МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Міжнародний гуманітарний університет
Кафедра комп’ютерної інженерії та інноваційних технологій

Йона Л.Г., Педяш В.В., Мазур Г.Д.

БЕЗПЕКА ПРОГРАМ ТА ДАНИХ

Конспект лекцій для здобувачів вищої освіти зі спеціальностей:
A4.09 Середня освіта (Інформатика),
F2 Інженерія програмного забезпечення,
F3 Комп'ютерні науки,
F7 Комп'ютерна інженерія,
F5 Кібербезпека та захист інформації,
G5 Електроніка, електронні комунікації, приладобудування та радіотехніка

Одеса 2025

Йона Л.Г., Педяш В.В., Мазур Г.Д.

Безпека програм та даних: конспект лекцій [Електронне видання] / Йона Л.Г., Педяш В.В., Мазур Г.Д. — Кафедра комп’ютерної інженерії та інноваційних технологій Міжнародного гуманітарного університету. — Одеса, 2025. — 192 с.

Конспект лекцій призначено для проведення навчальних занять з дисципліни «Безпека програм та даних» і спрямовано на формування у здобувачів вищої освіти системних знань та практичних навичок забезпечення захищеності програмного забезпечення, операційних систем, мережевих сервісів та даних у сучасному інформаційному середовищі.

Видання охоплює тематичні модулів, які послідовно розкривають ключові аспекти кібербезпеки: від архітектурних основ та моделі CIA до захисту розподілених хмарних середовищ. Навчальний матеріал включає питання налаштування ізольованого віртуалізованого середовища, аудиту безпеки операційних систем, реалізації моделей контролю доступу (DAC, MAC, RBAC), захисту від атак грубої сили та автоматизованих вторгнень, конфігурування мережевих фаєрволів, управління вразливостями та оновленнями, застосування криптографічних засобів захисту даних, управління ключами та інфраструктурою відкритих ключів (PKI), організації захищених VPN-з'єднань (IPsec, OpenVPN, WireGuard), використання безпечних протоколів передачі даних (SSH, TLS), а також впровадження принципів Zero Trust у контейнерних та хмарних середовищах.

Конспект лекцій може використовуватися під час проведення лекційних, практичних і лабораторних занять, а також при виконанні кваліфікаційних робіт та наукових досліджень. Видання адресоване студентам спеціальностей галузі інформаційних технологій, викладачам, аспірантам та фахівцям, які вивчають або викладають дисципліни, пов'язані з інформаційною та кібербезпекою.

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі безпека програмного забезпечення та даних виступає критичним фундаментом функціонування будь-якої інформаційної системи. Стрімка еволюція кіберзагроз вимагає від фахівців галузі інформаційних технологій не лише теоретичних знань, а й сформованих практичних навичок протидії реальним атакам. Дисципліна «Безпека програм та даних» покликана забезпечити здобувачів вищої освіти глибоким розумінням архітектурних принципів захисту, механізмів забезпечення цілісності та конфіденційності інформації, а також інструментарію для моніторингу та реагування на інциденти.

Цей конспект лекцій охоплює 14 тематичних модулів, що комплексно розкривають ключові аспекти кібербезпеки: від архітектурних основ та моделі CIA до захисту розподілених хмарних середовищ. Навчальний матеріал послідовно веде студента через налаштування безпечного віртуалізованого середовища, аудит операційних систем, управління доступом (DAC, MAC, RBAC), конфігурацію мережевих екранів, криптографічний захист даних, організацію VPN-з'єднань та впровадження архітектури Zero Trust.

Структура викладу матеріалу забезпечує тісний зв'язок між теорією та практикою. Кожен тематичний розділ містить «Ключові положення», де систематизовано фундаментальні концепції, та практично орієнтовані рекомендації щодо використання інструментів безпеки. Такий підхід дозволяє здобувачам не лише засвоїти принципи роботи захисних механізмів, а й навчитися застосовувати їх у реальних сценаріях.

Практична складова курсу реалізується через роботу з сучасним програмним забезпеченням у середовищі Linux (Debian 12, Linux Mint). Студенти опановують інструменти аудиту (Lynis, OpenSCAP), захисту від вторгнень (Fail2Ban), мережевої фільтрації (iptables, nftables), криптографії (GnuPG, OpenSSL), аналізу трафіку (Wireshark, tcpdump) та управління контейнерами (Docker, Kubernetes). Особлива увага приділяється автоматизації процесів безпеки та використанню інфраструктури як коду (IaC). Інтеграція теоретичних знань з практичними навичками сприяє формуванню професійних компетентностей, необхідних для побудови захищених інформаційних систем. Випускники зможуть самостійно виявляти вразливості, проєктувати системи захисту з урахуванням моделі загроз та оцінювати ефективність впроваджених заходів безпеки.

Отримані знання стануть надійним підґрунтям для вивчення спеціалізованих дисциплін з мережевої безпеки, криптографії, цифрової криміналістики та управління інцидентами. Крім того, матеріал посібника може бути використаний при виконанні кваліфікаційних робіт та у подальшій професійній діяльності фахівців з кібербезпеки та інженерії програмного забезпечення.

Тема 1. Архітектурні основи безпеки програм та даних

1.1. Поняття безпеки програмного забезпечення та даних. Модель CIA

Безпека програмного забезпечення та даних є фундаментальним поняттям сучасної кібербезпеки, що охоплює сукупність технічних, організаційних і процедурних заходів, спрямованих на захист програм і даних від несанкціонованого доступу, модифікації, знищення або розкриття. У цифровому суспільстві програмні системи стали невід'ємною частиною критичної інфраструктури держав, бізнес-процесів, охорони здоров'я, фінансових операцій та повсякденного спілкування. Відтак будь-яке порушення їхньої безпеки може спричинити значні матеріальні збитки, репутаційні втрати або навіть загрозу людському життю. Саме тому розуміння основ безпеки програм і даних є необхідною компетентністю сучасного фахівця з кібербезпеки.

Предмет безпеки програмного забезпечення охоплює щонайменше два взаємопов'язані аспекти. По-перше, це безпека самого програмного коду — забезпечення того, щоб програма функціонувала виключно відповідно до задуму розробника і не містила вразливостей, що можуть бути використані зловмисниками. По-друге, це захист даних, якими програма оперує, — забезпечення їхньої конфіденційності, цілісності та доступності протягом усього їхнього життєвого циклу. Обидва аспекти нерозривно пов'язані між собою: навіть бездоганно написана програма може стати інструментом витоку даних, якщо вона не реалізує механізмів їхнього захисту, і навіть надійно зашифровані дані можуть бути скомпрометовані через вразливість у програмі, що здійснює їхню обробку.

Важливо розуміти, що безпека програм і даних не є статичним станом, який досягається одноразовим впровадженням засобів захисту. Це безперервний процес, що включає аналіз загроз, оцінку ризиків, впровадження захисних заходів, моніторинг їхньої ефективності та постійне вдосконалення відповідно до змін у технологічному середовищі й ландшафті загроз. Принципово нові вразливості виявляються щодня, а зловмисники постійно вдосконалюють свої інструменти та методи. Тому фахівець із безпеки програм і даних мусить не лише знати чинні технічні рішення, але й розуміти фундаментальні принципи, на яких ці рішення засновані, — адже саме принципи залишаються незмінними, коли конкретні технології та стандарти змінюються.

Ключовою теоретичною моделлю, що лежить в основі більшості підходів до забезпечення інформаційної безпеки, є **модель CIA (Confidentiality, Integrity, Availability — конфіденційність, цілісність, доступність).** Ця тріада сформульована ще в 1970-х роках і досі залишається актуальною, слугуючи орієнтиром при проєктуванні систем захисту, проведенні аудитів безпеки та оцінці наслідків інцидентів. Кожна з трьох властивостей описує окремий вимір безпеки і потребує власних механізмів захисту. Водночас між ними існують

певні компроміси: надмірне посилення конфіденційності може ускладнити доступ до даних, знижуючи доступність; підвищення доступності через масштабну реплікацію може збільшити ризик порушення конфіденційності.

Конфіденційність означає, що інформація є доступною лише тим суб'єктам — особам, процесам або системам, — яким надано відповідні права. Механізмами забезпечення конфіденційності є шифрування, контроль доступу, аутентифікація та використання захищених каналів зв'язку. **Цілісність** полягає у тому, що дані не змінюються або не знищуються несанкціонованим чином ні під час зберігання, ні під час передачі. Для захисту цілісності застосовуються криптографічні хеш-функції, цифрові підписи, механізми контролю версій та журналювання. **Доступність** — це гарантія того, що авторизовані суб'єкти можуть отримати доступ до необхідних ресурсів у потрібний момент; вона забезпечується через резервування, балансування навантаження, захист від атак типу «відмова в обслуговуванні» та якісне планування відновлення після аварій.

На рисунку 1.1 зображена діаграма моделі CIA у вигляді рівностороннього трикутника з підписаними вершинами: «Конфіденційність» (Confidentiality), «Цілісність» (Integrity) та «Доступність» (Availability).

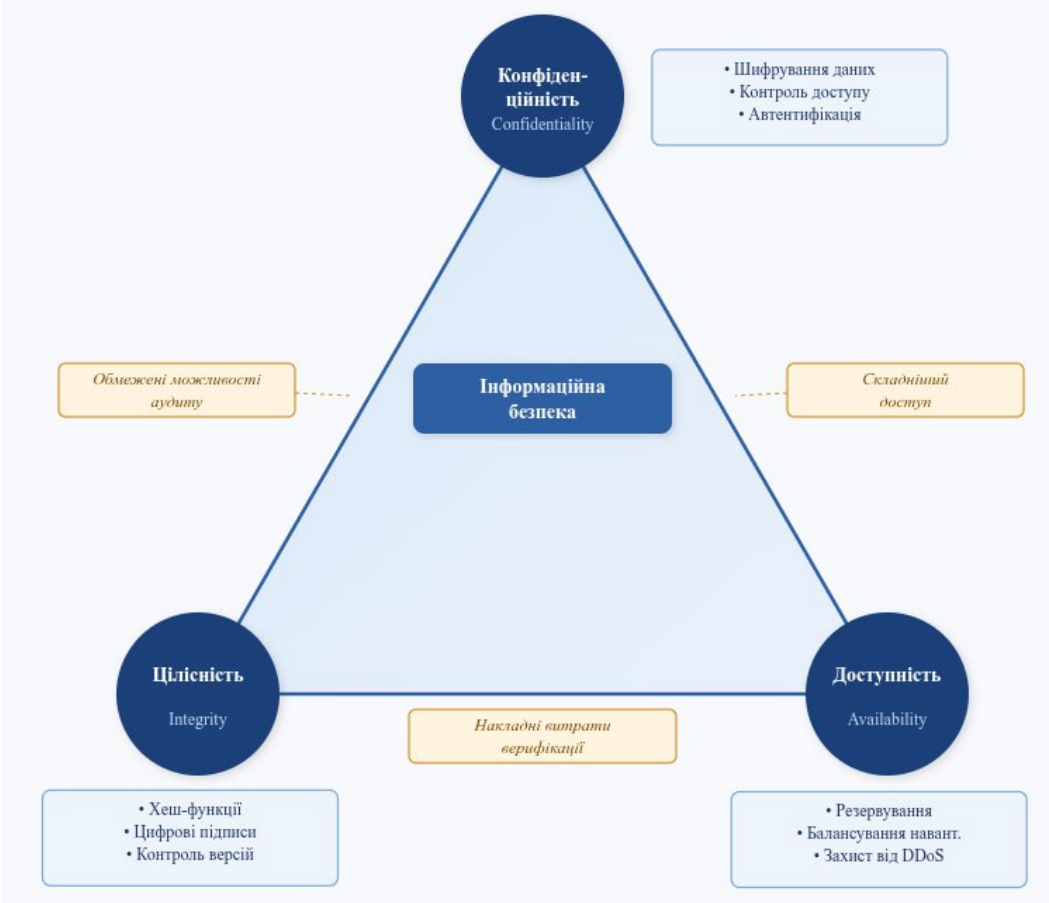

Рисунок 1.1 – Трикутник моделі CIA: властивості інформаційної безпеки та їхні компроміси

Модель CIA, незважаючи на свою простоту, є потужним інструментом аналізу та проєктування систем захисту. Нижче наведено повну характеристику

всіх трьох властивостей із зазначенням механізмів їхнього забезпечення та типових прикладів порушення (таблиця 1.1).

Таблиця 1.1 – Властивості моделі CIA: зміст, механізми забезпечення та приклади порушень

Властивість	**Зміст**	**Механізми забезпечення**	**Приклад порушення**
Конфіденційність (Confidentiality)	Доступ до інформації лише авторизованим суб'єктам	Шифрування, контроль доступу, автентифікація, захищені канали зв'язку	Витік персональних даних клієнтів через скомпрометовану БД
Цілісність (Integrity)	Відсутність несанкціонованого змінення або знищення інформації	Хеш-функції, цифрові підписи, контроль версій, журналювання	Підміна виконуваного файлу зловмисником для зміни поведінки програми
Доступність (Availability)	Готовність ресурсів до використання авторизованими суб'єктами у потрібний момент	Резервування, балансування навантаження, захист від DDoS, плани відновлення	DDoS-атака, що виводить з ладу вебсервер організації

Суттєвим практичним аспектом застосування моделі CIA є свідоме управління компромісами між її властивостями. При проєктуванні будь-якої системи захисту фахівець неминуче стикається з ситуацією, коли посилення однієї властивості послаблює іншу. Розуміння цих компромісів є необхідним для прийняття обґрунтованих архітектурних рішень. У таблиці 1.2 наведено типові ситуації таких компромісів.

Таблиця 1.2 – Типові компроміси між властивостями моделі CIA

Ситуація компромісу	Пояснення та практичний наслідок
Конфіденційність vs. Доступність	Надмірне шифрування та складні процедури автентифікації уповільнюють доступ до даних. Надмірно жорстка автентифікація збільшує час відгуку та погіршує зручність використання системи.
Доступність vs. Цілісність	Реплікація даних для підвищення доступності ускладнює забезпечення їхньої цілісності: узгодженість копій потребує додаткових механізмів синхронізації та верифікації.
Цілісність vs. Продуктивність	Постійна перевірка хеш-сум і цифрових підписів потребує обчислювальних ресурсів і може знижувати продуктивність системи, особливо при

	обробці великих обсягів даних у реальному часі.

При оцінці будь-якого потенційного ризику або при проєктуванні нової системи захисту фахівець повинен систематично ставити питання: як пропонований захід впливає на кожну з трьох властивостей? Чи не погіршує він одну з них, намагаючись посилити іншу? Наскільки цей компроміс є прийнятним з огляду на бізнес-вимоги та рівень загроз? Саме це системне мислення, засноване на моделі CIA, є основою компетентного підходу до забезпечення інформаційної безпеки і відрізняє фахівця від людини, яка просто виконує набір технічних процедур.

1.2. Поняття загроз, вразливостей і ризиків у безпеці програм та даних

Для ефективного забезпечення безпеки програм та даних необхідно чітко розуміти понятійний апарат, що описує природу небезпек та їхні взаємозв'язки. Три ключові поняття — загроза, вразливість і ризик — утворюють концептуальну основу будь-якого аналізу безпеки. Вони відрізняються за своєю природою та роллю, однак нерозривно пов'язані між собою, і розуміння цього зв'язку є критично важливим для правильної постановки задачі захисту, вибору адекватних заходів і пріоритизації інвестицій у безпеку.

Загроза (threat) — це будь-який чинник, дія або подія, що потенційно може завдати шкоди інформаційній системі або даним. Загрози поділяються на зовнішні й внутрішні, навмисні й ненавмисні. До зовнішніх навмисних загроз відносяться дії зловмисників, кіберзлочинних угруповань, конкурентів або спецслужб іноземних держав; внутрішні загрози можуть виходити від незадоволених або підкуплених співробітників, підрядників чи інших осіб із законним доступом до системи. Ненавмисні загрози включають помилки адміністраторів, некоректне налаштування програмного забезпечення або природні катастрофи. Важливо розуміти, що наявність загрози сама по собі ще не означає неминучості інциденту — для реалізації загрози необхідна наявність вразливості, яку ця загроза може використати.

У таблиці 1.3 наведена класифікація загроз за джерелом виникнення з прикладами та можливими наслідками. Ця класифікація є практично корисною при проведенні аналізу ризиків, оскільки дозволяє систематично охопити весь спектр потенційних загроз і не пропустити важливі категорії при оцінці захищеності системи.

Таблиця 1.3 – Класифікація загроз інформаційній безпеці за джерелом виникнення

Тип загрози	Джерело	Приклади	Наслідки
Зовнішня навмисна	Кіберзлочинці, APT-групи, конкуренти,	Фішинг, ransomware, атаки на ланцюг	Витік даних, фінансові збитки, порушення роботи

	спецслужби	постачання	
Внутрішня навмисна	Незадоволені або підкуплені співробітники, підрядники	Несанкціонований витік даних, саботаж систем	Пряма матеріальна шкода, репутаційні наслідки
Ненавмисна (людська помилка)	Персонал, адміністратори систем	Некоректне налаштування, випадкове видалення даних	Доступність та цілісність під загрозою
Технічна / природна	Апаратні збої, стихійні лиха	Відмова жорсткого диска, пожежа, повінь	Втрата доступності, можлива втрата даних

Вразливість (vulnerability) — це слабке місце в програмному забезпеченні, конфігурації або процедурах, яке може бути використане загрозою для завдання шкоди системі. Вразливості мають різну природу та можуть виникати на всіх рівнях програмного стеку. На рівні коду — це помилки програмування: переповнення буфера (buffer overflow), ін'єкція SQL (SQL injection), міжсайтовий скриптинг (XSS), некоректна перевірка вхідних даних, стан гонки (race condition). На рівні конфігурації — це використання паролів за замовчуванням, надмірно широкі права доступу, відкриті непотрібні мережеві порти. На архітектурному рівні — це відсутність ізоляції між критичними компонентами, надмірна централізація привілеїв. Нарешті, вразливості можуть виникати і на рівні процедур та людського чинника — через недостатню підготовку персоналу або відсутність регламентів.

Ризик (risk) є інтегральним поняттям, що поєднує ймовірність реалізації загрози через наявну вразливість і потенційну шкоду від такої реалізації. Формально ризик можна описати формулою: Ризик = Ймовірність × Вплив, де ймовірність залежить як від наявності та характеристик загрози, так і від ступеня вразливості системи, а вплив визначається цінністю активу та потенційними наслідками його компрометації. Саме ризик є головним об'єктом управління у сфері інформаційної безпеки: не можна усунути всі загрози і не завжди доцільно ліквідовувати всі вразливості — натомість метою є утримання ризику на прийнятному для організації рівні з урахуванням вартості захисних заходів.

На рисунку 1.2 зображена схема взаємозв'язку між загрозою, вразливістю та ризиком.

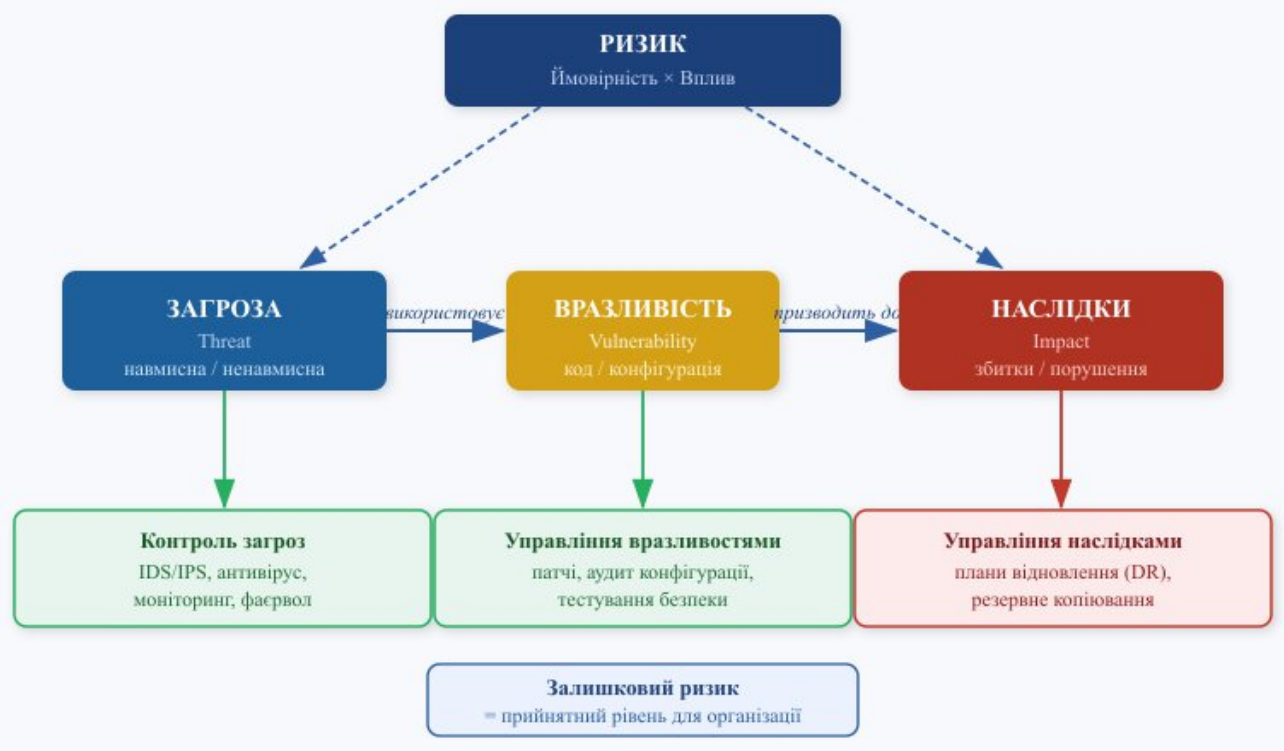

Рисунок 1.2 – Взаємозв'язок між загрозою, вразливістю та ризиком; роль контрзаходів

Важливою практичною концепцією, пов'язаною з управлінням вразливостями, є поняття поверхні атаки (attack surface) — сукупності всіх точок входу в систему, через які зловмисник потенційно може реалізувати загрозу. Поверхня атаки охоплює відкриті мережеві порти та сервіси, доступні API та інтерфейси програмного забезпечення, точки введення даних користувачем, привілейованих користувачів та їхні облікові дані. Зменшення поверхні атаки є одним зі стратегічних напрямів підвищення рівня захищеності системи і безпосередньо пов'язане з принципом мінімізації, що розглядається в наступному підрозділі.

