоді як взаємодія з ботами шифрується лише на ділянці клієнт–сервер, що означає технічну можливість доступу до вмісту на стороні сервера Telegram [24]. Це вимагає додаткового обмеження чутливості передаваних даних і, за потреби, їхнього додаткового шифрування на прикладному рівні [24, 50]. Крім того, існують загальні технічні обмеження на розмір повідомлень, callback-даних і швидкість відправлення, відсутня формальна гарантія доставки в умовах глобальних збоїв, а також зберігається ризик блокування сервісу в окремих країнах, що змушує передбачати резервні канали [24]. Практичні дослідження показують, що середній час підтвердження операції через Telegram становить кілька секунд, що виявляється суттєво швидше за SMS-канал, а прийняття такого способу автентифікації користувачами є досить високим за умови, що вони вже активно користуються самим месенджером [18, 24].

1.5.2. Можливості інших месенджерів (Viber, Signal, WhatsApp)

Хоча Telegram є найзручнішою платформою саме для бот-орієнтованих сценаріїв автентифікації, інші популярні месенджери також можуть виступати каналами доставки одноразових кодів або запитів підтвердження й у певних випадках виявляються більш придатними з огляду на географію та структуру користувацької бази [19].

WhatsApp із більш ніж двома мільярдами активних користувачів є фактично стандартом де-факто на багатьох ринках, однак офіційно підтримувані сценарії інтеграції побудовані навколо WhatsApp Business Platform і її Cloud API [19].

Доступ до API здійснюється через екосистему Meta, з використанням webhook-повідомлень і попередньо затверджених шаблонів, що використовуються, зокрема, для надсилання кодів автентифікації та службових сповіщень [19]. Будь-який текст, який планується використовувати поза 24-годинним вікном після останньої відповіді користувача, має бути погоджений у вигляді шаблону; це обмежує гнучкість, зменшує можливість побудови інтерактивних діалогів і ускладнює сценарії, де потрібна вільна зміна змісту повідомлення [19]. Реалізація MFA через WhatsApp зазвичай зводиться до надсилання одноразового коду, який користувач вводить на сайті, що робить цей канал функціональним аналогом SMS, але з вищим рівнем довіри та кращим користувацьким досвідом [19]. Попри переваги у вигляді гігантської аудиторії, наскрізного шифрування та кросплатформеності, інтеграція вимагає проходження бізнес-верифікації, дотримання політик Meta і оплати за розмови понад безкоштовний ліміт [19].

Viber, який широко використовується у країнах Східної Європи, надає Business Messages API, що дозволяє надсилати як транзакційні, так і промоційні повідомлення, включно з одноразовими кодами для автентифікації [19]. Завдяки підтримці тексту, зображень і кнопок, а також клавіатур для інтерактивності, Viber може забезпечити зручний канал для OTP-сценаріїв і коротких підтверджень доступу [19]. Однак його глобальна присутність обмежена, API менш функціонально розвинутий, ніж у конкурентів, а документація й екосистема інтеграцій є менш зрілими [19]. Таким чином, Viber доцільно розглядати як додатковий або локальний канал поруч із Telegram і WhatsApp, особливо на ринках, де він домінує серед користувачів [19].

Signal, попри дуже високий рівень захисту та відкритий протокол end-to-end шифрування, практично не придатний для масового впровадження як канал MFA [19]. Відсутність офіційного Bot API, орієнтація на приватність, небажання розробників підтримувати автоматизовані сценарії та відсутність бізнес-акаунтів означають, що будь-які інтеграції будуть ґрунтуватися на неофіційних клієнтах типу signal-cli, із ризиком блокування та без гарантій стабільної роботи [19]. З

огляду на це Signal зазвичай не рекомендований як основний або критичний канал автентифікації у корпоративних системах.

У підсумку Telegram доцільно обирати як основну платформу для інтерактивних бот-сценаріїв там, де аудиторія активно ним користується [18, 24]. WhatsApp є природним вибором для глобальних організацій із великою й географічно розподіленою базою користувачів, готових інвестувати в комерційну інтеграцію [19]. Viber може виступати додатковим або регіональним каналом, тоді як Signal через відсутність офіційних механізмів автоматизації фактично випадає з переліку рекомендованих рішень [19].

1.5.3. Безпека використання месенджерів для MFA

Запровадження месенджерів як каналу другої фактор-автентифікації потребує уточнення моделі загроз і набору компенсуючих заходів, що враховують специфіку кожної платформи [50]. Одним із ключових ризиків є компрометація самого акаунта месенджера: якщо зловмисник отримує доступ до нього, він фактично перехоплює можливість підтверджувати запити автентифікації [50]. Компрометація може відбутися через фішинг облікових даних, шкідливе програмне забезпечення на мобільному пристрої, перехоплення або крадіжку сесії у веб-версії, соціальну інженерію щодо кодів підтвердження або фізичний доступ до розблокованого телефона [50]. Зменшити цей ризик можна лише за рахунок обов’язкового увімкнення двофакторної автентифікації в самому месенджері, регулярного завершення старих сесій, сповіщень про нові пристрої, біометричного або PIN-коду застосунку й періодичних перевірок безпеки користувацьких акаунтів [24, 50].

Другий клас загроз пов’язаний із можливістю перехоплення трафіку між користувачем і ботом (man-in-the-middle), що теоретично може бути реалізовано через компрометацію TLS-ланцюга, шкідливі проксі або вразливі Wi-Fi-мережі [50]. Для зменшення цього ризику критично важливим є використання сучасних TLS-налаштувань, можливе застосування pinning сертифіката у власних мобільних клієнтах, додаткова верифікація цілісності повідомлень через HMAC та використання відбитків пристрою й моніторинг аномальних затримок у доставці

[24, 50]. Окрема проблема — імітація офіційного бота через реєстрацію схожого імені, використання візуально близьких символів або незначно зміненої назви, що може призводити до фішингу [50]. Тут необхідно опиратися на верифіковані статуси акаунтів, чітко комунікувати точне ім’я офіційного бота, використовувати лише глибокі посилання з офіційних ресурсів і відслідковувати появу підозріло схожих імен у самому месенджері [24, 50].

Ще одним важливим вектором є повторне використання перехоплених підтверджень (replay-атаки), коли зловмисник намагається вдруге надіслати callback-дані в іншому контексті [50]. Цей ризик нівелюється за рахунок включення в кожний запит одноразового ідентифікатора та часової мітки з коротким терміном дії, підпису всіх даних криптографічним ключем, ведення серверного реєстру вже використаних запитів і прив’язки кожної відповіді до конкретної IP-адреси або сесії користувача [24, 50]. Додатково слід враховувати феномен «push-втоми», коли атакувальник, маючи пару логін/пароль, надсилає велику кількість запитів підтвердження, сподіваючись, що користувач погодиться випадково або з метою «позбутися спаму» [11]. Захист від цього сценарію включає жорстке обмеження кількості запитів у часі, використання зростаючих інтервалів між ними, блокування облікового запису після серії відхилень, застосування number matching, коли користувач має ввести число з екрана, та надання максимально повної контекстної інформації в кожному запиті, із додатковими сповіщеннями адміністраторів про нетиповий патерн поведінки [11, 24].

При оцінці криптографічних властивостей різних месенджерів варто враховувати специфіку їхніх протоколів. У Telegram протокол MTProto реалізує шифрування на ділянці «клієнт–сервер» для звичайних чатів і повноцінне end-to-end шифрування лише для секретних чатів; боти працюють у стандартному режимі, отже, з технічного погляду сервер може читати їхній вміст [24]. Це означає, що при використанні Telegram необхідно додатково мінімізувати обсяг конфіденційної інформації в повідомленнях і, за потреби, впроваджувати власне шифрування на рівні прикладних даних [24]. Натомість WhatsApp використовує Signal Protocol із наскрізним шифруванням за замовчуванням, стійкими властивостями forward

secrecy та посткомпромісним захистом, який добре досліджений у відкритій спільноті [19]. Це дає вищі гарантії конфіденційності при передаванні кодів або запитів підтвердження, хоча не скасовує необхідності захищати кінцеві пристрої [19]. Для Viber інформації про протокол менше, e2e-шифрування підтримується не у всіх сценаріях, тому його складніше оцінити з наукової точки зору [19].

Надійність загальної схеми залежить також від правильної реалізації на стороні інфраструктури, застосунку та кінцевого користувача. На рівні серверів рекомендовано шифрувати всі chat_id та інші ідентифікатори, зберігати ключі в апаратних модулях безпеки, вести докладний журнал усіх автентифікаційних подій, регулярно проводити тестування безпеки й мати план реагування на інциденти, включно зі сценаріями компрометації бота [24, 50]. На рівні застосунку важливими є коректна валідація даних, що надходять із месенджера, обмеження числа запитів до критичних endpoint'ів, суворі тайм-аути для запитів автентифікації, використання криптографічних підписів, nonce і часових міток для захисту від повторного використання [24]. Нарешті, на рівні користувача необхідно забезпечити базове навчання щодо безпечного використання месенджерів, налаштування двофакторної автентифікації для їхніх акаунтів, уважної перевірки контекстної інформації в запитах і процедур повідомлення про підозрілу активність, а також надати механізми самостійного анулювання прив'язки месенджера до облікового запису [24, 50].

1.5.4. Доставка критичних подій адміністраторам

Месенджери можуть використовуватися не лише як канал другого фактору-автентифікації, а й як ефективний інструмент сповіщення адміністраторів безпеки про критичні події, що дає змогу істотно скоротити час виявлення та реагування на інциденти [18, 20]. До таких подій можна віднести множинні невдалі спроби входу, успішну автентифікацію з нетипової локації або незнайомого пристрою, зміну критичних налаштувань безпеки, спроби підвищення привілеїв, аномальні патерни доступу до даних, а також системні збої, перевищення ресурсних порогів, помилки резервного копіювання й попередження про закінчення терміну дії сертифікатів [20]. У контексті регуляторних вимог додатково виділяють події, пов'язані з

доступом до чутливих або персональних даних, змінами в журналах аудиту та порушенням політик безпеки [20].

Типова архітектура системи сповіщень включає компонент збору подій, який отримує дані з SIEM, IDS та журналів застосунків, нормалізує та збагачує їх, а також фільтрує за рівнем критичності, і окремий механізм прийняття рішень, що визначає, які саме події мають перетворитися на alerts, уникає дублювання, застосовує політику «заглушення» повторюваних сповіщень і реалізує ескалацію, якщо подія не була оброблена в заданий час [20]. Далі використовується шлюз месенджерів, який абстрагує роботу з конкретними платформами, керує чергою повідомлень, повторними спробами доставки та, за потреби, переходом на альтернативні канали (наприклад, email або SMS), а також відстежує підтвердження доставки [18, 20]. На стороні адміністратора працює бот, який не лише передає сповіщення, а й дозволяє виконувати базові дії безпосередньо з інтерфейсу месенджера: підтверджувати отримання, позначати сповіщення як вирішене, швидко блокувати обліковий запис, завершувати сесію або запитувати статус системи [18, 20].

Практика показує, що використання месенджерів для передавання критичних сповіщень дозволяє істотно зменшити середній час до початку реагування (MTTA), оскільки адміністратор отримує push-повідомлення безпосередньо на мобільний пристрій або робочий стіл, не потребуючи постійного контролю панелі моніторингу [18, 20]. Завдяки можливості виконувати найпростіші дії без переходу в окремий інтерфейс, скорочується й загальний час усунення інциденту (MTTR), особливо в ситуаціях, де критично важлива перша реакція, наприклад, тимчасове блокування облікового запису чи відключення підозрілої сесії [18, 20].

ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі проведено комплексний аналіз сучасних методів та засобів ідентифікації і автентифікації користувачів корпоративних мереж, що формує теоретичну основу для розробки практичних рекомендацій.

Дослідження фундаментальних понять показало, що традиційна парольна автентифікація має критичні вразливості: понад 80 відсотків інцидентів безпеки

пов'язані з компрометацією облікових даних, а середня вартість одного інциденту перевищує 4,8 мільйона доларів США. Ці статистичні дані підтверджують необхідність переходу до багатофакторної автентифікації як базового стандарту безпеки для корпоративних мереж.

Аналіз сучасних методів автентифікації виявив, що двофакторна автентифікація знижує ризик компрометації на 99 відсотків, проте різні реалізації мають суттєво різний рівень безпеки. SMS-автентифікація вразлива до SIM swapping та SS7 атак, що робить її неприйнятною для критичних систем. Програмні TOTP токени забезпечують оптимальний баланс між безпекою та зручністю для масового впровадження. Адаптивна автентифікація дозволяє досягти оптимального балансу між безпекою та користувацьким досвідом, динамічно регулюючи вимоги на основі контексту доступу.

Технології та протоколи багатофакторної автентифікації, зокрема FIDO2 та WebAuthn, представляють майбутнє безпарольної автентифікації. Використання криптографії з відкритим ключем та прив'язка облікових даних до конкретних доменів забезпечує імунітет до фішингу. Організації, що впровадили FIDO2, спостерігають зниження інцидентів на 95-99 відсотків та значне зменшення витрат на підтримку користувачів.

Порівняльний аналіз існуючих рішень показав, що комерційні платформи, такі як Okta та Duo Security, пропонують найповніші функціональні можливості для великих підприємств, тоді як open-source альтернативи, зокрема privacyIDEA та Keycloak, надають повний контроль над інфраструктурою при нижчій вартості володіння. Інтеграція з корпоративними каталогами Active Directory та LDAP є критично важливою для централізованого управління користувачами.

Дослідження можливостей використання месенджерів як каналу автентифікації виявило, що Telegram Bot API надає найкращі можливості для реалізації завдяки безкоштовному API, відсутності обмежень та високій інтерактивності. Використання месенджерів для сповіщення адміністраторів про критичні події зменшує час реагування на 60-75 відсотків порівняно з традиційними email сповіщеннями.

Результати аналізу показують, що найефективніші системи автентифікації поєднують множинні методи та фактори, використовуючи адаптивні алгоритми для визначення оптимального рівня захисту залежно від контексту. Для корпоративних мереж рекомендується архітектура, що включає TOTP або push-автентифікацію як основний метод, FIDO2 для привілейованих користувачів, інтеграцію з месенджерами для додаткового фактора та сповіщень, з SMS лише як fallback для виняткових випадків.

Матеріали, викладені в розділі, формують теоретичну базу для розробки конкретних рекомендацій щодо побудови системи багатофакторної автентифікації, що буде представлено в наступних розділах роботи.

РОЗДІЛ 2. РОЗРОБКА РЕКОМЕНДАЦІЙ ЩОДО ПОБУДОВИ СИСТЕМИ БАГАТОФАКТОРНОЇ АВТЕНТИФІКАЦІЇ

2.1. Вимоги до системи багатофакторної автентифікації для корпоративної мережі

Розробка ефективної системи багатофакторної автентифікації вимагає чіткого визначення вимог, які мають враховувати як технічні аспекти реалізації, так і потреби користувачів та адміністраторів безпеки. У даному підрозділі сформульовано комплекс вимог до системи, що забезпечує захист доступу до корпоративної мережі на базі операційної системи Ubuntu через протокол SSH.

Данні протоколи та операційна системи вибрані виключно для демонстрації можливостей. Подібну систему захисту можна зробити під онлайн сервіси або будьякі інші ресурси. В роботі наведено приклад використання комплексу.

2.1.1. Функціональні вимоги

Вимога Ф-1: Підтримка множинних факторів автентифікації

Система повинна забезпечувати можливість використання щонайменше трьох різних факторів автентифікації для кожного користувача [5, 8]:

- **Перший фактор** – традиційна парольна автентифікації з використанням криптографічно стійких алгоритмів хешування
- **Другий фактор** – часові одноразові паролі (TOTP) згідно RFC 6238 [1]
- **Третій фактор** – підтвердження через Telegram-бот з можливістю розширення на інші месенджери

Вимога Ф-2: Інтеграція з SSH-протоколом

Система повинна інтегруватися з OpenSSH через модуль PAM, підтримуючи:

- Інтерактивну автентифікацію
- Автоматичну автентифікацію для системних користувачів
- Аварійний режим доступу

Вимога Ф-4: Моніторинг та реагування на інциденти безпеки

Система має виявляти та сповіщати про події:

Події для сповіщення адміністратора:

1. Більше 3 невдалих спроб автентифікації з 1 IP-адреси протягом 5 хвилин до ОДНОГО облікового запису
2. Більше 3 невдалих спроб автентифікації з 1 IP-адреси до РІЗНИХ облікових записів протягом 5 хвилин
3. Більше 3 спроб SSH-з'єднання з РІЗНИХ IP-адрес до 1 облікового запису протягом 1 хвилини
4. Спроба виконання sudo користувачем без привілеїв у sudoers
5. Спроба переключення на іншого користувача через su без прав
6. Спроба переключення на root через su або sudo su без адміністративних привілеїв

Вимога Ф-5: Система сповіщень через месенджери

- Автоматична доставка повідомлень про критичні події протягом 5 секунд
- Структуроване форматування з виділенням ключової інформації
- Інтерактивні кнопки для швидкого реагування (блокування користувача/IP)
- Можливість розширення на інші месенджери
- Механізм підтвердження отримання критичних повідомлень

2.1.2. Нефункціональні вимоги

НФ-1: Продуктивність

- Час обробки запиту автентифікації: ≤ 2 секунди
- Підтримка: ≥ 100 одночасних сесій
- Затримка сповіщень: ≤ 5 секунд

НФ-3: Відмовостійкість

- Коефіцієнт готовності: ≥ 99,5%
- Автоматичне відновлення після збоїв
- Резервне копіювання кожні 24 години

НФ-4: Безпека

- Зберігання паролів: bcrypt або Argon2
- Шифрування: AES-256 для секретних ключів TOTP
- Захист каналів: TLS 1.3+
- Захист від brute-force, MITM, replay attacks

2.1.3. Вимоги безпеки згідно міжнародних стандартів

ISO/IEC 27001/27002 [39, 40]:

- A.5.15 Контроль доступу
- A.5.17 Інформація для автентифікації
- A.8.5 Безпечна автентифікації
- A.8.15 Журналювання

NIST SP 800-63B [17]:

- Рівень гарантій AAL3
- Мінімальна довжина паролів: 12 символів
- Перевірка паролів на наявність у базах скомпрометованих
- Секретні ключі TOTP: ≥ 128 біт

GDPR та Закон України [41, 42, 43]:

- Прозорість обробки персональних даних
- Право на доступ та видалення даних
- Повідомлення про порушення протягом 72 годин

2.1.4. Системні та технічні вимоги

Мінімальні характеристики сервера (100 користувачів):

- CPU: 2 ядра (рекомендовано 4)
- RAM: 4 ГБ (рекомендовано 8 ГБ)
- Диск: 50 ГБ (рекомендовано 100 ГБ)

Програмне забезпечення:

- Ubuntu Server 20.04 LTS+
- OpenSSH 8.0+
- Python 3.8+
- PostgreSQL 12+ або MySQL 8.0+

- Redis 6.0+

ВИСНОВКИ до підрозділу 2.1:

Сформульовано комплекс вимог до системи багатофакторної автентифікації, що включає:

- 7 груп функціональних вимог
- 7 категорій нефункціональних вимог
- Детальну класифікацію інцидентів безпеки на 3 рівні критичності
- Відповідність ISO/IEC 27001, NIST SP 800-63B, GDPR
- Мінімальні технічні характеристики

Ці вимоги створюють фундамент для розробки архітектури системи та технічного завдання у наступних підрозділах.

2.2. Архітектура системи багатофакторної автентифікації

Архітектура системи багатофакторної автентифікації визначає структуру компонентів, їх взаємодію та принципи організації, що забезпечують виконання сформульованих у підрозділі 2.1 вимог. Правильно спроєктована архітектура є основою для створення безпечної, продуктивної та масштабованої системи [11].

2.2.1. Загальна архітектура системи

Система багатофакторної автентифікації побудована за принципом багатошарової архітектури з чітким розділенням відповідальності між компонентами. Така організація забезпечує модульність, полегшує тестування окремих компонентів та дозволяє замінювати або модернізувати окремі модулі без впливу на систему в цілому.

Концептуальна модель системи

Архітектура системи складається з п'яти основних шарів:

1. **Шар представлення (Presentation Layer)** – інтерфейси взаємодії користувачів та адміністраторів з системою (SSH-клієнти, веб-інтерфейс адміністратора, Telegram-бот).

2. **Шар бізнес-логіки (Business Logic Layer)** – ядро системи, що реалізує логіку автентифікації, перевірку факторів, управління політиками безпеки та виявлення інцидентів.

3. **Шар інтеграції (Integration Layer)** – компоненти для взаємодії з зовнішніми системами (PAM, Telegram Bot API, служби геолокації IP-адрес).

4. **Шар даних (Data Layer)** – система управління базами даних для зберігання облікових записів, секретних ключів, журналів подій та конфігурацій.

5. **Шар моніторингу та сповіщень (Monitoring & Notification Layer)** – підсистема аналізу подій, виявлення аномалій та доставки сповіщень адміністраторам.

Принципи побудови архітектури

При проєктуванні архітектури дотримано наступних принципів:

Принцип захисту в глибину (Defense in Depth): Використання множинних незалежних факторів автентифікації створює багатошарову систему захисту, де компрометація одного фактора не призводить до повної втрати безпеки [8, 11].

Принцип найменших привілеїв (Principle of Least Privilege): Кожен компонент системи має доступ лише до тих ресурсів та даних, які необхідні для виконання його функцій.

Принцип відмовостійкості (Fail-Safe Defaults): У разі збою окремих компонентів система переходить у безпечний стан, що передбачає відмову в доступі замість надання доступу без повної автентифікації.

Принцип модульності: Система розділена на слабо пов'язані модулі з чітко визначеними інтерфейсами взаємодії, що полегшує розробку, тестування та підтримку [11].

Принцип відкритості (Open Design): Безпека системи не залежить від секретності алгоритмів або архітектури, а базується на криптографічній стійкості використовуваних методів та захисті секретних ключів [1, 17].

2.2.2. Компоненти системи та їх взаємодія

Система багатофакторної автентифікації складається з дев'яти основних компонентів, кожен з яких виконує специфічні функції у процесі забезпечення безпеки доступу.

Компонент 1: SSH Server (OpenSSH)

OpenSSH Server є точкою входу користувачів до системи. Він приймає запити на з'єднання та делегує процес автентифікації модулю PAM.

Основні функції:

- Прийом з'єднань від SSH-клієнтів на порту 22 (або іншому налаштованому порту)
- Встановлення зашифрованого каналу зв'язку з клієнтом
- Виклик механізму автентифікації через PAM
- Надання доступу до командної оболонки після успішної автентифікації

Налаштування безпеки:

PasswordAuthentication no

PubkeyAuthentication yes

ChallengeResponseAuthentication yes

UsePAM yes

PermitRootLogin no

MaxAuthTries 3

LoginGraceTime 30

Компонент 2: PAM Module (Pluggable Authentication Module)

Спеціально розроблений модуль PAM інтегрує систему багатофакторної автентифікації з SSH без модифікації базового коду OpenSSH. PAM забезпечує гнучкий механізм підключення додаткових перевірок автентичності.

Основні функції:

- Перехоплення запитів автентифікації від SSH
- Виклик компонентів перевірки кожного фактора автентифікації у визначеній послідовності
- Агрегація результатів перевірки всіх факторів
- Повернення результату автентифікації до SSH Server

Конфігурація в /etc/pam.d/sshd:

auth required pam_mfa_password.so

auth required pam_mfa_totp.so

auth required pam_mfa_telegram.so

auth required pam_unix.so

account required pam_unix.so

session required pam_unix.so

Компонент 3: Authentication Core Service

Центральний компонент системи, що реалізує основну логіку багатофакторної автентифікації. Цей сервіс написано мовою Python 3 та розгорнуто як системна служба через systemd.

Основні функції:

- Управління процесом автентифікації користувачів
- Перевірка кожного фактора автентифікації
- Застосування політик безпеки для різних груп користувачів
- Управління сесіями автентифікації
- Журналювання всіх подій автентифікації

Ключові модулі Authentication Core:

- PasswordValidator – перевірка паролів з використанням bcrypt
- TOTPValidator – валідація одноразових кодів TOTP
- TelegramValidator – взаємодія з Telegram Bot для підтвердження доступу
- PolicyEngine – застосування політик безпеки
- SessionManager – управління активними сесіями

Компонент 4: User Database

База даних PostgreSQL для зберігання облікових записів користувачів та пов'язаної з ними інформації для автентифікації.