Концепція вектора атаки (attack vector) є суміжною з поняттям поверхні атаки і описує конкретний шлях або метод, за допомогою якого зловмисник реалізує загрозу. Вектори атаки охоплюють мережеві канали (мережевий вектор), підключення фізичних носіїв (фізичний вектор), взаємодію з людьми через соціальну інженерію (соціальний вектор) та ланцюг постачання програмного забезпечення (вектор ланцюга постачання). Розуміння актуальних векторів атаки для конкретної системи дозволяє зосередити захисні зусилля на найбільш вірогідних шляхах реалізації загроз. Слід також розрізняти поняття атаки та інциденту: атака — це цілеспрямована дія зловмисника, спрямована на реалізацію загрози, тоді як інцидент — подія, що фактично спричинила порушення безпеки і потребує реагування.

1.3. Принципи безпечного проєктування програмних систем

Безпека програмної системи значною мірою визначається рішеннями, прийнятими на етапі її проєктування. Реалізація захисних механізмів у вже готовій системі є значно дорожчою та менш ефективною, ніж їхнє впровадження на стадії архітектурного планування. Емпірично підтверджено, що вартість усунення вразливості, виявленої на стадії проєктування, є в десятки разів меншою, ніж вартість усунення тієї самої вразливості після введення системи в експлуатацію. Саме тому принципи безпечного проєктування є не просто рекомендаціями, а основоположними настановами, що мають бути враховані при розробці будь-якої системи, яка обробляє чутливі дані або виконує критичні функції.

Принцип **глибокого захисту (Defense in Depth)** передбачає організацію захисту у вигляді кількох незалежних шарів, таким чином, що подолання одного рівня захисту не надає зловмиснику повного контролю над системою. Ця ідея запозичена з військової стратегії, де концепція ешелонованої оборони довела свою ефективність протягом тисячоліть. У програмних системах це означає поєднання засобів мережевого захисту (фаєрволи, IDS/IPS), захисту на рівні операційної системи (контроль доступу, аудит, обов'язкові профілі безпеки), захисту на рівні застосунку (вхідна валідація, управління сесіями, автентифікація) та захисту на рівні даних (шифрування, маскування чутливих полів). Якщо зловмисник подолає зовнішній мережевий бар'єр, він все одно зіткнеться з вимогою автентифікації; якщо він отримає несанкціонований доступ до файлів — дані залишаться зашифрованими. Такий підхід суттєво підвищує вартість і складність успішної атаки.

На рисунку 1.3 зображена схема принципу Defense in Depth у вигляді концентричних кілець (шарів) захисту.

Рисунок 1.3 – Схема принципу Defense in Depth: ешелоновані рівні захисту

Принцип найменших привілеїв (Principle of Least Privilege, PoLP) є одним з найбільш дієвих інструментів обмеження наслідків компрометації. Суть принципу полягає в тому, що кожному суб'єкту — будь то користувач, процес, служба або компонент системи — надається лише той мінімальний набір прав, який необхідний і достатній для виконання його легітимних функцій. Цей принцип безпосередньо обмежує потенційну шкоду від компрометації: якщо зловмисник отримає контроль над процесом з обмеженими привілеями, можливості для подальшого поширення атаки будуть суттєво звужені. На практиці реалізація цього принципу означає відмову від запуску сервісів з правами адміністратора, використання окремих облікових записів для різних задач, обмеження прав доступу до файлів і мережевих ресурсів, регулярний перегляд та ревізію наданих прав.

Принцип мінімізації поверхні атаки (Minimizing Attack Surface) означає цілеспрямоване скорочення кількості компонентів, інтерфейсів і служб, доступних потенційному зловмиснику. Кожен відкритий мережевий порт, кожна активна служба, кожен публічний інтерфейс програмного забезпечення є потенційною точкою входу для атаки. Практично цей принцип реалізується через відключення невикористовуваних сервісів і функцій, видалення зайвого

програмного забезпечення, закриття непотрібних мережевих портів, обмеження доступу до адміністративних інтерфейсів. Цей принцип також стосується обсягу коду: чим менше рядків коду виконується у привілейованому контексті, тим менше потенційних вразливостей у критичному шляху виконання. Концепція «hardening» (зміцнення) операційної системи та застосунків є практичним втіленням цього принципу.

У таблиці 1.4 подано зведену характеристику основних принципів безпечного проєктування з описом їхньої сутності та практичними прикладами реалізації. Разом ці принципи утворюють цілісну методологію безпечного проєктування, що дозволяє систематично підходити до побудови захищених програмних систем, не покладаючись виключно на окремі технічні засоби захисту.

Таблиця 1.4 – Принципи безпечного проєктування програмних систем

Принцип	Сутність	Практичний приклад
Defense in Depth (глибокий захист)	Кілька незалежних рівнів захисту; подолання одного не дає повного контролю	Фаєрвол + IDS + шифрування + автентифікація
Least Privilege (найменші привілеї)	Кожному суб'єкту — лише мінімум необхідних прав	Веб-сервер запущений від непривілейованого користувача
Minimize Attack Surface (мін. поверхня атаки)	Зменшення кількості активних служб, інтерфейсів, відкритих портів	Відключення SSH на продакшн-сервері з доступом лише через VPN
Fail-Safe Defaults (безпечні значення)	Стан за замовчуванням — максимальне обмеження; дозволяється лише явно	Нова роль користувача не має жодних прав, поки вони не призначені
Separation of Duties (розподіл обов'язків)	Критичні дії потребують участі кількох незалежних суб'єктів	Підтвердження фінансової транзакції двома різними співробітниками
Open Design (відкритий дизайн)	Безпека не залежить від таємниці алгоритму; стійкість — у ключах	Використання стандартних крипто-алгоритмів AES, RSA замість власних
Psychological Acceptability (зручність)	Захист не має бути надмірно обтяжливим, інакше його обходять	Менеджер паролів замість вимоги пам'ятати 30 складних паролів

Ще одним важливим принципом є принцип «відкритого дизайну» (Open Design), відомий у криптографії як принцип Керкгоффса. Він стверджує, що безпека системи не повинна залежати від таємниці її архітектури або

алгоритмів — стійкість має ґрунтуватися виключно на якості ключів і секретних параметрів. Системи, безпека яких побудована на «security through obscurity» (безпека через невідомість), є набагато менш надійними, оскільки архітектура рано чи пізно стає відомою зловмиснику, тоді як добре захищений ключ залишається таємним. Цей принцип є підставою для відкритого рецензування та стандартизації криптографічних алгоритмів: тільки алгоритми, що витримали публічний криптоаналіз протягом тривалого часу, можна вважати надійними.

Принцип психологічної прийнятності (Psychological Acceptability) підкреслює важливість людського чинника в безпеці: захисні механізми мають бути зручними у використанні. Якщо процедури безпеки надто обтяжливі або незручні, користувачі знаходять способи їх обійти — наприклад, записують складні паролі на папірці, вимикають антивірусне програмне забезпечення через уповільнення роботи або ігнорують попередження системи. Це зводить нанівець навіть найдосконаліші технічні механізми захисту. Тому при проєктуванні систем безпеки необхідно шукати баланс між рівнем захищеності та зручністю використання, враховуючи реальну поведінку людей, а не ідеалізовані моделі дотримання регламентів.

1.4. Архітектурна роль операційної системи у захисті програм та даних

Операційна система є центральним компонентом програмного середовища, що відіграє ключову роль у забезпеченні безпеки всіх рівнів обчислювальної системи. Вона виступає посередником між апаратним забезпеченням і прикладними програмами, контролюючи доступ до всіх критичних ресурсів: процесора, оперативної та постійної пам'яті, мережевих інтерфейсів, периферійних пристроїв і файлової системи. Без надійного і коректно налаштованого ядра операційної системи жоден із прикладних механізмів захисту не може гарантувати своєї ефективності: зловмисник, що отримав контроль над операційною системою, автоматично отримує доступ до всіх ресурсів, якими вона управляє, незалежно від того, які захисні механізми реалізовані на прикладному рівні.

Ключовим механізмом безпеки, реалізованим на рівні операційної системи, є розмежування режимів виконання коду. Сучасні процесори підтримують щонайменше два режими: привілейований (режим ядра, kernel mode) і непривілейований (режим користувача, user mode). У режимі ядра виконується код самої операційної системи, що має необмежений доступ до всіх апаратних ресурсів; у режимі користувача виконуються прикладні програми, доступ яких до ресурсів суворо опосередкований системними викликами. Цей поділ є фундаментальним механізмом ізоляції: прикладна програма не може безпосередньо звернутися до апаратного рівня або до пам'яті іншого процесу, не пройшовши через контрольований механізм операційної системи. Порушення цього кордону, зване підвищенням привілеїв (privilege

escalation), є однією з найнебезпечніших категорій атак, оскільки дозволяє зловмиснику отримати практично необмежений контроль над системою.

На рисунку 1.4 зображена архітектура рівнів виконання в операційній системі у вигляді горизонтальних шарів.

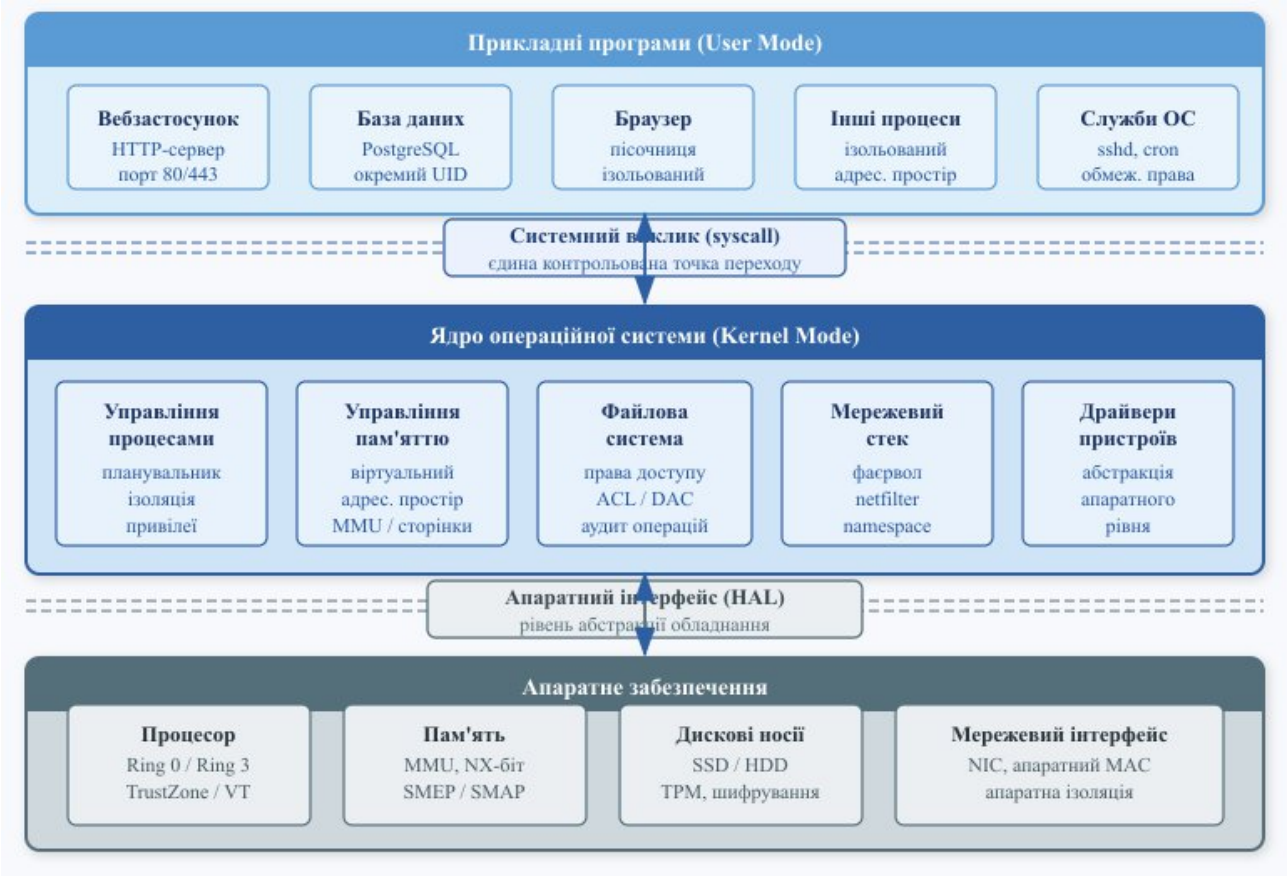

Рисунок 1.4 – Архітектура рівнів операційної системи: user mode, kernel mode та апаратний рівень

Управління пам'яттю є ще одним критично важливим механізмом безпеки операційної системи. Кожен процес отримує власний віртуальний адресний простір, ізольований від адресних просторів інших процесів: навіть якщо два процеси звертаються до однакових віртуальних адрес, вони фактично отримують доступ до різних фізичних ділянок пам'яті, контрольованих блоком управління пам'яттю (MMU) процесора. Механізм захисту пам'яті запобігає тому, щоб один процес міг прочитати або модифікувати дані іншого, що є базовою вимогою для ізоляції програм. Проте помилки в реалізації управління пам'яттю, такі як переповнення буфера або помилки використання пам'яті після звільнення (use-after-free), можуть дозволити зловмиснику порушити цю ізоляцію та отримати несанкціонований доступ до даних або виконання довільного коду.

Файлова система і система дозволів є ще одним рівнем захисту, що реалізується операційною системою. Більшість сучасних операційних систем підтримують дискреційний контроль доступу до файлів і каталогів, що дозволяє визначати, які користувачі та процеси можуть читати, записувати або

виконувати той чи інший файл. У системах на базі UNIX-подібних ядер кожен файл має власника та асоційований набір бітів дозволів для власника, групи та інших користувачів; у системах Windows реалізовані списки контролю доступу (ACL — Access Control List), що забезпечують більш тонке налаштування прав. Деякі системи додатково підтримують обов'язковий контроль доступу (MAC — Mandatory Access Control), реалізований через такі механізми, як SELinux або AppArmor у Linux, що дозволяє задавати централізовані політики безпеки, які не можуть бути скасовані власником файлу.

Підсистема аудиту та журналювання операційної системи відіграє важливу роль як для виявлення інцидентів безпеки, так і для їхнього розслідування. Операційна система реєструє події автентифікації, зміни конфігурації, звернення до критичних ресурсів, запуск і завершення процесів та інші значущі події. Ці журнали є першоджерелом для виявлення аномальної активності та відновлення хронологічної картини атаки після інциденту. Однак самі журнали також потребують захисту: зловмисник, що отримав привілейований доступ до системи, може спробувати їх видалити або модифікувати. Саме тому в системах з підвищеними вимогами до безпеки журнали передаються в реальному часі на централізований захищений сервер (SIEM-система), де вони зберігаються в незмінному вигляді та піддаються автоматизованому аналізу.

Мережевий стек операційної системи також є важливою складовою її архітектурної ролі в захисті. Вбудований фаєрвол дозволяє фільтрувати мережевий трафік на рівні операційної системи, обмежуючи вхідні та вихідні з'єднання відповідно до заданої політики безпеки. Механізми мережевої ізоляції — простори мережевих імен (network namespaces) у Linux або компартменти у Windows — дозволяють ізолювати мережеве середовище окремих процесів або контейнерів. У таблиці 1.5 подано зведений огляд ключових механізмів безпеки операційної системи, їхніх функцій і прикладів практичної реалізації.

Таблиця 1.5 – Ключові механізми безпеки операційної системи

Механізм	Функція захисту	Приклад реалізації
Розмежування режимів виконання	Ізоляція коду ядра (kernel mode) від коду застосунків (user mode)	Системні виклики як єдина точка переходу між режимами
Віртуальна пам'ять та ізоляція процесів	Кожен процес має власний адресний простір; неможливий прямий доступ до пам'яті іншого	MMU процесора + механізм сторінкових таблиць ОС
Файлові дозволи та ACL	Контроль доступу до файлів і каталогів на рівні ОС	chmod/chown у Linux; NTFS ACL у Windows
Аудит та журналювання	Реєстрація подій автентифікації, доступу, змін конфігурації	auditd (Linux), Windows Event Log, syslog
Вбудований фаєрвол	Фільтрація вхідного та вихідного мережевого	iptables/nftables (Linux), Windows Defender

	трафіку	Firewall
Обов'язковий контроль доступу (MAC)	Централізована політика безпеки, що не може бути скасована користувачем	SELinux, AppArmor (Linux); Mandatory Integrity Control (Windows)

Розуміння архітектурної ролі операційної системи у забезпеченні безпеки є необхідною передумовою для вивчення конкретних механізмів захисту, що розглядатимуться в подальших темах дисципліни. Ізоляція процесів і ресурсів, моделі контролю доступу, аудит безпеки та захист мережевого периметра — всі ці концепції ґрунтуються на розглянутих у цій темі архітектурних основах і набувають повного змісту лише в контексті системного розуміння рівнів і механізмів захисту операційної системи. Коректне налаштування цих механізмів — так зване «зміцнення» (hardening) операційної системи — є першим і найважливішим кроком у побудові будь-якої захищеної інформаційної системи.

Слід окремо підкреслити, що операційна система сама по собі є складним програмним продуктом і містить власні вразливості, що виявляються та усуваються протягом усього її життєвого циклу. Тому підтримання актуального стану операційної системи — своєчасне встановлення оновлень безпеки — є невід'ємною частиною її безпечної експлуатації. Відома концепція CVE (Common Vulnerabilities and Exposures) і рейтинг CVSS (Common Vulnerability Scoring System) дозволяють стандартизовано описувати та пріоритизувати виявлені вразливості, що детальніше розглядатиметься в темі 7 дисципліни. Таким чином, безпека на рівні операційної системи вимагає системного підходу: правильної архітектурної конфігурації, коректного налаштування дозволів, активного моніторингу та регулярного оновлення.

Тема 2. Ізоляція, віртуалізація та моделювання безпечного програмного середовища

2.1. Принципи ізоляції процесів і ресурсів в операційних системах

Сучасна операційна система є складним програмним середовищем, у якому одночасно виконуються десятки і сотні процесів, що належать різним користувачам і додаткам. Для забезпечення безпеки та стабільності роботи кожен процес має виконуватись у власному захищеному просторі та з мінімальним набором привілеїв, необхідних для виконання його функцій. Саме цю властивість прийнято називати ізоляцією процесів, і вона є однією з фундаментальних архітектурних гарантій, на яких будується вся решта захисних механізмів операційної системи. Без надійної ізоляції будь-яке скомпрометоване застосування ставало б відправною точкою для атаки на всю систему, що робить її не просто бажаним покращенням, а необхідною умовою безпечного функціонування.

Основою процесної ізоляції в Linux є концепція віртуального адресного простору. Кожен процес отримує власне відображення логічних адрес пам'яті на фізичні сторінки оперативної пам'яті і не може безпосередньо звертатись до пам'яті іншого процесу. Ядро управляє таблицями сторінок (page tables) для кожного процесу окремо: навіть якщо два процеси мають однакову логічну адресу, фізично ці адреси вказують на зовсім різні ділянки пам'яті. Механізм захисту підтримується на рівні процесора: будь-яка спроба процесу звернутись до сторінки, яка йому не належить, спричиняє апаратне переривання — page fault, — яке ядро обробляє, надсилаючи процесу сигнал SIGSEGV. Таким чином, порушення ізоляції пам'яті фізично неможливе без участі ядра або серйозної вразливості в ньому.

Для кращого розуміння структури пам'яті процесу корисно розглянути схему її організації, зображену на рисунку 2.1. Адресний простір процесу логічно поділяється на дві великі зони: простір ядра (Kernel Space) і простір користувача (User Space). Процес не може читати або записувати у простір ядра безпосередньо — будь-яке звернення до ресурсів ядра відбувається виключно через системні виклики (syscall).

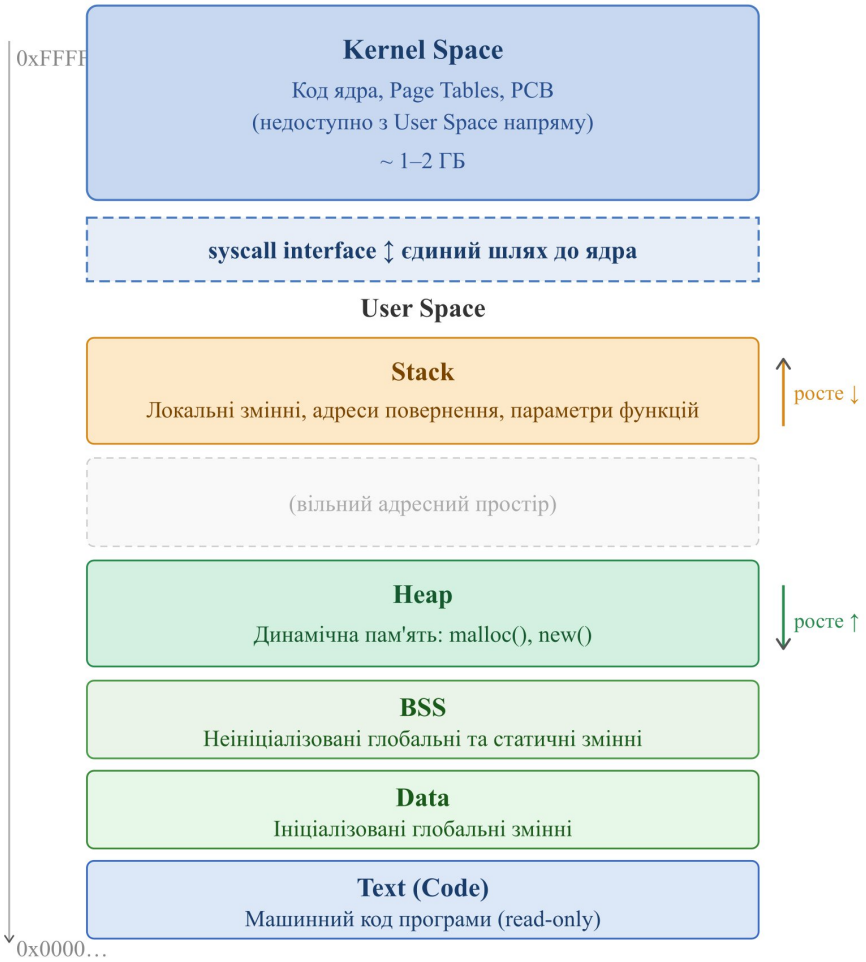

Рисунок 2.1 — Структура віртуального адресного простору процесу Linux

Окрім ізоляції пам'яті, Linux забезпечує ізоляцію системних ресурсів через механізм просторів імен (namespaces). Простір імен — це підсистема ядра, яка надає групі процесів відокремлений «погляд» на певний ресурс системи, приховуючи від них усе, що знаходиться поза їхнім простором імен. Поточна версія ядра Linux підтримує сім типів просторів імен: PID (ізоляція ідентифікаторів процесів), Network (ізоляція мережевого стека), Mount (ізоляція файлової системи), UTS (ізоляція імені хосту і домену), IPC (ізоляція міжпроцесної взаємодії), User (ізоляція UID/GID) та Cgroup (ізоляція ієрархії cgroups). Завдяки саме цим механізмам стає можливою контейнеризація: контейнер є не чим іншим, як групою процесів, поміщених у набір ізольованих просторів імен.