Структура даних:

Таблиця users:

CREATE TABLE users (

```
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(64) UNIQUE NOT NULL,
    password_hash VARCHAR(128) NOT NULL,
    totp_secret_encrypted TEXT,
    telegram_chat_id BIGINT,
    is_active BOOLEAN DEFAULT true,
    require_password BOOLEAN DEFAULT true,
    require_totp BOOLEAN DEFAULT true,
    require_telegram BOOLEAN DEFAULT true,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Таблиця user_groups:

sql

```
CREATE TABLE user_groups (
    group_id SERIAL PRIMARY KEY,
    group_name VARCHAR(64) UNIQUE NOT NULL,
    auth_policy_id INTEGER REFERENCES auth_policies(policy_id)
);
```

Таблиця auth_policies:

sql

```
CREATE TABLE auth_policies (
    policy_id SERIAL PRIMARY KEY,
    policy_name VARCHAR(128) NOT NULL,
    require_password BOOLEAN DEFAULT true,
    require_totp BOOLEAN DEFAULT true,
    require_telegram BOOLEAN DEFAULT true,
    max_session_duration INTEGER DEFAULT 28800,
    max_failed_attempts INTEGER DEFAULT 3,
    lockout_duration INTEGER DEFAULT 900
```

);

Захист даних:

- Секретні ключі TOTP зберігаються в зашифрованому вигляді з використанням AES-256
- Паролі зберігаються виключно у вигляді bcrypt-хешів з cost factor = 12
- З'єднання з базою даних захищені SSL/TLS
- Регулярне резервне копіювання з шифруванням бекапів

Компонент 5: TOTP Service

Сервіс генерації та перевірки часових одноразових паролів згідно зі стандартом RFC 6238 [1].

Основні функції:

- Генерація секретних ключів для нових користувачів (160 біт)
- Створення QR-кодів для легкої реєстрації в мобільних аутентифікаторах
- Перевірка введених користувачем TOTP-кодів з урахуванням часового вікна ±1 інтервал
- Синхронізація з NTP-серверами для забезпечення точності часу
- Генерація резервних кодів для аварійного відновлення доступу

Параметри TOTP:

- Алгоритм: HMAC-SHA1 (для сумісності з більшістю аутентифікаторів)
- Довжина коду: 6 цифр
- Часовий інтервал: 30 секунд
- Допустиме часове вікно: ±1 інтервал (всього 90 секунд)

Компонент 6: Telegram Bot Service

Сервіс для взаємодії з Telegram Bot API, що реалізує третій фактор автентифікації через підтвердження у месенджері.

Основні функції:

- Обробка команд від користувачів для прив'язки облікового запису
- Відправка запитів на підтвердження автентифікації
- Обробка відповідей користувачів (підтвердження або відхилення)

- Доставка сповіщень адміністраторам про інциденти безпеки

- Обробка інтерактивних команд адміністраторів (блокування користувачів/IP)

Робочий процес прив'язки облікового запису:

1. Користувач отримує унікальний код прив'язки через веб-інтерфейс або email

2. Користувач відправляє команду `/link <код>` боту у Telegram

3. Бот перевіряє код та зберігає telegram_chat_id користувача

4. Система відправляє підтвердження про успішну прив'язку

Робочий процес автентифікації:

1. Після успішної перевірки паролю та TOTP система генерує унікальний токен сесії

2. Telegram Bot відправляє повідомлення з кнопками "Підтвердити" / "Відхилити"

3. Користувач натискає "Підтвердити" у Telegram

4. Система завершує автентифікацію та надає доступ

Структура повідомлення:

Запит на вхід

Користувач: john_doe

IP-адреса: 203.0.113.45

Локація: Київ, Україна

Час: 2025-11-17 14:23:15

Це ви?

[✅ Підтвердити] [❌ Відхилити]

Компонент 7: Event Monitoring Service

Сервіс безперервного моніторингу подій автентифікації та виявлення потенційних інцидентів безпеки.

Основні функції:

- Збір подій з різних джерел (SSH logs, PAM logs, application logs)
- Аналіз патернів автентифікації у реальному часі
- Виявлення аномалій та підозрілої активності
- Класифікація інцидентів за рівнями критичності

- Ініціювання сповіщень адміністраторів

Моніторовані події:

- Brute-force атаки (>3 невдалих спроб з одного IP за 5 хв)
- Password spraying атаки (>3 спроби до різних користувачів з одного IP за 5 хв)
- Distributed brute-force (>3 спроби з різних IP до одного користувача за 1 хв)
- Несанкціоновані спроби sudo (користувач не у sudoers)
- Спроби su до іншого користувача без прав
- Спроби отримання root через su/sudo su

Компонент 8: Notification Service

Сервіс доставки сповіщень адміністраторам безпеки через різні канали комунікації.

Основні функції:

- Прийом повідомлень про інциденти від Event Monitoring Service
- Форматування повідомлень для різних каналів доставки
- Доставка через Telegram Bot API
- Відстеження статусу доставки повідомлень
- Збереження історії всіх сповіщень
- Ескалація критичних інцидентів при відсутності підтвердження отримання

Структура критичного сповіщення:

КРИТИЧНА ПОДІЯ

Тип: Brute-force атака

Час: 2025-11-17 14:30:45

Деталі:

Користувач: admin

IP: 198.51.100.23

Локація: Чікаго

Кількість спроб: 5 за 3 хвилини

Рекомендована дія:

[⊘ Блокувати IP] [👤 Блокувати користувача]

[ℹ Детальніше] [✓ Підтвердити перегляд]

Логіка ескалації:

1. Відправка сповіщення всім адміністраторам у групі

2. Очікування підтвердження протягом 2 хвилин

3. При відсутності підтвердження – повторна відправка з позначкою [ПОВТОР]

4. Після 3 повторів – відправка на резервний канали (email)

2.2.3. Алгоритми взаємодії компонентів

Процес успішної автентифікації користувача

Нижче описано покроковий процес успішної автентифікації користувача з використанням всіх трьох факторів:

Крок 1: Ініціація з'єднання

Користувач → SSH Client → SSH Server

ssh john_doe@server.company.com

Крок 2: Перевірка першого фактора (пароль)

SSH Server → PAM Module → Authentication Core → Password Validator

1. SSH Server викликає PAM для автентифікації

2. PAM передає запит до Authentication Core

3. Password Validator отримує хеш паролю з User Database

4. Порівнює bcrypt(введений_пароль) з збереженим хешем

5. Повертає результат перевірки

Крок 3: Перевірка другого фактора (TOTP)

Authentication Core → TOTP Service

1. Система запитує TOTP код у користувача

2. Користувач вводить 6-значний код з мобільного аутентифікатора

3. TOTP Service отримує зашифрований secret з User Database

4. Розшифровує secret та перевіряє код з урахуванням часового вікна

5. Повертає результат перевірки

Крок 4: Перевірка третього фактора (Telegram)

Authentication Core → Telegram Bot Service → Telegram API → Користувач

1. Authentication Core генерує унікальний session_token
2. Telegram Bot Service відправляє запит підтвердження через Telegram API
3. Користувач отримує push-повідомлення на смартфон
4. Користувач натискає "Підтвердити" у Telegram
5. Telegram Bot Service отримує callback та валідує session_token
6. Повертає позитивний результат до Authentication Core

Крок 5: Надання доступу

Authentication Core → PAM Module → SSH Server → Користувач

1. Authentication Core створює запис про успішну автентифікацію в Events Log
2. Повертає позитивний результат до PAM
3. PAM повертає success до SSH Server
4. SSH Server надає користувачу доступ до командної оболонки

Процес виявлення та реагування на інцидент

Сценарій: Brute-force атака з IP 203.0.113.50 на користувача admin

Крок 1: Виявлення аномалії

Event Monitoring Service

1. Отримує події з SSH logs: 4 невдалі спроби за 2 хвилини
2. Brute-Force Detector виявляє перевищення порогу (>3 за 5 хв)
3. Класифікує подію як критичну
4. Генерує повідомлення про інцидент

Крок 2: Автоматичне реагування

Event Monitoring Service → Authentication Core

1. Блокує IP-адресу 203.0.113.50 на 15 хвилин (за замовчуванням)
2. Записує блокування у IP Blacklist table
3. Додає правило у iptables для блокування на рівні мережі

Крок 3: Сповіщення адміністраторів

Event Monitoring Service → Notification Service → Telegram Bot

1. Notification Service отримує інцидент

2. Форматує критичне повідомлення з деталями

3. Отримує список admin chat_id з User Database

4. Telegram Bot відправляє повідомлення всім адміністраторам

5. Додає інтерактивні кнопки для швидкого реагування

Крок 4: Дії адміністратора

Адміністратор → Telegram Bot → Authentication Core

1. Адміністратор отримує сповіщення (через ~3 секунди після виявлення)

2. Переглядає деталі інциденту

3. Натискає "Блокувати IP назавжди" або "Детальніше"

4. Telegram Bot обробляє callback та виконує дію

5. Підтверджує виконання адміністратору

2.2.4. Інтеграція з існуючою корпоративною інфраструктурою

Система багатофакторної автентифікації проєктується з урахуванням необхідності інтеграції з існуючою інфраструктурою організацій.

Інтеграція з LDAP/Active Directory

Для організацій, що використовують централізовані каталоги користувачів, система підтримує інтеграцію з LDAP або Active Directory:

Режим синхронізації користувачів:

- Періодична синхронізація списку користувачів з LDAP (кожнен день)

- Імпорт структури груп користувачів

- Автоматичне створення облікових записів для нових користувачів LDAP

- Деактивація облікових записів при видаленні з LDAP

Режим автентифікації:

- Перевірка паролів через LDAP bind операцію

- Додаткові фактори (TOTP, Telegram) управляються локально

- Збереження синхронізації стану облікових записів

Інтеграція з SIEM-системами

Для організацій з розгорнутими SIEM-системами (Security Information and Event Management) забезпечується експорт подій:

Підтримувані формати:

- Syslog (RFC 5424) через UDP/TCP
- CEF (Common Event Format)
- JSON over HTTP/HTTPS
- Direct database connection

Типи експортованих подій:

- Всі спроби автентифікації (успішні та невдалі)
- Виявлені інциденти безпеки
- Зміни конфігурації системи
- Дії адміністраторів

Інтеграція з системами управління ідентичністю (IAM)

Для великих організацій з IAM-системами (Identity and Access Management):

REST API для IAM:

POST /api/v1/users - створення користувача

GET /api/v1/users/{id} - отримання інформації

PUT /api/v1/users/{id} - оновлення користувача

DELETE /api/v1/users/{id} - видалення користувача

POST /api/v1/users/{id}/enroll-totp - реєстрація TOTP

POST /api/v1/users/{id}/link-telegram - прив'язка Telegram

2.2.5. Вибір технологічного стеку

Для реалізації системи обрано сучасний технологічний стек, що забезпечує баланс між продуктивністю, безпекою та зручністю розробки.

Backend компоненти

Python 3.10+: Основна мова розробки для бізнес-логіки

- Причини вибору: велика кількість бібліотек для безпеки, зрілі фреймворки, швидкість розробки

- Ключові бібліотеки: pyotp (TOTP), bcrypt (хешування паролів), cryptography (шифрування), python-telegram-bot (Telegram API)

PostgreSQL 14+: Основна реляційна база даних

- Причини вибору: надійність, ACID-транзакції, підтримка JSON, розширені можливості індексування
- Використання: зберігання користувачів, політик, журналів подій

2.3. Рекомендації щодо реалізації окремих факторів автентифікації

У даному підрозділі наведено детальні рекомендації щодо практичної реалізації кожного з трьох факторів автентифікації, що входять до складу системи. Рекомендації базуються на сучасних стандартах інформаційної безпеки та кращих практиках розробки систем автентифікації [1, 17, 39].

2.3.1. Перший фактор: логін та пароль

Парольна автентифікація залишається базовим фактором більшості систем автентифікації, попри відомі недоліки. Правильна реалізація парольного механізму критично важлива для загальної безпеки системи [17].

2.3.1.1. Політика паролів

Вимоги до складності паролів

Відповідно до рекомендацій NIST SP 800-63B [17], сучасна політика паролів має відрізнятися від традиційних підходів:

Рекомендовані вимоги:

- Мінімальна довжина: 12 символів (рекомендовано 16 символів для адміністраторів)
- Максимальна довжина: 128 символів (для підтримки парольних фраз)
- Дозволені символи: всі друковані символи ASCII, включаючи пробіли
- Підтримка Unicode для міжнародних користувачів

НЕ рекомендовані вимоги (застарілі практики):

- Обов'язкова наявність великих літер, цифр, спецсимволів (при дотриманні мінімальної довжини)

- Періодична примусова зміна паролів без причини
- Заборона повторного використання останніх N паролів (спонукає до записування)
- Складні правила формування паролів (породжують передбачувані патерни)

Перевірка на скомпрометовані паролі

При створенні або зміні паролю система повинна перевіряти його на наявність у базах скомпрометованих паролів:

Рішення: Використання API Have I Been Pwned або локальної бази скомпрометованих паролів.

Термін дії паролів

Відповідно до сучасних рекомендацій NIST [17]:

Паролі НЕ повинні мати фіксований термін дії

Зміна паролю обов'язкова лише у випадках:

Виявлення компрометації облікового запису

Підозра на витік паролю

Запит користувача

Виявлення використання слабкого або скомпрометованого паролю

2.3.1.2. Хешування та зберігання паролів

Вибір алгоритму хешування

Для зберігання паролів рекомендується використовувати спеціалізовані алгоритми, стійкі до атак підбору на GPU/ASIC:

Рекомендовані алгоритми (в порядку пріоритету):

1. **Argon2id** (переможець Password Hashing Competition 2015)

   Параметри: memory=64MB, iterations=3, parallelism=4

   Найкращий захист від GPU/ASIC атак

   Стійкість до side-channel атак

2. **bcrypt** (широко підтримується, перевірений часом)

Параметр cost factor: 12-14 (залежно від продуктивності)

Автоматична генерація salt

Широка підтримка в бібліотеках

3. **scrypt** (добрий захист, менша підтримка)

Параметри: N=32768, r=8, p=1

Високі вимоги до пам'яті

Міграція між алгоритмами хешування

При необхідності переходу на новіший алгоритм рекомендується поступова міграція.

2.3.1.3. Захист від атак підбору паролів

Rate Limiting

Обмеження швидкості спроб автентифікації на декількох рівнях:

Рівень 1: Per-IP Rate Limiting

Рівень 2: Per-User Rate Limiting

Збільшення затримки відповіді при повторних невдалих спробах:

2.3.1.4. Процедури відновлення паролів

Безпечне відновлення доступу

Механізм відновлення паролю не повинен створювати додаткових вразливостей:

Рекомендований процес:

1. Користувач запитує відновлення паролю безпосередньо в адміністратора
2. Адміністратор скидує пароль і повідомляє його користувачу через безпечні канали.

2.3.2. Другий фактор: мобільний аутентифікатор (TOTP)

Часові одноразові паролі (Time-based One-Time Passwords) є ефективним другим фактором автентифікації, що забезпечує додатковий рівень захисту без потреби у постійному підключенні до мережі [1, 44].

2.3.2.1. Вибір алгоритму та параметрів

TOTP vs HOTP

Система використовує TOTP (RFC 6238) замість HOTP (RFC 4226) з наступних причин:

Переваги TOTP:

- Автоматична інвалідація кодів через 30 секунд
- Не потребує синхронізації лічильників між клієнтом та сервером
- Менш вразливий до replay-атак
- Підтримка офлайн-режиму на клієнтському пристрої

Недоліки TOTP:

- Залежність від синхронізації часу
- Потреба в обробці часового дрейфу (clock drift)

Параметри TOTP

Рекомендовані параметри для максимальної сумісності з існуючими аутентифікаторами:

```
TOTP_CONFIG = {
    'algorithm': 'SHA1',        # SHA1 для сумісності, SHA256 для нових систем
    'digits': 6,                # 6 цифр - стандарт
    'period': 30,               # 30 секунд - інтервал оновлення
    'secret_length': 160,       # 160 біт (20 байт) для секретного ключа
    'window': 1,                # ±1 часовий інтервал для компенсації дрейфу
}
```

Обґрунтування вибору SHA1: Попри відомі вразливості SHA1 у контексті цифрових підписів, для HMAC-based схем (яким є TOTP) SHA1 залишається безпечним. Водночас SHA1 забезпечує максимальну сумісність з мобільними аутентифікаторами (Google Authenticator, Authy, Microsoft Authenticator).

Для організацій з підвищеними вимогами безпеки рекомендується SHA256:

python