Схему взаємодії різних типів просторів імен ілюструє рисунок 2.2. Розуміння цієї ієрархії є ключовим для аналізу рівня ізоляції, який забезпечує той чи інший контейнер або тестове середовище.

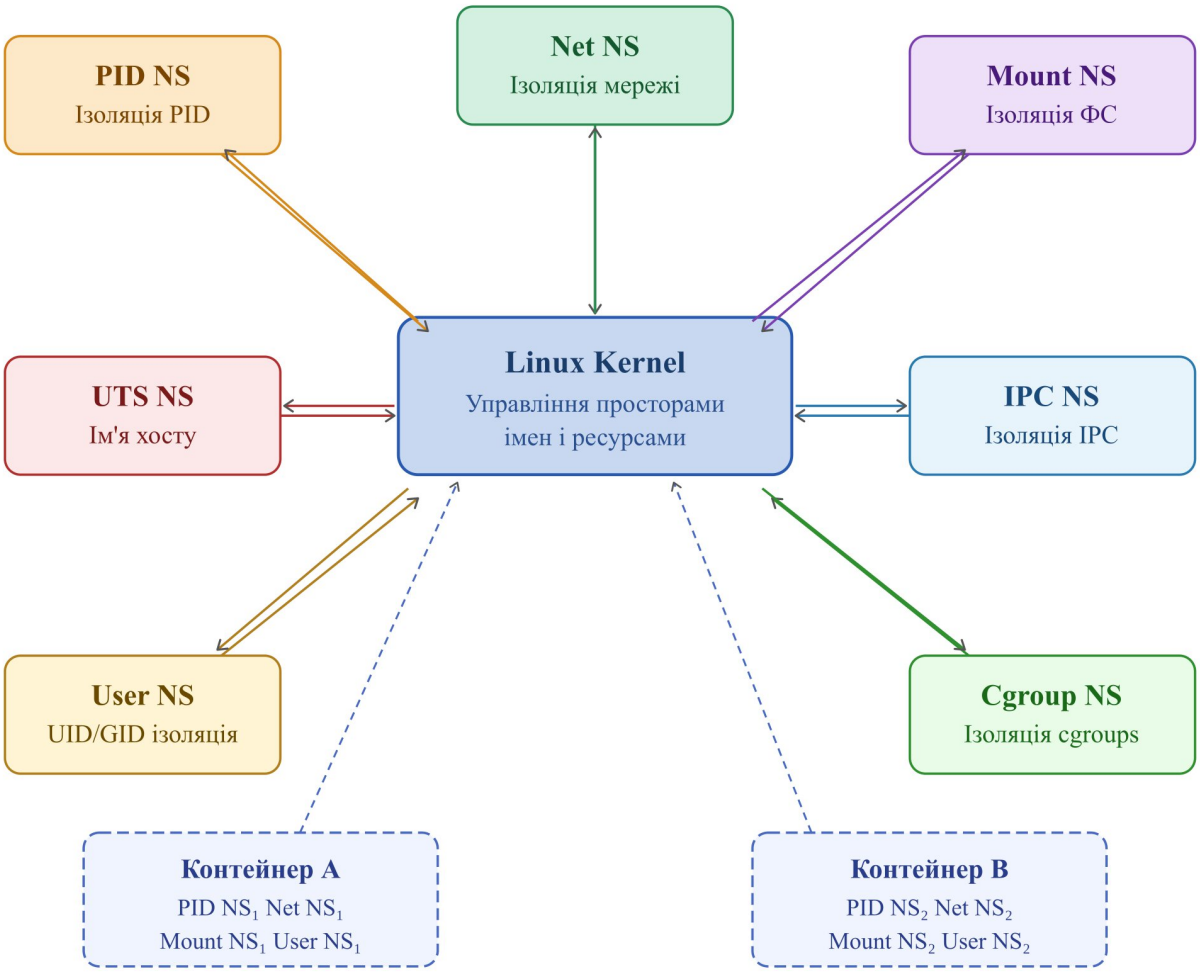

Рисунок 2.2 — Ієрархія Linux Namespaces та їх зв'язок із процесами

Практичне використання просторів імен можна дослідити безпосередньо в Linux Mint без будь-якого спеціалізованого програмного забезпечення. Утиліта unshare дозволяє запустити процес у новому, ізольованому просторі імен, а утиліта nsenter — навпаки, «зайти» у вже існуючий простір імен іншого процесу. Наприклад, команда sudo unshare --pid --fork --mount-proc bash запускає нову оболонку bash у власному просторі PID: команда ps aux всередині цієї оболонки покаже лише один процес (сам bash), тоді як хостова система при цьому продовжує нормально функціонувати. Це унаочнює, як саме Docker ізолює дерево процесів кожного контейнера від решти системи.

Технологія seccomp (Secure Computing Mode) надає ще один шар захисту — фільтрацію системних викликів на рівні ядра. Кожен процес може активувати для себе seccomp-фільтр, визначений у форматі BPF (Berkeley Packet Filter), який перехоплює кожен системний виклик і вирішує, чи дозволити його виконання, чи повернути помилку, чи завершити процес. Docker за замовчуванням застосовує власний seccomp-профіль, що забороняє понад 40 потенційно небезпечних системних викликів, зокрема ptrace (дозволяє один процес дебажити інший), mount (монтування файлових систем),

kexec_load (завантаження нового ядра) та mknod (створення спеціальних файлів пристроїв). Можна уявити seccomp як «білий список» операцій: якщо системний виклик не входить до дозволених, він просто не буде виконаний, навіть якщо зловмисник отримав контроль над кодом процесу. Це суттєво обмежує можливості подальшого розвитку атаки після початкової компрометації.

Механізм capabilities (привілеїв) доповнює seccomp, вирішуючи класичну проблему «все або нічого» при роботі від імені root. Традиційно в UNIX root-користувач мав абсолютні привілеї, і будь-який процес, що виконувався від його імені, міг робити з системою будь-що. Capabilities розбивають ці привілеї на близько 40 незалежних одиниць: CAP_NET_BIND_SERVICE (прив'язка до портів < 1024), CAP_SYS_PTRACE (відлагодження процесів), CAP_CHOWN (зміна власника файлів) тощо. Застосунок може отримати лише ті capabilities, які справді потрібні для його роботи, а решту — скинути. Комбінація seccomp, capabilities та AppArmor/SELinux утворює ешелонований захист, і навіть якщо зловмиснику вдасться обійти один шар, наступний обмежить масштаб завданої шкоди. Нижче у таблиці 2.1 зведено всі ключові механізми ізоляції Linux.

Таблиця 2.1 — Механізми ізоляції процесів та ресурсів у Linux

Механізм	**Що ізолює / контролює**	**Рівень захисту**	**Приклад використання**
Namespaces	Видимість ресурсів: PID, мережа, монтування, UTS, IPC, UID	Середній–Високий	Docker, LXC: ізоляція контейнерів; unshare для тестових mid середовищ
cgroups	Ліміт CPU, RAM, I/O, к-ть процесів для групи	Середній (DoS-захист)	Обмеження ресурсів контейнера; захист від «виснаження» хоста
seccomp	Фільтрація системних викликів (syscall whitelist/blacklist)	Високий	Chrome sandbox, Docker default profile; заборона mknod, ptrace тощо
Capabilities	Гранулярний набір привілеїв замість «все або нічого» root	Середній–Високий	docker run --cap-drop=ALL --cap-add=NET_BIND_SERVICE для веб-сервера
AppArmor / SELinux	MAC-профілі: обмежують	Високий	Обов'язкові профілі для демонів (nginx, sshd); захист контейнерів

Механізм	Що ізолює / контролює	Рівень захисту	Приклад використання
	доступ процесів до файлів і мережі		

⚠ Попередження з безпеки: Вимикання будь-якого з перелічених механізмів (наприклад, запуск контейнера з --privileged або --security-opt seccomp=unconfined) суттєво знижує рівень ізоляції та може дозволити зловмиснику вийти за межі ізольованого середовища на хостову систему. Ніколи не застосовуйте ці прапори у виробничих середовищах без обґрунтування та документування ризику.

2.2. Віртуалізація: типи, архітектура та вплив на безпеку

Віртуалізація — це технологія, яка дозволяє запускати одну або декілька операційних систем (гостьових ОС) на одному фізичному апаратному забезпеченні паралельно або замість основної системи. Ключовим елементом архітектури є гіпервізор — програмний або мікропрограмний шар, що виступає посередником між гостьовою ОС і реальним апаратним забезпеченням. Для кожної гостьової системи гіпервізор створює ілюзію роботи на виділеному апаратному ресурсі: власний процесор, пам'ять, мережевий інтерфейс та диск. З точки зору безпеки це означає, що навіть якщо гостьова ОС повністю скомпрометована зловмисником, усі його дії обмежені межами віртуальної машини, а хостова система залишається захищеною — за умови що сам гіпервізор не містить критичних вразливостей. Саме ця властивість робить віртуалізацію незамінним інструментом у сфері кібербезпеки.

Гіпервізори поділяються на два типи залежно від місця їх розміщення в архітектурі системи. Гіпервізори першого типу (Type-1, bare-metal) встановлюються безпосередньо на апаратне забезпечення і не потребують хостової ОС, маючи прямий доступ до ресурсів заліза. Гіпервізори другого типу (Type-2, hosted) виконуються поверх звичайної операційної системи як звичайні застосунки, використовуючи її сервіси для доступу до апаратного забезпечення. Обидва типи широко застосовуються у сфері безпеки, але в різних сценаріях. Для навчальних цілей та дослідницьких стендів найчастіше використовується VirtualBox (Type-2), встановлений на Linux Mint, тоді як у корпоративних і хмарних середовищах домінують ESXi або KVM (Type-1). Детальне порівняння наведено у таблиці 2.2.

Таблиця 2.2 — Порівняння гіпервізорів Типу 1 та Типу 2

Критерій	Гіпервізор Типу 1 (bare-metal)	Гіпервізор Типу 2 (hosted)
Приклади	VMware ESXi, Microsoft Hyper-V, KVM, Xen	VirtualBox, VMware Workstation, QEMU
Розміщення	Безпосередньо на апаратному забезпеченні	Поверх хостової ОС
Продуктивність	Висока (прямий доступ до HW)	Нижча (додатковий рівень ОС)
Поверхня атаки	Менша (немає хостової ОС)	Більша (вразливості ОС + гіпервізора)
Складність налаштування	Висока	Низька — зручний для навчання
Типові сценарії	Дата-центри, хмарна інфраструктура, продуктивні сервери	Навчання, розробка, дослідницькі стенди, тестування
Ризик VM escape	Нижчий	Вищий при outdated версіях

Особливої уваги заслуговує технологія KVM (Kernel-based Virtual Machine) — модуль ядра Linux, який перетворює ядро на повноцінний гіпервізор першого типу. Завдяки апаратним розширенням процесора Intel VT-x та AMD-V, KVM дозволяє гостьовим ОС виконувати привілейовані інструкції безпосередньо на процесорі (без емуляції), що забезпечує продуктивність, наближену до «нативної». З точки зору безпеки KVM цікавий тим, що частина коду гіпервізора перебуває у просторі ядра Linux: це означає, що вразливість у ядрі хоста може потенційно впливати на ізоляцію гостьових VM. Водночас рівень зрілості та якості аудиту коду ядра Linux дуже високий, а виявлені вразливості оперативно усуваються.

Вкрай важливим поняттям для розуміння меж безпеки віртуалізації є атака типу «VM escape» або «hypervisor escape» — клас вразливостей, при якому зловмисник, отримавши контроль над гостьовою ОС, знаходить і експлуатує помилку в гіпервізорі, щоб виконати код безпосередньо на хості або в іншій VM. Реальним прикладом є вразливість VENOM (CVE-2015-3456, CVSS 9.3): вона полягала в помилці переповнення буфера в емуляторі флоппі-дискети (FDC) QEMU/KVM, і дозволяла зловмиснику всередині гостьової VM надіслати спеціально сформовані команди до FDC, що призводило до виконання довільного коду у просторі гіпервізора. Попри те що реальні флоппі-диски нікому не потрібні, відповідний контролер за замовчуванням залишався увімкненим у конфігурації. Це ілюструє принцип мінімальної поверхні атаки:

кожна увімкнена функція або пристрій, навіть якщо вони не використовуються, є потенційним вектором атаки.

Вразливості класу Spectre та Meltdown (2018) відкрили ще один вимір загроз для середовищ віртуалізації — атаки через апаратні канали витоку (side-channel). Meltdown дозволяв процесу читати пам'ять ядра через особливості спекулятивного виконання процесора, а Spectre — маніпулювати спекулятивним виконанням для витоку даних між різними процесами або між VM і гіпервізором. Повна ізоляція від цих атак вимагала змін як у мікрокоді процесорів, так і в ядрі Linux (KPTI — Kernel Page-Table Isolation), що спричинило певне зниження продуктивності систем. Ці вразливості показали, що навіть архітектурно ізольовані VM не є повністю захищеними від зловмисника, який контролює іншу VM на тому ж фізичному сервері.

Варто також зупинитися на концепції вкладеної (nested) віртуалізації — запуску VM всередині VM. Це корисний інструмент для тестування гіпервізорів, навчання адміністраторів хмарних систем або відтворення складних виробничих середовищ у лабораторії. У VirtualBox та KVM nested virtualisation підтримується явно і вмикається окремим параметром. З точки зору безпеки вкладена віртуалізація збільшує складність системи, розширює поверхню атаки і ускладнює моніторинг: атака, що відбувається у «внутрішній» VM, може бути значно складнішою для виявлення інструментами моніторингу «зовнішньої» VM. Тому у виробничих безпекових середовищах nested virtualisation застосовується лише за наявності чіткого обґрунтування.

2.3. Контейнеризація як механізм легковагової ізоляції

Контейнери являють собою порівняно новий підхід до ізоляції, який набув масового поширення завдяки таким проектам, як Docker і Kubernetes. На відміну від повної апаратної віртуалізації, контейнери не запускають окрему гостьову ОС: вони використовують те ж саме ядро Linux, що й хостова система, а ізоляція досягається виключно за рахунок механізмів ядра — просторів імен і cgroups, розглянутих у розділі 2.1. Завдяки цьому контейнери запускаються за частки секунди, споживають на порядок менше ресурсів і дозволяють розмістити тисячі ізольованих середовищ на одному сервері. Архітектуру Docker-системи ілюструє рисунок 2.3.

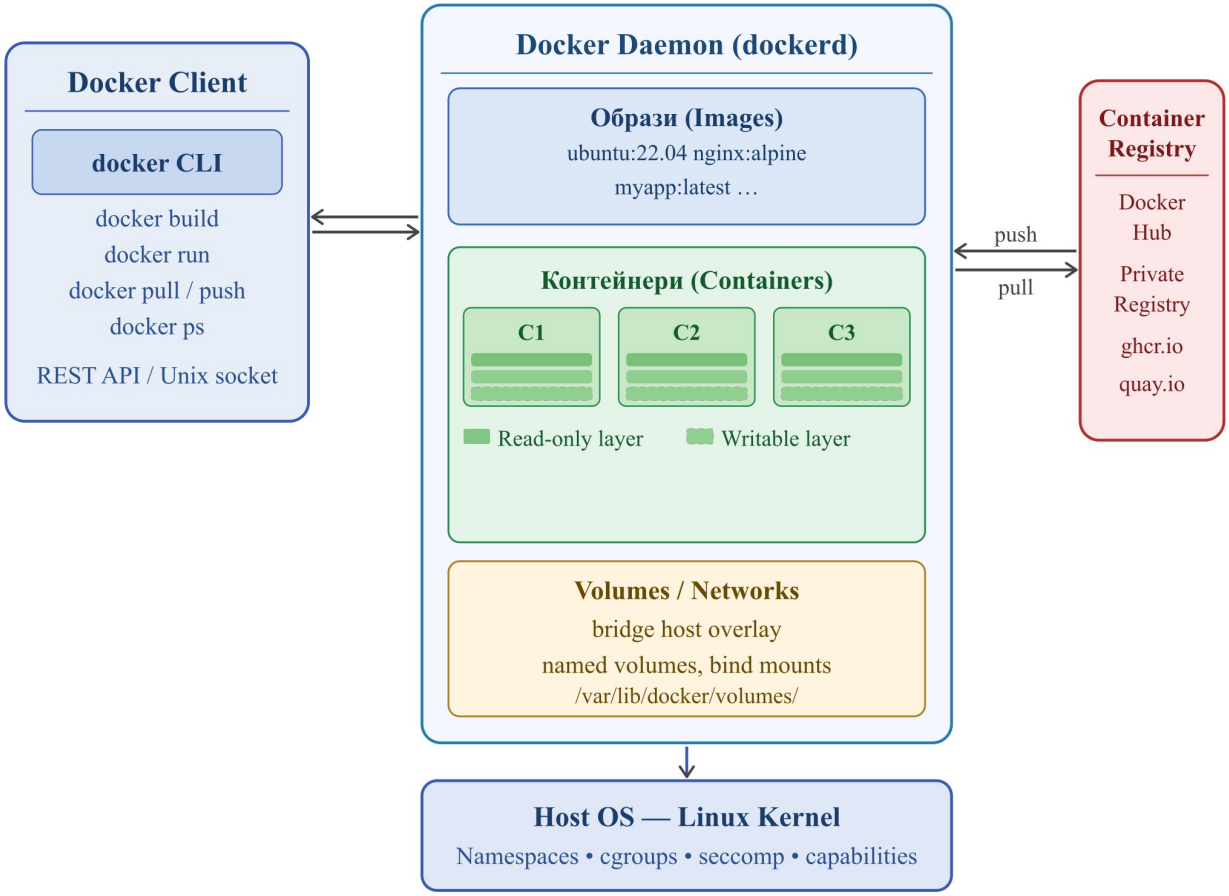

Рисунок 2.3 — Архітектура Docker: клієнт, демон, реєстр та контейнери

Архітектура образів Docker заснована на механізмі шарів файлової системи (Union File System, overlay2). Кожна інструкція Dockerfile (RUN, COPY, ADD) створює новий незмінний шар, що «накладається» поверх попереднього. Завдяки тому що кожен шар ідентифікується криптографічним хешем SHA-256, Docker автоматично перевіряє цілісність усіх компонентів образу: якщо хоча б один байт будь-якого шару змінено зловмисником, хеш відповідного шару зміниться, і Docker відмовиться використовувати такий образ без явної перевірки. Це є важливим елементом ланцюга довіри програмного забезпечення — аналогом перевірки підписів пакетів в apt. Постачальники програмного забезпечення можуть підписувати образи цифровим підписом (Docker Content Trust), а системи розгортання — верифікувати підпис перед запуском.

Таблиця 2.3 містить детальне порівняння повних VM та Docker-контейнерів за ключовими критеріями безпеки і ресурсоспоживання — це допоможе в кожному конкретному сценарії обирати оптимальний інструмент.

Таблиця 2.3 — Порівняльна характеристика VM та Docker-контейнерів

Критерій	Віртуальна машина (VM)	Docker-контейнер
Ізоляція ядра ОС	Повна — окреме ядро	Часткова — спільне ядро хоста
Час запуску	30 с — кілька хв	< 1 с
RAM на екземпляр	Сотні МБ — ГБ (повна ОС)	Мегабайти (лише процес + libs)
Ризик виходу з ізоляції	Нижчий (VM escape важко)	Вищий при неправильній конфігурації
Снімки стану (snapshot)	Повна підтримка	Часткова (шари образів)
Зберігання образу	Файл диску (.vmdk, .vdi)	Шарова файлова система (overlay2)
Типові сценарії безпеки	Аналіз шкідливого ПЗ, lab стенди, prod сервери	Мікросервіси, CI/CD, розгортання застосунків

Незважаючи на переваги легковаги, контейнери мають принципово ширшу поверхню атаки порівняно з VM: усі контейнери одного хоста розділяють ядро ОС. Вразливість у ядрі Linux, яка дозволяє ескалацію привілеїв, потенційно надає зловмиснику контроль над усіма контейнерами хоста одночасно. Рисунок 2.4 наочно демонструє різницю в архітектурі ізоляції VM та контейнерів.