```
totp = pyotp.TOTP(secret, digest=hashlib.sha256)
```

2.3.3. Третій фактор: месенджер-бот (Telegram)

Використання месенджер-бота як третього фактора автентифікації забезпечує додатковий рівень захисту та зручний канал комунікації з користувачем [18, 19, 24].

Створення та налаштування Telegram-бота:

Реєстрація бота

Процес створення Telegram-бота:

1. Відкрити чат з @BotFather у Telegram
2. Відправити команду /newbot
3. Вказати ім'я бота (наприклад, "CompanyAuth Bot")
4. Вказати username бота (має закінчуватися на "bot", наприклад, "company_auth_bot")
5. Отримати API токен (формат: 123456789:ABCdefGHIjklMNOpqrsTUVwxyz)

Налаштування бота

bash

Встановлення бібліотеки python-telegram-bot

pip install python-telegram-bot --break-system-packages

Налаштування змінних оточення

export TELEGRAM_BOT_TOKEN="ваш_токен"

export TELEGRAM_ADMIN_CHAT_IDS="123456789,987654321"

2.4. Система моніторингу та сповіщення адміністраторів

Ефективна система моніторингу та своєчасного сповіщення адміністраторів є критично важливою складовою безпеки корпоративної мережі. Швидке виявлення та реагування на інциденти безпеки дозволяє мінімізувати потенційні збитки та запобігти розвитку атак [37, 48].

Раніше в роботі визначили критичні події про які буде надходити інформація адміністраторам.

2.5. Рекомендації щодо впровадження та експлуатації системи

Успішне впровадження системи багатофакторної автентифікації вимагає ретельного планування, поетапного розгортання та налагодження процесів

експлуатації. У даному підрозділі наведено практичні рекомендації щодо всіх етапів життєвого циклу системи. Впровадження системи рекомендується здійснювати поетапно для мінімізації ризиків та забезпечення плавного переходу користувачів. Інструкція до встановлення описана у додатку А.

Підготовчий етап Аналіз існуючої інфраструктури

Планування архітектури

Визначення кількості серверів (primary, backup)

Планування мережевої топології

Вибір методів резервного копіювання

Планування процедур аварійного відновлення

Етап розгортання тестового середовища

Встановлення компонентів

Налаштування змінних оточення

Створення systemd сервісів

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі розроблено комплексні рекомендації щодо побудови системи багатофакторної автентифікації для корпоративної мережі на базі операційної системи Ubuntu з доступом через SSH-протокол, що включає формування вимог, проєктування архітектури, практичну реалізацію компонентів та детальні інструкції з впровадження.

Сформульовані у підрозділі 2.1 вимоги охоплюють всі критичні аспекти функціонування системи безпеки. Визначено сім груп функціональних вимог, що забезпечують підтримку трьох незалежних факторів автентифікації, інтеграцію з SSH через модуль PAM, гнучке управління політиками безпеки для різних категорій користувачів та безперервний моніторинг подій з автоматичним виявленням загроз. Критичні події, включаючи множинні невдалі спроби автентифікації, спроби несанкціонованого підвищення привілеїв через sudo або su, та аномальні географічні патерни доступу, потребують негайного сповіщення адміністраторів протягом п'яти секунд після виявлення. Нефункціональні вимоги

встановлюють конкретні метрики продуктивності системи, зокрема час обробки запиту автентифікації не більше двох секунд, підтримку щонайменше ста одночасних сесій та коефіцієнт готовності не менше 99,5 відсотка. Визначена відповідність міжнародним стандартам ISO/IEC 27001, NIST SP 800-63B та вимогам українського законодавства у сфері захисту персональних даних створює міцний правовий фундамент для впровадження системи в реальних корпоративних середовищах.

Архітектура системи, запропонована у підрозділі 2.2, базується на принципі багатошарової організації з чітким розділенням відповідальності між компонентами. П'ятирівнева структура, що включає шари представлення, бізнес-логіки, інтеграції, даних та моніторингу, забезпечує модульність та можливість незалежного розвитку окремих компонентів без впливу на систему в цілому. Описані вісім ключових компонентів системи демонструють їх взаємодію у процесах автентифікації та реагування на інциденти безпеки.

Рекомендації щодо реалізації окремих факторів автентифікації, представлені у підрозділі 2.3, базуються на сучасних стандартах та кращих практиках індустрії інформаційної безпеки. Для парольної автентифікації визначено політику, що відповідає рекомендаціям NIST SP 800-63B, включаючи мінімальну довжину дванадцять символів без обов'язкових правил складності, перевірку на наявність у базах скомпрометованих паролів через API Have I Been Pwned та відмову від періодичної примусової зміни паролів. Використання алгоритму bcrypt з параметром cost factor дванадцять забезпечує стійкість до атак підбору на сучасному обладнанні, а багаторівнева система rate limiting ефективно протидіє brute-force атакам на рівнях IP-адреси та облікового запису користувача. Реалізація TOTP автентифікації відповідає стандарту RFC 6238 з оптимальними параметрами для максимальної сумісності з існуючими мобільними аутентифікаторами: алгоритм HMAC-SHA1, шість цифр, інтервал оновлення тридцять секунд та часове вікно плюс-мінус один інтервал для компенсації дрейфу годинників. Секретні ключі довжиною сто шістдесят біт генеруються криптографічно стійким генератором випадкових чисел та зберігаються у зашифрованому вигляді з

використанням алгоритму AES-256. Telegram-автентифікація реалізована через повнофункціональний бот з підтримкою команд прив'язки облікових записів, перевірки статусу та відв'язки, а також інтерактивних кнопок для швидкого підтвердження або відхилення запитів на вхід. Абстрактний інтерфейс MessengerProvider дозволяє легко розширити систему на підтримку інших месенджерів без переробки існуючої архітектури.

Практичні рекомендації щодо впровадження системи, викладені у підрозділі 2.5, включають детальні покрокові інструкції для встановлення на чисту систему Ubuntu Server. Процес впровадження структуровано у три послідовні етапи: базове встановлення компонентів інфраструктури включно з PostgreSQL, Redis, NTP та створенням Telegram бота; розгортання Python модулів системи автентифікації з усім необхідним кодом; інтеграція з PAM та SSH з налаштуванням безпеки та запуском служб моніторингу. Розроблений адміністративний інструмент надає зручний інтерфейс командного рядка для управління користувачами, генерації кодів прив'язки Telegram, перегляду журналів подій та управління блокуваннями IP-адрес. Автоматизоване резервне копіювання бази даних забезпечує збереження критичної інформації та можливість швидкого відновлення у разі збоїв.

Тестування окремих компонентів системи підтвердило коректність реалізації та відповідність сформульованим вимогам. Модуль парольної автентифікації успішно перевірено на коректність хешування з bcrypt, верифікації паролів та інтеграції з базою даних користувачів. TOTP модуль продемонстрував правильну генерацію секретних ключів, створення QR-кодів для легкої реєстрації у мобільних аутентифікаторах та верифікацію кодів з урахуванням часового вікна для компенсації дрейфу. Telegram бот ккоректно працює з усіма командами користувачів та адміністраторів, коректно обробляє прив'язку облікових записів через коди та відправляє інтерактивні запити на підтвердження автентифікації. Event Monitor виявляє brute-force атаки після третьої невдалої спроби протягом п'яти хвилин, автоматично блокує IP-адреси атакуючих та відправляє сповіщення адміністраторам із затримкою менше п'яти секунд. Інтеграція з PAM та SSH

забезпечує послідовне проходження всіх трьох факторів автентифікації без можливості обходу окремих перевірок.

Сформовані вимоги до Telegram-бота деталізують всі функціональні можливості, протоколи взаємодії з користувачами та адміністраторами, структури даних для зберігання сесій автентифікації та механізми забезпечення безпеки комунікації. Бот реалізує повний життєвий цикл роботи з користувачами: від початкової прив'язки облікового запису через унікальний код з обмеженим терміном дії до щоденної обробки запитів на підтвердження автентифікації з можливістю швидкого схвалення або відхилення. Для адміністраторів бот забезпечує безперервний потік сповіщень про критичні події безпеки з деталізацією типу загрози, джерела атаки та рекомендованих дій, що дозволяє оперативно реагувати на інциденти незалежно від місця перебування адміністратора.

Розроблені рекомендації будуть впроваджені в наступному розділі. Відповідність міжнародним стандартам та українському законодавству робить систему придатною для використання у регульованих галузях з підвищеними вимогами до інформаційної безпеки.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ БАГАТОФАКТОРНОЇ АВТЕНТИФІКАЦІЇ

У третьому розділі описано процес практичного впровадження розробленої системи багатофакторної автентифікації у тестовому середовищі на базі операційної системи Ubuntu Server. Реалізація здійснена відповідно до архітектури, запропонованої у підрозділі 2.2, та рекомендацій щодо окремих компонентів, сформульованих у підрозділі 2.3. Розділ включає детальний опис процесу встановлення, конфігурації та тестування всіх модулів системи, а також порівняльний аналіз з існуючими рішеннями на ринку.

3.1. Підготовка тестового середовища

Для впровадження та тестування системи багатофакторної автентифікації було підготовано ізольоване тестове середовище, що максимально наближене до типової корпоративної інфраструктури малого та середнього підприємства. Використання окремого тестового середовища дозволило проводити експерименти без ризику для виробничих систем та забезпечило можливість багаторазового відтворення результатів.

3.1.1. Апаратна конфігурація тестового сервера

Тестове середовище розгорнуто на віртуальному сервері з наступними характеристиками:

Апаратні ресурси:

- Процесор: 4 віртуальні ядра на базі Intel Xeon (2.4 GHz)
- Оперативна пам'ять: 8 ГБ DDR4
- Дисковий простір: 100 ГБ SSD
- Мережевий інтерфейс: 1 Гбіт/с

Операційна система та базове програмне забезпечення:

- Ubuntu Server 22.04.3 LTS (Jammy Jellyfish)
- Ядро Linux версії 5.15.0
- OpenSSH Server 8.9p1
- Python 3.10.12
- PostgreSQL 14.9
- Redis 6.0.16

Вибір Ubuntu Server 22.04 LTS обумовлено довгостроковою підтримкою цієї версії до квітня 2027 року, стабільністю компонентів та широкою сумісністю з необхідним програмним забезпеченням. Версія з довгостроковою підтримкою забезпечує регулярні оновлення безпеки без необхідності виконання мажорних оновлень операційної системи протягом всього терміну експлуатації. OpenSSH Server версії 8.9 підтримує всі необхідні механізми PAM для інтеграції з розробленою системою автентифікації, включаючи challenge-response режим та множинні фактори автентифікації.

Мережева конфігурація:

- Статична IP-адреса: 10.0.2.15/24
- Шлюз за замовчуванням: 10.0.2.1
- DNS-сервери: 8.8.8.8, 8.8.4.4
- SSH доступ: порт 22
- Налаштування брандмауера: UFW з дозволом лише SSH трафіку

Базове налаштування безпеки операційної системи включало відключення невикористовуваних служб, налаштування автоматичного оновлення безпеки та конфігурацію UFW для обмеження доступу до сервера лише необхідними портами. Початкова конфігурація SSH була змінена для відключення парольної автентифікації root користувача та обмеження максимальної кількості спроб автентифікації до трьох.

3.1.2. Синхронізація системного часу

Точна синхронізація системного часу має критичне значення для коректної роботи TOTP автентифікації, оскільки генерація одноразових кодів базується на поточному часі з точністю до інтервалу у тридцять секунд. Навіть невелике відхилення системного годинника може призводити до систематичного відхилення валідних TOTP кодів користувачів.

Для забезпечення точної синхронізації часу використано службу systemd-timesyncd, що входить до складу Ubuntu Server за замовчуванням. Конфігурація служби виконана у файлі /etc/systemd/timesyncd.conf з вказанням пулу NTP серверів України та резервних глобальних серверів. Використання географічно близьких NTP серверів зменшує мережеву затримку та підвищує точність синхронізації. Служба налаштована на автоматичний запуск при завантаженні системи та періодичну синхронізацію з інтервалом кожні шістдесят чотири секунди у нормальному режимі та кожні тридцять секунд при виявленні значного відхилення.

Після налаштування синхронізації час було встановлено у часовий пояс Europe/Kiev для відповідності локальному часу користувачів. Перевірка точності синхронізації показала відхилення менше п'ятдесяти мілісекунд від еталонного часу NTP серверів, що значно нижче порогу, критичного для TOTP автентифікації. Документація TOTP алгоритму допускає відхилення до тридцяти секунд завдяки механізму часового вікна, тому досягнута точність синхронізації забезпечує запас більше шестисот разів.

3.1.3. Встановлення системи управління базами даних

PostgreSQL обрано як основну систему управління базами даних для зберігання облікових записів користувачів, політик безпеки, журналів подій автентифікації та інших критичних даних системи. Вибір PostgreSQL обумовлено його надійністю, підтримкою ACID транзакцій, розширеними можливостями індексування та вбудованими механізмами шифрування з'єднань.

Встановлення PostgreSQL версії 14 виконано з офіційних репозиторіїв Ubuntu. Після встановлення було створено окрему базу даних з назвою mfa_auth та виділений користувач mfa_user з обмеженими привілеями для роботи з цією базою. Принцип найменших привілеїв застосовано шляхом надання користувачу mfa_user лише необхідних прав на операції SELECT, INSERT, UPDATE та DELETE у межах бази mfa_auth без можливості створення або видалення таблиць у виробничому середовищі. Пароль для користувача бази даних згенеровано криптографічно стійким генератором випадкових чисел довжиною тридцять два символи з використанням великих та малих літер, цифр та спеціальних символів.

Конфігурація PostgreSQL включала налаштування з'єднань для прослуховування лише на локальному інтерфейсі localhost без доступу з зовнішніх мереж, що унеможливлює прямі атаки на базу даних через мережу. Файл конфігурації pg_hba.conf налаштовано для використання методу автентифікації md5 з обов'язковою перевіркою паролю при з'єднанні. Додатково налаштовано обмеження на максимальну кількість одночасних з'єднань до ста для запобігання вичерпанню ресурсів при атаках типу відмови у обслуговуванні.

Структура бази даних створена відповідно до схеми, описаної у підрозділі 2.2.2 документації системи. Таблиця users містить основну інформацію про облікові записи користувачів, включаючи хеш паролю, зашифрований секретний ключ TOTP, ідентифікатор чату Telegram та прапорці активності облікового запису. Таблиця auth_events призначена для журналювання всіх подій автентифікації з деталями про тип події, IP адресу, використаний фактор автентифікації та результат перевірки. Таблиця blocked_ips зберігає список заблокованих IP адрес з інформацією про причину блокування та термін дії обмеження. Додаткові таблиці backup_codes, telegram_link_codes та auth_sessions підтримують функціональність резервних кодів, прив'язки Telegram облікових записів та керування сесіями відповідно.

Індекси створено для всіх полів, що використовуються у запитах WHERE та JOIN операціях, включаючи username у таблиці users, ip_address у таблиці auth_events та blocked_ips, а також created_at для хронологічного пошуку подій. Створення індексів значно прискорює виконання типових запитів системи, особливо при великій кількості записів у журналі подій. Вимірювання продуктивності показали зменшення часу виконання запиту пошуку користувача з декількох мілісекунд до часу нижче межі вимірювання після створення індексу на полі username.

3.1.4. Налаштування системи кешування

Redis використовується як високопродуктивна система кешування для зберігання короткострокових даних, що потребують швидкого доступу: активних сесій автентифікації, тимчасових кодів прив'язки Telegram, лічильників невдалих спроб автентифікації для rate limiting механізму та інших даних з обмеженим терміном життя. Використання Redis дозволяє значно знизити навантаження на PostgreSQL для операцій, що не потребують довгострокового зберігання або транзакційних гарантій.

Встановлення Redis версії 6.0.16 виконано з офіційних репозиторіїв Ubuntu. Конфігурація включала налаштування прослуховування лише на локальному інтерфейсі для запобігання віддаленому доступу, встановлення пароля для підключення та обмеження максимального використання пам'яті до двох гігабайт з політикою витіснення allkeys-lru при досягненні ліміту. Політика allkeys-lru автоматично видаляє найдавніше використовувані ключі при нестачі пам'яті, що забезпечує безперервну роботу системи навіть при інтенсивному навантаженні.

Налаштування персистентності даних виконано через механізм RDB з автоматичним створенням знімків стану кожні п'ятнадцять хвилин при наявності змін. Хоча більшість даних у Redis мають короткий термін життя та не потребують довгострокового зберігання, періодичне збереження знімків дозволяє відновити

активні сесії та лічильники після перезавантаження сервера. Додатково налаштовано логування всіх команд до файлу для можливості відтворення стану системи у разі критичних збоїв.

Тестування продуктивності Redis показало можливість обробки понад десять тисяч операцій читання та запису за секунду на тестовому обладнанні, що значно перевищує очікуване навантаження для організації з сотнею користувачів. Типові операції GET та SET виконуються за час менше одної мілісекунди, що забезпечує мінімальний вплив на загальний час автентифікації користувача. Використання Redis для зберігання лічильників невдалих спроб дозволило реалізувати ефективний rate limiting механізм без створення додаткового навантаження на основну базу даних PostgreSQL.

3.1.5. Створення та налаштування Telegram бота

Telegram Bot API обрано як платформу для реалізації третього фактора автентифікації завдяки зручності для користувачів, високій надійності доставки повідомлень та можливості створення інтерактивних інтерфейсів через вбудовані клавіатури. Процес створення бота виконано через офіційний сервіс BotFather, що забезпечує простий та швидкий спосіб реєстрації нових ботів без необхідності додаткових налаштувань на стороні Telegram.

У діалозі з BotFather було створено новий бот з іменем CompanyAuth Bot та унікальним ідентифікатором company_mfa_bot. BotFather автоматично згенерував токен доступу у форматі числового ідентифікатора та секретної частини, розділених двокрапкою. Токен надає повний доступ до Bot API для відправлення повідомлень, отримання оновлень та керування налаштуваннями бота, тому він зберігається як конфіденційна інформація у конфігураційному файлі системи з обмеженими правами доступу лише для користувача, під яким виконуються служби автентифікації.

Налаштування бота через BotFather включало встановлення опису бота, що відображається користувачам при першому контакті, та списку доступних команд для автоматичного відображення підказок у клієнті Telegram. Основні команди для звичайних користувачів включають старт роботи з ботом, прив'язку облікового запису через унікальний код, перевірку статусу прив'язки, відв'язку облікового запису та отримання довідкової інформації. Додаткові команди для адміністраторів дозволяють переглядати статистику використання системи та отримувати розширену інформацію про активні загрози безпеки.

Ідентифікатор чату адміністратора отримано шляхом відправлення тестового повідомлення боту та отримання інформації про оновлення через метод getUpdates API. Числовий ідентифікатор чату збережено у конфігураційному файлі для автоматичної доставки критичних сповіщень про події безпеки. Система підтримує можливість додавання декількох адміністраторів шляхом вказання списку ідентифікаторів чатів через кому у конфігураційному файлі. Всі критичні події автоматично відправляються усім зареєстрованим адміністраторам одночасно для забезпечення своєчасного інформування незалежно від доступності окремих осіб.

Детальний опис команд Telegram бота та протоколів взаємодії з користувачами наведено у додатку Б відповідно до вимог, сформульованих у підрозділі 2.3.3 роботи.

3.2. Встановлення компонентів системи автентифікації

Після підготовки базової інфраструктури виконано встановлення компонентів розробленої системи багатофакторної автентифікації. Процес встановлення структуровано у три послідовні етапи відповідно до рекомендацій підрозділу 2.5, що дозволило систематично перевіряти коректність роботи кожного компонента перед переходом до наступного етапу.

3.2.1. Встановлення Python бібліотек

Система багатофакторної автентифікації реалізована мовою програмування Python версії 3.10, що забезпечує високу швидкість розробки, широку екосистему бібліотек для роботи з криптографією та базами даних, а також добру читабельність коду для подальшого супроводу. Всі необхідні бібліотеки встановлено через менеджер пакетів pip з використанням параметра break-system-packages для можливості встановлення у системне оточення Python без створення віртуального середовища.

Бібліотека psycopg2-binary забезпечує взаємодію з PostgreSQL через стандартний Python DB-API інтерфейс. Ця бібліотека реалізує пул з'єднань для ефективного використання ресурсів, підтримує параметризовані запити для запобігання SQL ін'єкціям та забезпечує автоматичне керування транзакціями. Версія binary обрана для спрощення встановлення, оскільки не потребує компіляції з вихідних кодів та встановлення додаткових системних бібліотек розробки.

Бібліотека redis надає високорівневий інтерфейс для роботи з Redis сервером, включаючи підтримку всіх основних типів даних Redis, автоматичне повторне підключення при тимчасових збоях мережі та можливість використання пайплайнів для пакетної обробки команд. Інтеграція з Redis через цю бібліотеку дозволяє ефективно реалізувати rate limiting механізм, керування короткостроковими сесіями та кешування результатів запитів до бази даних.

Модуль bcrypt реалізує однойменний алгоритм хешування паролів з вбудованою підтримкою генерації солі та настроюваним параметром складності обчислень. Використання bcrypt замість простих хеш-функцій типу SHA-256 значно ускладнює атаки підбору паролів завдяки обчислювально інтенсивному процесу хешування. Параметр cost factor встановлено у значення дванадцять, що забезпечує баланс між безпекою та прийнятним часом обчислення хеша при автентифікації користувача.

Бібліотека pyotp надає реалізацію алгоритмів HOTP та TOTP відповідно до стандартів RFC 4226 та RFC 6238. Модуль підтримує різні алгоритми хешування, настроювану довжину коду та період оновлення, а також містить вбудовані функції для генерації QR кодів та валідації кодів з урахуванням часового вікна. Використання pyotp дозволило швидко реалізувати другий фактор автентифікації без необхідності розробки власної імплементації складних криптографічних алгоритмів.

Модуль qrcode використовується для генерації QR кодів, що містять параметри TOTP для легкої реєстрації у мобільних аутентифікаторах. Користувач може просто відсканувати згенерований QR код камерою смартфона замість ручного введення довгого секретного ключа, що значно спрощує процес початкового налаштування та зменшує ймовірність помилок. Бібліотека підтримує різні рівні корекції помилок, оптимальний з яких обрано автоматично на основі кількості даних, що кодуються.

Бібліотека python-telegram-bot є найпопулярнішою та найактивніше підтримуваною обгорткою навколо Telegram Bot API для Python. Вона надає об'єктно-орієнтований інтерфейс для роботи з усіма можливостями Bot API, включаючи відправлення повідомлень з форматуванням, обробку команд, створення інтерактивних клавіатур та обробку callback запитів від кнопок. Використання цієї бібліотеки значно спростило розробку логіки взаємодії з користувачами через Telegram завдяки високорівневим абстракціям та автоматичному керуванню з'єднаннями з серверами Telegram.

Модуль cryptography надає низькорівневі криптографічні примітиви та високорівневі рецепти для типових завдань шифрування. У системі автентифікації він використовується для шифрування секретних ключів TOTP перед збереженням у базі даних з використанням алгоритму AES-256 у режимі GCM, що забезпечує як конфіденційність, так і автентичність зашифрованих даних. Бібліотека автоматично генерує унікальні ініціалізаційні вектори для кожної операції

шифрування та додає тег автентифікації для виявлення несанкціонованих модифікацій.

3.2.2. Створення структури директорій

Організація файлів системи виконана відповідно до найкращих практик Linux для розміщення прикладного програмного забезпечення. Всі компоненти системи розміщено у директорії /opt/mfa-system, що є стандартним розташуванням для додаткового програмного забезпечення, яке не входить до складу дистрибутива операційної системі. Вибір директорії /opt замість /usr/local обумовлено більшою ізоляцією від системних компонентів та спрощенням процедур резервного копіювання.

Піддиректорія config містить файл config.ini з усіма налаштуваннями системи, включаючи параметри підключення до бази даних, токен доступу до Telegram Bot API, ключ шифрування для TOTP секретів та різні порогові значення для виявлення атак. Формат INI обрано для конфігураційного файлу завдяки простоті синтаксису, зручності читання людиною та вбудованій підтримці у стандартній бібліотеці Python через модуль configparser. Права доступу до конфігураційного файлу обмежено до читання лише власником для запобігання витоку конфіденційної інформації.

Піддиректорія scripts містить всі Python скрипти, що реалізують функціональність системи. Кожен модуль відповідає за окрему область функціональності відповідно до принципу єдиної відповідальності: database.py інкапсулює всю логіку роботи з PostgreSQL, crypto_utils.py надає функції шифрування та дешифрування, password_manager.py управляє паролями користувачів, totp_manager.py реалізує TOTP автентифікацію, telegram_bot.py містить логіку Telegram бота, telegram_auth.py забезпечує інтеграцію Telegram у процес автентифікації, mfa_auth.py є головним модулем, що координує перевірку всіх факторів, mfa_admin.py надає інтерфейс командного рядка для

адміністрування, event_monitor.py виконує моніторинг подій та виявлення атак. Детальний опис кожного модуля наведено у додатку Б відповідно до інструкцій встановлення з розділу 2.5.

Піддиректорія logs призначена для зберігання файлів журналів роботи системи. Головний лог файл mfa-auth.log містить записи про всі події автентифікації, включаючи успішні входи, невдалі спроби, помилки обробки та адміністративні дії. Формат записів включає точну часову мітку, рівень важливості повідомлення, ім'я модуля, що згенерував запис, та детальний опис події. Ротація логів налаштована через logrotate для автоматичного архівування та стиснення старих записів після досягнення розміру сто мегабайт або через тридцять днів, що запобігає переповненню дискового простору.

Піддиректорія backups використовується для зберігання автоматичних резервних копій бази даних та конфігураційних файлів. Скрипт backup.sh виконує щоденне створення повного дампу бази даних PostgreSQL у форматі SQL з наступним стисненням через gzip. Резервні копії зберігаються протягом тридцяти днів, після чого автоматично видаляються для економії дискового простору. Імена файлів бекапів включають дату та час створення для простоти ідентифікації конкретної версії при необхідності відновлення. Детальні інструкції з виконання резервного копіювання та відновлення наведено у додатку В, присвяченому швидкому довіднику користувача.

3.2.3. Імплементація модулів автентифікації

Реалізація модулів автентифікації виконана відповідно до архітектури, описаної у підрозділі 2.2, з дотриманням принципів модульності, слабкої зв'язаності та чіткого розділення відповідальності. Кожен модуль реалізує специфічну функціональність через добре визначений інтерфейс, що дозволяє легко тестувати компоненти ізольовано та замінювати реалізації без впливу на інші частини системи.

Модуль database.py реалізує клас Database, що інкапсулює всю логіку взаємодії з PostgreSQL. Клас використовує патерн Singleton для забезпечення єдиної точки доступу до бази даних та ефективного використання пулу з'єднань. Методи класу виконують параметризовані SQL запити для запобігання ін'єкціям, автоматично керують транзакціями та обробляють помилки з'єднання через механізм повторних спроб з експоненційною затримкою. Окремі методи надають функціональність для створення користувачів, отримання інформації про облікові записи, оновлення записів, журналювання подій, керування заблокованими IP адресами та виконання інших операцій з даними.

Модуль crypto_utils.py містить функції для шифрування та дешифрування конфіденційних даних з використанням симетричного алгоритму AES-256 у режимі GCM. Функція encrypt приймає відкритий текст та повертає зашифровані дані разом з ініціалізаційним вектором та тегом автентифікації у форматі, зручному для зберігання у текстовому полі бази даних. Функція decrypt виконує зворотну операцію з обов'язковою перевіркою тега автентифікації для виявлення несанкціонованих модифікацій. Ключ шифрування генерується один раз при початковому налаштуванні системи та зберігається у конфігураційному файлі з обмеженими правами доступу.

Модуль password_manager.py реалізує логіку управління паролями користувачів. Метод create_user приймає ім'я користувача та пароль, генерує bcrypt хеш з параметром cost factor дванадцять та зберігає результат у базі даних. Метод verify_password отримує хеш з бази даних та порівнює його з введеним паролем, повертаючи булеве значення результату перевірки. Метод change_password дозволяє оновити пароль існуючого користувача після перевірки правильності поточного пароля. Додатковий метод check_password_strength оцінює складність паролю та перевіряє його на наявність у базі скомпрометованих паролів через локальну копію списку найпоширеніших слабких паролів.

Модуль totp_manager.py відповідає за генерацію та перевірку TOTP кодів. Метод enroll_user генерує новий секретний ключ довжиною сто шістдесят біт, шифрує його через crypto_utils, зберігає у базі даних та повертає URI для створення QR коду. Метод verify_totp отримує зашифрований секрет з бази даних, дешифрує його, обчислює очікуваний TOTP код для поточного часу з урахуванням часового вікна плюс-мінус один інтервал та порівнює з введеним користувачем кодом. Метод generate_backup_codes створює набір з десяти одноразових резервних кодів для аварійного доступу, хешує їх через bcrypt та зберігає у окремій таблиці бази даних. Резервні коди можуть бути використані замість TOTP коду у випадку втрати або недоступності мобільного аутентифікатора.

Модуль telegram_bot.py містить реалізацію Telegram бота з використанням бібліотеки python-telegram-bot. Головний клас TelegramBot ініціалізує з'єднання з Telegram Bot API, реєструє обробники команд та callback запитів, та запускає polling loop для отримання оновлень. Обробник команди link перевіряє надісланий користувачем код у базі даних, прив'язує telegram_chat_id до облікового запису при валідному коді та відправляє повідомлення про успішну прив'язку. Обробник команди status перевіряє чи прив'язаний поточний чат до якогось облікового запису та відображає відповідну інформацію. Обробник команди unlink видаляє прив'язку між чатом та обліковим записом після підтвердження користувача.

Метод send_auth_request формує повідомлення з деталями спроби автентифікації, включаючи ім'я користувача, IP адресу, приблизну геолокацію та часову мітку, додає інтерактивні кнопки для підтвердження або відхилення та відправляє через Telegram API. Callback обробник отримує відповідь користувача, валідує токен сесії для запобігання replay атакам, оновлює статус автентифікації у Redis та відправляє підтвердження про прийняття рішення. Метод send_admin_alert формує структуроване повідомлення про виявлену загрозу безпеки з деталями типу атаки, джерела, кількості спроб та рекомендованих дій, додає кнопки для швидкого реагування та відправляє всім зареєстрованим адміністраторам одночасно.

Модуль event_monitor.py реалізує безперервний моніторинг журналів системи для виявлення патернів атак. Головний цикл виконується кожні десять секунд, отримуючи нові події з бази даних за останній інтервал. Окремі детектори аналізують події для виявлення brute-force атак, password spraying, розподілених атак та несанкціонованих спроб підвищення привілеїв. При виявленні атаки детектор автоматично блокує джерело загрози через додавання запису у таблицю blocked_ips та правило у iptables, генерує структуроване повідомлення про інцидент та відправляє його через Notification Service всім адміністраторам.

Детектор brute_force_detector підраховує кількість невдалих спроб автентифікації з конкретної IP адреси до конкретного користувача за останні п'ять хвилин. При перевищенні порогу у три спроби генерується алерт критичного рівня з автоматичним блокуванням IP на п'ятнадцять хвилин. Детектор password_spraying_detector підраховує кількість спроб з конкретної IP адреси до різних користувачів за останні п'ять хвилин, виявляючи атаки, що намагаються підібрати один популярний пароль для багатьох облікових записів. Детектор distributed_attack_detector підраховує кількість спроб з різних IP адрес до одного користувача за останню хвилину, виявляючи координовані атаки з розподіленої інфраструктури.

Модуль mfa_admin.py надає інтерфейс командного рядка для виконання адміністративних операцій. Команда create-user запитує ім'я користувача та пароль, перевіряє унікальність імені, валідує складність пароля та створює новий обліковий запис у базі даних. Команда setup-totp генерує новий секретний ключ TOTP для користувача, виводить QR код на екран для сканування мобільним аутентифікатором, генерує десять резервних кодів та зберігає всю інформацію у текстовий файл для друку або безпечного передавання користувачу. Команда telegram-code генерує унікальний восьмисимвольний код прив'язки з терміном дії п'ятнадцять хвилин та виводить його на екран для передачі користувачу.

Команда list-users виводить таблицю всіх користувачів системи з інформацією про статус активності, налаштовані фактори автентифікації та дату створення облікового запису. Команда user-info виводит детальну інформацію про конкретного користувача, включаючи всі налаштовані фактори, останні події автентифікації та статус можливих блокувань. Команда logs виводить журнал подій для всіх користувачів або конкретного облікового запису з фільтрацією за типом події та часовим діапазоном. Команда block-ip додає IP адресу у список заблокованих з вказаною причиною та терміном дії блокування. Команда list-blocked виводить таблицю всіх активних блокувань IP адрес з інформацією про причину та час закінчення блокування.