Наведемо практичний приклад безпечного запуску контейнера в Linux Mint із застосуванням кількох захисних прапорів одночасно. Команда поєднує скидання всіх capabilities, перехід до режиму read-only файлової системи та застосування кастомного seccomp-профілю:

```
docker run --rm -it \
  --cap-drop=ALL \
  --cap-add=NET_BIND_SERVICE \
  --read-only \
  --tmpfs /tmp:rw,size=64m \
  --security-opt no-new-privileges \
  --security-opt seccomp=/etc/docker/seccomp-strict.json \
  --user 1000:1000 \
  nginx:alpine
```

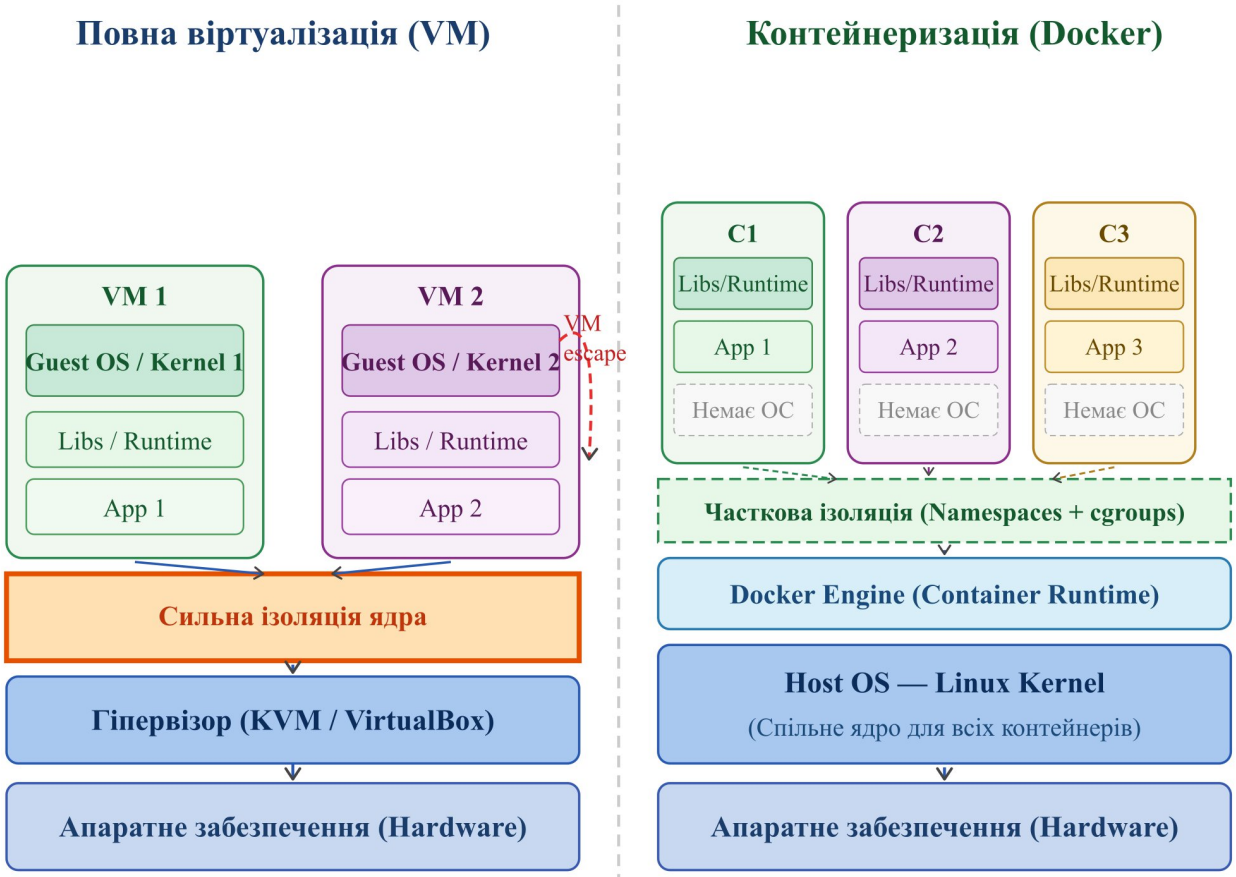

Рисунок 2.4 — Архітектурне порівняння ізоляції VM (зліва) та контейнерів (справа)

У цьому прикладі --cap-drop=ALL скидає всі privileges, а --cap-add=NET_BIND_SERVICE повертає лише ту, що потрібна веб-серверу для прив'язки до порту 80. Прапор --read-only робить файлову систему контейнера незмінною, а --tmpfs /tmp надає тимчасовий записуваний розділ лише для /tmp. Параметр --user 1000:1000 запускає процес від імені непривілейованого користувача навіть всередині контейнера, а --security-opt no-new-privileges забороняє процесу будь-яким способом підвищити свої привілеї (наприклад, через setuid-біт). Таблиця 2.4 систематизує найважливіші практики безпеки Docker-контейнерів.

2.4. Практичне використання ізольованих середовищ для аналізу та тестування

Однією з ключових прикладних задач для фахівця з кібербезпеки є безпечне дослідження шкідливого програмного забезпечення та перевірка ефективності захисних заходів без ризику завдати шкоди робочій інфраструктурі. Для цього широко застосовуються ізольовані «лабораторні стенди» (security labs) — середовища, побудовані на базі VM або контейнерів, у

яких можна відтворити реалістичне мережеве оточення, запустити потенційно небезпечний код і спостерігати за його поведінкою. Такий підхід реалізує принцип «безпечного провалу» (fail safe): навіть якщо аналізований зразок повністю скомпрометує гостьову ОС, уся шкідлива діяльність залишається всередині VM, а стан може бути відновлений зі знімку за лічені секунди.

Таблиця 2.4 — Практики безпеки Docker-контейнерів

Практика безпеки	**Команда / Налаштування**	**Що запобігає**
Не запускати як root	USER appuser в Dockerfile	Ескалація привілеїв через вразливість у застосунку
Скидати всі capabilities	--cap-drop=ALL --cap-add=NET_BIND	Використання небезпечних системних операцій
Read-only файлова система	--read-only --tmpfs /tmp	Запис шкідливого ПЗ або зміна конфігурацій
Seccomp-профіль	--security-opt seccomp=profile.json	Небезпечні syscalls (ptrace, mount, mknod)
Не монтувати Docker socket	Не використовувати -v /var/run/docker.sock	Повний доступ до хоста через контейнер
Сканування образів	docker scout cves IMAGE або trivy image IMAGE	Відомі CVE у базових образах та залежностях
Мінімальний базовий образ	FROM alpine або distroless	Зменшення поверхні атаки: менше пакетів = менше CVE

⚠ Попередження з безпеки: Монтування Docker-сокету (-v /var/run/docker.sock:/var/run/docker.sock) всередину контейнера фактично надає йому повний контроль над усіма контейнерами хоста та можливість запускати нові привілейовані контейнери. Ніколи не використовуйте цей прийом у виробничих або публічно доступних середовищах.

Типова конфігурація навчального стенда складається з двох або більше VM, об'єднаних у ізольовану внутрішню мережу (Host-Only або Internal Network у VirtualBox) без виходу до реального Інтернету. В такій конфігурації одна машина виконує роль атакуючого (наприклад, Kali Linux із набором інструментів тестування), а друга — цільової жертви (наприклад, навмисно вразлива система Metasploitable або DVWA). Весь трафік між ними проходить лише всередині гіпервізора. Схему такого стенда зображено на рисунку 2.5.

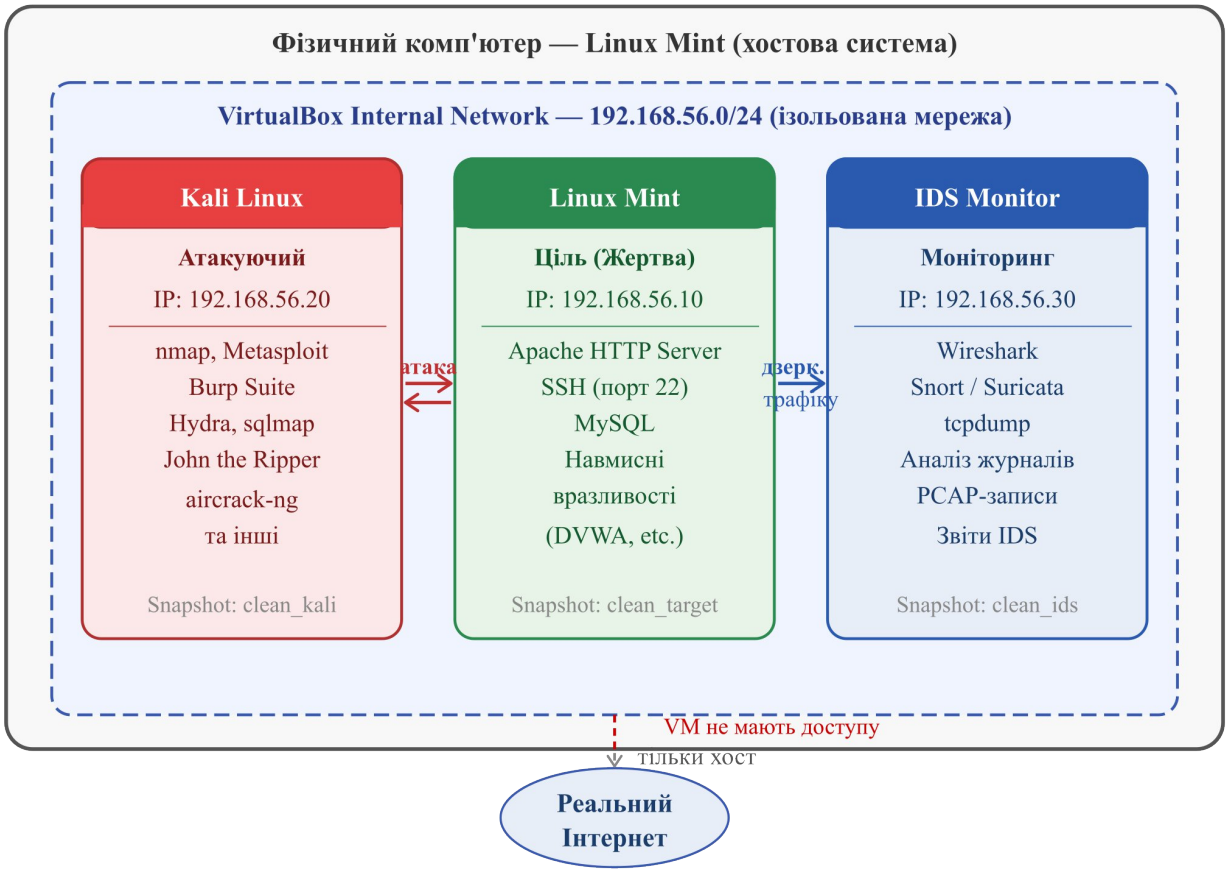

Рисунок 2.5 — Топологія навчального стенда безпеки на базі VirtualBox у Linux Mint

Для дослідження поведінки шкідливих програм застосовуються спеціалізовані платформи динамічного аналізу, найвідомішою з яких є Cuckoo Sandbox. Вона являє собою автоматизований аналізатор, який запускає підозрілий файл або URL у гостьовій VM і детально реєструє всі дії: системні виклики, звернення до файлової системи, мережеві з'єднання, спроби завантаження додаткових компонентів і модифікації системних файлів. Процес аналізу в Cuckoo ілюструє рисунок 2.6.

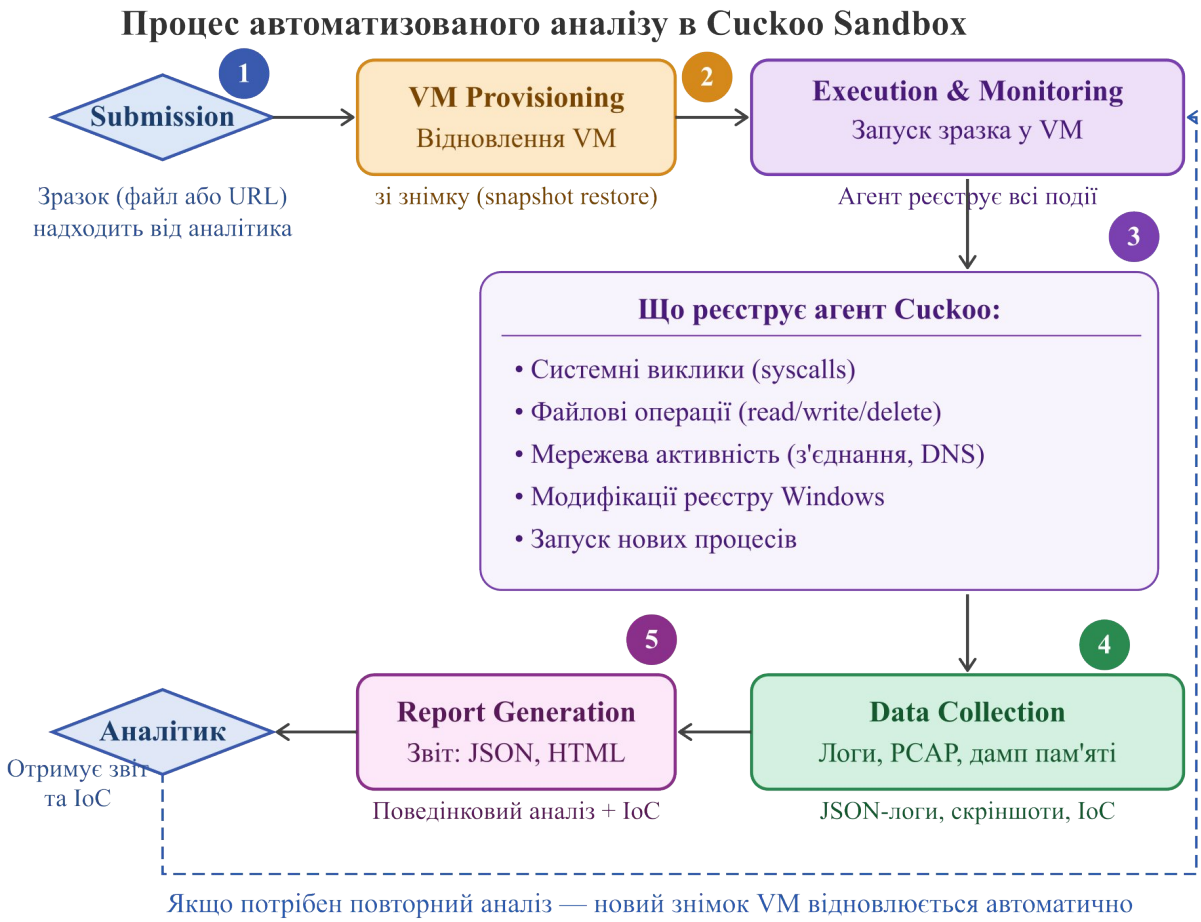

Рисунок 2.6 — Блок-схема процесу аналізу у Cuckoo Sandbox

Неодмінною складовою роботи з ізольованими середовищами є управління знімками стану (snapshots). У VirtualBox знімок зберігає повний стан VM у конкретний момент: вміст оперативної пам'яті, стан жорсткого диска, конфігурацію мережі. Якщо під час тестування стан системи небажано змінився, відновлення зі знімку займає лише кілька секунд і повертає машину до відомого «чистого» стану. Рекомендована практика — створювати базовий знімок одразу після встановлення ОС та базових інструментів, а потім — перед кожним новим практичним завданням. Для автоматизації розгортання стендів широко застосовується Vagrant — інструмент для програмного управління VM. Нижче наведено мінімальний Vagrantfile, який автоматично розгортає дві VM в ізольованій мережі:

```
Vagrant.configure('2') do |config|
  config.vm.define 'target' do |t|
    t.vm.box = 'ubuntu/jammy64'
    t.vm.network 'private_network', ip: '192.168.56.10'
    t.vm.provider 'virtualbox' do |v|
      v.memory = 1024
    end
  end
  config.vm.define 'attacker' do |a|
```

```
      a.vm.box = 'kalilinux/rolling'
      a.vm.network 'private_network', ip: '192.168.56.20'
    end
  end
```

Такий Vagrantfile описує інфраструктуру у вигляді коду (Infrastructure as Code): будь-хто, хто має цей файл і встановлений Vagrant, може відтворити ідентичне тестове середовище командою vagrant up. Це відповідає принципу відтворюваності (reproducibility), який є важливим у дослідженнях безпеки: якщо аналітик виявив вразливість у конкретному середовищі, він може поділитись Vagrantfile і дозволити колезі відтворити умови атаки незалежно. Порівняно зі «ручним» розгортанням VM це значно скорочує час підготовки і виключає людські помилки при налаштуванні.

Підсумовуючи матеріал теми, слід підкреслити, що ізоляція, віртуалізація і контейнеризація — не взаємовиключні, а взаємодоповнювальні технології захисту. Вибір між повною VM і контейнером залежить від конкретного завдання: для дослідження шкідливого ПЗ або розгортання цільової системи-жертви краще підходить VM із суворою ізоляцією ядра; для мікросервісних застосунків і CI/CD-пайплайнів — легковаговий контейнер. У реальних виробничих середовищах обидві технології часто застосовуються спільно: Kubernetes-кластер може виконуватись поверх VM у хмарі, де кожна VM є вузлом кластера, а всередині неї запущені сотні контейнерів. Розуміння архітектурних переваг і обмежень кожного підходу, включно з притаманними їм векторами атак та механізмами захисту, дозволяє фахівцеві з кібербезпеки обирати оптимальне рішення залежно від моделі загроз і операційних вимог конкретної системи. Ці знання безпосередньо знадобляться при вивченні наступних тем, де розглядаються моделі контролю доступу, аудит систем та мережева фільтрація.

Тема 3. Моделі контролю доступу та безпека облікових записів

3.1. Ідентифікація, автентифікація та авторизація

Забезпечення безпеки будь-якої інформаційної системи починається з відповіді на три фундаментальні запитання: хто ти є, чи справді ти є тим, за кого себе видаєш, і що саме тобі дозволено робити. Саме ці три запитання відповідають трьом ключовим процесам — ідентифікації, автентифікації та авторизації, які разом утворюють основу механізму контролю доступу. Розуміння відмінностей між ними є принциповим для фахівця з кібербезпеки, оскільки помилки на будь-якому з цих рівнів здатні повністю зруйнувати систему захисту. Ці три процеси завжди виконуються послідовно: спочатку система дізнається, кого перед нею, потім перевіряє це твердження, і лише після цього надає або відмовляє у доступі до певних ресурсів. Такий чіткий поділ дозволяє будувати багаторівневі системи захисту, де кожен рівень є незалежним і може бути налаштований окремо.

Ідентифікація — це процес, під час якого суб'єкт (користувач, програма або пристрій) заявляє про свою особу шляхом надання певного унікального ідентифікатора. Найпоширенішим прикладом є введення імені користувача (логіна) у форму входу до системи. Важливо розуміти, що на цьому етапі система ще не перевіряє, чи справді суб'єкт є тим, за кого себе видає — вона лише отримує заявку про особу. В операційній системі Linux Mint кожен користувач має унікальне ім'я (username) та пов'язаний з ним числовий ідентифікатор UID (User ID), що зберігається у файлі /etc/passwd. Ці ідентифікатори є основою для всіх подальших перевірок і дозволів у системі. Система ідентифікує не лише людей, а й процеси, служби та мережеві вузли — у кожного з них є свій унікальний ідентифікатор, і саме на основі цих ідентифікаторів операційна система розмежовує доступ до ресурсів.

Автентифікація — це процес перевірки того, що суб'єкт справді є тим, ким він себе назвав. Для цього система перевіряє один або кілька факторів автентифікації. Традиційно виділяють три категорії таких факторів: те, що суб'єкт знає (пароль, PIN-код, секретна відповідь), те, чим він володіє (смарт-картка, апаратний токен, мобільний телефон), і те, ким він є (біометричні дані: відбиток пальця, розпізнавання обличчя, голосовий відбиток). Кожна категорія має свої переваги та недоліки: паролі зручні, але можуть бути вкрадені або підібрані; апаратні токени надійніші, але можуть бути втрачені; біометрія унікальна, але незмінна у разі компрометації. Вибір методу автентифікації завжди є компромісом між рівнем безпеки та зручністю для користувача, і цей компроміс повинен бути усвідомленим рішенням, що ґрунтується на аналізі ризиків.

Авторизація настає після успішної автентифікації і визначає, які ресурси та операції доступні автентифікованому суб'єкту. Навіть якщо користувач успішно підтвердив свою особу, це не означає, що він має право виконувати будь-які дії в системі. Рішення про доступ приймається на основі попередньо встановлених правил і політик, які описують, що кому дозволено. Наприклад, у

Linux Mint звичайний користувач може читати свої файли та запускати програми, але не може змінювати системні конфігурації — ця операція вимагає привілеїв суперкористувача (root) або виконання команди через sudo. Авторизація може бути реалізована на різних рівнях: операційної системи, бази даних, застосунку, мережевого пристрою — і кожен рівень додає власний шар контролю.

Для кращого розуміння відмінностей між трьома процесами наведемо їх порівняльну характеристику в таблиці 3.1.

Таблиця 3.1 – Порівняльна характеристика ідентифікації, автентифікації та авторизації

Характеристика	**Ідентифікація**	**Автентифікація**	**Авторизація**
Запитання	Хто ти?	Чи справді ти — це ти?	Що тобі дозволено?
Механізм	Логін, UID, ім'я процесу	Пароль, токен, біометрія	ACL, ролі, політики
Момент виконання	Початок сеансу	Після ідентифікації	Після автентифікації
Приклад у Linux	Введення імені користувача	Перевірка пароля через PAM	Перевірка прав файлу (chmod)

У Linux-системах автентифікація традиційно реалізована через модуль PAM (Pluggable Authentication Modules) — гнучку архітектуру, що дозволяє підключати різні методи автентифікації без зміни самих програм. PAM розділяє логіку автентифікації від застосунків, тому одну й ту саму конфігурацію можна застосувати до різних програм: входу в систему, SSH, sudo тощо. Конфігураційні файли PAM розташовані у директорії /etc/pam.d/, і кожен файл відповідає конкретному сервісу. Таке архітектурне рішення є прикладом хорошого проєктування з точки зору безпеки: зміна методу автентифікації в одному місці автоматично поширюється на всі сервіси, що його використовують, — це спрощує управління та зменшує ризик помилок при масштабуванні системи.

Структуру файлів облікових записів у Linux, в яких зберігаються дані для ідентифікації та автентифікації, унаочнює рисунок 3.1. На рисунку показано два прямокутні блоки, розташовані вертикально один під одним. Верхній блок підписано "/etc/passwd" і містить рядок-зразок у форматі: "alice:x:1001:1001:Alice Smith:/home/alice:/bin/bash", де кожне поле відокремлено двокрапкою і підписано знизу: "ім'я", "x (пароль у shadow)", "UID", "GID", "коментар", "домашній каталог", "оболонка". Нижній блок підписано "/etc/shadow" і містить рядок-зразок: "alice:6salt$hashvalue:19800:0:99999:7:::", де поля підписано: "ім'я", "хеш пароля", "дата зміни", "мін. днів", "макс. днів", "попередження". Між блоками зображена стрілка з написом "поле x замінює пароль", що вказує на зв'язок між файлами. Праворуч від нижнього блоку розміщено примітку: "Доступ: root only (chmod 640)".