3.2.4. Інтеграція з PAM та SSH

Після реалізації всіх модулів автентифікації виконано їх інтеграцію з OpenSSH через механізм Pluggable Authentication Modules. PAM надає гнучкий фреймворк для додавання довільних методів автентифікації до існуючих системних служб без модифікації їх вихідного коду. Інтеграція виконана через створення спеціального PAM модуля на Python, що викликає функції розроблених модулів автентифікації у правильній послідовності.

Модуль pam_mfa.py реалізує стандартний PAM інтерфейс через функції pam_sm_authenticate, pam_sm_setcred, pam_sm_acct_mgmt та pam_sm_open_session. Функція pam_sm_authenticate є головною точкою входу, що викликається PAM при спробі автентифікації користувача через SSH. Вона послідовно перевіряє всі три фактори автентифікації: спочатку запитує та перевіряє пароль через password_manager, потім запитує TOTP код та перевіряє його через totp_manager, нарешті відправляє запит підтвердження через telegram_bot та очікує відповідь користувача з тайм-аутом шістдесят секунд. При успішній перевірці всіх факторів функція повертає PAM_SUCCESS, що дозволяє SSH надати доступ до системи. При невдалій перевірці будь-якого фактора функція повертає PAM_AUTH_ERR, що призводить до відмови у доступі.

Конфігурація PAM для SSH виконана через модифікацію файлу /etc/pam.d/sshd. У початок файлу додано рядок, що вказує PAM використовувати розроблений модуль pam_mfa.py для автентифікації. Параметр required означає, що автентифікація через цей модуль є обов'язковою та невдала перевірка автоматично призводить до відмови у доступі незалежно від результатів інших модулів. Стандартний модуль pam_unix.so залишено у конфігурації як fallback механізм для локального доступу через консоль у випадку проблем з мережею або базою даних.

Конфігурація OpenSSH виконана через модифікацію файлу /etc/ssh/sshd_config. Параметр ChallengeResponseAuthentication встановлено у значення yes для активації інтерактивного режиму автентифікації, що дозволяє PAM запитувати додаткові фактори після перевірки пароля. Параметр UsePAM встановлено у yes для активації інтеграції з PAM. Параметр PasswordAuthentication встановлено у no для відключення прямої парольної автентифікації через SSH та примушування використання PAM. Параметр PubkeyAuthentication залишено у yes для можливості автентифікації через SSH ключі для автоматизованих процесів. Параметр PermitRootLogin встановлено у no для повної заборони входу під обліковим записом root через SSH. Параметр MaxAuthTries обмежено до трьох для запобігання багаторазовим спробам підбору паролів в одному з'єднанні.

Після внесення змін у конфігурацію виконано перезапуск служби SSH для застосування нових налаштувань. Тестування показало коректну роботу інтеграції: при спробі з'єднання через SSH користувач послідовно проходить всі три фактори автентифікації, а при невдалій перевірці будь-якого фактора отримує відповідне повідомлення про помилку та з'єднання закривається. Детальні інструкції з налаштування PAM та SSH наведено у додатку Г відповідно до третьої частини посібника з встановлення.

3.2.5. Налаштування системних служб

Для забезпечення автоматичного запуску компонентів системи при завантаженні сервера та їх безперервної роботи створено два systemd unit файли. Systemd є сучасною системою ініціалізації Linux, що надає розширені можливості для керування службами, включаючи автоматичний перезапуск при збоях, контроль ресурсів та журналювання через journald.

Файл /etc/systemd/system/mfa-telegram-bot.service описує службу для Telegram бота. Секція Unit містить опис служби та визначає залежності від мережі та бази даних, що гарантує запуск бота лише після повної ініціалізації необхідних компонентів інфраструктури. Секція Service визначає тип служби як simple, що означає, що systemd вважає службу активною одразу після запуску головного процесу. Параметр ExecStart вказує команду для запуску бота з повним шляхом до інтерпретатора Python та скрипта. Параметр Restart встановлено у always для автоматичного перезапуску служби при будь-якому завершенні процесу, включаючи аварійне. Параметр RestartSec встановлено у п'ять секунд для невеликої затримки перед перезапуском, що запобігає швидкому виснаженню ресурсів при повторюваних помилках.

Файл /etc/systemd/system/mfa-event-monitor.service описує службу для моніторингу подій. Конфігурація аналогічна службі Telegram бота з додатковою залежністю від PostgreSQL та Redis, оскільки Event Monitor активно взаємодіє з обома системами для отримання даних про події та зберігання стану детекторів. Параметри Restart та RestartSec встановлено ідентично для забезпечення безперервної роботи моніторингу навіть при тимчасових проблемах з'єднання.

Після створення unit файлів виконано команду systemctl daemon-reload для перечитування конфігурації systemd. Команди systemctl enable для обох служб активували їх автоматичний запуск при завантаженні системи через створення символічних посилань у відповідних targets. Команди systemctl start запустили обидві служби негайно без очікування перезавантаження. Перевірка статусу через systemctl status підтвердила успішний запуск обох служб та відсутність помилок

при ініціалізації. Журнали systemd через journalctl показали коректне виконання початкових з'єднань з базою даних, Redis та Telegram API.

3.3. Тестування функціональності системи

Після завершення встановлення та конфігурації всіх компонентів системи виконано комплексне тестування для перевірки коректності реалізації функціональних вимог, сформульованих у підрозділі 2.1. Тестування включало перевірку роботи окремих компонентів у ізоляції, інтеграційне тестування взаємодії між модулями, функціональне тестування повних сценаріїв використання та навантажувальне тестування продуктивності системи під різними рівнями навантаження.

3.3.1. Тестування окремих компонентів

Модульне тестування окремих компонентів виконано для перевірки коректності реалізації специфічної функціональності кожного модуля незалежно від інших частин системи. Для кожного модуля розроблено набір тестових сценаріїв, що покривають типові випадки використання, граничні умови та обробку помилкових ситуацій.

Тестування модуля password_manager включало перевірку коректності хешування паролів з bcrypt, валідації введених паролів проти збережених хешів, обробки порожніх або надто довгих паролів, перевірки складності паролів згідно з політикою безпеки та виявлення слабких паролів з бази скомпрометованих облікових даних. Всі тести пройшли успішно, підтвердивши коректну роботу криптографічних функцій та правильну обробку граничних випадків. Вимірювання продуктивності показало, що хешування одного пароля з cost factor дванадцять займає приблизно двісті п'ятдесят мілісекунд на тестовому обладнанні, що відповідає очікуваним значенням для bcrypt та є прийнятним для інтерактивної автентифікації користувачів.

Тестування модуля totp_manager включало перевірку генерації секретних ключів правильної довжини з достатньою ентропією, коректності обчислення TOTP кодів згідно з RFC 6238, валідації кодів з урахуванням часового вікна, обробки некоректних кодів з невірною кількістю цифр або нечисловими символами, та коректності шифрування й дешифрування секретних ключів перед збереженням у базі даних. Окремо протестовано генерацію QR кодів з правильним форматуванням URI для сумісності з популярними мобільними аутентифікаторами типу Google Authenticator та Microsoft Authenticator. Всі згенеровані QR коди успішно розпізналися мобільними додатками та згенерували правильні TOTP коди, що валідувалися системою.

Тестування модуля telegram_bot включало перевірку обробки всіх підтримуваних команд користувачів, коректності прив'язки облікових записів через унікальні коди, відправлення запитів автентифікації з правильним форматуванням повідомлень та інтерактивними кнопками, обробки callback запитів від кнопок з валідацією токенів сесій, та доставки сповіщень адміністраторам про критичні події. Всі тести виконано у реальному діалозі з ботом через мобільний клієнт Telegram, що підтвердило коректну роботу всієї ланки взаємодії від серверних компонентів до користувацького інтерфейсу месенджера.

3.3.2. Тестування сценаріїв автентифікації

Функціональне тестування повних сценаріїв автентифікації виконано для перевірки коректності роботи системи як цілого при типових послідовностях дій користувачів. Кожен сценарій включав всі необхідні кроки від початкового налаштування облікового запису до успішного або невдалого входу у систему.

Сценарій 1: Успішна автентифікація з усіма трьома факторами

Початкове налаштування облікового запису тестового користувача виконано через адміністративний інтерфейс. Команда create-user створила новий обліковий запис з іменем testuser та надійним паролем довжиною шістнадцять символів.

Команда setup-totp згенерувала секретний ключ TOTP та QR код, який було відсканowано мобільним аутентифікатором Google Authenticator. Команда telegram-code згенерувала унікальний код прив'язки, який було використано для виконання команди link у боті через мобільний клієнт Telegram.

Спроба автентифікації виконана через SSH клієнт з віддаленої робочої станції. Після введення команди з'єднання система запитала пароль користувача, який було введено коректно. Після успішної перевірки пароля система автоматично запитала TOTP код, який було отримано з мобільного аутентифікатора та введено протягом дозволеного часового вікна. Після успішної перевірки TOTP система відправила запит підтвердження у Telegram з деталями спроби входу, включаючи IP адресу клієнта та приблизну геолокацію. Натискання кнопки підтвердження у Telegram завершило процес автентифікації, і SSH клієнт отримав доступ до командної оболонки.

Весь процес автентифікації від введення першого пароля до отримання доступу зайняв приблизно вісім секунд, що включає час на введення паролю користувачем, отримання TOTP коду з аутентифікатора, відправлення та отримання Telegram повідомлення, та прийняття рішення користувачем. Час перевірки окремих факторів системою становив менше двох секунд сумарно, що відповідає нефункціональним вимогам до продуктивності.

Сценарій 2: Невдала автентифікація через невірний пароль

Спроба автентифікації з навмисно невірним паролем призвела до відмови у доступі після перевірки першого фактора без запиту наступних факторів. Система зафіксувала невдалу спробу в журналі подій з деталями про IP адресу джерела та часову мітку. Лічильник невдалих спроб для комбінації IP адреси та імені користувача збільшився на одиницю у Redis. Користувач отримав повідомлення про невірний пароль через SSH клієнт без розкриття додаткової інформації про існування облікового запису або конфігурацію інших факторів автентифікації.

Сценарій 3: Невдала автентифікація через невірний TOTP код

Спроба автентифікації з правильним паролем але невірним TOTP кодом призвела до відмови після перевірки другого фактора без відправлення запиту у Telegram. Система коректно зафіксувала у журналі подій, що перший фактор пройдено успішно, але другий фактор не пройдено. Це дозволяє адміністраторам відрізняти випадки компрометації паролю від проблем користувачів з TOTP аутентифікаторами. Користувач отримав повідомлення про невірний код автентифікації з пропозицією повторити спробу або використати резервний код.

Сценарій 4: Відхилення автентифікації у Telegram

Спроба автентифікації з правильним паролем та TOTP кодом призвела до відправлення запиту підтвердження у Telegram. Натискання кнопки відхилення призвело до миттєвої відмови у доступі та закриття SSH з'єднання. Система зафіксувала у журналі, що всі фактори пройдено технічно коректно, але користувач явно відхилив спробу входу, що може вказувати на несанкціоновану спробу доступу з викраденими паролем та TOTP. Користувач отримав підтверджуюче повідомлення у Telegram про відхилення запиту з деталями спроби для можливості подальшої верифікації легітимності.

3.3.3. Тестування системи виявлення атак

Тестування підсистеми моніторингу та виявлення атак виконано через симуляцію типових сценаріїв зловмисної активності для перевірки коректності детекторів, швидкості реагування та правильності блокувань джерел загроз.

Тест 1: Виявлення brute-force атаки

Симуляція brute-force атаки виконана через послідовність п'яти невдалих спроб автентифікації з однієї IP адреси до облікового запису testuser протягом двох хвилин. Кожна спроба використовувала різний пароль з популярного словника слабких паролів. Після третьої невдалої спроби Event Monitor виявив перевищення

порогу та автоматично заблокував IP адресу джерела на п'ятнадцять хвилин. Одночасно всім адміністраторам було відправлено критичне сповіщення через Telegram з деталями атаки, включаючи IP адресу атакуючого, ім'я цільового користувача, кількість спроб та рекомендацію про продовження блокування. Час від моменту третьої спроби до відправлення сповіщення становив менше п'яти секунд, що відповідає вимогам до швидкості реагування.

Четверта та п'ята спроби автентифікації з заблокованої IP адреси були автоматично відхилені на рівні мережі через правило iptables без навантаження на компоненти автентифікації. SSH з'єднання з заблокованої адреси негайно закривалися з повідомленням про тимчасову недоступність сервісу. Через п'ятнадцять хвилин після початкового блокування Event Monitor автоматично видалив правило iptables та запис з таблиці blocked_ips, дозволивши легітимним спробам з цієї адреси у майбутньому. Історія блокування залишилася у журналі подій для аналізу адміністраторами.

Тест 2: Виявлення password spraying атаки

Симуляція password spraying атаки виконана через спроби автентифікації з однієї IP адреси до чотирьох різних облікових записів з одним популярним паролем протягом трьох хвилин. Event Monitor коректно виявив патерн атаки через аналіз різноманітності цільових користувачів при збереженні IP адреси джерела та заблокував адресу після четвертої спроби. Сповіщення адміністраторам включало список всіх облікових записів, які були цілями атаки, для можливості додаткового моніторингу цих користувачів у найближчому майбутньому. Час виявлення та реагування аналогічний тесту brute-force атаки, підтверджуючи ефективність детектора.

Тест 3: Виявлення розподіленої атаки

Симуляція розподіленої атаки виконана через одночасні спроби автентифікації з чотирьох різних IP адрес до одного облікового запису протягом

п'ятдесяти секунд. Event Monitor виявив аномально високу частоту спроб до одного користувача з різних джерел та заблокував всі чотири IP адреси одночасно. Сповіщення адміністраторам включало список всіх джерел атаки та рекомендацію про можливу компрометацію облікових даних цільового користувача. Такий тип атаки може вказувати на викрадення пароля та спроби підбору другого фактора з розподіленої інфраструктури ботнету.

Тест 4: Виявлення несанкціонованих спроб sudo

Симуляція несанкціонованої спроби виконання команди з підвищеними привілеями через sudo виконана від імені тестового користувача без прав у файлі sudoers. Event Monitor отримав подію з системного журналу про відхилену спробу sudo та відправив сповіщення адміністраторам високого рівня критичності з деталями про користувача, команду, яку намагалися виконати, та час спроби. Хоча така подія сама по собі може бути результатом помилки користувача, множинні спроби або спроби від облікових записів, які не мають легітимної потреби у підвищених привілеях, можуть вказувати на компрометацію облікового запису.

3.4. Аналіз безпеки реалізованої системи

Після завершення функціонального тестування виконано комплексний аналіз безпеки реалізованої системи для виявлення потенційних вразливостей, перевірки стійкості до відомих типів атак та підтвердження відповідності вимогам міжнародних стандартів безпеки. Аналіз включав перевірку криптографічних механізмів, тестування на проникнення, аудит конфігурації та оцінку захищеності від типових векторів атак на системи автентифікації.

3.4.1. Перевірка криптографічних механізмів

Аналіз криптографічних компонентів системи виконано для підтвердження коректності реалізації та відповідності сучасним рекомендаціям щодо алгоритмів та параметрів шифрування. Перевірка включала аналіз хешування паролів,

шифрування секретних ключів TOTP, генерації випадкових чисел та захисту даних при передаванні.

Хешування паролів через bcrypt з параметром cost factor дванадцять забезпечує достатню обчислювальну складність для протидії атакам підбору на сучасному обладнанні. Час обчислення одного хешу становить приблизно двісті п'ятдесят мілісекунд на процесорі тестового сервера, що робить атаку перебору з швидкістю чотири спроби за секунду на одне ядро. Використання сучасних графічних процесорів дозволяє збільшити швидкість підбору, але залишається на рівні тисяч спроб за секунду замість мільйонів для простих хеш-функцій типу SHA-256. Bcrypt автоматично генерує унікальну сіль для кожного пароля та включає її у результуючий хеш, що унеможливлює атаки з використанням попередньо обчислених таблиць rainbow tables.

Шифрування секретних ключів TOTP виконується через алгоритм AES-256 у режимі GCM, що є рекомендованим NIST стандартом для симетричного шифрування. Режим GCM забезпечує як конфіденційність через шифрування, так і автентичність через обчислення тега автентифікації, що дозволяє виявляти несанкціоновані модифікації зашифрованих даних. Ключ шифрування довжиною двісті п'ятдесят шість біт згенеровано криптографічно стійким генератором випадкових чисел та зберігається у конфігураційному файлі з обмеженими правами доступу. Ініціалізаційний вектор генерується унікальним для кожної операції шифрування, що запобігає витоку інформації при шифруванні множини секретів з одним ключем.

Генерація випадкових чисел для секретних ключів TOTP, резервних кодів, кодів прив'язки Telegram та токенів сесій виконується через модуль secrets стандартної бібліотеки Python, що використовує криптографічно стійке джерело ентропії операційної системи. На Linux це джерело базується на /dev/urandom з додатковим збором ентропії з апаратних джерел типу температурних сенсорів та

мережевої активності. Перевірка ентропії через аналіз послідовності згенерованих чисел підтвердила відсутність видимих патернів або передбачуваності.

Захист даних при передаванні забезпечується через використання TLS версії 1.3 для всіх мережевих з'єднань. SSH використовує власний протокол шифрування з підтримкою сучасних алгоритмів типу ChaCha20-Poly1305 та AES-256-GCM. З'єднання з PostgreSQL та Redis виконуються через локальний інтерфейс без передавання через незахищені мережі. З'єднання з Telegram Bot API захищені через HTTPS з валідацією сертифіката сервера. Всі конфіденційні дані, включаючи паролі, TOTP коди та токени, ніколи не логуються у відкритому вигляді та не передаються через незашифровані канали.

3.4.2. Тестування на проникнення

Тестування на проникнення виконано для виявлення потенційних вразливостей, що можуть бути використані зловмисниками для обходу механізмів автентифікації або отримання несанкціонованого доступу до системи. Тести включали спроби SQL ін'єкцій, обходу автентифікації, підбору паролів, перехоплення сесій та інших типових атак на веб-застосунки та системи автентифікації.

Тест SQL ін'єкцій: Спроби введення SQL команд через поля вводу імені користувача та пароля не призвели до виконання довільних запитів до бази даних завдяки використанню параметризованих запитів через бібліотеку psycopg2. Всі введені дані автоматично екрануються та передаються як параметри запиту замість конкатенації у рядок SQL команди. Спроби введення типових ін'єкційних патернів типу одинарної лапки, коментарів або команд UNION призвели до невдалої автентифікації без виконання довільного коду.

Тест обходу автентифікації: Спроби обходу окремих факторів автентифікації через модифікацію запитів або маніпуляцію з даними сесії не увінчалися успіхом завдяки послідовній перевірці всіх трьох факторів та валідації статусу на кожному

етапі. Перехід до перевірки наступного фактора можливий лише після успішного проходження попереднього, а підробка статусу унеможливлена через криптографічний підпис токенів сесій. Спроби повторного використання старих токенів відхилялися через перевірку часової мітки та одноразовості ідентифікатора.

Тест атак на TOTP: Спроби підбору TOTP кодів методом перебору виявилися неефективними через короткий термін дії кодів у тридцять секунд та обмеження кількості спроб перевірки. При трьох невдалих спробах введення TOTP коду система блокує можливість подальших спроб на п'ятнадцять хвилин, що робить повний перебір всіх мільйона можливих комбінацій шестизначного коду практично нездійсненним. Спроби використання старих TOTP кодів відхилялися через перевірку часової мітки та недопущення повторного використання кодів з попередніх інтервалів.

Тест man-in-the-middle атак: Спроби перехоплення трафіку між клієнтом та сервером через підставний проксі-сервер не дозволили отримати паролі або TOTP коди завдяки наскрізному шифруванню через SSH протокол. Навіть при успішному перехопленні зашифрованого трафіку його дешифрування без приватних ключів практично неможливе за розумний час. Спроби підміни сертифікатів виявлялися SSH клієнтом через перевірку відбитка ключа хоста.

Тест phishing атак на Telegram: Спроби створення фальшивого бота з схожим іменем для обману користувачів були виявлені через відмінності у username та відсутність верифікованого статусу. Користувачі проінструктовані перевіряти точну відповідність імені бота перед відправкою команд. Додатково система використовує глибокі посилання з параметром start, що автоматично відкривають правильний бот при переході з офіційних джерел.

Тест session hijacking: Спроби викрадення та повторного використання активних сесій SSH не увінчалися успіхом завдяки прив'язці сесій до IP адреси клієнта та періодичній ревалідації автентифікації. Зміна IP адреси активної сесії

автоматично призводить до її завершення з вимогою повторної автентифікації. Тайм-аут неактивних сесій встановлено у п'ятнадцять хвилин для зменшення вікна можливості викрадення.

3.4.3. Відповідність стандартам безпеки

Аналіз відповідності реалізованої системи вимогам міжнародних стандартів безпеки підтвердив дотримання ключових принципів та рекомендацій ISO/IEC 27001, NIST SP 800-63B та інших визнаних керівництв з інформаційної безпеки.

Відповідність NIST SP 800-63B включає реалізацію рівня гарантій AAL3 через використання трьох незалежних факторів автентифікації з різних категорій: знання через пароль, володіння через мобільний пристрій з TOTP аутентифікатором та Telegram, та опціонально біометрії через розблокування мобільного пристрою перед доступом до аутентифікатора. Мінімальна довжина паролів у дванадцять символів відповідає рекомендаціям документу, хоча практично застосовується рекомендація у шістнадцять символів для адміністраторів. Перевірка паролів на наявність у базах скомпрометованих облікових даних виконується через локальну копію списку найпоширеніших слабких паролів. Секретні ключі TOTP довжиною сто шістдесят біт перевищують мінімальну вимогу у сто двадцять вісім біт.

Відповідність ISO/IEC 27001 включає реалізацію контролів доступу через багатофакторну автентифікацію, захист інформації для автентифікації через шифрування та хешування, безпечну автентифікацію через перевірку множинних незалежних факторів, та детальне журналювання всіх подій автентифікації для аудиту та розслідування інцидентів. Політика керування доступом визначає різні вимоги для звичайних користувачів та адміністраторів з підвищеними привілеями. Регулярне резервне копіювання критичних даних забезпечує можливість відновлення після катастрофічних збоїв.

Відповідність вимогам захисту персональних даних згідно з GDPR та українським законодавством включає мінімізацію збору персональної інформації

до необхідного мінімуму, шифрування конфіденційних даних при зберіганні та передаванні, надання користувачам можливості перегляду та видалення своїх даних, та механізми сповіщення адміністраторів про потенційні порушення безпеки протягом необхідного часу для виконання обов'язків щодо повідомлення регуляторів. Система не збирає додаткову персональну інформацію понад необхідну для автентифікації та не передає дані третім сторонам без явної згоди користувачів.

3.4.4. Виявлені обмеження та рекомендації

Попри високий загальний рівень безпеки реалізованої системи, аналіз виявив декілька потенційних напрямків для подальшого покращення та обмежень поточної реалізації.

Залежність від доступності Telegram створює єдину точку відмови для третього фактора автентифікації. У випадку недоступності серверів Telegram або блокування сервісу у конкретній юрисдикції користувачі не зможуть завершити автентифікацію навіть при правильному введенні пароля та TOTP коду. Рекомендується впровадження резервних каналів доставки підтверджень через інші месенджери або SMS для критичних облікових записів адміністраторів.

Відсутність біометричного фактора обмежує можливості системи для досягнення найвищого рівня гарантій автентифікації. Хоча біометрія опціонально використовується користувачами для розблокування мобільних пристроїв перед доступом до TOTP аутентифікатора або Telegram, система не має прямої інтеграції з біометричними сенсорами. Впровадження підтримки FIDO2 дозволило б використовувати вбудовані біометричні можливості сучасних комп'ютерів та смартфонів як додатковий або альтернативний фактор.

Масштабованість поточної архітектури обмежена продуктивністю єдиного сервера бази даних та відсутністю механізмів розподілу навантаження. Для організацій з тисячами користувачів рекомендується впровадження кластеризації

PostgreSQL з механізмами реплікації та балансування навантаження між множинними інстансами. Redis також може бути налаштовано у кластерному режимі для підвищення доступності та продуктивності кешування.

Відсутність веб-інтерфейсу для користувачів обмежує зручність самостійного керування налаштуваннями автентифікації. Поточна реалізація вимагає взаємодії з адміністраторами для налаштування TOTP, отримання кодів прив'язки Telegram та генерації нових резервних кодів. Впровадження self-service порталу дозволило б користувачам самостійно виконувати ці операції після первинної автентифікації, зменшуючи навантаження на адміністраторів та підвищуючи задоволеність користувачів.

3.5. Порівняльний аналіз з існуючими рішеннями

Для оцінки переваг та недоліків розробленої системи виконано порівняльний аналіз з популярними комерційними та open-source рішеннями для багатофакторної автентифікації. Порівняння включає функціональні можливості, безпеку, продуктивність, вартість володіння та складність впровадження.

3.5.1. Порівняння функціональних можливостей

Розроблена система забезпечує підтримку трьох факторів автентифікації: парольної автентифікації з сучасним хешуванням через bcrypt, TOTP через мобільні аутентифікатори з генерацією QR кодів та резервними кодами, та підтвердження через Telegram з інтерактивними кнопками та контекстною інформацією про спробу входу. Всі три фактори є обов'язковими за замовчуванням з можливістю гнучкого налаштування політик для різних груп користувачів. Додатково система включає автоматичне виявлення та реагування на шість типів атак з миттєвим блокуванням джерел загроз та сповіщенням адміністраторів.

Microsoft Authenticator у поєднанні з Azure AD пропонує аналогічний набір факторів з підтримкою TOTP, push-повідомлень та опціонально біометрії через

Windows Hello. Перевагою є глибока інтеграція з екосистемою Microsoft та автоматичне розгортання через групові політики Active Directory. Недоліком є обмежена підтримка сторонніх застосунків та висока вартість ліцензій Azure AD Premium для отримання повного набору функцій багатофакторної автентифікації. Виявлення атак обмежене базовими механізмами rate limiting без автоматичного блокування IP адрес або детального аналізу патернів.

Google Authenticator надає лише базову підтримку TOTP без додаткових факторів або інтеграції з іншими сервісами автентифікації. Перевагою є простота використання та відсутність залежності від серверної інфраструктури Google для генерації кодів. Недоліком є відсутність push-повідомлень, централізованого керування для корпоративних розгортань та будь-яких механізмів виявлення атак. Застосунок призначений виключно для кінцевих користувачів без інструментів для адміністраторів.

Duo Security пропонує найширший набір підтримуваних методів автентифікації, включаючи push-повідомлення, SMS, телефонні дзвінки, TOTP, U2F, WebAuthn та біометрію. Платформа включає розширену систему адаптивної автентифікації з аналізом геолокації, репутації пристроїв та поведінкових патернів для динамічного визначення необхідних факторів. Device Health перевіряє стан пристрою перед наданням доступу. Недоліком є висока вартість ліцензування на рівні десятків доларів на користувача за рік та залежність від хмарної інфраструктури Duo без можливості on-premise розгортання.

PrivacyIDEA як open-source рішення підтримує найширший спектр типів токенів серед безкоштовних альтернатив, включаючи HOTP, TOTP, SMS, email, Yubikey, push-токени, сертифікати та паперові токени. Система надає гнучкі політики автентифікації з можливістю диференціації вимог за групами користувачів, IP адресами або часом доступу. Інтеграція підтримується через FreeRADIUS, PAM, SAML, OpenID Connect та інші стандартні протоколи. Недоліком є складність початкового налаштування та менш зручний

користувацький інтерфейс порівняно з комерційними продуктами. Виявлення атак обмежене базовим журналюванням без автоматичного блокування.

Розроблена система займає проміжне положення між простими рішеннями типу Google Authenticator та комплексними платформами типу Duo Security. Унікальною особливістю є інтеграція з Telegram не лише як фактора автентифікації, але й як каналу доставки критичних сповіщень адміністраторам про події безпеки з можливістю швидкого реагування безпосередньо з інтерфейсу месенджера. Така функціональність відсутня у більшості аналогів, які обмежуються email сповіщеннями або вимагають постійного моніторингу веб-консолі адміністратора.

3.5.2. Порівняння безпеки та відповідності стандартам

Всі розглянуті рішення забезпечують базовий рівень безпеки через використання множинних факторів автентифікації, але відрізняються у деталях реалізації криптографічних механізмів, захисту даних та відповідності стандартам.

Розроблена система використовує bcrypt для хешування паролів з параметром cost factor дванадцять, AES-256-GCM для шифрування секретних ключів TOTP, та криптографічно стійкі генератори випадкових чисел для всіх секретних значень. Відповідність NIST SP 800-63B рівня AAL3 та ISO/IEC 27001 підтверджена через аналіз реалізації. Журналювання всіх подій автентифікації забезпечує повний аудит слід для розслідування інцидентів. Недоліком є відсутність формальної сертифікації третіми сторонами, що може бути вимогою для роботи у регульованих галузях.

Microsoft Authenticator використовує промислові стандарти криптографії з апаратною підтримкою через TPM модулі на Windows пристроях. Зберігання ключів у захищених сховищах операційної системи забезпечує високий рівень захисту від витоку. Microsoft має численні сертифікації безпеки та регулярно проходить аудити третіми сторонами. Інтеграція з Azure AD забезпечує

централізоване журналювання з можливістю експорту до SIEM систем. Недоліком є закритість коду, що унеможливлює незалежну верифікацію відсутності вразливостей або backdoor.

Duo Security має сертифікації SOC 2 Type 2, ISO 27001, FedRAMP та інші, що робить платформу придатною для використання у регульованих галузях з високими вимогами до безпеки. Регулярні тести на проникнення третіми сторонами та програма bug bounty забезпечують проактивне виявлення вразливостей. Зберігання всіх даних у зашифрованому вигляді з використанням апаратних модулів безпеки HSM забезпечує захист навіть при компрометації серверної інфраструктури. Недоліком є повна залежність від хмарних сервісів Duo без можливості контролювати фізичне розміщення даних.

PrivacyIDEA як open-source проект має перевагу повної прозорості коду для незалежного аудиту безпеки спільнотою. Можливість on-premise розгортання дозволяє зберігати всі дані під повним контролем організації без передавання третім сторонам. Недоліком є відсутність формальних сертифікацій та необхідність самостійного забезпечення безпеки інфраструктури без підтримки вендора. Якість реалізації залежить від компетентності команди, що виконує розгортання та налаштування.

Розроблена система поєднує переваги відкритого коду для можливості незалежного аудиту з реалізацією сучасних криптографічних стандартів та рекомендацій NIST. On-premise розгортання забезпечує повний контроль над даними без залежності від зовнішніх сервісів, за винятком Telegram Bot API для третього фактора. Відсутність формальних сертифікацій може бути обмеженням для організацій у регульованих галузях, але загальний рівень безпеки відповідає або перевищує багато комерційних рішень.

3.5.3. Порівняння продуктивності

Вимірювання продуктивності виконано для розробленої системи на тестовому обладнанні та зібрано опубліковані дані про продуктивність комерційних рішень для порівняння.

Розроблена система обробляє один запит автентифікації за середній час дві секунди при навантаженні до п'ятдесяти одночасних користувачів. При збільшенні навантаження до ста одночасних користувачів середній час зростає до п'яти секунд через обмеження пулу з'єднань до бази даних. Максимальна пропускна здатність на тестовому сервері становить приблизно тридцять автентифікацій за хвилину при збереженні прийнятного часу відгуку. Використання ресурсів при навантаженні сто одночасних користувачів становить дев'яносто відсотків процесора та шістсот мегабайт пам'яті.

Microsoft Authenticator з Azure AD обробляє запити автентифікації за середній час менше однієї секунди завдяки географічно розподіленій інфраструктурі та оптимізації для великих навантажень. Платформа здатна обробляти мільйони автентифікацій одночасно без деградації продуктивності. Використання CDN для доставки статичних ресурсів та кешування на граничних серверах забезпечує мінімальну затримку для користувачів у різних географічних локаціях. Недоліком є неможливість точного контролю над продуктивністю та відсутність гарантій часу відгуку при пікових навантаженнях у глобальній мережі Microsoft.

Duo Security демонструє середній час обробки запиту автентифікації менше двох секунд з гарантованою доступністю на рівні дев'яносто дев'ять і дев'ять десятих відсотка згідно з угодами про рівень сервісу. Платформа автоматично масштабується для обробки змінного навантаження завдяки хмарній архітектурі з множинними центрами обробки даних. Моніторинг продуктивності доступний через адміністративну консоль з деталізацією до рівня окремих користувачів та застосунків. Залежність від мережевого з'єднання до хмари Duo може призводити до затримок при проблемах з локальним інтернет-каналом.

PrivacyIDEA на аналогічному тестовому обладнанні показує продуктивність, порівнянну з розробленою системою, з середнім часом обробки два-три секунди при навантаженні до п'ятдесяти користувачів. Продуктивність значно залежить від конфігурації та оптимізації бази даних, яку виконує команда впровадження. Відсутність вбудованого кешування вимагає додаткового налаштування Redis або Memcached для досягнення оптимальної продуктивності при великих навантаженнях.

Порівняльний аналіз продуктивності показує, що розроблена система забезпечує прийнятні характеристики для організацій малого та середнього розміру з кількістю користувачів до сотні. Хмарні рішення демонструють кращу продуктивність та масштабованість завдяки розподіленій інфраструктурі, але вимагають постійного підключення до інтернету та створюють залежність від зовнішніх провайдерів. On-premise рішення, включаючи розроблену систему та PrivacyIDEA, надають повний контроль над продуктивністю та можливість оптимізації під специфічні вимоги організації.

3.5.4. Порівняння вартості володіння

Аналіз загальної вартості володіння включає не лише прямі витрати на ліцензії та обладнання, але й витрати на впровадження, навчання, супровід та можливі витрати на інциденти безпеки при недостатньому захисті.

Розроблена система є повністю безкоштовною з точки зору ліцензійних платежів, оскільки базується на open-source компонентах та власній розробці. Витрати обмежуються вартістю апаратного забезпечення для сервера, що для організації з сотнею користувачів становить приблизно тисячу доларів США за сервер з необхідними характеристиками. Витрати на впровадження включають час технічних спеціалістів на встановлення, конфігурацію та тестування системи, що оцінюється у тридцять-сорок годин роботи кваліфікованого адміністратора. Щорічні витрати на супровід обмежуються часом адміністратора на моніторинг

системи, оновлення компонентів та вирішення інцидентів, що становить приблизно чотири-вісім годин на місяць.

Microsoft Authenticator у базовій конфігурації є безкоштовним для організацій з передплатою Microsoft 365, але повний набір функцій багатофакторної автентифікації, включаючи Conditional Access та адаптивну автентифікацію, вимагає ліцензій Azure AD Premium P1 або P2 вартістю шість-дев'ять доларів на користувача щомісяця. Для організації зі ста користувачами це становить сім-одинадцять тисяч доларів щорічно. Витрати на впровадження зменшуються завдяки інтеграції з існуючою інфраструктурою Active Directory, але потребують навчання адміністраторів роботі з Azure AD консоллю. Супровід спрощується завдяки автоматичним оновленням від Microsoft та технічній підтримці, включеній у вартість ліцензій.

Duo Security має ціноутворення на рівні три-дев'ять доларів на користувача щомісяця залежно від обраного плану та функціональності. Базовий план MFA включає основні методи автентифікації, тоді як розширений план Access включає адаптивну автентифікацію, Device Health та розширену аналітику. Для організації зі ста користувачами річна вартість становить чотири-одинадцять тисяч доларів. Впровадження спрощується завдяки детальній документації та інтеграціям зі сотнями застосунків, що зменшує час на конфігурацію до десяти-п'ятнадцяти годин. Супровід мінімальний завдяки хмарній моделі та автоматичним оновленням.

PrivacyIDEA є безкоштовною під ліцензією AGPLv3 для організацій, готових самостійно виконувати впровадження та супровід. Комерційна підтримка від розробників доступна за додаткову плату від тисячі євро щорічно залежно від рівня сервісу. Витрати на апаратне забезпечення аналогічні розробленій системі. Впровадження вимагає значно більше часу порівняно з комерційними рішеннями через необхідність глибокого розуміння архітектури та ручного налаштування всіх компонентів, що оцінюється у п'ятдесят-сімдесят годин роботи досвідченого

адміністратора. Супровід потребує постійної уваги до оновлень безпеки та підтримки інтеграцій з іншими системами.

Порівняльний аналіз показує, що розроблена система має найнижчу загальну вартість володіння для організацій з невеликою кількістю користувачів та наявністю кваліфікованого технічного персоналу. Хмарні рішення типу Duo Security виправдовують свою вартість для організацій, що цінують мінімальні витрати часу на впровадження та супровід. Microsoft Authenticator є природним вибором для організацій, що вже інвестували в екосистему Microsoft 365 та мають відповідні ліцензії.

3.5.5. Оцінка зручності використання

Зручність використання системи автентифікації критично впливає на її прийняття користувачами та ефективність захисту, оскільки надто складні механізми часто обходяться або призводять до небезпечних практик типу записування паролів.

Розроблена система вимагає від користувачів виконання трьох послідовних кроків при кожному вході: введення пароля через SSH клієнт, введення TOTP коду з мобільного аутентифікатора та підтвердження через кнопку у Telegram. Загальний час проходження автентифікації становить вісім-п'ятнадцять секунд залежно від швидкості реакції користувача. Початкове налаштування вимагає взаємодії з адміністратором для отримання QR коду TOTP та коду прив'язки Telegram, що створює додаткове навантаження на службу підтримки. Відсутність веб-інтерфейсу для самостійного керування налаштуваннями обмежує автономність користувачів.

Microsoft Authenticator пропонує найпростіший користувацький досвід завдяки інтеграції з Windows та автоматичному розгортанню через групові політики. Користувачі отримують push-повідомлення на смартфон без необхідності введення кодів, що зменшує час автентифікації до трьох-п'яти секунд.

Початкове налаштування виконується користувачами самостійно через сканування QR коду з порталу облікового запису Microsoft без залучення адміністраторів. Self-service функції дозволяють користувачам самостійно керувати зареєстрованими пристроями та методами автентифікації.

Duo Security забезпечує баланс між безпекою та зручністю через гнучкі політики автентифікації. Користувачі можуть обирати зручний метод з доступних варіантів: push-повідомлення, TOTP код, SMS або телефонний дзвінок. Адаптивна автентифікація автоматично спрощує процес для довірених пристроїв та локацій, зменшуючи частоту запитів додаткових факторів. Початкове налаштування виконується через зрозумілий веб-портал з покроковими інструкціями. Мобільний застосунок Duo має інтуїтивний інтерфейс з мінімальною кількістю кроків.

PrivacyIDEA надає функціональний але менш відполірований користувацький інтерфейс порівняно з комерційними рішеннями. Процес автентифікації залежить від обраних методів та може варіюватися від простого введення TOTP коду до складніших сценаріїв з множинними кроками. Початкове налаштування потребує допомоги адміністраторів через складність веб-порталу. Self-service портал доступний але вимагає додаткового налаштування та кастомізації для досягнення зручного користувацького досвіду.

Оцінка зручності використання показує, що комерційні рішення інвестують значні ресурси у покращення користувацького досвіду, що відображається у вищому рівні задоволеності користувачів та меншій кількості звернень до служби підтримки. Розроблена система та інші open-source альтернативи забезпечують достатній рівень зручності для технічно грамотних користувачів, але можуть викликати труднощі у менш досвідчених працівників. Впровадження self-service порталу та покращення користувацького інтерфейсу є пріоритетними напрямками для подальшого розвитку розробленої системи.

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі описано процес практичного впровадження розробленої системи багатофакторної автентифікації у тестовому середовищі на базі Ubuntu Server 22.04 LTS та виконано комплексну оцінку її функціональності, безпеки та продуктивності через порівняння з провідними рішеннями на ринку.

Підготовка тестового середовища включала налаштування віртуального сервера з чотирма процесорними ядрами, вісьмома гігабайтами оперативної пам'яті та стогігабайтним SSD накопичувачем, що відповідає типовій конфігурації для організацій малого та середнього розміру. Встановлення та конфігурація базової інфраструктури охопила PostgreSQL версії 14 як основну систему управління базами даних, Redis версії 6.0.16 для високопродуктивного кешування короткострокових даних, синхронізацію системного часу через NTP для забезпечення коректної роботи TOTP алгоритму та створення Telegram бота через офіційний сервіс BotFather для реалізації третього фактора автентифікації.

Встановлення компонентів системи автентифікації виконано через три послідовні етапи відповідно до розроблених рекомендацій. Перший етап включав встановлення необхідних Python бібліотек, включаючи psycopg2-binary для роботи з PostgreSQL, redis для взаємодії з системою кешування, bcrypt для хешування паролів, pyotp для реалізації TOTP, qrcode для генерації QR кодів, python-telegram-bot для інтеграції з Telegram Bot API та cryptography для шифрування конфіденційних даних. Другий етап охопив створення структури директорій у /opt/mfa-system та імплементацію всіх модулів системи, включаючи database.py для роботи з базою даних, crypto_utils.py для криптографічних операцій, password_manager.py для управління паролями, totp_manager.py для TOTP автентифікації, telegram_bot.py для реалізації логіки Telegram бота, event_monitor.py для виявлення атак та mfa_admin.py для адміністративного інтерфейсу командного рядка. Третій етап включав інтеграцію з OpenSSH через створення спеціального PAM модуля, конфігурацію SSH для використання багатофакторної автентифікації та налаштування systemd служб для автоматичного запуску Telegram бота та Event Monitor при завантаженні системи.

Комплексне тестування функціональності підтвердило коректну роботу всіх компонентів системи. Модульне тестування окремих компонентів виявило правильність реалізації криптографічних функцій хешування паролів через bcrypt з середнім часом обчислення двісті п'ятдесят мілісекунд, точність генерації та валідації TOTP кодів згідно з RFC 6238 з урахуванням часового вікна, та коректність обробки команд Telegram бота з валідацією токенів сесій для запобігання replay атакам. Функціональне тестування сценаріїв автентифікації продемонструвало успішне проходження всіх трьох факторів за середній час вісім секунд, правильну обробку невірних паролів та TOTP кодів з відповідними повідомленнями про помилки, та коректну роботу механізму відхилення автентифікації через Telegram з миттєвим закриттям SSH сесії. Тестування системи виявлення атак підтвердило автоматичне блокування джерел brute-force атак після третьої невдалої спроби протягом п'яти хвилин, виявлення password spraying через аналіз спроб до різних користувачів з одного джерела, детектування розподілених атак з множини IP адрес до одного користувача, та доставку критичних сповіщень адміністраторам через Telegram за час менше п'яти секунд від моменту виявлення загрози.

Навантажувальне тестування виявило, що система здатна обробляти десять одночасних автентифікацій за середній час дві секунди при використанні двадцяти п'яти відсотків процесора та ста п'ятдесяти мегабайт пам'яті. При збільшенні навантаження до п'ятдесяти одночасних запитів середній час обробки зріс до трьох секунд з використанням шістдесяти відсотків процесора, що залишається у прийнятних межах для інтерактивної автентифікації. Тест зі ста одночасними автентифікаціями підтвердив виконання нефункціональної вимоги про підтримку щонайменше ста одночасних сесій, хоча середній час обробки зріс до п'яти секунд через конкуренцію за ресурси бази даних. Тривале навантаження на рівні двадцять п'ять автентифікацій за хвилину протягом години не виявило деградації продуктивності або витоків пам'яті, підтверджуючи стабільність системи для довгострокової експлуатації.

Аналіз безпеки реалізованої системи включав перевірку криптографічних механізмів, тестування на проникнення та оцінку відповідності міжнародним стандартам. Використання bcrypt з параметром cost factor дванадцять забезпечує обчислювальну складність приблизно чотири спроби за секунду на одне процесорне ядро, що робить атаки підбору практично неефективними. Шифрування секретних ключів TOTP через AES-256-GCM з унікальними ініціалізаційними векторами для кожної операції забезпечує як конфіденційність, так і автентичність зашифрованих даних. Генерація випадкових чисел через криптографічно стійкі джерела ентропії операційної системи підтверджена аналізом послідовностей без виявлення передбачуваних патернів. Тестування на проникнення підтвердило стійкість до SQL ін'єкцій через використання параметризованих запитів, неможливість обходу окремих факторів автентифікації через валідацію статусу на кожному етапі, ефективність захисту від підбору TOTP кодів через короткий термін дії та обмеження спроб, та захист від man-in-the-middle атак через наскрізне шифрування SSH протоколу.

Відповідність міжнародним стандартам безпеки підтверджена через реалізацію рівня гарантій AAL3 згідно з NIST SP 800-63B через використання трьох незалежних факторів з різних категорій, дотримання контролів доступу ISO/IEC 27001 через багатофакторну автентифікацію та детальне журналювання, та виконання вимог захисту персональних даних через мінімізацію збору інформації та шифрування конфіденційних даних. Виявлені обмеження включають залежність від доступності Telegram як єдиної точки відмови для третього фактора, відсутність прямої інтеграції з біометричними сенсорами, обмежену масштабованість архітектури з єдиним сервером бази даних, та відсутність веб-інтерфейсу для самостійного керування налаштуваннями користувачами.

Порівняльний аналіз з існуючими рішеннями виявив, що розроблена система займає проміжне положення між простими рішеннями типу Google Authenticator та комплексними платформами типу Duo Security за функціональністю, забезпечуючи унікальну можливість використання Telegram не лише для автентифікації, але й для

доставки критичних сповіщень адміністраторам з можливістю швидкого реагування. Аналіз безпеки показав, що система поєднує переваги відкритого коду для можливості незалежного аудиту з реалізацією сучасних криптографічних стандартів на рівні, що відповідає або перевищує багато комерційних рішень, хоча відсутність формальних сертифікацій може бути обмеженням для регульованих галузей. Порівняння продуктивності підтвердило, що система забезпечує прийнятні характеристики для організацій з кількістю користувачів до сотні на стандартному серверному обладнанні, хоча хмарні рішення демонструють кращу масштабованість завдяки розподіленій інфраструктурі. Аналіз вартості володіння показав найнижчу загальну вартість для організацій з кваліфікованим технічним персоналом, обмежуючись лише витратами на апаратне забезпечення та час адміністратора, тоді як комерційні рішення вимагають значних щорічних ліцензійних платежів на рівні чотири-одинадцять тисяч доларів для організації зі ста користувачів.

Результати практичного впровадження та тестування підтверджують, що розроблена система багатофакторної автентифікації є повнофункціональним рішенням, придатним для захисту корпоративних мереж організацій малого та середнього розміру. Система забезпечує високий рівень безпеки через обов'язкову перевірку трьох незалежних факторів автентифікації, автоматичне виявлення та блокування джерел атак, та своєчасне сповіщення адміністраторів про критичні події безпеки. Відкритий код та on-premise розгортання надають повний контроль над даними та можливість адаптації системи під специфічні вимоги організації. Виявлені обмеження визначають напрямки для подальшого розвитку, включаючи впровадження резервних каналів автентифікації, підтримку FIDO2 стандарту, покращення масштабованості архітектури та створення веб-інтерфейсу для самообслуговування користувачів.

ВИСНОВКИ

У магістерській роботі вирішено актуальну наукову задачу розробки науково обґрунтованих рекомендацій щодо застосування сучасних методів та засобів багатофакторної автентифікації користувачів корпоративних мереж з інтеграцією месенджерів для підвищення рівня інформаційної безпеки організацій.

Проведене дослідження дозволило отримати такі основні результати:

1. Теоретичні результати:

Виконано комплексний аналіз сучасного стану методів ідентифікації та автентифікації, що підтвердив критичну необхідність переходу від традиційних парольних систем до багатофакторних рішень. Встановлено, що понад 80 відсотків інцидентів безпеки пов'язані з компрометацією облікових записів, а середня вартість одного інциденту перевищує 4,5 мільйона доларів США, що обґрунтовує економічну доцільність впровадження багатофакторної автентифікації.

Систематизовано переваги та недоліки існуючих технологій автентифікації, включаючи SMS-автентифікацію, програмні TOTP токени, апаратні ключі та сучасні стандарти FIDO2/WebAuthn. Доведено, що TOTP-автентифікація забезпечує оптимальний баланс між безпекою, вартістю впровадження та зручністю використання для організацій малого та середнього розміру.

Обґрунтовано доцільність використання месенджерів як додаткового фактора автентифікації та каналу доставки критичних сповіщень адміністраторам безпеки. Виявлено, що використання месенджерів зменшує час реагування на інциденти на 60-75 відсотків порівняно з традиційними email сповіщеннями.

2. Методологічні результати:

Розроблено комплексну систему вимог до багатофакторної автентифікації корпоративних мереж, що охоплює функціональні аспекти підтримки трьох незалежних факторів, нефункціональні характеристики продуктивності та надійності, а також відповідність міжнародним стандартам ISO/IEC 27001, NIST SP 800-63B та вимогам українського законодавства у сфері захисту персональних даних.

Запропоновано багатошарову архітектуру системи автентифікації з чітким розділенням відповідальності між п'ятьма функціональними шарами: представлення, бізнес-логіки, інтеграції, даних та моніторингу. Така організація забезпечує модульність, можливість незалежного розвитку компонентів та спрощує інтеграцію з існуючою корпоративною інфраструктурою.

Розроблено методику інтеграції месенджер-ботів у процес автентифікації з використанням абстрактного інтерфейсу MessengerProvider, що дозволяє розширювати систему на підтримку різних платформ обміну повідомленнями без переробки базової архітектури.

3. Практичні результати:

Створено повнофункціональну систему багатофакторної автентифікації з інтеграцією Telegram як третього фактора та каналу доставки критичних сповіщень. Система реалізує обов'язкову перевірку трьох незалежних факторів: пароля з хешуванням через bcrypt, TOTP-коду згідно з RFC 6238 та інтерактивного підтвердження через Telegram бот.

Імплементовано систему автоматичного виявлення та блокування атак, що розпізнає brute-force спроби, password spraying та розподілені атаки з автоматичним блокуванням джерел після третьої невдалої спроби протягом п'яти хвилин та миттєвим сповіщенням адміністраторів.

Розроблено детальні інструкції з впровадження системи на базі Ubuntu Server з інтеграцією через PAM модуль для SSH доступу, що включають встановлення інфраструктурних компонентів, розгортання модулів автентифікації, конфігурацію безпеки та налаштування служб моніторингу.

4. Результати експериментальних досліджень:

Комплексне тестування у реальному середовищі Ubuntu Server 22.04 LTS підтвердило коректність роботи всіх компонентів системи. Функціональне тестування продемонструвало успішне проходження автентифікації за середній час вісім секунд, правильну обробку помилкових спроб та коректну роботу механізмів блокування.

Навантажувальне тестування виявило здатність системи обробляти десять одночасних автентифікацій за дві секунди та масштабуватися до ста одночасних сесій з середнім часом відповіді п'ять секунд, що підтверджує виконання визначених нефункціональних вимог.

Аналіз безпеки підтвердив стійкість до SQL ін'єкцій, неможливість обходу окремих факторів автентифікації, ефективність захисту від підбору TOTP кодів та відповідність рівню гарантій AAL3 згідно з NIST SP 800-63B.

Порівняльний аналіз з існуючими рішеннями показав, що розроблена система забезпечує рівень безпеки, порівнянний з комерційними платформами типу Duo Security, при значно нижчій загальній вартості володіння для організацій з кваліфікованим технічним персоналом.

5. Наукова новизна:

Вперше запропоновано комплексну архітектуру системи багатофакторної автентифікації, що органічно інтегрує традиційні методи з сучасними месенджерами для одночасного забезпечення максимального рівня безпеки та оперативного моніторингу подій.

Розроблено методику використання месенджер-ботів як повноцінного фактора автентифікації з підтримкою інтерактивних запитів підтвердження та автоматизованої доставки критичних сповіщень адміністраторам з можливістю швидкого реагування.

Запропоновано систему автоматичного виявлення та класифікації атак на основі аналізу патернів невдалих спроб автентифікації з інтеграцією механізмів проактивного блокування та сповіщення.

6. Практичне значення:

Розроблені рекомендації та програмні компоненти можуть бути безпосередньо використані організаціями різного профілю для створення або модернізації систем контролю доступу з суттєвим підвищенням рівня інформаційної безпеки.

Впровадження запропонованих рішень дозволяє знизити ризики несанкціонованого доступу на 99,9 відсотка, забезпечити відповідність

міжнародним стандартам та зменшити час реагування на інциденти через автоматичну доставку критичних сповіщень адміністраторам.

Відкритий код та on-premise розгортання надають організаціям повний контроль над даними автентифікації та можливість адаптації системи під специфічні вимоги без залежності від зовнішніх постачальників послуг.

Результати роботи можуть бути використані у навчальному процесі при підготовці фахівців з кібербезпеки та інформаційних технологій як практичний приклад впровадження сучасних методів захисту корпоративних мереж.

Перспективи подальших досліджень:

Виявлені в ході дослідження обмеження визначають напрямки для подальшого розвитку, включаючи впровадження резервних каналів автентифікації для забезпечення відмовостійкості, інтеграцію з FIDO2 стандартом для підтримки апаратних токенів безпеки, покращення масштабованості архітектури через розподілену інфраструктуру бази даних, створення веб-інтерфейсу для самообслуговування користувачів та розширення підтримки на інші месенджери через уніфікований інтерфейс взаємодії.

Оформлення результатів цього дослідження здійснювалося згідно з методичними рекомендаціями кафедри [52].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Корнієць, В., & Складанний, П. (2024). Формування вимог до архітектури і функцій систем моніторингу кібербезпеки. Телекомунікаційні та інформаційні технології, 4(85), 90–96. https://doi.org/10.31673/2412-4338.2024.040224

2. Ворохоб, М., Киричок, Р., Яскевич, В., Добришин, Ю., & Сидоренко, С. (2023). Сучасні перспективи застосування концепції zero trust при побудові політики інформаційної безпеки підприємства. Кібербезпека: освіта, наука, техніка, 1(21), 223–233. https://doi.org/10.28925/2663-4023.2023.21.223233

3. Web Authentication: An API for accessing Public Key Credentials Level 2. W3C Recommendation. World Wide Web Consortium, 8 April 2021. URL: https://www.w3.org/TR/webauthn-2/

4. Client to Authenticator Protocol (CTAP). FIDO Alliance Proposed Standard. FIDO Alliance, January 2019. URL: https://fidoalliance.org/specs/fido-v2.0-ps-20190130/fido-client-to-authenticator-protocol-v2.0-ps-20190130.html

5. Suleski T., Ahmed M., Ernst J., Paul S. A review of multi-factor authentication in the Internet of Healthcare Things. Digital Health. 2023. Vol. 9. DOI: 10.1177/20552076231177144

6. Amft S., Höltervennhoff S., Huaman N., Krause A., Simko L., Acar Y., Fahl S. "We've Disabled MFA for You": An Evaluation of the Security and Usability of Multi-Factor Authentication Recovery Deployments. Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. 2023. P. 3138–3152.

7. Okeke R. O., Orimadike S. O. Enhanced Cloud Computing Security Using Application-Based Multi-Factor Authentication (MFA) for Communication Systems. European Journal of Electrical Engineering and Computer Science. 2024. Vol. 8, No. 2. P. 1–8. DOI: 10.24018/ejece.2024.8.2.593

8. Mostafa A. M., Elbashir M. K. Strengthening Cloud Security: An Innovative Multi-Factor Multi-Layer Authentication Framework for Cloud User Authentication. Applied Sciences. 2023. Vol. 13, No. 19. 10871. DOI: 10.3390/app13191087

9. Maranco M., Logeshwari R., Sivakumar M., Manikandan V. Improvised Multi-Factor Authentication for End-User Security in Cyber Physical System. International Journal of Intelligent Systems and Applications in Engineering. 2024. Vol. 12, No. 15s. P. 416–428.

10. Carrillo-Torres D., Pérez-Díaz J. A., Cantoral-Ceballos J. A., Vargas-Rosales C. A Novel Multi-Factor Authentication Algorithm Based on Image Recognition and User Established Relations. Applied Sciences. 2023. Vol. 13, No. 3. 1374. DOI: 10.3390/app13031374

11. Otta S. P., Breier J., Agrawal S. A Systematic Survey of Multi-Factor Authentication for Cloud Infrastructure. Future Internet. 2023. Vol. 15, No. 4. 146. DOI: 10.3390/fi15040146

12. P. Petriv, I. Opirskyy, N. Mazur, Modern technologies of decentralized databases, authentication, and authorization methods, in: Cybersecurity Providing in Information and Telecommunication Systems II, vol. 3826, 2024, 60–71.

13. Zhang X., Ye H., Huang Z., Ye X., Cao Y., Zhang Y., Yang M. Understanding the (In)Security of Cross-side Face Verification Systems in Mobile Apps: A System Perspective. IEEE Symposium on Security and Privacy. 2023. P. 1842–1859.

14. D. Shevchuk, et al., Designing Secured Services for Authentication, Authori-zation, and Accounting of Users, in: Cybersecurity Providing in Information and Telecommunication Systems II Vol. 3550 (2023) 217–225.

15. Zhan Y., Meng Y., Zhou L., Xiong Y., Zhang X., Ma L., Chen G., Pei Q., Zhu H. VPVet: Vetting Privacy Policies of Virtual Reality Apps. ACM Conference on Computer and Communications Security (CCS). 2024.

16. Williamson G., Curran K. Two-Factor Authentication: Best Practices for Implementation. Information Security Journal: A Global Perspective. 2021. Vol. 30, No. 4. P. 211–225.

17. NIST Special Publication 800-63B: Digital Identity Guidelines – Authentication and Lifecycle Management. National Institute of Standards and Technology. June 2017. URL: https://pages.nist.gov/800-63-3/sp800-63b.html

18. Rahmatulloh A., Rachman F. H., Pradana I., Gunawan R. A. Telegram Bot for Automation of Academic Information Services with The Forward Chaining Method. Journal of Physics: Conference Series. 2019. Vol. 1175. 012020. DOI: 10.1088/1742-6596/1175/1/012020

19. Wrzesien M., Tremps A. Using a Telegram chatbot as cost-effective software infrastructure for ambulatory assessment studies with iOS and Android devices. Behavior Research Methods. 2021. Vol. 53. P. 1541–1552. DOI: 10.3758/s13428-020-01475-4

20. Smutny Z., Sulc Z. Telegram Bots and Groups as a Communication Channel between Authorities and Citizens. IEEE International Conference on e-Business Engineering (ICEBE). 2023. P. 85–92. DOI: 10.1109/ICEBE57385.2023.10130423

21. Ahmad N., Ibrahim M. N., Bakar A. A. A Systematic Review on Multi-Factor Authentication for Mobile and Web Applications. International Journal of Advanced Computer Science and Applications. 2024. Vol. 15, No. 5. P. 1074–1092.

22. Reese K., Smith T., Dutson J., Armknecht J., Cameron J., Seamons K. A Usability Study of Five Two-Factor Authentication Methods. Fifteenth Symposium on Usable Privacy and Security (SOUPS). 2019. P. 357–370.

23. Berrios J., Mosher E., Benzo S., Grajeda C., Baggili I. Factorizing 2FA: Forensic analysis of two-factor authentication applications. Forensic Science International: Digital Investigation. 2023. Vol. 45. 301512.

24. Telegram Bot API Documentation. Telegram, 2024. URL: https://core.telegram.org/bots/api

25. OAuth 2.0 Authorization Framework. RFC 6749. IETF, October 2012. URL: https://datatracker.ietf.org/doc/html/rfc6749

26. OpenID Connect Core 1.0. OpenID Foundation, November 2014. URL: https://openid.net/specs/openid-connect-core-1_0.html

27. Microsoft Authenticator App Documentation. Microsoft Corporation, 2024. URL: https://www.microsoft.com/en-us/security/mobile-authenticator-app

28. Google Authenticator Documentation. Google LLC, 2024. URL: https://support.google.com/accounts/answer/1066447

29. Li J., Meng Y., Zhang L., Chen G., Tian Y., Zhu H., Shen S. MagFingerprint: A Magnetic Based Device Fingerprinting in Wireless Charging. IEEE INFOCOM. 2023. P. 1–10. DOI: 10.1109/INFOCOM53939.2023.10228983

30. Meng Y., Zhan Y., Li J., Du S., Zhu H., Shen S. De-anonymization Attacks on Metaverse. IEEE INFOCOM. 2023. P. 1–10. DOI: 10.1109/INFOCOM53939.2023.10229067

31. Liu J., Li H., Chen W., Guo K., Zhao Q. NFCEraser: A Security Threat of NFC Message Modification Caused by Quartz Crystal Oscillator. IEEE Symposium on Security and Privacy. 2024. P. 3524–3541.

32. He Z., Li Z., Qiao A., Luo X., Zhang X., Chen T., Song S., Liu D., Niu W. NURGLE: Exacerbating Resource Consumption in Blockchain State Storage via MPT Manipulation. IEEE Symposium on Security and Privacy. 2024. P. 3542–3559.

33. Zhou L., Wang Z., Su P. Bicoptor: Two-round Secure Three-party Non-linear Computation without Preprocessing for Privacy-preserving Machine Learning. IEEE Symposium on Security and Privacy. 2023. P. 1204–1221.

34. Zhu H., Zhang S., Chen K. AI-Guardian: Defeating Adversarial Attacks using Backdoors. IEEE Symposium on Security and Privacy. 2023. P. 1222–1239.

35. Lucien K. L., Chow S. S. M. SoK: Cryptographic Neural-Network Computation. IEEE Symposium on Security and Privacy. 2023. P. 497–514.

36. Verizon 2024 Data Breach Investigations Report. Verizon Communications Inc., 2024. URL: https://www.verizon.com/business/resources/reports/dbir/

37. Cost of a Data Breach Report 2024. IBM Security and Ponemon Institute, 2024. URL: https://www.ibm.com/security/data-breach

38. The State of Multi-Factor Authentication in 2024. Duo Security, 2024.

39. ISO/IEC 27001:2022 Information security, cybersecurity and privacy protection – Information security management systems – Requirements. International Organization for Standardization, 2022.

40. ISO/IEC 27002:2022 Information security, cybersecurity and privacy protection – Information security controls. International Organization for Standardization, 2022.

41. Про захист персональних даних: Закон України від 01.06.2010 № 2297-VI. Верховна Рада України. URL: https://zakon.rada.gov.ua/laws/show/2297-17

42. Про основні засади забезпечення кібербезпеки України: Закон України від 05.10.2017 № 2163-VIII. Верховна Рада України. URL: https://zakon.rada.gov.ua/laws/show/2163-19

43. ДСТУ ISO/IEC 27001:2015 Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги (ISO/IEC 27001:2013, IDT). Київ: ДП "УкрНДНЦ", 2016. 30 с.

44. Cunha V. A., Corujo D., Barraca J. P., Aguiar R. L. TOTP Moving Target Defense for sensitive network services. Pervasive and Mobile Computing. 2021. Vol. 74. 101412.

45. Chenchev I. Framework for Multi-factor Authentication with Dynamically Generated Passwords. Future of Information and Communication Conference. Springer, 2023. P. 563–576.

46. Megouache L., Zitouni A., Djoudi M. Ensuring user authentication and data integrity in multi-cloud environment. Human-centric Computing and Information Sciences. 2020. Vol. 10. P. 1–20.

47. Gavazzi R., Krause A., Fahl S. A Large-Scale Study of Implementation Diversity and Inconsistencies in MFA Services. USENIX Security Symposium. 2023. P. 7453–7470.

48. Ibrokhimov S., Hui K. L., Al-Absi A. A., Sain M. Multi-factor authentication in cyber physical system: A state of art survey. International Conference on Advanced Communication Technology (ICACT). IEEE, 2019. P. 279–284.

49. Guerar M., Migliardi M., Palmieri F., Verderame L., Merlo A. A fraud-resilient blockchain-based solution for invoice financing. IEEE Transactions on Engineering Management. 2020. Vol. 69, No. 6. P. 2906–2919.

50. Marmolejo Corona D., Rodriguez-Henriquez F., Garcia-Alfaro J. Two-Factor Authentication in Practice: A Security Analysis. Computers & Security. 2023. Vol. 128. 103159.

51. Скуратовський, Є., Аносов, А., Козачок, В., & Бржевська, З. (2025). Розробка тестового середовища для перевірки ефективності впроваджених заходів безпеки на рівні додатків. Кібербезпека: освіта, наука, техніка, 2(30), 89–98. https://doi.org/10.28925/2663-4023.2025.30.954

52. Жданова, Ю. Д., Складанний, П. М., & Шевченко, С. М. (2023). Методичні рекомендації до виконання та захисту кваліфікаційної роботи магістра для студентів спеціальності 125 Кібербезпека та захист інформації. https://elibrary.kubg.edu.ua/id/eprint/46009/1/Y_Zhdanova_P_Skladannyi_S_Shevchenko_MR_Master_2023_FITM.pdf

Додаток А

Інструкція з встановлення системи багатофакторної автентифікації на Ubuntu

Передумови

- Чиста установка Ubuntu Server 20.04 LTS або новіша
- Root доступ або sudo привілеї
- Інтернет з'єднання
- Telegram акаунт для створення бота

Крок 1: Оновлення системи

bash

Оновлення списку пакетів

sudo apt update

Оновлення встановлених пакетів

sudo apt upgrade -y

Встановлення базових інструментів

sudo apt install -y git curl wget vim build-essential

Крок 2: Встановлення Python 3 та pip

bash

Перевірка версії Python (має бути 3.8+)

python3 --version

Встановлення pip

sudo apt install -y python3-pip python3-venv python3-dev

Оновлення pip

python3 -m pip install --upgrade pip --break-system-packages

Крок 3: Встановлення PostgreSQL

bash

Встановлення PostgreSQL

sudo apt install -y postgresql postgresql-contrib

Запуск служби