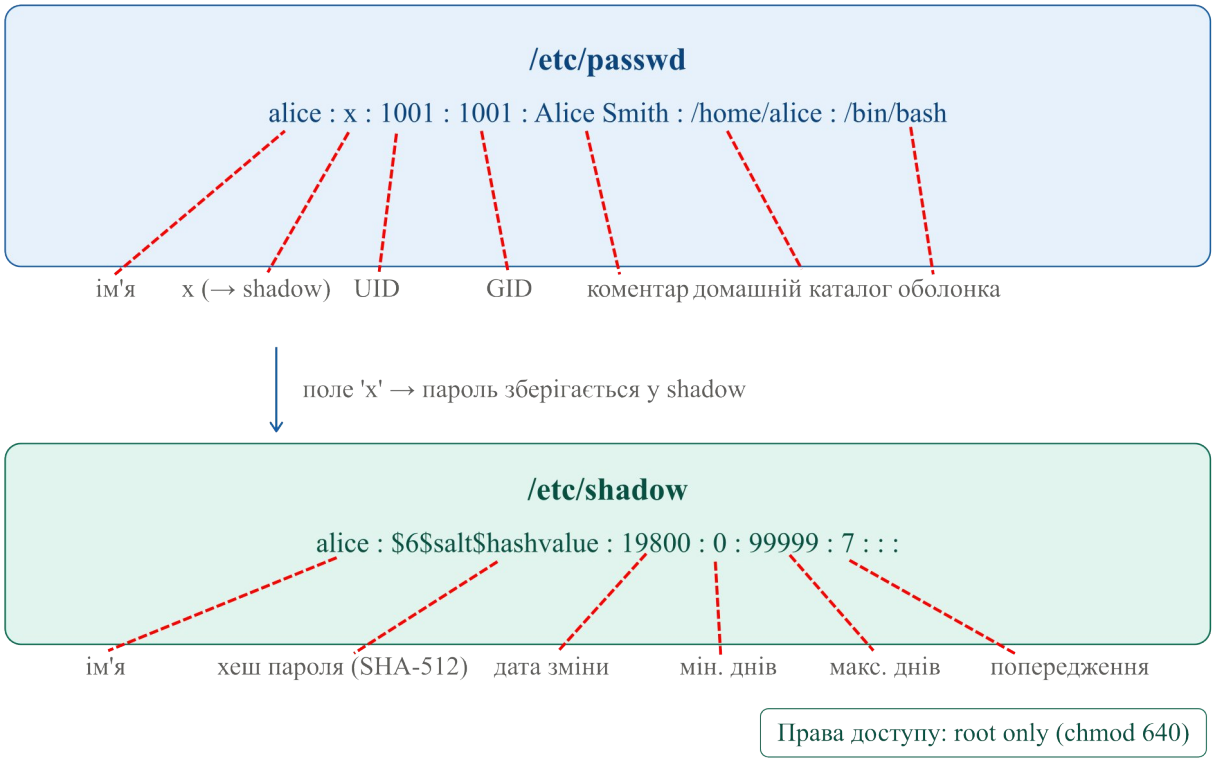

Рисунок 3.1 – Структура файлів /etc/passwd та /etc/shadow

3.2. Моделі контролю доступу: DAC, MAC та RBAC

Контроль доступу є центральним елементом будь-якої системи захисту інформації. Під контролем доступу розуміють сукупність механізмів, що регулюють, які суб'єкти (користувачі, процеси, програми) можуть виконувати які операції (читання, запис, виконання, видалення) над якими об'єктами (файлами, базами даних, мережевими ресурсами). Протягом десятиліть розвитку інформаційної безпеки було розроблено декілька принципово різних підходів до реалізації контролю доступу, кожен з яких ґрунтується на відмінних концепціях управління правами. Три основних моделі — дискреційна, обов'язкова та рольова — сьогодні використовуються як окремо, так і в комбінації, залежно від вимог конкретної системи та рівня чутливості оброблюваних даних.

3.2.1. Дискреційна модель контролю доступу (DAC)

Дискреційна модель контролю доступу (Discretionary Access Control, DAC) — це найбільш інтуїтивна та широко поширена модель, в якій власник ресурсу самостійно вирішує, кому і які права надати. Слово «дискреційна» (від англ. discretion — розсуд) відображає суть моделі: рішення про доступ залишається на розсуд власника об'єкта. Класичним прикладом DAC є система дозволів у UNIX/Linux: кожен файл має власника (user), групу (group) і решту користувачів (others), для кожної з яких встановлюються окремі права на читання (r), запис (w) та виконання (x). Якщо ви створили файл report.txt у

Linux Mint, ви автоматично стали його власником і маєте повне право змінити дозволи — дозволити чи заборонити доступ іншим користувачам командою chmod.

Реалізація DAC у Linux ґрунтується на трирівневій структурі дозволів. Команда ls -l може показати рядок такого вигляду: -rw-r--r-- 1 alice staff 1024 Mar 10 report.txt. Тут перші три символи після типу файлу (rw-) означають права власника: читання і запис, але не виконання. Наступні три (r--) — права групи staff: лише читання. Останні три (r--) — права решти користувачів: також лише читання. Практичний кейс: щоб організувати спільну роботу команди над проектом у директорії /srv/project, адміністратор може створити групу project_team, додати до неї всіх учасників командою usermod -aG project_team alice, призначити директорію групі командою chown :project_team /srv/project та встановити права 2770 (SGID-біт забезпечує успадкування групи новими файлами). Така конфігурація дозволяє всім учасникам групи читати та змінювати файли проекту, тоді як сторонні користувачі не матимуть жодного доступу.

Додатково Linux підтримує більш гнучкий механізм — розширені списки контролю доступу (ACL, Access Control Lists), що дозволяють задавати права для конкретних користувачів або груп незалежно від базової структури. Переглянути та змінити ACL можна командами getfacl і setfacl. Наприклад, команда setfacl -m u:bob:r-- report.txt надає користувачу bob право лише на читання файлу report.txt, навіть якщо він не є власником і не входить до групи файлу. Це розширення усуває основне обмеження стандартної UNIX-моделі, де не можна задати різні права для двох різних користувачів, що не є власником файлу. Головна перевага DAC — простота і гнучкість, проте ця ж гнучкість є і головним недоліком: ніщо не заважає власнику файлу випадково або навмисно надати надлишкові права, зокрема дозволити доступ шкідливому програмному забезпеченню.

3.2.2. Обов'язкова модель контролю доступу (MAC)

Обов'язкова модель контролю доступу (Mandatory Access Control, MAC) кардинально відрізняється від DAC тим, що рішення про доступ приймає не власник ресурсу, а система відповідно до централізовано встановленої політики безпеки. Жоден користувач, навіть власник файлу, не може надати іншому суб'єкту права, яких немає у нього самого або які суперечать системній політиці. В основі MAC лежить концепція міток безпеки (security labels): кожен об'єкт і кожен суб'єкт отримують мітку, що визначає рівень секретності та категорії доступу. Класичні приклади рівнів: несекретно (Unclassified), для службового користування (Confidential), таємно (Secret), цілком таємно (Top Secret). Суб'єкт може читати лише об'єкти з рівнем, що не перевищує його власний, і записувати лише на об'єкти з рівнем, не нижчим за свій — цей принцип відомий як модель Белла–ЛаПадула і спрямований на запобігання витоку секретних даних до менш захищених рівнів.

У Linux MAC реалізується через підсистему LSM (Linux Security Modules) та конкретні її реалізації — SELinux (Security-Enhanced Linux) і AppArmor. В Ubuntu та Linux Mint за замовчуванням активний AppArmor — більш проста у налаштуванні система MAC. AppArmor призначає кожній програмі профіль, що описує, до яких файлів і системних викликів вона має доступ, незалежно від того, від якого користувача вона запущена. Переглянути активні профілі AppArmor можна командою sudo aa-status. Завдяки MAC навіть якщо зловмисник знайде вразливість у браузері і виконає довільний код, цей код залишається обмеженим профілем AppArmor і не зможе завдати шкоди решті системи. Реальний профіль AppArmor для Firefox виглядає наступним чином: він дозволяє читати файли у /home/*/Downloads/, звертатися до мережі через TCP, але явно забороняє читати /etc/shadow та інші системні файли.

SELinux, що використовується за замовчуванням у Red Hat/Fedora/CentOS, реалізує більш складну і потужну модель MAC із контекстами безпеки. Кожен файл, процес і мережевий порт у SELinux отримує контекст безпеки у форматі user:role:type:level, і правила доступу визначаються через зіставлення цих контекстів. SELinux може працювати у трьох режимах: enforcing (активне блокування порушень), permissive (лише журналювання без блокування, корисно для налагодження) і disabled (вимкнений). MAC-системи значно підвищують стійкість до атак типу privilege escalation та обмежують наслідки компрометації окремих компонентів, але вимагають ретельного налаштування та значно ускладнюють адміністрування порівняно з DAC.

3.2.3. Рольова модель контролю доступу (RBAC)

Рольова модель контролю доступу (Role-Based Access Control, RBAC) — це підхід, при якому права доступу призначаються не безпосередньо користувачам, а ролям, а користувачі отримують права через призначення їм відповідних ролей. Роль являє собою абстракцію, що об'єднує набір дозволів, пов'язаних із певною функцією або посадою: наприклад, «адміністратор бази даних», «оператор резервного копіювання», «аудитор» або «розробник». Такий підхід відображає реальну організаційну структуру підприємства: кожен співробітник виконує певну функцію, і саме від функції залежать необхідні права. RBAC є стандартом де-факто для корпоративних систем і широко використовується у базах даних, хмарних платформах та корпоративних застосунках.

Основна перевага RBAC полягає у значному спрощенні адміністрування прав доступу. Якщо у компанії є 100 бухгалтерів, і всім їм потрібен однаковий набір прав, то замість того щоб призначати права кожному окремо (що займе час і неминуче призведе до помилок), адміністратор один раз налаштовує роль «бухгалтер», а потім просто призначає цю роль 100 користувачам. Коли з'являється новий бухгалтер, достатньо призначити йому роль; коли бухгалтер іде — відкликати роль. Якщо необхідно розширити права всіх бухгалтерів, достатньо змінити одну роль. Таке управління є не лише зручнішим, але й значно безпечнішим, оскільки зменшує ймовірність помилок і спрощує аудит.

Стандарт NIST RBAC (NIST SP 800-53) виділяє чотири компоненти моделі: користувачі, ролі, дозволи та операції.

Порівняльну характеристику трьох основних моделей контролю доступу за ключовими параметрами наведено в таблиці 3.2.

Таблиця 3.2 – Порівняльна характеристика моделей контролю доступу

Параметр	**DAC**	**MAC**	**RBAC**
Хто вирішує доступ	Власник ресурсу	Система (централізована політика)	Адміністратор (ролі)
Гнучкість	Висока	Низька	Середня
Складність адміністрування	Низька	Висока	Середня
Захист від insider threats	Слабкий	Сильний	Середній
Реалізація у Linux	chmod, ACL	AppArmor, SELinux	sudo, групи, СУБД
Типове застосування	ОС загального призначення	Держ. та військові системи	Корпоративні ІС, СУБД

Взаємозв'язок між трьома моделями та логіку вибору між ними ілюструє рисунок 3.2. На рисунку зображена вертикальна блок-схема прийняття рішення (дерево рішень). На вершині розміщено прямокутник з написом «Вибір моделі контролю доступу». Від нього вниз відходить ромб із запитанням: «Є вимоги регуляторів або обробляються секретні дані?». Гілка «Так» праворуч веде до блоку «MAC (AppArmor / SELinux)». Гілка «Ні» продовжується вниз до наступного ромба: «Є чітка організаційна структура ролей?». Гілка «Так» праворуч веде до блоку «RBAC (групи, ролі в СУБД)». Гілка «Ні» продовжується вниз до блоку «DAC (chmod, ACL)». Всі три кінцеві блоки пофарбовані у різні кольори: MAC — червоний, RBAC — синій, DAC — зелений. Праворуч від кожного кінцевого блоку розміщено невелику примітку: для MAC — «Найвищий рівень безпеки, складне адміністрування», для RBAC — «Баланс безпеки і зручності», для DAC — «Максимальна гнучкість, відповідальність на власнику».

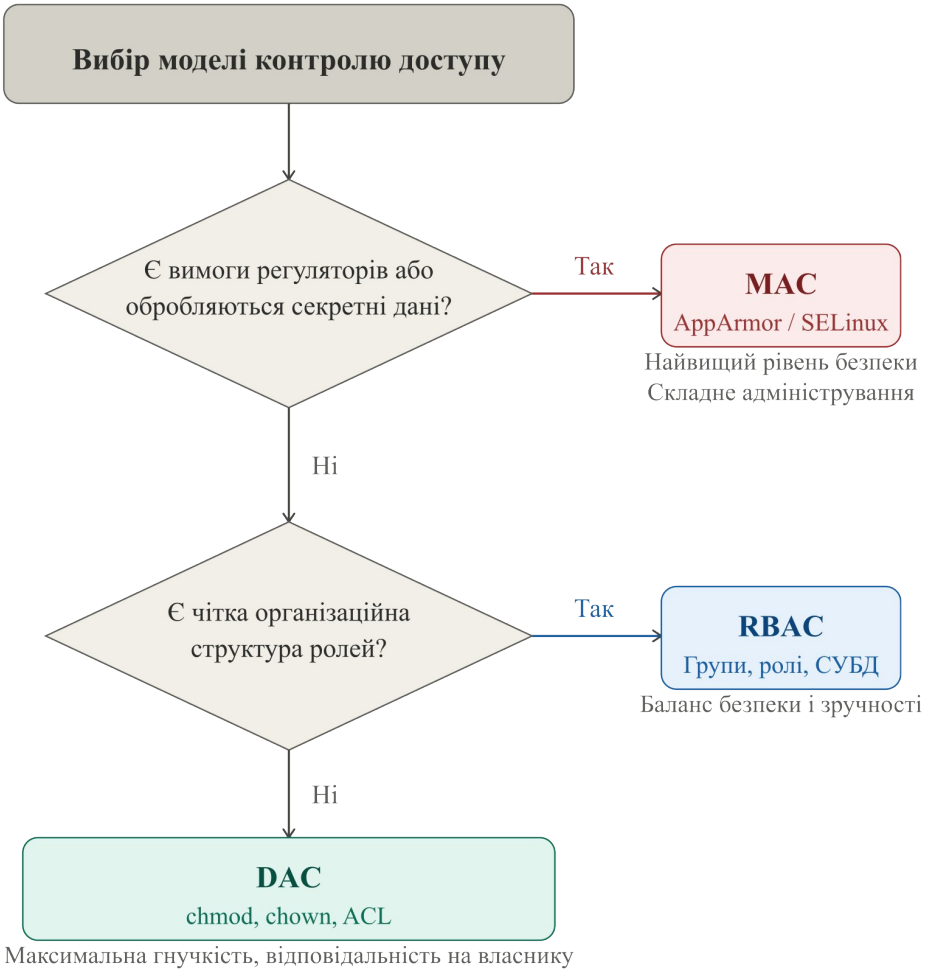

Рисунок 3.2 – Схема вибору моделі контролю доступу

На практиці сучасні системи часто поєднують декілька моделей. Наприклад, у корпоративному середовищі може одночасно використовуватися RBAC для управління правами на рівні застосунку, DAC для налаштування прав доступу до конкретних файлів і MAC у формі AppArmor для обмеження дій системних процесів. Така комбінація дозволяє використати переваги кожної моделі та компенсувати їх недоліки. Розуміння того, яка модель або комбінація моделей є найбільш придатною для конкретної ситуації, є важливою компетенцією фахівця з кібербезпеки, яка формується через практичний досвід і знання нормативних вимог до конкретних класів інформаційних систем.

3.3. Управління обліковими записами, політики паролів та багатофакторна автентифікація

Облікові записи є первинним механізмом ідентифікації суб'єктів у системі, і тому управління ними — один із найкритичніших аспектів забезпечення безпеки. Поняття «управління обліковими записами» охоплює весь їхній життєвий цикл: від створення та налаштування початкових прав доступу до регулярного перегляду привілеїв, тимчасового блокування і, нарешті, видалення після завершення потреби у доступі. Недостатня увага до будь-якого з цих етапів може стати причиною серйозних уразливостей. Якщо

обліковий запис звільненого співробітника не деактивувати вчасно, колишній працівник може зберігати доступ до корпоративних ресурсів, що є грубим порушенням безпеки. Аналогічна проблема виникає із так званими «сирітськими» обліковими записами — записами, власники яких вже не є частиною організації, але самі записи залишаються активними і потенційно доступними для зловмисного використання.

Повний життєвий цикл облікового запису, всі його етапи та типові дії адміністратора на кожному з них показано на рисунку 3.3. На рисунку зображена горизонтальна стрічкова діаграма, що складається з шести прямокутних блоків, з'єднаних стрілками зліва направо. Перший блок підписано «Створення» і містить перелік дій: useradd, встановлення пароля, визначення груп. Другий блок — «Налаштування прав»: призначення ролей, sudo, ACL. Третій блок — «Активне використання»: моніторинг активності, перевірка журналів. Четвертий блок — «Перегляд привілеїв»: регулярний аудит прав, видалення зайвих. П'ятий блок — «Тимчасове блокування»: usermod -L, відкликання ключів. Шостий блок — «Видалення»: userdel -r, архівація даних. Під стрічкою розміщено горизонтальну шкалу часу. Між четвертим і першим блоками зображена дугоподібна стрілка з написом «Повторний перегляд (periodically)», що показує циклічність процесу. Блок «Видалення» обведено червоною рамкою з поміткою «Критичний момент — своєчасне видалення».

Рисунок 3.3 – Життєвий цикл облікового запису користувача

У Linux Mint управління користувачами відбувається через набір стандартних інструментів командного рядка. Команда useradd створює нового користувача, passwd встановлює або змінює пароль, usermod дозволяє змінювати параметри існуючого облікового запису (групи, домашній каталог, термін дії тощо), а userdel видаляє обліковий запис. Інформація про облікові записи зберігається у кількох файлах: /etc/passwd містить базову інформацію (ім'я, UID, GID, домашній каталог, оболонку), тоді як хеші паролів зберігаються у /etc/shadow, доступ до якого мають лише адміністратори. Такий поділ є свідомим рішенням з точки зору безпеки: якщо /etc/passwd доступний усім для читання (це потрібно для коректної роботи деяких програм), то /etc/shadow, що містить чутливі дані, захищений від несанкціонованого читання звичайними користувачами.

Основні команди для управління обліковими записами в Linux Mint разом із практичними прикладами наведено в таблиці 3.3.

Таблиця 3.3 – Команди управління обліковими записами в Linux

Команда	Приклад використання	Призначення
useradd	sudo useradd -m -s /bin/bash alice	Створення користувача з домашнім каталогом
passwd	sudo passwd alice	Встановлення або зміна пароля
usermod	sudo usermod -aG sudo alice	Додавання користувача до групи sudo
usermod -L/U	sudo usermod -L alice	Блокування / розблокування облікового запису
userdel	sudo userdel -r alice	Видалення користувача разом з домашнім каталогом
chage	chage -l alice	Перегляд параметрів терміну дії пароля
id	id alice	Відображення UID, GID та груп користувача
last	last alice	Перегляд журналу входів користувача в систему

Окрему увагу варто приділити обліковим записам служб (service accounts) — спеціальним записам, від імені яких виконуються системні демони та сервіси. Наприклад, веб-сервер Apache або Nginx виконується від імені власного системного користувача (www-data у Debian/Ubuntu), а не від імені root. Це є важливим заходом безпеки: якщо зловмисник знайде вразливість у веб-сервері та отримає можливість виконувати код, він зможе діяти лише в рамках прав облікового запису www-data, а не отримає повний доступ до системи. Для службових облікових записів рекомендовано встановити оболонку /sbin/nologin або /bin/false — щоб неможливо було увійти в систему інтерактивно — та суворо обмежувати привілеї згідно з принципом найменших привілеїв. Команда для створення системного облікового запису служби: sudo useradd -r -s /sbin/nologin -d /var/lib/myservice myservice.

3.3.1. Політики паролів

Паролі залишаються найпоширенішим механізмом автентифікації, і тому правильна політика паролів є критично важливим елементом безпеки. Слабкі паролі та неправильне управління ними — одна з найчастіших причин успішних атак. Важливо розуміти різні типи атак на паролі, оскільки ефективна політика повинна протидіяти кожному з них.

Основні типи атак на паролі та відповідні методи захисту від них наведено в таблиці 3.4.

Таблиця 3.4 – Типи атак на паролі та методи захисту

Тип атаки	Опис	Метод захисту
Brute Force	Послідовний перебір усіх можливих комбінацій символів	Обмеження кількості спроб, затримки між спробами, довгі паролі
Dictionary Attack	Перебір паролів зі словника поширених паролів	Перевірка нових паролів по базам відомих паролів, складність
Credential Stuffing	Використання пар логін-пароль з попередніх витоків	MFA, перевірка паролів по базах витоків (HaveIBeenPwned)
Phishing	Виманювання паролів через підроблені сайти та листи	MFA, апаратні ключі FIDO2, навчання користувачів
Keylogging	Перехоплення паролів при введенні з клавіатури	MFA, двофакторна автентифікація, антивірусний захист

Сучасні рекомендації щодо паролів суттєво змінилися порівняно з тим, що вчили 10–15 років тому. Актуальні настанови NIST (SP 800-63B), що є міжнародним стандартом де-факто, рекомендують: мінімальна довжина пароля повинна становити щонайменше 8 символів для звичайних користувачів і 15 символів для адміністраторів; паролі не слід примусово змінювати за часовим розкладом, якщо немає підстав вважати, що пароль був скомпрометований — таке примусове оновлення призводить до передбачуваних патернів типу «Пароль2024!» → «Пароль2025!»; необхідно перевіряти нові паролі на присутність у базах відомих витоків. Натомість акцент робиться на довгих парольних фразах (passphrase), наприклад «КіньЛітаєПонадХмарами!», які легко запам'ятати, але важко підібрати через велику ентропію.

Основні параметри налаштування політики паролів у Linux через файли /etc/security/pwquality.conf та /etc/login.defs наведено в таблиці 3.5.

Таблиця 3.5 – Параметри політики паролів у Linux

Параметр	Файл налаштування	Рекомендоване значення	Опис
minlen	/etc/security/pwquality.conf	12	Мінімальна довжина пароля
dcredit	/etc/security/pwquality.conf	-1	Мінімальна кількість цифр (від'ємне = вимога)
ucredit	/etc/security/pwquality.conf	-1	Мінімальна кількість великих літер
ocredit	/etc/security/pwquality.conf	-1	Мінімальна кількість спецсимволів
dictcheck	/etc/security/pwquality.conf	1	Перевірка паролю по словнику (1=увімкнено)
PASS_MAX_DAYS	/etc/login.defs	90	Максимальний термін дії пароля в днях
PASS_MIN_DAYS	/etc/login.defs	7	Мінімальний термін між змінами пароля
PASS_WARN_AGE	/etc/login.defs	14	Попередження про закінчення пароля за N днів

3.3.2. Багатофакторна автентифікація

Навіть найсильніший пароль не є абсолютним захистом: він може бути викрадений через фішинг, перехоплений при передачі або отриманий у результаті злому бази даних сервісу. Рішенням цієї проблеми є багатофакторна автентифікація (Multi-Factor Authentication, MFA) — вимога підтвердити особу за допомогою двох або більше незалежних факторів із різних категорій. Логіка проста: навіть якщо зловмисник отримав пароль, він не зможе автентифікуватися без фізичного доступу до другого фактору. Дослідження Google та Microsoft показують, що MFA блокує понад 99% автоматизованих атак на облікові записи. Саме тому MFA є обов'язковою вимогою для більшості стандартів безпеки — PCI DSS, HIPAA, ISO 27001.

Порівняльну характеристику трьох категорій факторів автентифікації за основними критеріями наведено в таблиці 3.6.