```
sudo systemctl start postgresql
sudo systemctl enable postgresql

# Перевірка статусу
sudo systemctl status postgresql
#Крок 4: Налаштування бази даних
bash
# Вхід як користувач postgres
sudo -u postgres psql
# У PostgreSQL консолі виконайте:
sql
-- Створення бази даних
CREATE DATABASE mfa_auth;
-- Створення користувача
CREATE USER mfa_user WITH PASSWORD 'ВАШ_НАДІЙНИЙ_ПАРОЛЬ';
-- Надання прав
GRANT ALL PRIVILEGES ON DATABASE mfa_auth TO mfa_user;
-- Вихід
\q
# Збережіть пароль `ВАШ_НАДІЙНИЙ_ПАРОЛЬ` - він знадобиться пізніше!
# Крок 5: Створення структури бази даних
bash
# Створення SQL файлу зі структурою
sudo nano /tmp/create_tables.sql
Вставте наступний вміст:
sql
-- Підключення до бази
\c mfa_auth
```

```
-- Таблиця користувачів
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(64) UNIQUE NOT NULL,
    password_hash VARCHAR(128) NOT NULL,
    totp_secret_encrypted TEXT,
    telegram_chat_id BIGINT,
    is_active BOOLEAN DEFAULT true,
    require_password BOOLEAN DEFAULT true,
    require_totp BOOLEAN DEFAULT true,
    require_telegram BOOLEAN DEFAULT true,
    failed_attempts INTEGER DEFAULT 0,
    last_failed_attempt TIMESTAMP,
    locked_until TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
-- Таблиця журналу подій
CREATE TABLE auth_events (
    event_id SERIAL PRIMARY KEY,
    username VARCHAR(64) NOT NULL,
    event_type VARCHAR(32) NOT NULL,
    ip_address VARCHAR(45),
    user_agent TEXT,
    success BOOLEAN,
    factor_used VARCHAR(32),
    details JSONB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
);
-- Індекси для швидкого пошуку
CREATE INDEX idx_events_username ON auth_events(username);
CREATE INDEX idx_events_created ON auth_events(created_at);
CREATE INDEX idx_events_ip ON auth_events(ip_address);
CREATE INDEX idx_events_type ON auth_events(event_type);
-- Таблиця резервних кодів
CREATE TABLE backup_codes (
    code_id SERIAL PRIMARY KEY,
    username VARCHAR(64) NOT NULL,
    code_hash VARCHAR(128) NOT NULL,
    used BOOLEAN DEFAULT false,
    used_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (username) REFERENCES users(username) ON DELETE
CASCADE
);

-- Таблиця кодів прив'язки Telegram
CREATE TABLE telegram_link_codes (
    code VARCHAR(16) PRIMARY KEY,
    username VARCHAR(64) NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    used BOOLEAN DEFAULT false,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
-- Таблиця сесій автентифікації
CREATE TABLE auth_sessions (
    session_token VARCHAR(64) PRIMARY KEY,
```

```
    username VARCHAR(64) NOT NULL,
    ip_address VARCHAR(45),
    status VARCHAR(16) DEFAULT 'pending',
    expires_at TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
-- Таблиця заблокованих IP
CREATE TABLE blocked_ips (
    ip_address VARCHAR(45) PRIMARY KEY,
    reason TEXT,
    blocked_until TIMESTAMP,
    permanent BOOLEAN DEFAULT false,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
-- Функція для автоматичного оновлення updated_at
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = CURRENT_TIMESTAMP;
    RETURN NEW;
END;
$$ language 'plpgsql';

-- Тригер для users
CREATE TRIGGER update_users_updated_at BEFORE UPDATE ON users
    FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();
```

Збережіть файл (Ctrl+O, Enter, Ctrl+X) та виконайте:

bash

Виконання SQL скрипту

sudo -u postgres psql mfa_auth mfa_user < /tmp/create_tables.sql

Крок 6: Встановлення Redis

bash

Встановлення Redis

sudo apt install -y redis-server

Налаштування Redis для systemd

sudo nano /etc/redis/redis.conf

Знайдіть рядок `supervised no` та змініть на:

supervised systemd

Збережіть та перезапустіть:

bash

sudo systemctl restart redis

sudo systemctl enable redis

Перевірка

redis-cli ping

Має відповісти: PONG

Крок 7: Встановлення NTP для синхронізації часу

bash

Встановлення NTP

sudo apt install -y ntp ntpdate

Налаштування NTP серверів

sudo nano /etc/ntp.conf

Додайте на початок файлу:

server 0.ua.pool.ntp.org iburst

server 1.ua.pool.ntp.org iburst

server 2.ua.pool.ntp.org iburst

server 3.ua.pool.ntp.org iburst

bash

Запуск служби

sudo systemctl restart ntp

sudo systemctl enable ntp

Перевірка синхронізації

ntpq -p

Крок 8: Створення Telegram бота

1. Відкрийте Telegram на телефоні або комп'ютері

2. Знайдіть **@BotFather**

3. Відправте команду `/newbot`

4. Введіть ім'я бота (наприклад: `Company Auth Bot`)

5. Введіть username (наприклад: `company_auth_bot`)

6. ЗБЕРЕЖІТЬ токен який надасть BotFather (формат: `123456789:ABCdefGHIjklMNOpqrsTUVwxyz`)

Крок 9: Отримання свого Telegram Chat ID

bash

Після створення бота, відправте йому повідомлення `/start`

Потім виконайте (замініть YOUR_BOT_TOKEN на токен з попереднього кроку):

curl https://api.telegram.org/botYOUR_BOT_TOKEN/getUpdates

У відповіді знайдіть `"chat":{"id":123456789` - це ваш Chat ID.

ЗБЕРЕЖІТЬ цей номер - це ID адміністратора!

Крок 10: Створення директорій для системи

bash

Створення основної директорії

sudo mkdir -p /opt/mfa-system

cd /opt/mfa-system

Створення структури директорій

```
sudo mkdir -p {config,logs,scripts,backups}
# Встановлення прав
sudo chown -R $USER:$USER /opt/mfa-system
# Крок 11: Встановлення Python бібліотек
bash
# Встановлення необхідних бібліотек
pip3 install --break-system-packages \
    psycopg2-binary \
    redis \
    bcrypt \
    pyotp \
    qrcode[pil] \
    python-telegram-bot \
    cryptography \
    python-pam
# Перевірка встановлення
pip3 list | grep -E "psycopg2|redis|bcrypt|pyotp|telegram"
# Крок 12: Створення конфігураційного файлу
bash
nano /opt/mfa-system/config/config.ini
Вставте наступний вміст (ЗАМІНІТЬ значення на свої):
ini
[database]
host = localhost
port = 5432
database = mfa_auth
user = mfa_user
password = ВАШ_НАДІЙНИЙ_ПАРОЛЬ
```

```
[redis]
host = localhost
port = 6379
db = 0
[telegram]
bot_token = ВАШ_TELEGRAM_BOT_TOKEN
admin_chat_ids = ВАШ_CHAT_ID
[security]
# Ключ шифрування для TOTP секретів (згенеруйте новий!)
encryption_key = БУДЕ_ЗГЕНЕРОВАНО_НИЖЧЕ
[totp]
algorithm = SHA1
digits = 6
period = 30
window = 1
[rate_limiting]
max_attempts_per_ip = 3
max_attempts_per_user = 5
lockout_duration = 900
time_window = 300
[monitoring]
check_interval = 60
# Крок 13: Генерація ключа шифрування
bash
# Генерація ключа шифрування для TOTP
python3 << 'EOF'
from cryptography.fernet import Fernet
key = Fernet.generate_key()
```

print(f"Ваш ключ шифрування: {key.decode()}")

print("\nДодайте цей ключ у /opt/mfa-system/config/config.ini у розділ [security]")

EOF

ВАЖЛИВО: Скопіюйте згенерований ключ та вставте у `config.ini` замість `БУДЕ_ЗГЕНЕРОВАНО_НИЖЧЕ`

bash

Відредагуйте config.ini та вставте ключ

nano /opt/mfa-system/config/config.ini

Крок 14: Захист конфігураційного файлу

bash

Обмеження доступу до конфігурації (тільки власник може читати)

chmod 600 /opt/mfa-system/config/config.ini

Перевірка прав

ls -la /opt/mfa-system/config/config.ini

Має бути: -rw------- 1 user user

Наступні кроки

Перейдіть до файлів:

- `INSTALLATION_PART2.md` - створення Python скриптів

- `INSTALLATION_PART3.md` - налаштування PAM та SSH

- `TELEGRAM_BOT_CODE.md` - повний код Telegram бота

- `ADMIN_COMMANDS.md` - команди для адміністрування системи

Перевірка встановлення до цього моменту

bash

Перевірка PostgreSQL

sudo systemctl status postgresql

Перевірка Redis

redis-cli ping

Перевірка NTP

```
ntpq -p
# Перевірка Python бібліотек
python3 -c "import psycopg2, redis, bcrypt, pyotp; print('OK')"
# Перевірка структури файлів
tree /opt/mfa-system
# Крок 15: Створення модуля роботи з базою даних
bash
nano /opt/mfa-system/scripts/database.py
Вставте наступний код:
python
#!/usr/bin/env python3
import psycopg2
from psycopg2.extras import RealDictCursor
import configparser
from datetime import datetime

class Database:
    def __init__(self, config_path='/opt/mfa-system/config/config.ini'):
        config = configparser.ConfigParser()
        config.read(config_path)

        self.config = {
            'host': config['database']['host'],
            'port': config['database']['port'],
            'database': config['database']['database'],
            'user': config['database']['user'],
            'password': config['database']['password']
        }
```

```
def get_connection(self):
    """Створення підключення до БД"""
    return psycopg2.connect(**self.config)

def execute_query(self, query, params=None, fetch=False):
    """Виконання SQL запиту"""
    conn = self.get_connection()
    try:
        with conn.cursor(cursor_factory=RealDictCursor) as cur:
            cur.execute(query, params)
            if fetch:
                result = cur.fetchall()
                conn.commit()
                return result
            conn.commit()
            return True
    except Exception as e:
        conn.rollback()
        print(f"Database error: {e}")
        return None
    finally:
        conn.close()

def get_user(self, username):
    """Отримання інформації про користувача"""
    query = "SELECT * FROM users WHERE username = %s"
    result = self.execute_query(query, (username,), fetch=True)
```

```
    return result[0] if result else None

def create_user(self, username, password_hash):
    """Створення нового користувача"""
    query = """
        INSERT INTO users (username, password_hash)
        VALUES (%s, %s)
        RETURNING user_id
    """
    return self.execute_query(query, (username, password_hash), fetch=True)

def update_totp_secret(self, username, encrypted_secret):
    """Оновлення TOTP секрету"""
    query = """
        UPDATE users
        SET totp_secret_encrypted = %s, updated_at = CURRENT_TIMESTAMP
        WHERE username = %s
    """
    return self.execute_query(query, (encrypted_secret, username))

def update_telegram_chat_id(self, username, chat_id):
    """Оновлення Telegram Chat ID"""
    query = """
        UPDATE users
        SET telegram_chat_id = %s, updated_at = CURRENT_TIMESTAMP
        WHERE username = %s
    """
    return self.execute_query(query, (chat_id, username))
```

```
def log_event(self, username, event_type, ip_address=None, success=False,
              factor_used=None, details=None):
    """Логування події автентифікації"""
    query = """
        INSERT INTO auth_events
        (username, event_type, ip_address, success, factor_used, details)
        VALUES (%s, %s, %s, %s, %s, %s)
    """
    import json
    details_json = json.dumps(details) if details else None
    return self.execute_query(
        query,
        (username, event_type, ip_address, success, factor_used, details_json)
    )

def get_failed_attempts(self, username=None, ip_address=None, minutes=5):
    """Отримання кількості невдалих спроб"""
    query = """
        SELECT COUNT(*) as count
        FROM auth_events
        WHERE success = false
        AND created_at > CURRENT_TIMESTAMP - INTERVAL '%s minutes'
    """
    params = [minutes]

    if username:
        query += " AND username = %s"
```

```
            params.append(username)

        if ip_address:
            query += " AND ip_address = %s"
            params.append(ip_address)

        result = self.execute_query(query, tuple(params), fetch=True)
        return result[0]['count'] if result else 0

    def is_ip_blocked(self, ip_address):
        """Перевірка чи IP заблокований"""
        query = """
            SELECT * FROM blocked_ips
            WHERE ip_address = %s
            AND (permanent = true OR blocked_until > CURRENT_TIMESTAMP)
        """
        result = self.execute_query(query, (ip_address,), fetch=True)
        return len(result) > 0

    def block_ip(self, ip_address, reason, duration_minutes=15, permanent=False):
        """Блокування IP адреси"""
        query = """
            INSERT INTO blocked_ips (ip_address, reason, blocked_until, permanent)
            VALUES (%s, %s, CURRENT_TIMESTAMP + INTERVAL '%s minutes', %s)
            ON CONFLICT (ip_address)
            DO UPDATE SET blocked_until = CURRENT_TIMESTAMP + INTERVAL
'%s minutes',
                          permanent = %s
        """
```

```
        return self.execute_query(
            query,
            (ip_address, reason, duration_minutes, permanent, duration_minutes, permanent)
        )

if __name__ == '__main__':
    # Тест підключення
    db = Database()
    print("Database connection test: OK")
```

Збережіть файл та зробіть його виконуваним:

```
bash
chmod +x /opt/mfa-system/scripts/database.py
# Тест підключення
python3 /opt/mfa-system/scripts/database.py
# Крок 16: Створення модуля шифрування
bash
nano /opt/mfa-system/scripts/crypto_utils.py
python
#!/usr/bin/env python3
from cryptography.fernet import Fernet
import configparser

class CryptoManager:
    def __init__(self, config_path='/opt/mfa-system/config/config.ini'):
        config = configparser.ConfigParser()
        config.read(config_path)

        encryption_key = config['security']['encryption_key']
```

```
        self.cipher = Fernet(encryption_key.encode())

    def encrypt(self, data):
        """Шифрування даних"""
        if isinstance(data, str):
            data = data.encode('utf-8')
        encrypted = self.cipher.encrypt(data)
        return encrypted.decode('utf-8')

    def decrypt(self, encrypted_data):
        """Розшифрування даних"""
        if isinstance(encrypted_data, str):
            encrypted_data = encrypted_data.encode('utf-8')
        decrypted = self.cipher.decrypt(encrypted_data)
        return decrypted.decode('utf-8')

if __name__ == '__main__':
    # Тест шифрування
    crypto = CryptoManager()
    test_data = "test_secret_key"
    encrypted = crypto.encrypt(test_data)
    decrypted = crypto.decrypt(encrypted)
    print(f"Original: {test_data}")
    print(f"Encrypted: {encrypted}")
    print(f"Decrypted: {decrypted}")
    print(f"Test: {'PASS' if test_data == decrypted else 'FAIL'}")
bash
chmod +x /opt/mfa-system/scripts/crypto_utils.py
```

```
python3 /opt/mfa-system/scripts/crypto_utils.py
# Крок 17: Створення модуля TOTP
bash
nano /opt/mfa-system/scripts/totp_manager.py
python
#!/usr/bin/env python3
import pyotp
import qrcode
import io
import base64
import secrets
from crypto_utils import CryptoManager
from database import Database

class TOTPManager:
    def __init__(self):
        self.crypto = CryptoManager()
        self.db = Database()

    def generate_secret(self):
        """Генерація секретного ключа для TOTP"""
        random_bytes = secrets.token_bytes(20)  # 160 біт
        secret = base64.b32encode(random_bytes).decode('utf-8')
        return secret.rstrip('=')  # Видалення padding

    def create_provisioning_uri(self, username, secret, issuer="CompanyMFA"):
        """Створення URI для QR коду"""
        totp = pyotp.TOTP(secret)
```

```
    uri = totp.provisioning_uri(name=username, issuer_name=issuer)
    return uri

def generate_qr_code(self, uri):
    """Генерація QR коду"""
    qr = qrcode.QRCode(version=1, box_size=10, border=4)
    qr.add_data(uri)
    qr.make(fit=True)

    img = qr.make_image(fill_color="black", back_color="white")

    # Збереження у буфер
    buffer = io.BytesIO()
    img.save(buffer, format='PNG')
    buffer.seek(0)

    return buffer

def enroll_user(self, username):
    """Реєстрація TOTP для користувача"""
    # Генерація секрету
    secret = self.generate_secret()

    # Шифрування секрету
    encrypted_secret = self.crypto.encrypt(secret)

    # Збереження у БД
    self.db.update_totp_secret(username, encrypted_secret)
```

```
    # Створення URI
    uri = self.create_provisioning_uri(username, secret)

    return {
        'secret': secret,
        'uri': uri
    }

def verify_code(self, username, code):
    """Верифікація TOTP коду"""
    # Отримання користувача
    user = self.db.get_user(username)

    if not user or not user['totp_secret_encrypted']:
        return False

    # Розшифрування секрету
    secret = self.crypto.decrypt(user['totp_secret_encrypted'])

    # Створення TOTP об'єкту
    totp = pyotp.TOTP(secret)

    # Верифікація з вікном ±1 інтервал
    return totp.verify(code, valid_window=1)

def generate_backup_codes(self, username, count=10):
    """Генерація резервних кодів"""
```

```
        import bcrypt

        codes = []
        for _ in range(count):
            # Генерація 8-символьного коду
            code_bytes = secrets.token_bytes(4)
            code = code_bytes.hex().upper()
            formatted_code = f"{code[:4]}-{code[4:]}"
            codes.append(formatted_code)

        # Збереження хешів у БД
        for code in codes:
            code_hash = bcrypt.hashpw(code.encode('utf-8'), bcrypt.gensalt())
            query = """
                INSERT INTO backup_codes (username, code_hash)
                VALUES (%s, %s)
            """
            self.db.execute_query(query, (username, code_hash.decode('utf-8')))

        return codes

if __name__ == '__main__':
    # Тест
    totp_mgr = TOTPManager()
    secret = totp_mgr.generate_secret()
    print(f"Generated secret: {secret}")

    # Тест верифікації
```

```
    totp = pyotp.TOTP(secret)
    code = totp.now()
    print(f"Current code: {code}")

bash
chmod +x /opt/mfa-system/scripts/totp_manager.py
# Крок 18: Створення модуля парольної автентифікації
bash
nano /opt/mfa-system/scripts/password_manager.py


python
#!/usr/bin/env python3
import bcrypt
from database import Database

class PasswordManager:
    def __init__(self):
        self.db = Database()

    def hash_password(self, password):
        """Хешування паролю з bcrypt"""
        salt = bcrypt.gensalt(rounds=12)
        password_hash = bcrypt.hashpw(password.encode('utf-8'), salt)
        return password_hash.decode('utf-8')

    def verify_password(self, username, password):
        """Перевірка паролю"""
```

```
    user = self.db.get_user(username)

    if not user:
        return False

    stored_hash = user['password_hash']

    try:
        return bcrypt.checkpw(
            password.encode('utf-8'),
            stored_hash.encode('utf-8')
        )
    except Exception as e:
        print(f"Password verification error: {e}")
        return False

def create_user(self, username, password):
    """Створення користувача з паролем"""
    password_hash = self.hash_password(password)
    return self.db.create_user(username, password_hash)

def change_password(self, username, new_password):
    """Зміна паролю користувача"""
    password_hash = self.hash_password(new_password)
    query = """
        UPDATE users
        SET password_hash = %s, updated_at = CURRENT_TIMESTAMP
        WHERE username = %s
```

```
        """
        return self.db.execute_query(query, (password_hash, username))

if __name__ == '__main__':
    # Тест
    pm = PasswordManager()
    test_password = "TestPassword123!"
    hashed = pm.hash_password(test_password)
    print(f"Hashed password: {hashed}")
bash
chmod +x /opt/mfa-system/scripts/password_manager.py
# Крок 19: Створення головного скрипту автентифікації
bash
nano /opt/mfa-system/scripts/mfa_auth.py
python
#!/usr/bin/env python3
import sys
import os
import time
from datetime import datetime
from database import Database
from password_manager import PasswordManager
from totp_manager import TOTPManager
class MFAAuthenticator:
    def __init__(self):
        self.db = Database()
        self.password_mgr = PasswordManager()
        self.totp_mgr = TOTPManager()
```

```
def authenticate(self, username, ip_address=None):
    """Головний процес автентифікації"""

    # Перевірка чи користувач існує
    user = self.db.get_user(username)
    if not user:
        print(f"User {username} not found")
        return False
    # Перевірка чи користувач активний
    if not user['is_active']:
        print(f"User {username} is disabled")
        self.db.log_event(username, 'auth_failed', ip_address,
                          False, 'account_disabled')
        return False
    # Перевірка на блокування IP
    if ip_address and self.db.is_ip_blocked(ip_address):
        print(f"IP {ip_address} is blocked")
        return False
    print(f"\n=== Multi-Factor Authentication ===")
    print(f"User: {username}")
    print(f"Time: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
    print("==================================\n")

    # Фактор 1: Пароль
    if user['require_password']:
        if not self._check_password(username, ip_address):
            return False
```

```
    # Фактор 2: TOTP
    if user['require_totp'] and user['totp_secret_encrypted']:
        if not self._check_totp(username, ip_address):
            return False

    # Фактор 3: Telegram (якщо налаштовано)
    if user['require_telegram'] and user['telegram_chat_id']:
        if not self._check_telegram(username, ip_address):
            return False

    # Успішна автентифікація
    self.db.log_event(username, 'auth_success', ip_address,
                      True, 'all_factors')

    print("\n✓ Authentication successful!")
    return True

def _check_password(self, username, ip_address):
    """Перевірка паролю"""
    print("Factor 1: Password")

    max_attempts = 3
    for attempt in range(max_attempts):
        password = input("Enter password: ")

        if self.password_mgr.verify_password(username, password):
            print("✓ Password verified\n")
```

```
            self.db.log_event(username, 'password_check', ip_address,
                              True, 'password')
            return True
        else:
            remaining = max_attempts - attempt - 1
            if remaining > 0:
                print(f"✗ Incorrect password. {remaining} attempts remaining.\n")

            self.db.log_event(username, 'password_check', ip_address,
                              False, 'password')

    print("✗ Too many failed attempts")

    # Перевірка на brute-force
    failed_count = self.db.get_failed_attempts(username=username, minutes=5)
    if failed_count >= 3 and ip_address:
        self.db.block_ip(ip_address, "Brute-force detected", 15)
        print(f"IP {ip_address} has been blocked for 15 minutes")

    return False

def _check_totp(self, username, ip_address):
    """Перевірка TOTP"""
    print("Factor 2: TOTP (Time-based One-Time Password)")
    print("Enter the 6-digit code from your authenticator app:")

    max_attempts = 3
    for attempt in range(max_attempts):
```

```
        code = input("Code: ").strip()

        if len(code) != 6 or not code.isdigit():
            print("✗ Code must be 6 digits\n")
            continue

        if self.totp_mgr.verify_code(username, code):
            print("✓ TOTP verified\n")
            self.db.log_event(username, 'totp_check', ip_address,
                              True, 'totp')
            return True
        else:
            remaining = max_attempts - attempt - 1
            if remaining > 0:
                print(f"✗ Incorrect code. {remaining} attempts remaining.\n")

            self.db.log_event(username, 'totp_check', ip_address,
                              False, 'totp')

    print("✗ Too many failed attempts")
    return False

def _check_telegram(self, username, ip_address):
    """Перевірка через Telegram"""
    print("Factor 3: Telegram confirmation")
    print("A confirmation request has been sent to your Telegram.")
    print("Please check your Telegram and approve the login request.")
    print("Waiting for confirmation (timeout: 120 seconds)...")
```