Таблиця 3.6 – Порівняння категорій факторів автентифікації

Категорія	Приклади	Переваги	Недоліки	Рівень безпеки
Знання (щось знаєш)	Пароль, PIN-код, секретне питання	Простота, не потрібен пристрій	Вразливий до крадіжки, фішингу, підбору	Низький–середній
Володіння (щось маєш)	TOTP-токен, смарт-картка, телефон	Фізичний контроль над фактором	Може бути втрачений або вкрадений	Середній–високий
Біометрія (хто ти є)	Відбиток пальця, обличчя, голос	Унікальність, не потрібно пам'ятати	Неможливо змінити при компрометації	Високий

Процес автентифікації з увімкненою багатофакторною автентифікацією на прикладі SSH-входу на сервер Linux наочно зображено на рисунку 3.4. На рисунку показана вертикальна покрокова схема з шести блоків, з'єднаних стрілками зверху вниз. Перший блок підписано «Крок 1: Ідентифікація» і містить напис "SSH-клієнт надсилає ім'я користувача". Другий блок — «Крок 2: Перший фактор (знання)»: "Сервер запитує пароль; користувач вводить пароль". Від другого блоку відходить гілка ліворуч з написом «Пароль невірний» до блоку «Відмова у доступі» (червоний). Третій блок — «Крок 3: Перевірка PAM»: "PAM перевіряє хеш пароля у /etc/shadow". Четвертий блок — «Крок 4: Другий фактор (володіння)»: "Сервер запитує TOTP-код; користувач відкриває Google Authenticator і вводить 6-значний код". Від четвертого блоку відходить гілка ліворуч «Код невірний або протермінований» до блоку «Відмова у доступі». П'ятий блок — «Крок 5: Перевірка TOTP»: "PAM порівнює код з обчисленим значенням на основі секрету та поточного часу". Шостий блок — «Крок 6: Доступ надано» (зелений): "SSH-сеанс встановлено, журналюється у /var/log/auth.log". Праворуч від схеми розміщено примітку: "Кожен фактор — окремий бар'єр; зловмисник повинен подолати обидва".

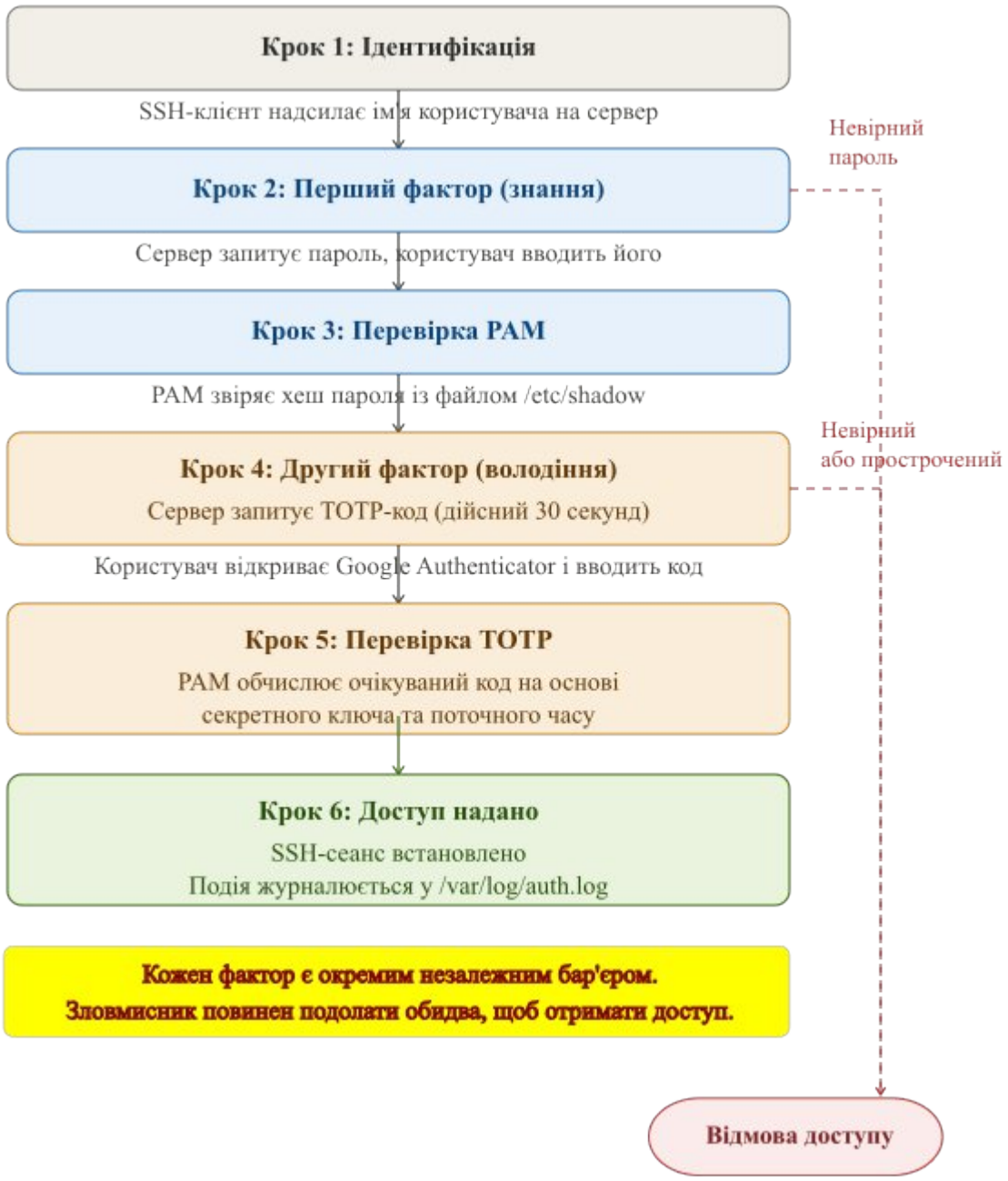

Рисунок 3.4 – Схема процесу багатофакторної автентифікації при SSH-вході

Найпоширенішим видом другого фактору є одноразові паролі, що базуються на часі, — TOTP (Time-based One-Time Password). Алгоритм TOTP, стандартизований у RFC 6238, генерує 6-значний код кожні 30 секунд на основі поточного часу та секретного ключа, що зберігається у застосунку-автентифікаторі (Google Authenticator, Authy або FreeOTP). Для SSH у Linux Mint TOTP підключається через модуль PAM libpam-google-authenticator: після команди sudo apt install libpam-google-authenticator та запуску google-authenticator від імені кожного користувача до файлу /etc/pam.d/sshd додається рядок "auth required pam_google_authenticator.so", а у файлі /etc/ssh/sshd_config встановлюється "AuthenticationMethods publickey,keyboard-interactive". Після цього навіть наявність SSH-ключа без TOTP-коду не дозволить увійти на сервер, що суттєво підвищує безпеку віддаленого доступу.

Апаратні ключі безпеки (hardware security keys), такі як YubiKey або Token2, реалізують стандарт FIDO2/WebAuthn і є найнадійнішим варіантом другого фактору. На відміну від TOTP-кодів, які можуть бути перехоплені під час фішингу в реальному часі, апаратні ключі перевіряють справжність сайту або сервісу і не надають відповідь, якщо домен не збігається з тим, для якого ключ був зареєстрований. Таким чином, навіть якщо користувача обдурять і він зайде на підроблений сайт, апаратний ключ відмовить в автентифікації, ефективно нейтралізуючи навіть найдосконаліший фішинг. У корпоративних середовищах, де захист від цілеспрямованих атак є критичним, апаратні ключі стають стандартом де-факто для захисту привілейованих облікових записів.

3.4. Принцип найменших привілеїв та управління привілейованим доступом

Принцип найменших привілеїв (Principle of Least Privilege, PoLP) є одним із фундаментальних принципів інформаційної безпеки, вперше сформульованих у 1975 році дослідниками Солтцером і Шредером у роботі «The Protection of Information in Computer Systems». Суть принципу проста: кожен суб'єкт (користувач, процес, програма) повинен мати лише той мінімальний набір привілеїв, який є абсолютно необхідним для виконання його законних функцій, і нічого більше. На перший погляд це звучить очевидно, проте на практиці цей принцип порушується надзвичайно часто — через зручність, лінощі або нерозуміння ризиків. Типові прояви порушення принципу: робота в системі постійно від імені адміністратора, надання розробникам прямого доступу до виробничих баз даних, запуск веб-серверів від імені root, надання всім співробітникам прав редагування на загальних файлових ресурсах.

Значення принципу найменших привілеїв важко переоцінити з точки зору обмеження наслідків інцидентів безпеки. Уявіть дві ситуації: в першій зловмисник компрометує обліковий запис бухгалтера, що має права адміністратора (тому що «так зручніше»), — і отримує повний контроль над усією системою. В другій ситуації зловмисник компрометує обліковий запис того самого бухгалтера, що має лише права на доступ до бухгалтерського програмного забезпечення та відповідних документів, — і не може вийти за межі цих ресурсів. Принцип найменших привілеїв не запобігає злому, але суттєво обмежує його масштаб і дає команді безпеки більше часу на виявлення і реагування. В термінах моделі CIA це підвищує рівень конфіденційності та цілісності даних, що не стосуються скомпрометованого облікового запису.

В операційних системах Linux реалізація принципу найменших привілеїв починається з фундаментального правила: ніколи не працювати постійно від імені root. Для виконання адміністративних задач використовується механізм sudo (substitute user do), що дозволяє виконувати окремі команди з підвищеними привілеями після підтвердження власного пароля. Конфігурація sudo знаходиться у файлі /etc/sudoers (редагується командою visudo, що запобігає синтаксичним помилкам) та файлах у директорії /etc/sudoers.d/. Важливо, що sudo дозволяє тонко налаштовувати, які саме команди може виконувати

конкретний користувач: наприклад, можна дозволити оператору перезапускати лише службу nginx командою "operator ALL=(ALL) NOPASSWD: /bin/systemctl restart nginx", але не надавати жодних інших адміністративних прав. Це є практичним втіленням принципу найменших привілеїв на рівні системного адміністрування.

Особливу небезпеку становлять вектори підвищення привілеїв (privilege escalation) — шляхи, що дозволяють зловмиснику з непривілейованого облікового запису досягти прав root. Основні вектори атак та відповідні контрзаходи наведено в таблиці 3.7.

Таблиця 3.7 – Вектори підвищення привілеїв та методи захисту в Linux

Вектор атаки	**Приклад**	**Метод виявлення**	**Захист**
SUID/SGID-файли	Вразливий бінарник з битом SUID	find / -perm -4000 2>/dev/null	Регулярний аудит, видалення зайвих SUID
Некоректний sudo	NOPASSWD: ALL або sudo до редактора	sudo -l (від імені атакованого)	Принцип найменших привілеїв у sudoers
Служби від root	Apache або MySQL без ізоляції	ps aux | grep root	Окремий системний користувач + AppArmor
Вразливості ядра	DirtyPipe (CVE-2022-0847)	uname -r, перевірка CVE	Своєчасне оновлення ядра ОС
Writeable PATH	Шкідливий файл у /tmp в PATH	echo $PATH, перевірка прав	Виключити записувані каталоги з PATH

Регулярний аудит облікових записів і привілеїв є обов'язковою складовою підтримки безпечного стану системи. Практичний чек-лист аудиту облікових записів у Linux з відповідними командами наведено в таблиці 3.8.

Таблиця 3.8 – Чек-лист аудиту безпеки облікових записів у Linux

Перевірка	**Команда**	**Очікуваний безпечний результат**
Користувачі з UID 0 (root)	awk -F: '$3==0{print $1}' /etc/passwd	Лише "root"; інших не повинно бути
Облікові записи без пароля	sudo awk -F: '($2==""){print $1}' /etc/shadow	Порожній список (всі мають паролі)
Неактивні облікові записи	sudo lastlog | grep "Never"	Перевірити та заблокувати зайві
SUID/SGID-файли	find / -perm -4000 2>/dev/null	Лише стандартні системні утиліти
Список користувачів sudo	getent group sudo	Лише потрібні адміністратори
Терміни дії паролів	sudo chage -l <username>	Перевірити PASS_MAX_DAYS ≤ 90
Налаштування sudo	sudo cat /etc/sudoers	Відсутність NOPASSWD: ALL без потреби
Журнал sudo-команд	grep sudo /var/log/auth.log	Аналіз на підозрілі дії

Концепція управління привілейованим доступом (Privileged Access Management, PAM) у корпоративному середовищі виходить далеко за рамки простих механізмів sudo. Корпоративні PAM-рішення (CyberArk, HashiCorp Vault, BeyondTrust) надають централізований контроль, аудит і управління привілейованими обліковими записами в масштабах всього підприємства. Ключові функції таких систем охоплюють: сховища паролів привілейованих облікових записів (password vault) із автоматичною ротацією, сесійне управління із записом усіх дій привілейованих користувачів (session recording), видачу тимчасових (just-in-time) привілеїв лише на час виконання конкретного завдання, а також детальний аудит і сповіщення при підозрілих діях. Такий підхід не лише зменшує ризик зловживань, але й забезпечує відповідність нормативним вимогам, що часто вимагають документування всіх дій з привілейованими обліковими записами.

Ще одним важливим аспектом принципу найменших привілеїв є сегрегація обов'язків (Separation of Duties, SoD) — принцип, за яким критично важливі операції мають вимагати участі щонайменше двох різних осіб. Класичний приклад з фінансової сфери: одна людина ініціює платіж, а інша — авторизує його. В контексті IT-безпеки це може означати, що розробник може писати код, але не може самостійно розгортати його у виробничому середовищі; адміністратор може налаштовувати систему, але не може переглядати чутливі дані; аудитор може читати журнали подій, але не може їх видаляти. SoD ускладнює як зловмисні дії, так і випадкові помилки з серйозними наслідками, і є рекомендованою практикою у всіх серйозних стандартах безпеки, включаючи ISO/IEC 27001 та PCI DSS.

Тема 4. Аудит безпеки та оцінка стану захищеності операційних систем

4.1 Поняття аудиту безпеки та його місце в житті системи захисту

Будь-яка інформаційна система з часом змінюється: встановлюються нові програми, оновлюються конфігурації, з'являються нові користувачі та сервіси, а дослідники постійно виявляють раніше невідомі вразливості. Усі ці зміни можуть непомітно знижувати рівень захищеності навіть тоді, коли на початку систему було налаштовано ретельно. Саме тому одноразове налаштування безпеки є принципово недостатнім — потрібен систематичний процес перевірки поточного стану захищеності. Цей процес і називається аудитом безпеки, і він є одним із ключових інструментів фахівця з кібербезпеки.

Аудит безпеки (security audit) — це систематична, незалежна та задокументована перевірка стану захищеності інформаційної системи, яка охоплює оцінювання конфігурацій, механізмів контролю доступу, програмного забезпечення, журналів подій та відповідності встановленим вимогам і стандартам. Ключовою характеристикою аудиту є його систематичність: перевірка проводиться за чітко визначеною методикою, результати фіксуються та порівнюються з попередніми перевірками або з встановленими еталонними значеннями. Без цієї систематичності будь-яка перевірка перетворюється на хаотичне та неповне сканування.

Важливо з самого початку розрізняти аудит безпеки та тестування на проникнення (penetration testing), оскільки ці два підходи нерідко плутають. Аудит зосереджений на перевірці відповідності системи певним стандартам і вимогам: аудитор аналізує конфігурації, журнали та документацію, переважно не вживаючи активних атакуючих дій. Тестування на проникнення, навпаки, є активним — фахівець намагається реально проексплуатувати вразливості так, як це зробив би зловмисник. Обидва підходи є цінними й доповнюють один одного, однак у цій темі зосереджуємось саме на аудиті як основі регулярної оцінки захищеності.

Аудит безпеки займає центральне місце в життєвому циклі системи захисту. Цей цикл охоплює такі фази: планування та проєктування захисту, впровадження засобів захисту, операційна фаза, моніторинг та аудит, а також вдосконалення на основі отриманих результатів. Таким чином, аудит є зворотним зв'язком у системі управління безпекою: він показує, наскільки реальний стан системи відповідає запланованому рівню захищеності, і формує підстави для коригуючих дій. Без регулярного аудиту навіть добре спроєктована система безпеки поступово деградує.

Розрізняють кілька видів аудиту залежно від виконавця та мети. Внутрішній аудит проводиться власними фахівцями організації або її підрозділом інформаційної безпеки і може виконуватись досить регулярно — наприклад, щоквартально. Зовнішній аудит здійснює незалежна третя сторона — спеціалізована компанія або сертифікований аудитор, і його результати мають вищу юридичну силу при підтвердженні відповідності стандартам. Аудит на відповідність (compliance audit) перевіряє, чи виконує організація

вимоги конкретного стандарту, наприклад ISO/IEC 27001. Технічний аудит зосереджується безпосередньо на конфігураціях та вразливостях програмного й апаратного забезпечення. Для об'єктивності рекомендується поєднувати обидва типи. У таблиці 4.4 наведено детальне порівняння внутрішнього та зовнішнього аудиту за ключовими критеріями.

Таблиця 4.4 – Порівняння видів аудиту безпеки

Критерій	Внутрішній аудит	Зовнішній аудит
Виконавець	Власні фахівці організації або підрозділ ІБ	Незалежна сертифікована третя сторона
Об'єктивність	Середня (можливий конфлікт інтересів)	Висока (незалежний погляд)
Вартість	Нижча (оплата власного персоналу)	Вища (оплата зовнішніх послуг)
Частота проведення	Регулярно (наприклад, щоквартально)	Раз на рік або за потребою
Глибина перевірки	Висока (фахівці знають систему)	Обмежена відведеним часом
Юридична сила результатів	Нижча	Вища (може підтвердити відповідність стандартам)

Операційна система є центральним елементом захисту всієї інформаційної системи, тому аудит безпеки ОС є базовим та найпоширенішим видом технічного аудиту. ОС контролює доступ до ресурсів, запускає програми, ізолює процеси та веде журнали подій. Якщо ОС налаштована неправильно — з надмірними привілеями, застарілими компонентами або слабкими паролями — це відкриває зловмиснику широкі можливості навіть за наявності інших засобів захисту на рівні мережі чи застосунків. Тому оцінка стану захищеності ОС є першочерговим завданням будь-якого технічного аудиту.

4.2 Етапи проведення аудиту безпеки операційних систем

Аудит безпеки — це не одна дія, а послідовний багатоетапний процес. Порушення цієї послідовності або пропуск будь-якого з етапів знижує якість і достовірність результатів. У загальному вигляді процес охоплює: підготовку та визначення обсягу перевірки, збір інформації про систему, технічний аналіз конфігурацій та вразливостей, оцінку ризиків, формування звіту та рекомендацій, і нарешті — впровадження коригуючих дій з наступною верифікацією. Цей процес є циклічним: після верифікації цикл починається знову.

Підготовчий етап визначає межі та цілі аудиту. Аудитор разом із замовником з'ясовує, які системи підлягають перевірці, які ресурси є критичними для захисту, якими стандартами або вимогами слід керуватись і які

обмеження існують (наприклад, неможливість перезавантаження виробничого сервера під час перевірки). На цьому ж етапі узгоджується методологія, призначаються відповідальні особи та підписуються необхідні дозвільні документи. Без чіткого визначення обсягу перевірка ризикує або охопити занадто багато об'єктів і бути поверховою, або пропустити критично важливі компоненти.

Збір інформації — найбільш трудомісткий технічний етап. Він охоплює отримання відомостей про версію ОС та встановлені оновлення, перелік запущених служб і процесів, відкриті мережеві порти та активні з'єднання, облікові записи користувачів та їх привілеї, налаштування прав доступу до файлів і каталогів, параметри конфігурації системних служб, а також вміст і параметри журналів подій. У Linux основні команди для збору інформації охоплюють кілька категорій: аналіз системи, служб, мережевих з'єднань, облікових записів і файлових прав. Для зручності орієнтування у цих командах таблиця 4.5 систематизує основні інструменти збору інформації за відповідними етапами аудиту.

Таблиця 4.5 – Команди Linux для збору інформації під час аудиту

Етап	Мета	Приклади команд
Збір інформації про ОС	Версія ОС, ядро, оновлення	uname -a, hostnamectl, cat /etc/os-release
Аналіз служб	Виявлення активних сервісів	systemctl list-units --type=service, ps aux
Перевірка портів	Виявлення відкритих мережевих портів	ss -tulnp, netstat -tulpn
Аналіз користувачів	Перевірка облікових записів і груп	cat /etc/passwd, id, last, who
Перевірка прав доступу	Аналіз файлових дозволів	ls -la /etc, find / -perm -4000 -type f
Аналіз журналів	Виявлення підозрілих подій	journalctl -xe, cat /var/log/auth.log

Технічний аналіз зібраних даних передбачає порівняння реального стану системи з еталонним — з рекомендованими налаштуваннями безпеки. Аудитор перевіряє, чи є застарілі компоненти з відомими вразливостями, чи немає зайвих відкритих служб і портів, чи відповідають права доступу принципу найменших привілеїв, чи правильно налаштовані журнали аудиту. Особливу увагу приділяють так званим «слабким місцям за замовчуванням» — налаштуванням, які ОС встановлює автоматично й які часто є надмірно дозволяючими. Наприклад, деякі конфігурації за замовчуванням дозволяють вхід до системи для облікового запису root через мережу, або запускають служби, що не потрібні для основного призначення системи.

Серед типових знахідок аудиту безпеки Linux-систем можна виділити кілька груп. До найнебезпечніших належать: активна служба Telnet, що передає дані у відкритому вигляді; облікові записи з порожніми або тривіальними паролями; дозвіл на вхід під обліковим записом root через SSH; відсутній або вимкнений фаєрвол; та SUID-біти на файлах, що не потребують підвищених привілеїв. Кожна з цих знахідок є самостійним вектором атаки. Таблиця 4.6 наводить типові знахідки аудиту, їх рівень ризику та конкретні команди для усунення.

Таблиця 4.6 – Типові знахідки аудиту та способи їх усунення

Знахідка	Рівень ризику	Команда / спосіб виправлення
Служба Telnet активна	Критичний	sudo systemctl disable --now telnet.socket
Обліковий запис із порожнім паролем	Критичний	sudo passwd username (встановити надійний пароль)
SSH дозволяє вхід root	Високий	Встановити PermitRootLogin no у /etc/ssh/sshd_config
Фаєрвол вимкнено або відсутній	Високий	sudo apt install ufw && sudo ufw enable
Застарілі пакети з вразливостями	Середній	sudo apt update && sudo apt upgrade
SUID-файли поза стандартним переліком	Середній	find / -perm -4000 -type f, потім аналіз кожного

Оцінка ризиків перетворює технічні знахідки на управлінську інформацію, яку може сприймати керівництво організації. Для кожної виявленої проблеми оцінюються два параметри: ймовірність її використання зловмисником та потенційний збиток у разі успішної атаки. Добуток цих параметрів дає рівень ризику. Такий підхід дозволяє розставити пріоритети: не всі виявлені вразливості потребують негайного усунення — деякі можна виправити планово або компенсувати іншими засобами захисту. На рисунку 4.1 наведено класичну матрицю ризиків, яка унаочнює залежність загального рівня ризику від ймовірності та впливу загрози.