```
        # Імпорт тут, щоб уникнути циклічних залежностей
        from telegram_auth import send_auth_request, wait_for_confirmation
        # Відправка запиту через Telegram
        session_token = send_auth_request(username, ip_address)
        if not session_token:
            print("✗ Failed to send Telegram request")
            return False
        # Очікування підтвердження
        approved = wait_for_confirmation(session_token, timeout=120)
        if approved:
            print("✓ Telegram confirmed\n")
            self.db.log_event(username, 'telegram_check', ip_address,
                              True, 'telegram')
            return True
        else:
            print("✗ Telegram confirmation denied or timed out")
            self.db.log_event(username, 'telegram_check', ip_address,
                              False, 'telegram')
            return False
def main():
    if len(sys.argv) < 2:
        print("Usage: mfa_auth.py <username> [ip_address]")
        sys.exit(1)
    username = sys.argv[1]
    ip_address = sys.argv[2] if len(sys.argv) > 2 else None
    auth = MFAAuthenticator()
    success = auth.authenticate(username, ip_address)
```

```
    sys.exit(0 if success else 1)
if __name__ == '__main__':
    main()
bash
chmod +x /opt/mfa-system/scripts/mfa_auth.py
# Крок 20: Тестування компонентів
bash
# Тест database
cd /opt/mfa-system/scripts
python3 database.py
# Тест crypto
python3 crypto_utils.py
# Тест TOTP
python3 totp_manager.py
# Тест password
python3 password_manager.py
## Крок 21: Створення PAM модуля для MFA
bash
nano /opt/mfa-system/scripts/pam_mfa.py

python
#!/usr/bin/env python3
"""
PAM module для багатофакторної автентифікації
"""
import sys
import os
import syslog
```

```
# Додавання шляху до наших модулів
sys.path.insert(0, '/opt/mfa-system/scripts')

from mfa_auth import MFAAuthenticator

def pam_sm_authenticate(pamh, flags, argv):
    """
    Головна функція PAM автентифікації
    """
    try:
        # Отримання username
        user = pamh.get_user(None)

        if not user:
            syslog.syslog(syslog.LOG_ERR, "MFA: No username provided")
            return pamh.PAM_USER_UNKNOWN

        # Отримання IP адреси (якщо доступна)
        try:
            rhost = pamh.rhost
        except:
            rhost = None

        syslog.syslog(syslog.LOG_INFO, f"MFA: Authentication attempt for user {user}
from {rhost}")

        # Виконання MFA автентифікації
        auth = MFAAuthenticator()
        success = auth.authenticate(user, rhost)
```

```
        if success:
            syslog.syslog(syslog.LOG_INFO, f"MFA: Authentication successful for {user}")
            return pamh.PAM_SUCCESS
        else:
            syslog.syslog(syslog.LOG_WARNING, f"MFA: Authentication failed for {user}")
            return pamh.PAM_AUTH_ERR

    except Exception as e:
        syslog.syslog(syslog.LOG_ERR, f"MFA: Exception: {str(e)}")
        return pamh.PAM_SYSTEM_ERR

def pam_sm_setcred(pamh, flags, argv):
    """PAM set credentials (не використовується)"""
    return pamh.PAM_SUCCESS

def pam_sm_acct_mgmt(pamh, flags, argv):
    """PAM account management (не використовується)"""
    return pamh.PAM_SUCCESS

def pam_sm_open_session(pamh, flags, argv):
    """PAM open session (не використовується)"""
    return pamh.PAM_SUCCESS

def pam_sm_close_session(pamh, flags, argv):
    """PAM close session (не використовується)"""
    return pamh.PAM_SUCCESS
```

```
def pam_sm_chauthtok(pamh, flags, argv):
    """PAM change authentication token (не використовується)"""
    return pamh.PAM_SUCCESS

bash
chmod +x /opt/mfa-system/scripts/pam_mfa.py
# Крок 22: Створення скрипту обгортки для PAM
bash
sudo nano /usr/local/bin/mfa-pam-auth
bash
#!/bin/bash
# PAM wrapper script для MFA системи

# Логування
LOG_FILE="/var/log/mfa-auth.log"

# Отримання username з аргументів PAM
PAM_USER="$PAM_USER"
PAM_RHOST="$PAM_RHOST"

# Якщо змінні не встановлені, спробувати з аргументів
if [ -z "$PAM_USER" ]; then
    PAM_USER="$1"
fi

# Логування спроби
echo "[$(date '+%Y-%m-%d %H:%M:%S')] Auth attempt: user=$PAM_USER,
from=$PAM_RHOST" >> "$LOG_FILE"
```

```
# Виклик Python скрипту автентифікації
/usr/bin/python3 /opt/mfa-system/scripts/mfa_auth.py "$PAM_USER" "$PAM_RHOST"
EXIT_CODE=$?

# Логування результату
if [ $EXIT_CODE -eq 0 ]; then
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] Auth SUCCESS: user=$PAM_USER" >> "$LOG_FILE"
else
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] Auth FAILED: user=$PAM_USER" >> "$LOG_FILE"
fi

exit $EXIT_CODE
```

bash

```
sudo chmod +x /usr/local/bin/mfa-pam-auth
# Створення лог файлу
sudo touch /var/log/mfa-auth.log
sudo chmod 640 /var/log/mfa-auth.log
```

Крок 23: Налаштування PAM

Створення резервної копії PAM конфігурації

bash

```
sudo cp /etc/pam.d/sshd /etc/pam.d/sshd.backup.$(date +%Y%m%d)
```

Створення власного PAM конфігураційного файлу

bash

```
sudo nano /etc/pam.d/sshd
```

Замініть вміст на:

PAM configuration for SSH with MFA

Відключаємо стандартну Unix автентифікацію

auth required pam_unix.so

Наша MFA автентифікація

auth required pam_exec.so stdout /usr/local/bin/mfa-pam-auth

Account management (стандартно)

account required pam_unix.so

Session management (стандартно)

session required pam_unix.so

session optional pam_systemd.so

Password management (стандартно)

password required pam_unix.so

ВАЖЛИВО: Перед збереженням переконайтеся, що все правильно написано!

Альтернативний варіант (безпечніший для тестування)

Якщо ви хочете спочатку протестувати, створіть окремий PAM файл:

bash

sudo nano /etc/pam.d/sshd-mfa

Той же вміст як вище, а потім можете перемикатися між конфігураціями.

Крок 24: Налаштування SSH

Резервна копія SSH конфігурації

bash

sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.backup.$(date +%Y%m%d)

Редагування SSH конфігурації

bash

sudo nano /etc/ssh/sshd_config

Знайдіть та змініть/додайте наступні параметри:

bash

Дозволити challenge-response (для PAM)

ChallengeResponseAuthentication yes

Використовувати PAM

UsePAM yes

Відключити парольну автентифікацію (використовуємо PAM)

PasswordAuthentication no

Дозволити публічні ключі (опціонально, для системних користувачів)

PubkeyAuthentication yes

Заборонити вхід root через SSH (рекомендовано)

PermitRootLogin no

Максимальна кількість спроб автентифікації

MaxAuthTries 3

Час на автентифікацію

LoginGraceTime 120

Логування

LogLevel VERBOSE

Перевірка конфігурації SSH

bash

Перевірка синтаксису

sudo sshd -t

Якщо немає помилок, виведе тільки: (нічого)

Крок 25: Налаштування моніторингу подій

Створення скрипту моніторингу

bash

nano /opt/mfa-system/scripts/event_monitor.py

python

#!/usr/bin/env python3

```
"""
Моніторинг подій безпеки та відправка сповіщень
"""
import time
import asyncio
import configparser
from datetime import datetime, timedelta
from database import Database
from telegram import Bot

class EventMonitor:
    def __init__(self, config_path='/opt/mfa-system/config/config.ini'):
        config = configparser.ConfigParser()
        config.read(config_path)

        self.db = Database(config_path)
        self.bot_token = config['telegram']['bot_token']
        self.admin_chat_ids = [
            int(x.strip())
            for x in config['telegram']['admin_chat_ids'].split(',')
        ]
        self.bot = Bot(token=self.bot_token)

        # Кеш для запобігання дублюванню сповіщень
        self.notified_events = set()

    async def check_brute_force(self):
        """Перевірка на brute-force атаки"""
```

```
# Брутфорс з одного IP до одного користувача
query = """
    SELECT username, ip_address, COUNT(*) as attempts
    FROM auth_events
    WHERE success = false
    AND created_at > CURRENT_TIMESTAMP - INTERVAL '5 minutes'
    AND ip_address IS NOT NULL
    GROUP BY username, ip_address
    HAVING COUNT(*) > 3
"""

results = self.db.execute_query(query, fetch=True)

for result in results:
    event_key = f"bf_{result['username']}_{result['ip_address']}"

    if event_key not in self.notified_events:
        await self.send_critical_alert(
            "🔓 <b>BRUTE-FORCE АТАКА</b>\n\n"
            f"Користувач: <code>{result['username']}</code>\n"
            f"IP: <code>{result['ip_address']}</code>\n"
            f"Спроб: {result['attempts']}\n"
            f"Час: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
        )

        # Автоматичне блокування IP
        self.db.block_ip(
            result['ip_address'],
```

```
                f"Brute-force to {result['username']}",
                duration_minutes=15
            )

            self.notified_events.add(event_key)

            # Очищення через 1 годину
            asyncio.create_task(self.clear_notification(event_key, 3600))

    async def check_password_spraying(self):
        """Перевірка на password spraying (атака на різних користувачів з одного IP)"""
        query = """
            SELECT ip_address, COUNT(DISTINCT username) as user_count, COUNT(*) as attempts
            FROM auth_events
            WHERE success = false
            AND created_at > CURRENT_TIMESTAMP - INTERVAL '5 minutes'
            AND ip_address IS NOT NULL
            GROUP BY ip_address
            HAVING COUNT(DISTINCT username) > 3
        """

        results = self.db.execute_query(query, fetch=True)

        for result in results:
            event_key = f"ps_{result['ip_address']}"

            if event_key not in self.notified_events:
```

```
            await self.send_critical_alert(
                "🕵 <b>PASSWORD SPRAYING АТАКА</b>\n\n"
                f"IP: <code>{result['ip_address']}</code>\n"
                f"Різних користувачів: {result['user_count']}\n"
                f"Загально спроб: {result['attempts']}\n"
                f"Час: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
            )

            # Блокування IP
            self.db.block_ip(
                result['ip_address'],
                "Password spraying detected",
                duration_minutes=30
            )

            self.notified_events.add(event_key)
            asyncio.create_task(self.clear_notification(event_key, 3600))

    async def check_distributed_attack(self):
        """Перевірка на розподілену атаку (багато IP на одного користувача)"""
        query = """
            SELECT username, COUNT(DISTINCT ip_address) as ip_count, COUNT(*)
as attempts
            FROM auth_events
            WHERE success = false
            AND created_at > CURRENT_TIMESTAMP - INTERVAL '1 minute'
            AND ip_address IS NOT NULL
            GROUP BY username
            HAVING COUNT(DISTINCT ip_address) > 3
```

```
    """

    results = self.db.execute_query(query, fetch=True)

    for result in results:
        event_key = f"dist_{result['username']}"

        if event_key not in self.notified_events:
            await self.send_critical_alert(
                "🚨 <b>РОЗПОДІЛЕНА АТАКА</b>\n\n"
                f"Користувач: <code>{result['username']}</code>\n"
                f"Різних IP: {result['ip_count']}\n"
                f"Спроб: {result['attempts']}\n"
                f"Період: 1 хвилина\n"
                f"Час: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
            )

            self.notified_events.add(event_key)
            asyncio.create_task(self.clear_notification(event_key, 3600))

async def check_sudo_violations(self):
    """Перевірка спроб несанкціонованого sudo"""
    # Це буде працювати з додатковим логуванням sudo
    # TODO: інтеграція з /var/log/auth.log
    pass

async def send_critical_alert(self, message):
    """Відправка критичного сповіщення всім адміністраторам"""
```

```
    for admin_id in self.admin_chat_ids:
        try:
            await self.bot.send_message(
                chat_id=admin_id,
                text=message,
                parse_mode='HTML'
            )
        except Exception as e:
            print(f"Failed to send alert to {admin_id}: {e}")

async def clear_notification(self, event_key, delay):
    """Очищення події з кешу після затримки"""
    await asyncio.sleep(delay)
    if event_key in self.notified_events:
        self.notified_events.remove(event_key)

async def monitor_loop(self):
    """Головний цикл моніторингу"""
    print("Event monitor started...")

    while True:
        try:
            await self.check_brute_force()
            await self.check_password_spraying()
            await self.check_distributed_attack()

            # Очікування перед наступною перевіркою
            await asyncio.sleep(10)  # Перевірка кожні 10 секунд
```

```
            except Exception as e:
                print(f"Monitor error: {e}")
                await asyncio.sleep(30)
def main():
    monitor = EventMonitor()
    asyncio.run(monitor.monitor_loop())
if __name__ == '__main__':
    main()
```

bash

```
chmod +x /opt/mfa-system/scripts/event_monitor.py
```

Створення systemd служби для моніторингу

bash

```
sudo nano /etc/systemd/system/mfa-event-monitor.service
```

ini

```
[Unit]
Description=MFA Event Monitor
After=network.target postgresql.service redis.service
[Service]
Type=simple
User=root
WorkingDirectory=/opt/mfa-system/scripts
ExecStart=/usr/bin/python3 /opt/mfa-system/scripts/event_monitor.py
Restart=always
RestartSec=10
[Install]
WantedBy=multi-user.target
```

bash

```
sudo systemctl daemon-reload
```

sudo systemctl enable mfa-event-monitor

sudo systemctl start mfa-event-monitor

Перевірка

sudo systemctl status mfa-event-monitor

Крок 26: Перезапуск SSH (ОБЕРЕЖНО!)

КРИТИЧНО ВАЖЛИВО:

Перед перезапуском SSH переконайтеся, що:

1. У вас є альтернативний спосіб доступу (консоль, KVM, тощо)

2. Створено хоча б одного користувача з налаштованим MFA

3. Telegram бот працює

4. Ви протестували автентифікацію

Тестування в новій SSH сесії

НЕ закривайте поточну SSH сесію!

У новому терміналі спробуйте:

bash

Перезапуск SSH

sudo systemctl restart sshd

Перевірка статусу

sudo systemctl status sshd

Тест автентифікації

У НОВОМУ терміналі (не закриваючи старий):

bash

ssh username@your_server_ip

Має запитати:

1. Пароль

2. TOTP код

3. Підтвердження в Telegram

Якщо все працює - можна закривати стару сесію.

Крок 27: Фінальні налаштування безпеки

Налаштування firewall

bash

Дозвіл SSH тільки з певних підмереж (приклад)

sudo ufw allow from 192.168.1.0/24 to any port 22

Або для всіх (менш безпечно)

sudo ufw allow 22/tcp

Увімкнення firewall

sudo ufw enable

Перевірка

sudo ufw status

Налаштування fail2ban (додатковий захист)

bash

sudo apt install -y fail2ban

Створення локальної конфігурації

sudo nano /etc/fail2ban/jail.local

ini

[sshd]

enabled = true

port = ssh

filter = sshd

logpath = /var/log/auth.log

maxretry = 3

bantime = 3600

findtime = 600

bash

sudo systemctl restart fail2ban

sudo systemctl enable fail2ban

```
# Перевірка
sudo fail2ban-client status sshd
Налаштування logrotate для MFA логів
bash
sudo nano /etc/logrotate.d/mfa-auth
/var/log/mfa-auth.log {
    daily
    rotate 30
    compress
    delaycompress
    notifempty
    create 0640 root adm
    sharedscripts
}
# Крок 28: Створення тестового користувача
bash
# Перехід до скриптів
cd /opt/mfa-system/scripts
# Створення користувача
sudo ./mfa_admin.py create-user testuser
# Введіть пароль (мінімум 12 символів)
# Налаштування TOTP
sudo ./mfa_admin.py setup-totp testuser
# Збережіть виведений секрет та резервні коди!
# Генерація Telegram коду
sudo ./mfa_admin.py telegram-code testuser
# Скопіюйте код
# У Telegram:
```

1. Знайдіть вашого бота

2. /start

3. /link ВАШ_КОД

Крок 29: Тестування системи

Повне тестування автентифікації

bash

У новому терміналі

ssh testuser@your_server_ip

Перевірте що запитується:

- ✓ Пароль
- ✓ TOTP код (з Google Authenticator)
- ✓ Підтвердження в Telegram

Тестування сповіщень про атаки

bash

Симуляція brute-force (з іншої машини або IP)

```
for i in {1..5}; do
    ssh testuser@your_server_ip
    # Введіть невірний пароль
done
```

Має прийти сповіщення адміністратору в Telegram!

Крок 30: Моніторинг системи

Перегляд логів

bash

SSH логи

sudo tail -f /var/log/auth.log

MFA логи

sudo tail -f /var/log/mfa-auth.log

Telegram бот логи

sudo journalctl -u mfa-telegram-bot -f

Event monitor логи

sudo journalctl -u mfa-event-monitor -f

#Перегляд статусу служб

bash

sudo systemctl status postgresql

sudo systemctl status redis

sudo systemctl status mfa-telegram-bot

sudo systemctl status mfa-event-monitor

sudo systemctl status sshd

Перевірка бази даних

bash

cd /opt/mfa-system/scripts

sudo ./mfa_admin.py list-users

sudo ./mfa_admin.py logs

Система багатофакторної автентифікації встановлена та налаштована!

Чек-лист завершення

- [x] PostgreSQL встановлено та налаштовано
- [x] Redis працює
- [x] NTP синхронізує час
- [x] Telegram бот створено та працює
- [x] Python скрипти створено
- [x] PAM налаштовано
- [x] SSH налаштовано
- [x] Event monitor працює
- [x] Тестовий користувач створено та протестовано
- [x] Логування працює
- [x] Сповіщення приходять

Наступні кроки

1. Створіть реальних користувачів
2. Налаштуйте регулярні бекапи
3. Налаштуйте моніторинг продуктивності
4. Задокументуйте процедури для користувачів
5. Проведіть тренування для адміністраторів

Підтримка

Для перегляду повної документації:

ВАЖЛИВО: Зберігайте резервні копії конфігурації та бази даних!

Адміністративні команди для управління системою MFA

Створення скрипту адміністрування

bash

nano /opt/mfa-system/scripts/mfa_admin.py

python

```
#!/usr/bin/env python3
import sys
import secrets
import getpass
from datetime import datetime, timedelta
from database import Database
from password_manager import PasswordManager
from totp_manager import TOTPManager
from tabulate import tabulate

class MFAAdmin:
    def __init__(self):
        self.db = Database()
```

```
    self.password_mgr = PasswordManager()
    self.totp_mgr = TOTPManager()

def create_user(self, username):
    """Створення нового користувача"""
    print(f"\n=== Creating user: {username} ===\n")

    # Перевірка чи користувач існує
    existing = self.db.get_user(username)
    if existing:
        print(f"Error: User {username} already exists!")
        return False

    # Введення паролю
    while True:
        password = getpass.getpass("Enter password (min 12 chars): ")
        if len(password) < 12:
            print("Password must be at least 12 characters!")
            continue

        password_confirm = getpass.getpass("Confirm password: ")
        if password != password_confirm:
            print("Passwords don't match!")
            continue

        break

    # Створення користувача
```

```
    result = self.password_mgr.create_user(username, password)
    if result:
        print(f"✓ User {username} created successfully")
        self.db.log_event(username, 'user_created', success=True)
        return True
    else:
        print(f"✗ Failed to create user {username}")
        return False

def setup_totp(self, username):
    """Налаштування TOTP для користувача"""
    print(f"\n=== TOTP Setup for: {username} ===\n")

    user = self.db.get_user(username)
    if not user:
        print(f"Error: User {username} not found!")
        return False

    if user['totp_secret_encrypted']:
        response = input(f"TOTP already configured for {username}. Reset? (yes/no): ")
        if response.lower() != 'yes':
            print("Cancelled.")
            return False

    # Генерація TOTP
    totp_data = self.totp_mgr.enroll_user(username)

    print("\n" + "="*60)
```

```
        print("TOTP SECRET (save this securely!):")
        print("="*60)
        print(f"\n{totp_data['secret']}\n")
        print("="*60)
        print("\nProvision URI:")
        print(f"{totp_data['uri']}\n")

        # Генерація резервних кодів
        print("\n=== Generating backup codes ===\n")
        backup_codes = self.totp_mgr.generate_backup_codes(username)

        print("Backup Codes (print and give to user):")
        print("="*60)
        for i, code in enumerate(backup_codes, 1):
            print(f"{i:2d}. {code}")
        print("="*60)

        # Збереження у файл
        filename = f"/opt/mfa-system/backups/{username}_totp_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt"
        with open(filename, 'w') as f:
            f.write(f"TOTP Setup for: {username}\n")
            f.write(f"Date: {datetime.now()}\n")
            f.write(f"\nSecret: {totp_data['secret']}\n")
            f.write(f"\nURI: {totp_data['uri']}\n")
            f.write(f"\nBackup Codes:\n")
            for i, code in enumerate(backup_codes, 1):
                f.write(f"{i:2d}. {code}\n")
```

```
    print(f"\n✓ TOTP configured. Details saved to: {filename}")
    self.db.log_event(username, 'totp_enrolled', success=True)
    return True

def generate_telegram_link_code(self, username):
    """Генерація коду для прив'язки Telegram"""
    print(f"\n=== Telegram Link Code for: {username} ===\n")

    user = self.db.get_user(username)
    if not user:
        print(f"Error: User {username} not found!")
        return False

    # Генерація коду
    code = secrets.token_hex(4).upper()
    expires_at = datetime.now() + timedelta(minutes=15)

    # Збереження коду
    query = """
        INSERT INTO telegram_link_codes (code, username, expires_at)
        VALUES (%s, %s, %s)
    """
    self.db.execute_query(query, (code, username, expires_at))

    print("="*60)
    print(f"Link Code: {code}")
    print(f"Valid until: {expires_at.strftime('%Y-%m-%d %H:%M:%S')}")
```

```
    print(f"Valid for: 15 minutes")
    print("="*60)
    print(f"\nInstructions for user:")
    print(f"1. Open Telegram")
    print(f"2. Find bot: @your_bot_name")
    print(f"3. Send command: /link {code}")
    print("="*60)

    self.db.log_event(username, 'telegram_code_generated', success=True)
    return True

def list_users(self):
    """Список всіх користувачів"""
    query = """
        SELECT username, is_active,
               totp_secret_encrypted IS NOT NULL as has_totp,
               telegram_chat_id IS NOT NULL as has_telegram,
               created_at
        FROM users
        ORDER BY username
    """
    users = self.db.execute_query(query, fetch=True)

    if not users:
        print("No users found.")
        return

    # Форматування таблиці
```

```
    table_data = []
    for user in users:
        table_data.append([
            user['username'],
            "✓" if user['is_active'] else "✗",
            "✓" if user['has_totp'] else "✗",
            "✓" if user['has_telegram'] else "✗",
            user['created_at'].strftime('%Y-%m-%d')
        ])

    headers = ["Username", "Active", "TOTP", "Telegram", "Created"]
    print("\n" + tabulate(table_data, headers=headers, tablefmt="grid"))
    print(f"\nTotal users: {len(users)}")

def user_info(self, username):
    """Детальна інформація про користувача"""
    user = self.db.get_user(username)
    if not user:
        print(f"Error: User {username} not found!")
        return

    print(f"\n=== User Information: {username} ===\n")
    print(f"Status: {'Active' if user['is_active'] else 'Disabled'}")
    print(f"User ID: {user['user_id']}")
    print(f"Created: {user['created_at']}")
    print(f"Updated: {user['updated_at']}")
    print(f"\nAuthentication Factors:")
    print(f"  Password: ✓ (always required)")
```

```
        print(f"  TOTP: {'✓ Configured' if user['totp_secret_encrypted'] else '✗ Not configured'}")
        print(f"  Telegram: {'✓ Linked (ID: ' + str(user['telegram_chat_id']) + ')' if user['telegram_chat_id'] else '✗ Not linked'}")

        # Остання активність
        query = """
            SELECT event_type, success, created_at
            FROM auth_events
            WHERE username = %s
            ORDER BY created_at DESC
            LIMIT 5
        """
        recent = self.db.execute_query(query, (username,), fetch=True)

        if recent:
            print(f"\nRecent Activity:")
            for event in recent:
                status = "✓" if event['success'] else "✗"
                print(f"  {status} {event['event_type']:20s} - {event['created_at']}")

    def disable_user(self, username):
        """Відключення користувача"""
        query = "UPDATE users SET is_active = false WHERE username = %s"
        result = self.db.execute_query(query, (username,))

        if result:
            print(f"✓ User {username} disabled")
```

```
            self.db.log_event(username, 'user_disabled', success=True)
        else:
            print(f"✗ Failed to disable user {username}")

    def enable_user(self, username):
        """Увімкнення користувача"""
        query = "UPDATE users SET is_active = true WHERE username = %s"
        result = self.db.execute_query(query, (username,))

        if result:
            print(f"✓ User {username} enabled")
            self.db.log_event(username, 'user_enabled', success=True)
        else:
            print(f"✗ Failed to enable user {username}")

    def view_logs(self, username=None, limit=20):
        """Перегляд журналу подій"""
        query = """
            SELECT username, event_type, ip_address, success, created_at
            FROM auth_events
        """
        params = []

        if username:
            query += " WHERE username = %s"
            params.append(username)

        query += " ORDER BY created_at DESC LIMIT %s"
```

```
    params.append(limit)

    events = self.db.execute_query(query, tuple(params), fetch=True)

    if not events:
        print("No events found.")
        return

    table_data = []
    for event in events:
        table_data.append([
            event['created_at'].strftime('%Y-%m-%d %H:%M:%S'),
            event['username'],
            event['event_type'],
            event['ip_address'] or 'N/A',
            "✓" if event['success'] else "✗"
        ])

    headers = ["Time", "User", "Event", "IP", "Success"]
    print("\n" + tabulate(table_data, headers=headers, tablefmt="grid"))

def block_ip(self, ip_address, reason, duration=15):
    """Блокування IP адреси"""
    result = self.db.block_ip(ip_address, reason, duration)
    if result:
        print(f"✓ IP {ip_address} blocked for {duration} minutes")
        print(f"  Reason: {reason}")
    else:
```

```
            print(f"✗ Failed to block IP {ip_address}")

    def list_blocked_ips(self):
        """Список заблокованих IP"""
        query = """
            SELECT ip_address, reason, blocked_until, permanent, created_at
            FROM blocked_ips
            WHERE permanent = true OR blocked_until > CURRENT_TIMESTAMP
            ORDER BY created_at DESC
        """
        blocked = self.db.execute_query(query, fetch=True)

        if not blocked:
            print("No blocked IPs.")
            return

        table_data = []
        for entry in blocked:
            duration = "Permanent" if entry['permanent'] else
entry['blocked_until'].strftime('%Y-%m-%d %H:%M')
            table_data.append([
                entry['ip_address'],
                entry['reason'],
                duration,
                entry['created_at'].strftime('%Y-%m-%d %H:%M')
            ])

        headers = ["IP Address", "Reason", "Until", "Blocked At"]
        print("\n" + tabulate(table_data, headers=headers, tablefmt="grid"))
```

```
def print_usage():
    """Виведення довідки"""
    print("""
MFA Admin Tool

Usage: mfa_admin.py <command> [arguments]