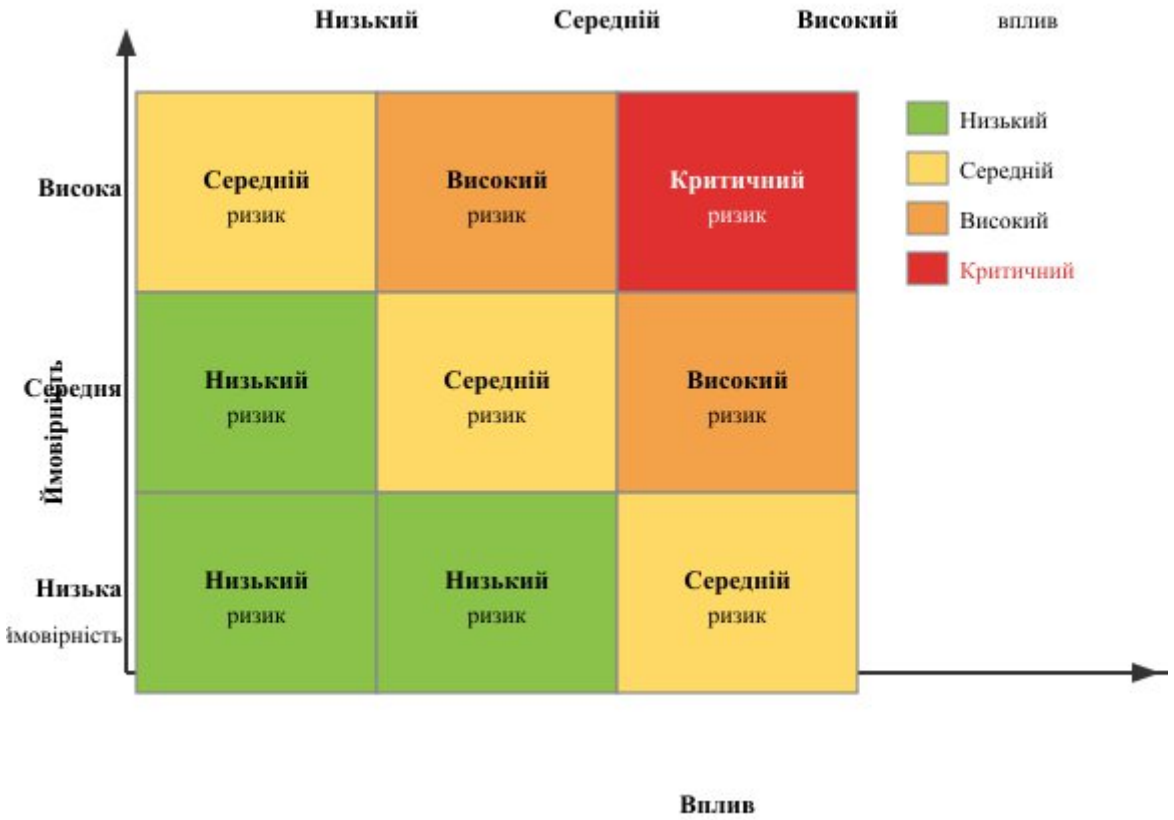

Рисунок 4.1 – Матриця ризиків (ймовірність × вплив)

На рисунку 4.1 зображено матрицю ризиків у вигляді таблиці 3×3. По вертикальній осі розташована «Ймовірність» із трьома рівнями знизу догори: низька, середня, висока. По горизонтальній осі — «Вплив» із рівнями зліва направо: низький, середній, високий. Клітинки матриці забарвлені відповідно до рівня ризику: комбінація «низька ймовірність + низький вплив» — зелений колір (низький ризик); комбінації, де хоча б один параметр є середнім — жовтий колір (середній ризик); комбінації «висока ймовірність + середній або високий вплив» та «середня ймовірність + високий вплив» — помаранчевий і червоний кольори (високий та критичний ризик). Стрілки по краях матриці показують напрям зростання ймовірності та впливу.

Завершальний етап — формування звіту та рекомендацій. Звіт аудиту є офіційним документом, що містить: опис методології, перелік перевірених об'єктів, виявлені вразливості та невідповідності з оцінкою їхньої критичності, а також конкретні рекомендації щодо усунення. Якісний звіт орієнтований на різні аудиторії: технічна частина для адміністраторів містить точні команди та параметри конфігурації, тоді як виконавча частина для керівництва описує ризики у ясних бізнес-термінах без технічного жаргону. Рекомендації мають бути конкретними та здійсненними — вказуючи не лише на проблему, але й на спосіб її вирішення.

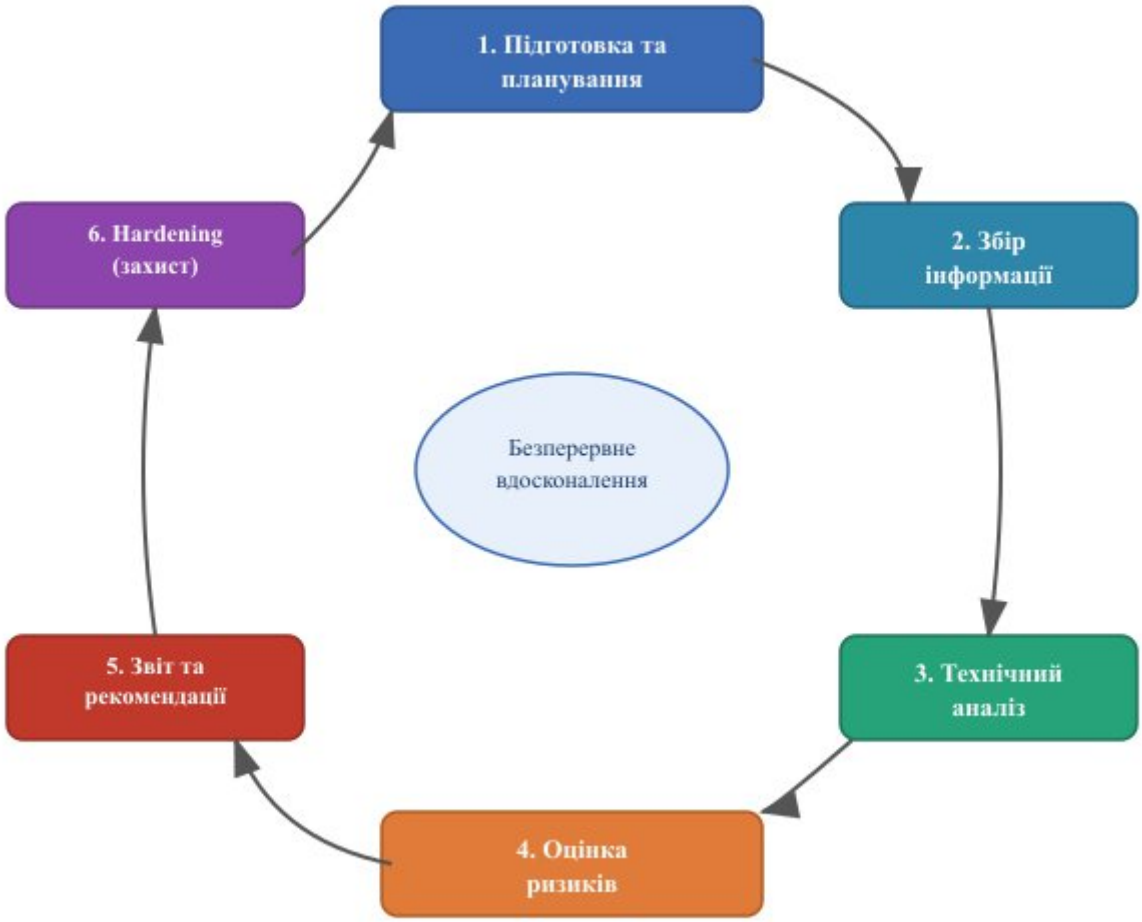

Рисунок 4.2 – Цикл аудиту безпеки операційної системи

4.3 Стандарти, методики та автоматизовані інструменти аудиту

Проведення аудиту безпеки без спирання на загальновизнані стандарти та методики є малоефективним: індивідуальні підходи легко пропускають цілі класи вразливостей, а результати різних аудитів неможливо об'єктивно порівняти між собою. Стандарти виконують роль структурованих контрольних списків, що охоплюють усі аспекти безпеки — від керування обліковими записами до конфігурації мережевих служб. Крім цього, аудит на основі загальновизнаних стандартів дає можливість доводити відповідність вимогам регуляторів, що є обов'язковою умовою для багатьох галузей.

Одним із найбільш практично орієнтованих наборів рекомендацій є CIS Benchmarks, розроблені організацією Center for Internet Security. CIS Benchmarks являють собою детальні посібники з безпечного налаштування конкретних систем: окремі документи для різних версій Linux, для веб-серверів, баз даних, хмарних платформ тощо. Кожен пункт Benchmark описує конкретне налаштування, пояснює його значення для безпеки, вказує, як його перевірити, та наводить команду або конфігурацію для виправлення. CIS Benchmarks розподілені на два профілі: Level 1 (базовий рівень, мінімальний вплив на функціональність) та Level 2 (підвищений рівень, для систем із жорсткими

вимогами безпеки). Для бажаючих дослідити CIS Benchmarks детальніше — повний список доступний на сайті cisecurity.org.

Американський Національний інститут стандартів і технологій публікує серію спеціальних публікацій SP 800, присвячених різним аспектам інформаційної безпеки. Для аудиту безпеки операційних систем ключовими є два документи: NIST SP 800-53 «Security and Privacy Controls for Information Systems and Organizations» — вичерпний каталог засобів контролю безпеки, організованих за сімнадцятьма сімействами, та NIST SP 800-115 «Technical Guide to Information Security Testing and Assessment» — методологія технічного тестування безпеки. Ці документи є основою для аудиту в державних установах США і широко застосовуються у приватному секторі по всьому світу.

У таблиці 4.1 наведено порівняльний огляд основних стандартів та методик аудиту безпеки.

Таблиця 4.1 – Стандарти та методики аудиту безпеки

Стандарт / методика	Організація-розробник	Основне призначення
CIS Benchmarks	Center for Internet Security	Практичні рекомендації з безпечного налаштування ОС, застосунків та мережевих пристроїв
NIST SP 800-53	NIST (США)	Каталог засобів контролю безпеки та конфіденційності для інформаційних систем
NIST SP 800-115	NIST (США)	Технічний посібник з тестування та оцінювання безпеки інформаційних систем
ISO/IEC 27001	ISO / IEC	Вимоги до системи управління інформаційною безпекою (СУІБ)
ISO/IEC 27002	ISO / IEC	Кодекс практики засобів контролю інформаційної безпеки
OWASP ASVS	OWASP Foundation	Стандарт верифікації безпеки застосунків, зокрема веб-сервісів

Автоматизовані інструменти аудиту суттєво прискорюють перевірку та дозволяють охопити сотні параметрів конфігурації за лічені хвилини. Один із найпопулярніших безкоштовних інструментів для Linux — Lynis. Це скриптова програма, яка запускається безпосередньо на аудійованій системі та перевіряє параметри ядра ОС, конфігурацію служб SSH, Apache, MySQL, параметри мережі, стан оновлень, права доступу до критичних файлів тощо. За результатами перевірки Lynis формує звіт із вказівкою виявлених недоліків і рекомендаціями. Для запуску аудиту достатньо виконати команду sudo lynis audit system — і через кілька хвилин буде готовий детальний звіт. Нижче наведено приклад характерного виведення Lynis:

```
=== Результати сканування Lynis ===
[+] Boot and services
  - Service SSH is enabled [OK]
  - Service Telnet is disabled [OK]
```

```
[!] Users and groups
  - Empty password detected for user 'guest' [WARNING]

[+] SSH support
  - SSH protocol version 2 enforced [OK]
  - Root login permitted [WARNING]

[===] Scan completed ===
  Hardening index: 72/100
  Tests performed: 247
  Warnings:        12
  Suggestions:      8
```

Наведений приклад ілюструє ключові характеристики звіту Lynis. Знаки «[OK]» та «[WARNING]» одразу вказують на проблемні місця. Загальний «Hardening index» дає інтегральну числову оцінку стану захищеності системи — чим він вищий, тим краще налаштована система. Перелік попереджень та пропозицій є основою для планування коригуючих дій. Такий формат звіту є зрозумілим навіть для початківця, що робить Lynis ідеальним інструментом для навчальних цілей.

Інший потужний інструмент — OpenSCAP, що реалізує специфікацію SCAP (Security Content Automation Protocol), розроблену NIST. OpenSCAP дозволяє завантажувати готові профілі безпеки (наприклад, профіль CIS або STIG для Linux) та автоматично перевіряти систему на їх відповідність. Результатом є детальний HTML-звіт із результатами перевірки кожного правила (passed/failed/not applicable) та посиланнями на відповідні стандарти. У таблиці 4.2 подано порівняння найпоширеніших автоматизованих інструментів аудиту.

Таблиця 4.2 – Порівняння автоматизованих інструментів аудиту безпеки

Інструмент	Тип / ліцензія	Платформа	Основні можливості
Lynis	Відкрите ПЗ (GPL)	Linux, macOS	Аудит конфігурації ОС, пошук вразливостей, перевірка відповідності
OpenSCAP	Відкрите ПЗ (LGPL)	Linux	Перевірка відповідності профілям SCAP, генерація HTML/XML звітів
Nessus Essentials	Безкоштовна (обмежена)	Linux, Win, macOS	Сканування вразливостей, перевірка конфігурацій, детальна звітність
OpenVAS / Greenbone	Відкрите ПЗ (GPL)	Linux	Повнофункціональний сканер вразливостей із базою NVT
CIS-CAT Lite	Безкоштовна (обмежена)	Linux, Win	Автоматизована перевірка відповідності CIS Benchmarks

Попри всі переваги, автоматизовані інструменти мають суттєве обмеження: вони виявляють лише відомі типи проблем, описані у їхніх базах

перевірок. Вони не здатні знайти логічні помилки у бізнес-логіці застосунків, нестандартні вектори атак або проблеми, специфічні для конкретного середовища. Тому результати автоматизованих перевірок завжди мають доповнюватись ручним аналізом досвідченого аудитора — особливо для критичних систем.

Система CVSS (Common Vulnerability Scoring System) є стандартним способом кількісного вираження серйозності вразливостей. Оцінка може приймати значення від 0.0 до 10.0. Вона складається з трьох груп метрик. Базові метрики описують фундаментальні характеристики: складність експлуатації, необхідність автентифікації, вплив на конфіденційність, цілісність і доступність. Тимчасові метрики враховують поточний стан: чи є публічний експлойт, чи існує патч. Метрики середовища коригують оцінку відповідно до специфіки конкретної організації. Шкала оцінок та рекомендовані дії наведені у таблиці 4.3.

Таблиця 4.3 – Рівні критичності вразливостей за CVSS 3.1

Рівень критичності	Діапазон CVSS	Рекомендовані дії
Критичний (Critical)	9.0 – 10.0	Негайне усунення або застосування тимчасових компенсуючих заходів
Високий (High)	7.0 – 8.9	Усунення у пріоритетному порядку в найближчому плановому циклі оновлення
Середній (Medium)	4.0 – 6.9	Планове усунення з урахуванням контексту та наявних компенсуючих заходів
Низький (Low)	0.1 – 3.9	Документування та усунення за наявності ресурсів без жорсткого дедлайну
Відсутній (None)	0.0	Не потребує коригуючих дій; факт реєструється для повноти обліку

Рисунок 4.3 – Шкала оцінок CVSS з кольоровим кодуванням

На рисунку 4.3 зображена горизонтальна числова шкала від 0.0 до 10.0, поділена на чотири кольорові сегменти. Перший сегмент від 0.1 до 3.9 позначений зеленим кольором і підписаний «Низький (Low)». Другий сегмент від 4.0 до 6.9 — жовтим, «Середній (Medium)». Третій від 7.0 до 8.9 — помаранчевим, «Високий (High)». Четвертий від 9.0 до 10.0 — червоним, «Критичний (Critical)». Між сегментами вертикальними рисками позначені

порогові значення 3.9, 6.9 та 8.9. Над шкалою розміщені числові мітки 0, 4, 7, 9, 10. Значення 0.0 окремо позначено як «None» — відсутність вразливості.

Для практичного застосування CVSS важливо розуміти, що базова оцінка не завжди відображає реальний ризик для конкретної організації. Наприклад, вразливість із оцінкою 9.0 у сервісі, що не використовується в організації та ізольований від мережі, несе значно менший реальний ризик, ніж вразливість із оцінкою 6.5 у критичному публічному веб-сервісі. Тому CVSS рекомендується використовувати разом із контекстним аналізом, а не як єдиний критерій прийняття рішень. У таблиці 4.7 наведено приклади реальних вразливостей у Linux-системах, що мали значний вплив на безпеку.

Таблиця 4.7 – Приклади реальних вразливостей Linux (CVE)

CVE	Опис	Оцінка CVSS	Вразливі версії
CVE-2021-4034 (PwnKit)	Локальне підвищення привілеїв у компоненті polkit	7.8	glibc (2011–2021)
CVE-2021-3156 (Baron Samedit)	Переповнення буфера в утиліті sudo	7.8	sudo < 1.9.5p2
CVE-2022-0847 (Dirty Pipe)	Перезапис файлів, доступних тільки для читання	7.8	ядро Linux 5.8+
CVE-2022-3786 / 3602 (SpookySSL)	Переповнення буфера в OpenSSL 3.x при перевірці сертифікатів	7.5	OpenSSL 3.0 – 3.0.6

Наведені у таблиці 4.7 вразливості ілюструють важливу закономірність: навіть добре відомі і широко використовувані компоненти Linux, такі як sudo, ядро або OpenSSL, можуть містити серйозні вразливості, що дозволяють зловмиснику підвищити привілеї або отримати несанкціонований доступ до даних. Всі три перші вразливості мають оцінку CVSS 7.8 (високий рівень) і при цьому вимагають лише локального доступу для експлуатації — тобто вони є особливо небезпечними у середовищах із багатьма користувачами або за умови, що зловмисник вже зміг якимось чином опинитись у системі. Регулярний аудит і своєчасне оновлення системи є єдиним надійним захистом від таких загроз.

4.4 Інтерпретація результатів та заходи з підвищення рівня захищеності

Отримані в ході аудиту дані мають цінність лише тоді, коли вони правильно інтерпретовані та перетворені на конкретні дії. Технічний звіт із сотнями виявлених проблем без чіткого розуміння їхньої пріоритетності на практиці виявляється некорисним: адміністратор просто не знає, з чого починати. Тому ключовим завданням після завершення перевірки є класифікація знахідок за критичністю, групування пов'язаних проблем та визначення чіткого плану усунення. Без цих кроків аудит залишається лише фіксацією проблем, а не інструментом їх вирішення.

Першим кроком інтерпретації є розподіл знахідок на категорії: критичні вразливості, що потребують негайного усунення; важливі невідповідності для виправлення найближчим часом; рекомендовані покращення, що суттєво підвищать рівень безпеки; та інформаційні спостереження без необхідності конкретних дій. Така класифікація дозволяє раціонально розподілити обмежені ресурси команди. Наступний крок — встановлення кореневих причин: нерідко кілька різних симптомів мають спільну першопричину. Наприклад, надмірні права доступу до конфігураційних файлів, відсутність журналювання та нестандартні налаштування служб можуть бути наслідком відсутності єдиної формалізованої політики безпеки.

Після класифікації та аналізу першопричин формується план коригуючих дій із трьома горизонтами: короткостроковий (негайне усунення критичних знахідок протягом 24–72 годин), середньостроковий (виправлення важливих невідповідностей протягом тижня-місяця) та довгостроковий (системні покращення, що потребують значних ресурсів або архітектурних змін). На рисунку 4.4 наведено схему опрацювання результатів аудиту та формування плану підвищення захищеності.

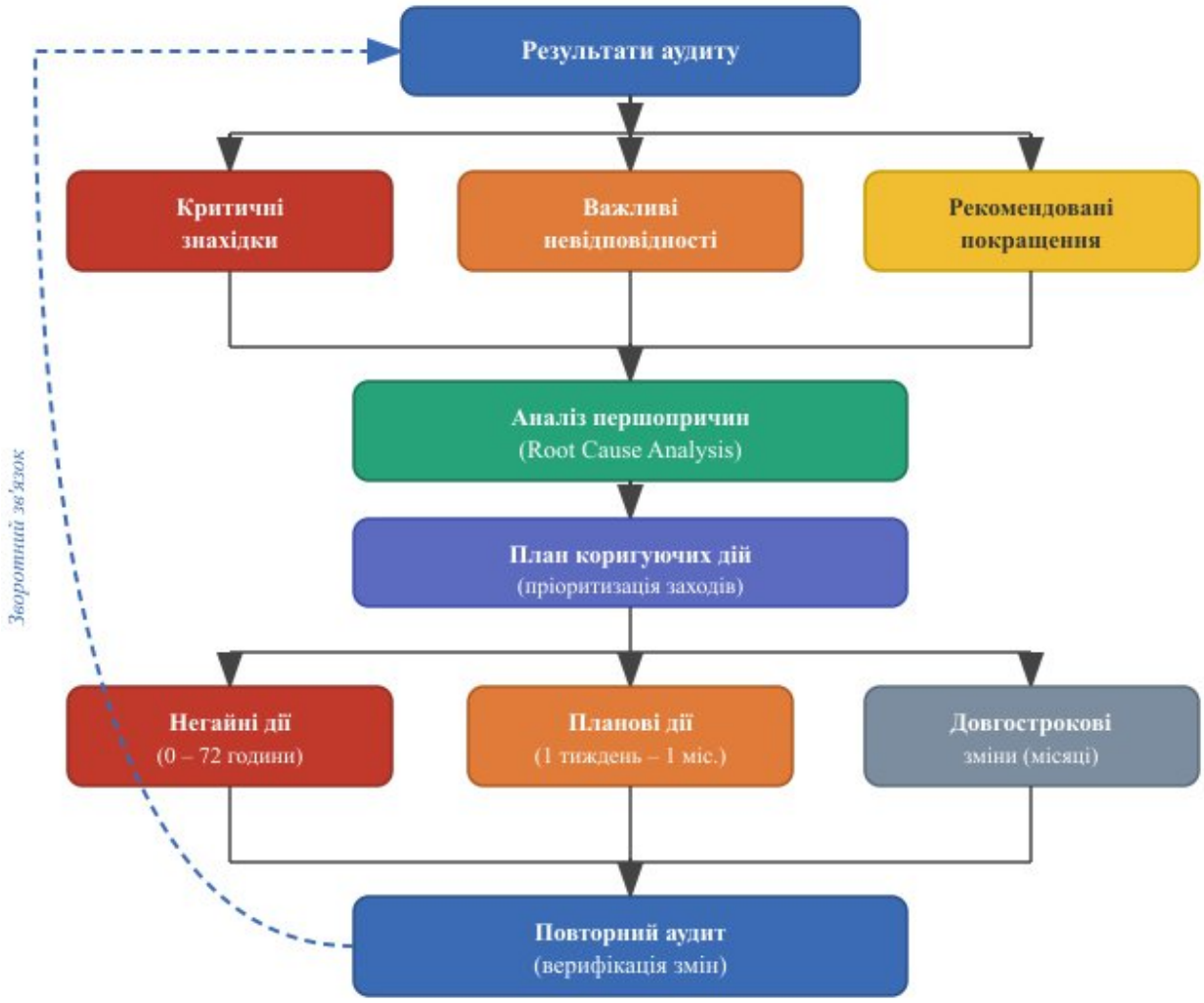

Рисунок 4.4 – Процес опрацювання результатів аудиту та формування плану hardening

На рисунку 4.4 зображено схему у вигляді вертикального дерева. У верхній частині розміщено прямокутний блок «Результати аудиту», від якого відходять три стрілки до блоків другого рівня: «Критичні знахідки» (червоний), «Важливі невідповідності» (помаранчевий) та «Рекомендовані покращення» (жовтий). Від кожного з цих блоків стрілки сходяться до блоку «Аналіз першопричин». Від нього одна стрілка веде до блоку «План коригуючих дій», що у свою чергу розгалужується на три горизонти: «0–72 год.», «1 тиждень – 1 місяць» та «Довгострокові зміни». Від кожного горизонту стрілка спрямована до фінального блоку «Повторний аудит», що замикає цикл, повертаючись стрілкою нагору до «Результатів аудиту».