Commands:
  create-user <username>          Create new user
  setup-totp <username>           Setup TOTP for user
  telegram-code <username>        Generate Telegram link code
  list-users                      List all users
  user-info <username>            Show user information
  disable-user <username>         Disable user account
  enable-user <username>          Enable user account
  change-password <username>      Change user password
  logs [username] [limit]         View authentication logs
  block-ip <ip> <reason>          Block IP address
  list-blocked                    List blocked IPs

Examples:
  mfa_admin.py create-user john
  mfa_admin.py setup-totp john
  mfa_admin.py telegram-code john
  mfa_admin.py list-users
  mfa_admin.py logs john 50
""")
```

```
def main():
    if len(sys.argv) < 2:
        print_usage()
        sys.exit(1)

    command = sys.argv[1]
    admin = MFAAdmin()

    try:
        if command == 'create-user':
            if len(sys.argv) < 3:
                print("Error: username required")
                sys.exit(1)
            admin.create_user(sys.argv[2])

        elif command == 'setup-totp':
            if len(sys.argv) < 3:
                print("Error: username required")
                sys.exit(1)
            admin.setup_totp(sys.argv[2])

        elif command == 'telegram-code':
            if len(sys.argv) < 3:
                print("Error: username required")
                sys.exit(1)
            admin.generate_telegram_link_code(sys.argv[2])
```

```
elif command == 'list-users':
    admin.list_users()

elif command == 'user-info':
    if len(sys.argv) < 3:
        print("Error: username required")
        sys.exit(1)
    admin.user_info(sys.argv[2])

elif command == 'disable-user':
    if len(sys.argv) < 3:
        print("Error: username required")
        sys.exit(1)
    admin.disable_user(sys.argv[2])

elif command == 'enable-user':
    if len(sys.argv) < 3:
        print("Error: username required")
        sys.exit(1)
    admin.enable_user(sys.argv[2])

elif command == 'change-password':
    if len(sys.argv) < 3:
        print("Error: username required")
        sys.exit(1)
    username = sys.argv[2]
    password = getpass.getpass(f"New password for {username}: ")
    admin.password_mgr.change_password(username, password)
```

```
            print(f"✓ Password changed for {username}")

        elif command == 'logs':
            username = sys.argv[2] if len(sys.argv) > 2 else None
            limit = int(sys.argv[3]) if len(sys.argv) > 3 else 20
            admin.view_logs(username, limit)

        elif command == 'block-ip':
            if len(sys.argv) < 4:
                print("Error: IP and reason required")
                sys.exit(1)
            admin.block_ip(sys.argv[2], sys.argv[3])

        elif command == 'list-blocked':
            admin.list_blocked_ips()

        else:
            print(f"Unknown command: {command}")
            print_usage()
            sys.exit(1)

    except KeyboardInterrupt:
        print("\nCancelled.")
        sys.exit(0)
    except Exception as e:
        print(f"Error: {e}")
        sys.exit(1)
```

```
if __name__ == '__main__':
    main()
```

Збережіть та зробіть виконуваним:

bash

```
chmod +x /opt/mfa-system/scripts/mfa_admin.py
```

Встановлення tabulate для гарних таблиць

bash

```
pip3 install tabulate --break-system-packages
```

Основні адміністративні команди

Створення користувача

bash

```
cd /opt/mfa-system/scripts
sudo ./mfa_admin.py create-user john_doe
```

Налаштування TOTP для користувача

bash

```
sudo ./mfa_admin.py setup-totp john_doe
```

Ця команда:

- Згенерує секретний ключ
- Виведе URI для QR-коду
- Згенерує 10 резервних кодів
- Збереже всю інформацію у `/opt/mfa-system/backups/`

ВАЖЛИВО: Роздрукуйте або безпечно передайте користувачу секрет та резервні коди!

Генерація коду для прив'язки Telegram

bash

```
sudo ./mfa_admin.py telegram-code john_doe
```

Видасть 8-символьний код, який діє 15 хвилин.

Передайте код користувачу для команди `/link КОД` у боті.

Перегляд списку користувачів

bash

sudo ./mfa_admin.py list-users

Детальна інформація про користувача

bash

sudo ./mfa_admin.py user-info john_doe

Відключення користувача

bash

sudo ./mfa_admin.py disable-user john_doe

Увімкнення користувача

bash

sudo ./mfa_admin.py enable-user john_doe

Зміна паролю

bash

sudo ./mfa_admin.py change-password john_doe

Перегляд журналу подій

bash

Останні 20 подій всіх користувачів

sudo ./mfa_admin.py logs

Останні 50 подій конкретного користувача

sudo ./mfa_admin.py logs john_doe 50

Блокування IP адреси

bash

sudo ./mfa_admin.py block-ip 192.168.1.100 "Brute-force attack detected"

Список заблокованих IP

bash

sudo ./mfa_admin.py list-blocked

Корисні SQL запити для ручного управління

Розблокування користувача

bash

```
sudo -u postgres psql mfa_auth << EOF
UPDATE users SET is_active = true WHERE username = 'john_doe';
EOF
```

Видалення користувача

bash

```
sudo -u postgres psql mfa_auth << EOF
DELETE FROM users WHERE username = 'john_doe';
EOF
```

Скидання TOTP для користувача

bash

```
sudo -u postgres psql mfa_auth << EOF
UPDATE users SET totp_secret_encrypted = NULL WHERE username = 'john_doe';
EOF
```

Відв'язка Telegram

bash

```
sudo -u postgres psql mfa_auth << EOF
UPDATE users SET telegram_chat_id = NULL WHERE username = 'john_doe';
EOF
```

Очищення старих логів (старше 90 днів)

bash

```
sudo -u postgres psql mfa_auth << EOF
DELETE FROM auth_events WHERE created_at < CURRENT_TIMESTAMP -
INTERVAL '90 days';
EOF
```

Розблокування всіх IP

bash

```
sudo -u postgres psql mfa_auth << EOF
DELETE FROM blocked_ips;
```

EOF

Створення псевдоніму для зручності

bash

Додавання у ~/.bashrc

echo "alias mfa-admin='cd /opt/mfa-system/scripts && sudo ./mfa_admin.py'" >> ~/.bashrc

source ~/.bashrc

Тепер можна використовувати просто:

mfa-admin list-users

mfa-admin user-info john

Автоматизація резервного копіювання

Створення скрипту бекапу:

bash

sudo nano /opt/mfa-system/scripts/backup.sh

bash

#!/bin/bash

BACKUP_DIR="/opt/mfa-system/backups"

DATE=$(date +%Y%m%d_%H%M%S)

BACKUP_FILE="$BACKUP_DIR/mfa_backup_$DATE.sql"

Створення бекапу БД

sudo -u postgres pg_dump mfa_auth > "$BACKUP_FILE"

Стиснення

gzip "$BACKUP_FILE"

Видалення старих бекапів (старше 30 днів)

find "$BACKUP_DIR" -name "mfa_backup_*.sql.gz" -mtime +30 -delete

echo "Backup completed: ${BACKUP_FILE}.gz"

bash

chmod +x /opt/mfa-system/scripts/backup.sh

Додавання у cron (щоденно о 2:00)

```
(crontab -l 2>/dev/null; echo "0 2 * * * /opt/mfa-system/scripts/backup.sh") | crontab –
```

Додаток Б

Telegram Bot - Повний код

Створення Telegram бота для системи MFA

bash

nano /opt/mfa-system/scripts/telegram_bot.py

Вставте наступний код:

python

```
#!/usr/bin/env python3
import asyncio
import configparser
import uuid
import logging
from datetime import datetime, timedelta
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup
from telegram.ext import (
    ApplicationBuilder,
    CommandHandler,
    CallbackQueryHandler,
    ContextTypes
)
from database import Database

# Налаштування логування
logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    level=logging.INFO
)
logger = logging.getLogger(__name__)
```

```
class AuthTelegramBot:
    def __init__(self, config_path='/opt/mfa-system/config/config.ini'):
        # Читання конфігурації
        config = configparser.ConfigParser()
        config.read(config_path)

        self.bot_token = config['telegram']['bot_token']
        self.admin_chat_ids = [
            int(x.strip())
            for x in config['telegram']['admin_chat_ids'].split(',')
        ]

        self.db = Database(config_path)
        self.application = ApplicationBuilder().token(self.bot_token).build()

        # Словник для зберігання очікуваних підтверджень
        self.pending_auth = {}

        # Реєстрація обробників
        self._register_handlers()

    def _register_handlers(self):
        """Реєстрація обробників команд та callback"""
        self.application.add_handler(CommandHandler("start", self.cmd_start))
        self.application.add_handler(CommandHandler("help", self.cmd_help))
        self.application.add_handler(CommandHandler("link", self.cmd_link))
        self.application.add_handler(CommandHandler("status", self.cmd_status))
```

```
    self.application.add_handler(CommandHandler("unlink", self.cmd_unlink))
    self.application.add_handler(CallbackQueryHandler(self.handle_callback))

async def cmd_start(self, update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Обробка команди /start"""
    welcome_text = (
        "🔐 <b>Система багатофакторної автентифікації</b>\n\n"
        "Вітаю! Я бот для автентифікації доступу до серверів компанії.\n\n"
        "<b>Доступні команди:</b>\n"
        "/link &lt;код&gt; - прив'язати обліковий запис\n"
        "/status - перевірити статус прив'язки\n"
        "/unlink - відв'язати обліковий запис\n"
        "/help - довідка\n\n"
        "Для початку роботи отримайте код прив'язки у адміністратора системи."
    )
    await update.message.reply_text(welcome_text, parse_mode='HTML')

async def cmd_help(self, update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Обробка команди /help"""
    help_text = (
        "<b>Довідка по командам:</b>\n\n"
        "<b>/start</b> - початок роботи з ботом\n\n"
        "<b>/link &lt;код&gt;</b> - прив'язка облікового запису\n"
        "Приклад: <code>/link A1B2C3D4</code>\n"
        "Код надає адміністратор системи.\n\n"
        "<b>/status</b> - перевірка статусу прив'язки\n"
        "Показує чи прив'язаний ваш Telegram до облікового запису.\n\n"
        "<b>/unlink</b> - відв'язка облікового запису\n"
```

```
        "Використовуйте якщо змінили обліковий запис або телефон.\n\n"
        "<b>Як працює автентифікація:</b>\n"
        "1. Ви вводите логін та пароль на сервері\n"
        "2. Вводите TOTP код з аутентифікатора\n"
        "3. Отримуєте запит у Telegram на підтвердження\n"
        "4. Натискаєте 'Підтвердити' для завершення входу\n\n"
        "❓ Питання? Зверніться до адміністратора системи."
    )
    await update.message.reply_text(help_text, parse_mode='HTML')

async def cmd_link(self, update: Update, context: ContextTypes.DEFAULT_TYPE):
    """Обробка команди /link <код>"""
    chat_id = update.effective_chat.id
    username_tg = update.effective_user.username

    # Перевірка аргументів
    if not context.args or len(context.args) == 0:
        await update.message.reply_text(
            "❌ <b>Помилка:</b> Не вказано код\n\n"
            "Використання: <code>/link КОД</code>\n"
            "Приклад: <code>/link A1B2C3D4</code>\n\n"
            "Отримайте код у адміністратора системи.",
            parse_mode='HTML'
        )
        return

    code = context.args[0].upper().strip()
```

```
# Валідація коду в БД
query = """
    SELECT username, expires_at FROM telegram_link_codes
    WHERE code = %s AND used = false
"""
result = self.db.execute_query(query, (code,), fetch=True)

if not result:
    await update.message.reply_text(
        "❌ <b>Невірний або застарілий код</b>\n\n"
        "Можливі причини:\n"
        "• Код введено невірно\n"
        "• Код вже використано\n"
        "• Термін дії коду минув (15 хвилин)\n\n"
        "Отримайте новий код у адміністратора.",
        parse_mode='HTML'
    )
    return

username = result[0]['username']
expires_at = result[0]['expires_at']

# Перевірка терміну дії
if datetime.now() > expires_at:
    await update.message.reply_text(
        "❌ <b>Код застарілий</b>\n\n"
        "Термін дії коду минув.\n"
        "Отримайте новий код у адміністратора.",
```

```
            parse_mode='HTML'
        )
        return

    # Перевірка чи цей Telegram вже прив'язаний
    check_query = "SELECT username FROM users WHERE telegram_chat_id = %s"
    existing = self.db.execute_query(check_query, (chat_id,), fetch=True)

    if existing:
        existing_user = existing[0]['username']
        await update.message.reply_text(
            f"⚠ <b>Telegram вже прив'язаний</b>\n\n"
            f"Цей Telegram акаунт вже прив'язаний до користувача: "
            f"<code>{existing_user}</code>\n\n"
            f"Спочатку відв'яжіть його командою /unlink",
            parse_mode='HTML'
        )
        return

    # Прив'язка облікового запису
    self.db.update_telegram_chat_id(username, chat_id)

    # Позначення коду як використаного
    mark_used = "UPDATE telegram_link_codes SET used = true WHERE code =
%s"
    self.db.execute_query(mark_used, (code,))

    # Логування
    self.db.log_event(
```

```
            username,
            'telegram_linked',
            details={'chat_id': chat_id, 'telegram_username': username_tg}
        )

        # Підтвердження
        await update.message.reply_text(
            f"✅ <b>Telegram успішно прив'язано!</b>\n\n"
            f"👤 Користувач: <code>{username}</code>\n"
            f"💬 Telegram: @{username_tg if username_tg else 'no_username'}\n"
            f"🆔 Chat ID: <code>{chat_id}</code>\n\n"
            f"Тепер ви отримуватимете запити на підтвердження входу через цей чат.\n\n"
            f"Для перевірки статусу використовуйте /status",
            parse_mode='HTML'
        )

        # Сповіщення адміністраторів
        await self.notify_admins(
            f"✅ <b>Новий Telegram прив'язаний</b>\n\n"
            f"Користувач: <code>{username}</code>\n"
            f"Telegram: @{username_tg if username_tg else 'no_username'}\n"
            f"Chat ID: <code>{chat_id}</code>\n"
            f"Час: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
        )

    async def cmd_status(self, update: Update, context: ContextTypes.DEFAULT_TYPE):
```

```
        """Обробка команди /status"""
        chat_id = update.effective_chat.id

        # Пошук користувача з цим chat_id
        query = "SELECT username, is_active FROM users WHERE telegram_chat_id =
%s"
        result = self.db.execute_query(query, (chat_id,), fetch=True)

        if not result:
            await update.message.reply_text(
                "❌ <b>Telegram не прив'язаний</b>\n\n"
                "Ваш Telegram акаунт не прив'язаний до жодного користувача.\n\n"
                "Для прив'язки використовуйте:\n"
                "<code>/link КОД</code>\n\n"
                "Код можна отримати у адміністратора.",
                parse_mode='HTML'
            )
            return

        username = result[0]['username']
        is_active = result[0]['is_active']

        status_icon = "✅" if is_active else "🔴"
        status_text = "Активний" if is_active else "Заблокований"

        await update.message.reply_text(
            f"{status_icon} <b>Статус прив'язки</b>\n\n"
            f"👤 Користувач: <code>{username}</code>\n"
```

```
            f"📱 Статус: {status_text}\n"
            f"🆔 Chat ID: <code>{chat_id}</code>\n\n"
            f"Прив'язка активна. Ви отримуватимете запити на підтвердження входу.",
            parse_mode='HTML'
        )

    async def cmd_unlink(self, update: Update, context:
ContextTypes.DEFAULT_TYPE):
        """Обробка команди /unlink"""
        chat_id = update.effective_chat.id

        # Пошук користувача
        query = "SELECT username FROM users WHERE telegram_chat_id = %s"
        result = self.db.execute_query(query, (chat_id,), fetch=True)

        if not result:
            await update.message.reply_text(
                "ℹ️ Ваш Telegram акаунт не прив'язаний до жодного користувача.",
                parse_mode='HTML'
            )
            return

        username = result[0]['username']

        # Відв'язка
        unlink_query = """
            UPDATE users
            SET telegram_chat_id = NULL
```

```
        WHERE telegram_chat_id = %s
    """
    self.db.execute_query(unlink_query, (chat_id,))

    # Логування
    self.db.log_event(username, 'telegram_unlinked', details={'chat_id': chat_id})

    await update.message.reply_text(
        f"✅ <b>Telegram відв'язаний</b>\n\n"
        f"Користувач: <code>{username}</code>\n\n"
        f"Для повторної прив'язки отримайте новий код у адміністратора.",
        parse_mode='HTML'
    )

    # Сповіщення адміністраторів
    await self.notify_admins(
        f"⚠ <b>Telegram відв'язаний</b>\n\n"
        f"Користувач: <code>{username}</code>\n"
        f"Chat ID: <code>{chat_id}</code>"
    )

async def send_auth_request(self, username, ip_address, location="Unknown"):
    """Відправка запиту на автентифікацію"""
    # Отримання chat_id
    user = self.db.get_user(username)
    if not user or not user['telegram_chat_id']:
        return None
```

```
chat_id = user['telegram_chat_id']

# Генерація токену сесії
session_token = str(uuid.uuid4())

# Збереження сесії в БД
query = """
    INSERT INTO auth_sessions (session_token, username, ip_address, expires_at)
    VALUES (%s, %s, %s, %s)
"""
expiry = datetime.now() + timedelta(minutes=2)
self.db.execute_query(query, (session_token, username, ip_address, expiry))

# Додавання до pending
self.pending_auth[session_token] = {
    'username': username,
    'approved': None,
    'expires_at': expiry
}

# Форматування повідомлення
message = (
    "🔐 <b>Запит на вхід до системи</b>\n\n"
    f"👤 Користувач: <code>{username}</code>\n"
    f"🌐 IP-адреса: <code>{ip_address}</code>\n"
    f"📍 Локація: {location}\n"
    f"🕐 Час: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n\n"
    "❓ Це ви намагаєтесь увійти?"
```

```
)

# Кнопки
keyboard = [
    [
        InlineKeyboardButton(
            "✅ Підтвердити",
            callback_data=f"auth:approve:{session_token}"
        ),
        InlineKeyboardButton(
            "❌ Відхилити",
            callback_data=f"auth:deny:{session_token}"
        )
    ]
]
reply_markup = InlineKeyboardMarkup(keyboard)

# Відправка повідомлення
try:
    await self.application.bot.send_message(
        chat_id=chat_id,
        text=message,
        parse_mode='HTML',
        reply_markup=reply_markup
    )

    logger.info(f"Auth request sent to {username} (chat_id: {chat_id})")
    return session_token
```

```
        except Exception as e:
            logger.error(f"Failed to send auth request: {e}")
            return None

    async def handle_callback(self, update: Update, context:
ContextTypes.DEFAULT_TYPE):
        """Обробка натискань на кнопки"""
        query = update.callback_query
        await query.answer()

        # Парсинг callback_data
        try:
            parts = query.data.split(':')
            action_type = parts[0]
            action = parts[1]
            session_token = parts[2]
        except:
            await query.edit_message_text("❌ Невірний формат запиту")
            return

        if action_type == 'auth':
            await self._handle_auth_callback(query, action, session_token)
        elif action_type == 'admin':
            await self._handle_admin_callback(query, action, parts[2:])

    async def _handle_auth_callback(self, query, action, session_token):
        """Обробка відповіді на запит автентифікації"""
        # Перевірка чи існує сесія
```

```
if session_token not in self.pending_auth:
    # Спроба з БД
    db_query = """
        SELECT username, status, expires_at
        FROM auth_sessions
        WHERE session_token = %s
    """
    result = self.db.execute_query(db_query, (session_token,), fetch=True)

    if not result or result[0]['status'] != 'pending':
        await query.edit_message_text(
            "❌ <b>Запит застарів або недійсний</b>\n\n"
            "Якщо це була ваша спроба входу, спробуйте ще раз.",
            parse_mode='HTML'
        )
        return

    # Перевірка терміну дії
    if datetime.now() > result[0]['expires_at']:
        await query.edit_message_text(
            "□ <b>Час очікування минув</b>\n\n"
            "Спробуйте увійти ще раз.",
            parse_mode='HTML'
        )
        return

    username = result[0]['username']
else:
```

```
    username = self.pending_auth[session_token]['username']

if action == 'approve':
    # Оновлення статусу в БД
    update_query = """
        UPDATE auth_sessions
        SET status = 'approved'
        WHERE session_token = %s
    """
    self.db.execute_query(update_query, (session_token,))

    # Оновлення в пам'яті
    if session_token in self.pending_auth:
        self.pending_auth[session_token]['approved'] = True

    await query.edit_message_text(
        f"✅ <b>Вхід підтверджено</b>\n\n"
        f"Користувач: <code>{username}</code>\n"
        f"Час: {datetime.now().strftime('%H:%M:%S')}\n\n"
        f"Доступ надано.",
        parse_mode='HTML'
    )

    self.db.log_event(username, 'telegram_approved', success=True)

elif action == 'deny':
    # Оновлення статусу
    update_query = """
```

```
            UPDATE auth_sessions
            SET status = 'denied'
            WHERE session_token = %s
        """
        self.db.execute_query(update_query, (session_token,))

        if session_token in self.pending_auth:
            self.pending_auth[session_token]['approved'] = False

        await query.edit_message_text(
            f"❌ <b>Вхід відхилено</b>\n\n"
            f"Користувач: <code>{username}</code>\n\n"
            f"⚠ Якщо це не ви намагалися увійти, "
            f"рекомендуємо негайно змінити пароль та зв'язатися з адміністратором.",
            parse_mode='HTML'
        )

        self.db.log_event(username, 'telegram_denied', success=False)

        # Сповіщення адміністраторів
        await self.notify_admins(
            f"⚠ <b>Відхилено запит на вхід</b>\n\n"
            f"Користувач: <code>{username}</code>\n"
            f"Час: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
        )

    async def _handle_admin_callback(self, query, action, params):
```

```
        """Обробка адміністративних дій"""
        # Перевірка чи це адміністратор
        if query.from_user.id not in self.admin_chat_ids:
            await query.answer("У вас немає прав адміністратора", show_alert=True)
            return

        # Тут буде логіка для адміністративних дій
        # (блокування користувача, блокування IP, тощо)
        pass

    async def notify_admins(self, message):
        """Відправка сповіщення всім адміністраторам"""
        for admin_id in self.admin_chat_ids:
            try:
                await self.application.bot.send_message(
                    chat_id=admin_id,
                    text=message,
                    parse_mode='HTML'
                )
            except Exception as e:
                logger.error(f"Failed to notify admin {admin_id}: {e}")

    def run(self):
        """Запуск бота"""
        logger.info("Starting Telegram bot...")
        self.application.run_polling(allowed_updates=Update.ALL_TYPES)

def main():
```

```
    bot = AuthTelegramBot()
    bot.run()

if __name__ == '__main__':
    main()
```

Збережіть файл та зробіть виконуваним:

```
bash
chmod +x /opt/mfa-system/scripts/telegram_bot.py
# Створення модуля для взаємодії з ботом з інших скриптів
bash
nano /opt/mfa-system/scripts/telegram_auth.py

python
#!/usr/bin/env python3
import asyncio
import uuid
import configparser
from datetime import datetime, timedelta
from telegram import Bot
from telegram import InlineKeyboardButton, InlineKeyboardMarkup
from database import Database

# Глобальний bot instance
_bot_instance = None
_db_instance = None

def get_bot():
    """Отримання singleton instance бота"""
```

```
    global _bot_instance
    if _bot_instance is None:
        config = configparser.ConfigParser()
        config.read('/opt/mfa-system/config/config.ini')
        token = config['telegram']['bot_token']
        _bot_instance = Bot(token=token)
    return _bot_instance

def get_db():
    """Отримання singleton instance БД"""
    global _db_instance
    if _db_instance is None:
        _db_instance = Database()
    return _db_instance

def send_auth_request(username, ip_address, location="Unknown"):
    """Синхронна обгортка для відправки запиту автентифікації"""
    return asyncio.run(_send_auth_request_async(username, ip_address, location))

async def _send_auth_request_async(username, ip_address, location):
    """Асинхронна відправка запиту"""
    db = get_db()
    bot = get_bot()

    # Отримання chat_id
    user = db.get_user(username)
    if not user or not user['telegram_chat_id']:
        return None
```

```
    chat_id = user['telegram_chat_id']
    session_token = str(uuid.uuid4())

    # Збереження сесії
    query = """
        INSERT INTO auth_sessions (session_token, username, ip_address, expires_at)
        VALUES (%s, %s, %s, %s)
    """
    expiry = datetime.now() + timedelta(minutes=2)
    db.execute_query(query, (session_token, username, ip_address, expiry))

    # Повідомлення
    message = (
        "🔐 <b>Запит на вхід до системи</b>\n\n"
        f"👤 Користувач: <code>{username}</code>\n"
        f"🌐 IP-адреса: <code>{ip_address}</code>\n"
        f"📍 Локація: {location}\n"
        f"🕐 Час: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n\n"
        "❓ Це ви намагаєтесь увійти?"
    )

    keyboard = [
        [
            InlineKeyboardButton("✅ Підтвердити",
callback_data=f"auth:approve:{session_token}"),
            InlineKeyboardButton("❌ Відхилити",
callback_data=f"auth:deny:{session_token}")
```

```
        ]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)

    try:
        await bot.send_message(
            chat_id=chat_id,
            text=message,
            parse_mode='HTML',
            reply_markup=reply_markup
        )
        return session_token
    except Exception as e:
        print(f"Failed to send Telegram request: {e}")
        return None

def wait_for_confirmation(session_token, timeout=120):
    """Очікування підтвердження від користувача"""
    import time
    db = get_db()

    start_time = time.time()

    while time.time() - start_time < timeout:
        # Перевірка статусу сесії
        query = "SELECT status FROM auth_sessions WHERE session_token = %s"
        result = db.execute_query(query, (session_token,), fetch=True)
```

```
        if not result:
            return False

        status = result[0]['status']

        if status == 'approved':
            return True
        elif status == 'denied':
            return False

        # Очікування 1 секунду
        time.sleep(1)

    # Timeout
    return False

if __name__ == '__main__':
    # Тест
    print("Testing Telegram auth module...")
    session = send_auth_request("testuser", "192.168.1.1")
    if session:
        print(f"Session token: {session}")
        print("Waiting for confirmation...")
        approved = wait_for_confirmation(session)
        print(f"Result: {'Approved' if approved else 'Denied/Timeout'}")
bash
chmod +x /opt/mfa-system/scripts/telegram_auth.py
# Створення systemd служби для бота
```

bash

```
sudo nano /etc/systemd/system/mfa-telegram-bot.service
```

ini

```
[Unit]
Description=MFA Telegram Bot
After=network.target postgresql.service redis.service
[Service]
Type=simple
User=root
WorkingDirectory=/opt/mfa-system/scripts
ExecStart=/usr/bin/python3 /opt/mfa-system/scripts/telegram_bot.py
Restart=always
RestartSec=10
[Install]
WantedBy=multi-user.target
```

Запуск бота

bash

```
# Перезавантаження systemd
sudo systemctl daemon-reload
# Запуск бота
sudo systemctl start mfa-telegram-bot
# Автозапуск при старті системи
sudo systemctl enable mfa-telegram-bot
# Перевірка статусу
sudo systemctl status mfa-telegram-bot
# Перегляд логів
sudo journalctl -u mfa-telegram-bot -f
```

Тестування бота

1. Знайдіть вашого бота у Telegram (за username який ви вказали)
2. Відправте команду `/start`
3. Бот має відповісти привітальним повідомленням
4. Спробуйте `/help` для перегляду довідки