Комплекс заходів з підвищення рівня захищеності операційної системи на основі результатів аудиту прийнято називати hardening (загартування системи). Hardening охоплює кілька стандартних напрямків. По-перше, вимикання невикористовуваних служб: якщо сервер не є поштовим, служба Postfix має бути вимкнена (sudo systemctl disable --now postfix). По-друге, посилення парольної політики: мінімальна довжина пароля, вимога до складності та термін дії налаштовуються у файлах /etc/login.defs та /etc/pam.d/common-password. По-третє, обмеження прав доступу до критичних файлів конфігурації (sudo chmod 600 /etc/ssh/sshd_config). По-четверте, налаштування фаєрволу для дозволу лише необхідного трафіку. По-п'яте, увімкнення та правильне налаштування підсистеми аудиту ОС.

Послідовність дій у процесі hardening є принципово важливою, оскільки деякі кроки залежать від попередніх. На рисунку 4.5 зображено рекомендовану послідовність операцій hardening у вигляді блок-схеми.

Рисунок 4.5 – Рекомендована послідовність hardening операційної системи

На рисунку 4.5 зображена вертикальна блок-схема з вісьмома блоками, з'єднаними стрілками зверху вниз.

Особливо важливим практичним аспектом є налаштування підсистеми аудиту Linux — демона auditd. Ця підсистема дозволяє реєструвати такі події: спроби доступу до файлів, виконання системних викликів, зміна привілеїв, вхід та вихід користувачів. Конфігурація правил зберігається у файлі /etc/audit/rules.d/audit.rules. Правило -w /etc/passwd -p wa -k passwd_changes реєструватиме будь-яке звернення до файлу /etc/passwd на запис або зміну атрибутів: параметр -w задає шлях до відстежуваного файлу, -p wa визначає типи відстежуваних операцій (w — запис, a — зміна атрибутів), а -k задає ключ для пошуку в журналах. Перегляд журналів здійснюється командою ausearch -k passwd_changes, а формування зрозумілих звітів — командою aureport.

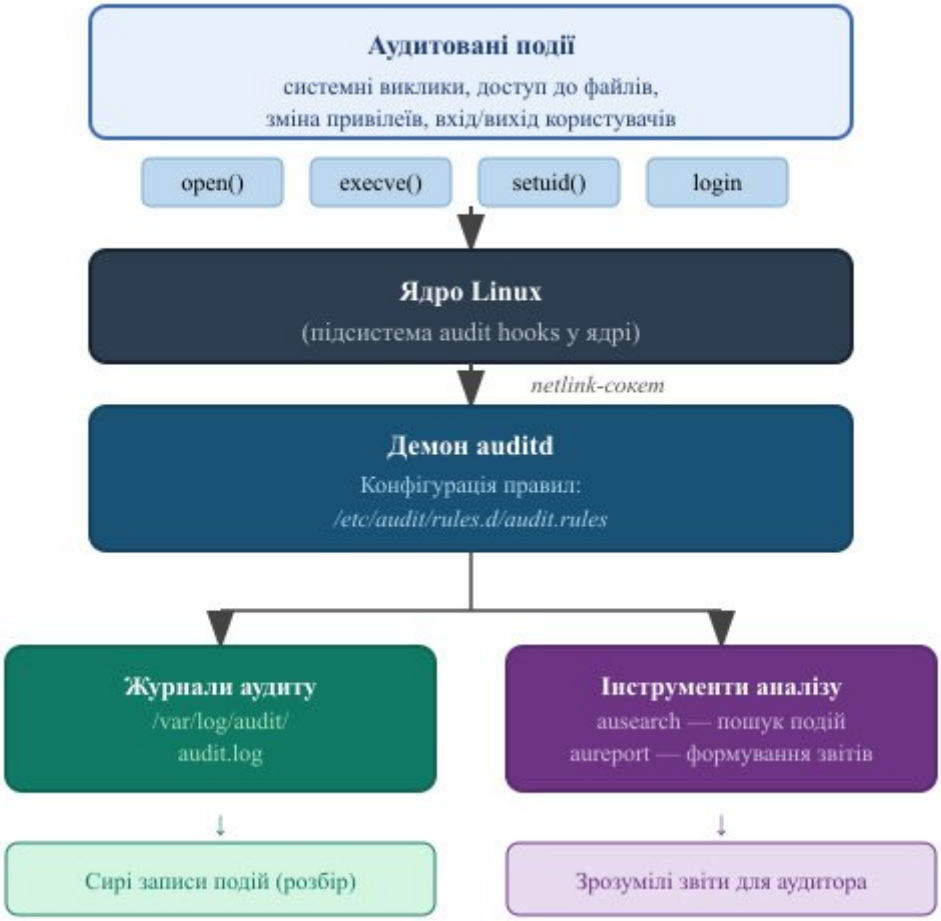

Рисунок 4.6 – Архітектура підсистеми аудиту Linux (auditd)

На рисунку 4.6 зображено вертикальну схему з чотирма рівнями, з'єднаними стрілками.

Ще одним важливим аспектом hardening є управління оновленнями безпеки. Аудит часто виявляє, що на системі встановлені застарілі версії програм із відомими вразливостями. У Linux перевірити наявність оновлень можна командою sudo apt list --upgradable. Для критичних середовищ рекомендується налаштувати автоматичне встановлення оновлень безпеки за допомогою пакета unattended-upgrades: після його встановлення та налаштування система автоматично завантажуватиме та встановлюватиме виправлення безпеки без ручного втручання адміністратора. Це суттєво скорочує так зване «вікно вразливості» — проміжок часу між публікацією патча та його встановленням на системі.

Після виконання всіх запланованих коригуючих дій проводиться повторний аудит для верифікації — підтвердження того, що внесені зміни справді усунули виявлені проблеми і не спричинили нових. Верифікація може бути вибірковою (лише параметри, що виправлялись) або повною (у разі масштабних змін). Результати повторного аудиту порівнюються з попередніми, що дає об'єктивне підтвердження прогресу. Ця практика є особливо важливою при проходженні зовнішніх сертифікаційних аудитів: перевіряючий очікує побачити не лише звіт із проблемами, але й докази їх усунення.

Таким чином, аудит безпеки операційної системи є комплексним та циклічним процесом, що охоплює планування, збір інформації, технічний аналіз, оцінку ризиків і цілеспрямовані заходи з підвищення захищеності. Ефективний аудит неможливий без опори на загальновизнані стандарти, використання автоматизованих інструментів у поєднанні з ручним аналізом і чіткого механізму переведення технічних знахідок у конкретні коригуючі дії. Освоєння цих навичок є обов'язковою умовою для фахівця з кібербезпеки, оскільки аудит забезпечує впевненість у тому, що система захищена не лише на папері, а й у реальному операційному середовищі.

Тема 5. Захист від атак грубої сили та автоматизованих вторгнень

5.1. Класифікація та характеристика атак на механізми автентифікації

Автентифікація є одним із ключових рубежів захисту будь-якої інформаційної системи: саме вона дозволяє відрізнити легітимного користувача від зловмисника. Проте механізми автентифікації самі по собі стають об'єктом атак — насамперед тих, що спрямовані на підбір або компрометацію облікових даних. Такі атаки об'єднують у широку категорію «атак на автентифікацію», і розуміння їхньої природи є необхідною умовою для побудови ефективного захисту. Загальна мета зловмисника в усіх випадках однакова — отримати доступ до системи від імені іншого користувача, минаючи встановлені політики безпеки.

Атака грубої сили (англ. brute force) є найпростішою з концептуальної точки зору: зловмисник послідовно перебирає всі можливі комбінації символів, поки не знайде ту, що збігається з паролем. Якщо пароль складається з 6 символів латинського алфавіту нижнього регістру, то кількість варіантів становить $26^{+} \approx 309$ мільйонів. Сучасне обладнання здатне перевіряти мільярди хеш-значень за секунду на відеокарті, тому без відповідних контрзаходів навіть відносно складні паролі можуть бути підібрані за прийнятний для зловмисника час. При цьому атака може проводитися як у режимі онлайн (безпосередньо проти служби автентифікації), так і в режимі офлайн (проти перехопленого хешу пароля), і ці два сценарії суттєво відрізняються за своєю ефективністю та заходами протидії.

Атака за словником (англ. dictionary attack) є суттєво ефективнішою версією атаки грубої сили: замість вичерпного перебору використовується заздалегідь підготовлений список слів, фраз і типових паролів. Такі списки формуються на основі аналізу реальних витоків паролів, лінгвістичних закономірностей різних мов та поширених шаблонів складання паролів — наприклад, «слово + рік» або «слово + знак оклику». Найвідоміший публічно доступний словник «rockyou.txt» містить понад 14 мільйонів записів і є стандартним інструментом у практиці тестування на проникнення. Атака за словником набагато швидша за повний перебір, адже охоплює паролі, що реально використовують люди, а не всі математично можливі комбінації.

Credential stuffing (укр. «вливання облікових даних») — якісно відмінний тип атаки, що базується не на підборі, а на використанні реальних пар логін/пароль, отриманих унаслідок попередніх витоків даних. Через те, що значна частина користувачів застосовує однакові паролі на різних сайтах і сервісах, зловмисник може взяти базу даних з одного скомпрометованого ресурсу і автоматично перевірити ці облікові дані на десятках інших. Атаки credential stuffing є особливо небезпечними, адже вони «виглядають» як звичайний успішний вхід і погано виявляються традиційними засобами — система бачить правильний логін і правильний пароль, а не підозрілу кількість помилок. За різними оцінками, щороку в мережі публікуються від кількох

мільярдів до десятків мільярдів пар логін/пароль, що робить credential stuffing масовим і автоматизованим явищем.

Password spraying — ще один варіант, при якому зловмисник вибирає кілька надзвичайно поширених паролів і перевіряє їх проти великої кількості різних облікових записів. Цей підхід дозволяє обійти політику блокування після N невдалих спроб для одного облікового запису, адже кожен акаунт атакується лише один-два рази. Саме тому password spraying часто залишається непоміченим для стандартних засобів захисту, що реагують лише на велику кількість спроб для конкретного користувача. Наприклад, атака на організацію з тисячею облікових записів із паролем «Welcome1» дасть статистично кілька десятків успішних входів — і жодного сигналу від системи блокування по кількості спроб.

Важливим є також розмежування між атаками в режимі онлайн і офлайн. Онлайн-атака проводиться безпосередньо проти живого сервісу: кожна спроба надсилається мережею і потребує відповіді від сервера, що суттєво обмежує швидкість перебору — зазвичай кілька сотень або тисяч спроб на секунду, адже мережеві затримки та затримки обробки запитів на стороні сервера стають пляшковим горлечком. Офлайн-атака відбувається після того, як зловмисник отримав базу даних хешів паролів і проводить перебір локально на власному обладнанні, не взаємодіючи з мережею — швидкість у цьому разі визначається лише продуктивністю процесора та відеокарти і може сягати мільярдів хешів на секунду для MD5 або сотень мільйонів для bcrypt. Саме тому вибір алгоритму хешування паролів є критично важливим для безпеки зберігання облікових даних, проте це тема, що виходить за межі поточного розгляду і детальніше охоплюється у темах, присвячених криптографії. Для даної теми важливо розуміти, що захист від онлайн-атак будується на обмеженні швидкості та кількості спроб, тоді як захист від офлайн-атак — насамперед на правильному зберіганні облікових даних.

Для наочного порівняння розглянутих типів атак у таблиці 5.1 наведено їх основні характеристики. Важливо розуміти, що всі описані атаки в сучасних умовах є значною мірою автоматизованими: зловмисники використовують ботнети для розподілених атак із тисяч різних IP-адрес, що суттєво ускладнює виявлення та блокування. На рисунку 5.1 схематично зображено відмінності між чотирма типами атак за характером перебору — від лінійного вичерпного переліку у brute force до використання готових баз даних у credential stuffing.

Таблиця 5.1 — Порівняльна характеристика типів атак на механізми автентифікації

Тип атаки	Принцип дії	Швидкість	Залежність від словника/БД
Brute Force	Повний перебір усіх комбінацій символів	Низька	Ні
Dictionary Attack	Перебір слів і фраз зі спеціального словника	Висока	Так (словник)

Тип атаки	Принцип дії	Швидкість	Залежність від словника/БД
Credential Stuffing	Використання пар логін/пароль з витоків даних	Висока	Так (БД витоків)
Password Spraying	Один пароль проти великої кількості акаунтів	Середня	Частково

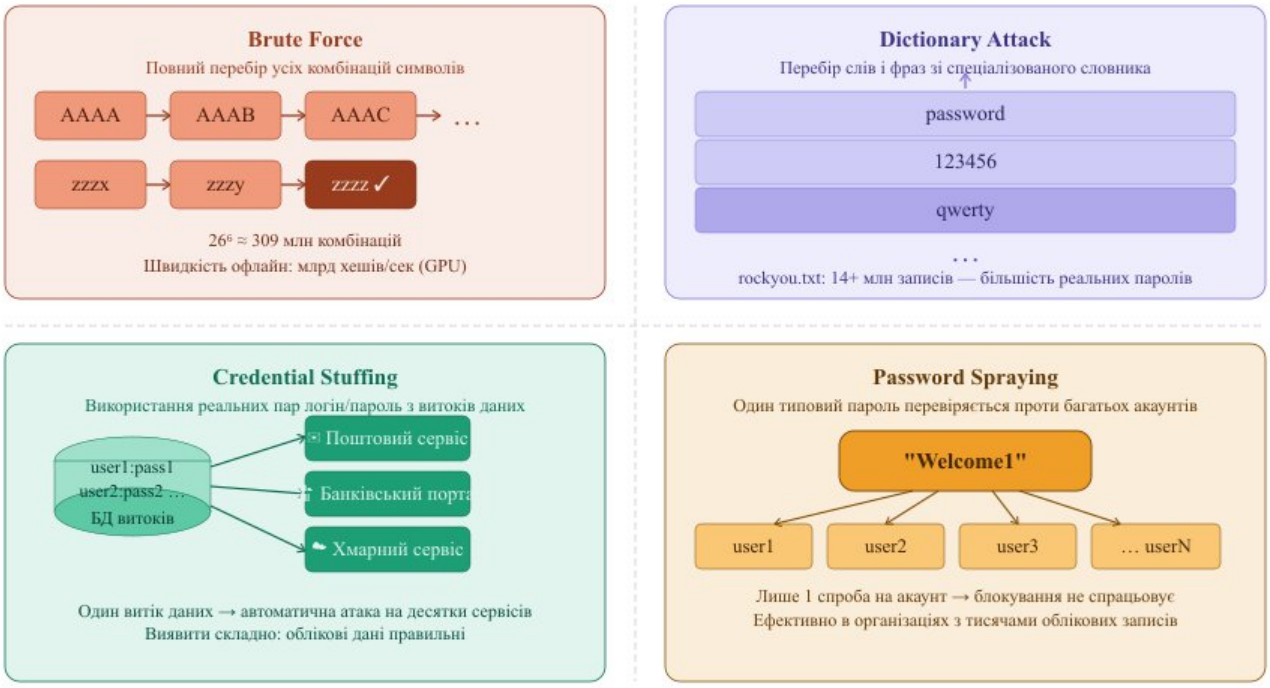

Рисунок 5.1 — Порівняння принципів роботи чотирьох типів атак на автентифікацію

Рисунок 5.1 є концептуальною схемою, що складається з чотирьох блоків, розташованих у два рядки. Верхній лівий блок «Brute Force» показує лінійний перебір символьних комбінацій (AAAA → AAAB → AAAC → … → zzzz) зі стрілкою, що вказує на нескінченну послідовність. Верхній правий блок «Dictionary Attack» зображує впорядкований список: password, 123456, qwerty, monkey, …, rockyou зі стрілкою вниз, що означає послідовний перебір. Нижній лівий блок «Credential Stuffing» містить піктограму бази даних витоків (user1:pass1, user2:pass2 …) зі стрілками до кількох іконок сервісів (пошта, банк, хмара), символізуючи перевірку тих самих даних на різних платформах. Нижній правий блок «Password Spraying» показує один пароль «Welcome1» у центрі та розходяться стрілки до великої кількості іконок користувачів, позначених user1, user2, … userN.

5.2. Джерела даних для виявлення атак: журнали та мережеві події

Виявлення автоматизованих атак на автентифікацію ґрунтується на аналізі даних, які генерує операційна система у процесі своєї роботи. Такі дані зосереджені насамперед у системних журналах — текстових файлах або структурованих базах даних, куди записуються події різного типу: успішні і невдалі спроби входу, зміни конфігурації, запуск процесів, мережева активність тощо. Без ведення журналів виявити атаку практично неможливо, адже зловмисна активність не залишає інших слідів. Саме тому налагодження повноцінного журналювання є першим і обов'язковим кроком до побудови будь-якої системи виявлення вторгнень.

У Linux основним джерелом інформації про автентифікацію є файл /var/log/auth.log. Кожна невдала спроба входу через SSH фіксується рядком, що містить позначку часу, ім'я хоста, назву служби (sshd), повідомлення «Failed password» або «Invalid user», а також IP-адресу, з якої надійшов запит. Аналізуючи цей файл, можна легко помітити патерн атаки грубої сили: сотні рядків з різними іменами користувачів, що надходять з однієї IP-адреси за короткий проміжок часу. Для перегляду актуальних подій у реальному часі застосовують команду tail -f /var/log/auth.log або journalctl -fu ssh, що виводить нові рядки одразу після їхньої появи в журналі.

Системний журнал /var/log/syslog містить значно ширший спектр подій — запуск і зупинку служб, повідомлення ядра, активність планувальника завдань. Хоча він безпосередньо не реєструє спроби входу, він є корисним для відстеження активності у суміжних процесах та виявлення аномалій, пов'язаних із атакою: наприклад, якщо зловмисник після успішного злому намагається встановити програмне забезпечення або змінити конфігурацію, сліди цих дій часто з'являються саме у syslog. Таким чином, для повноцінного аналізу інциденту необхідно паралельно аналізувати кілька джерел. У таблиці 5.2 систематизовано основні джерела даних, їх розташування у Linux та типові події для аналізу.

Таблиця 5.2 — Джерела даних для виявлення атак на автентифікацію у Linux

Джерело даних	Шлях / інструмент (Linux)	Ключові події для аналізу
Журнал автентифікації	/var/log/auth.log	Failed password, Invalid user, session opened
Системний журнал	/var/log/syslog	Запуск служб, зміни конфігурацій, помилки
PAM-підсистема	/var/log/auth.log (PAM)	pam_unix: authentication failure
Журнал брандмауера	iptables log / journalctl	Заблоковані з'єднання, портові скани
Journald (systemd)	journalctl -u ssh	Стан служби SSH, помилки входу

Мережеві події є ще одним важливим джерелом даних. Брандмауер операційної системи (наприклад, iptables або nftables у Linux) може бути налаштований на реєстрацію всіх або вибіркових мережевих з'єднань — зокрема спроб підключення до портів, що не використовуються, або з'єднань від підозрілих адрес. Аналіз мережевих логів дозволяє виявляти не тільки атаки на автентифікацію, а й горизонтальне переміщення (lateral movement) по мережі, сканування портів та інші ознаки присутності зловмисника в системі.

Для зручного аналізу логів у Linux широко використовуються стандартні текстові утиліти. Наведемо кілька практичних прикладів таких команд. Команда grep підраховує кількість невдалих спроб входу в auth.log: виконавши grep "Failed password" /var/log/auth.log | wc -l, адміністратор отримає загальне число невдалих спроб за весь час ведення журналу. Для визначення найагресивніших джерел атак застосовується конвеєр із grep, awk, sort та head: він виводить десятку IP-адрес з найбільшою кількістю невдалих спроб, що дозволяє швидко ідентифікувати атакуючих навіть без спеціалізованих інструментів. Для перегляду актуальних подій SSH у реальному часі команда journalctl -fu ssh дозволяє спостерігати за потоком подій, що є незамінним при оперативному реагуванні на інцидент. Нижче наведено приклади відповідних команд:

```
# Загальна кількість невдалих спроб входу
grep "Failed password" /var/log/auth.log | wc -l

# ТОП-10 IP-адрес, що атакували систему
grep "Failed password" /var/log/auth.log | \
  awk '{print $(NF-3)}' | sort | uniq -c | sort -nr | head -10

# Список імен користувачів, що атакувалися
grep "Invalid user" /var/log/auth.log | \
  awk '{print $8}' | sort | uniq -c | sort -nr | head -20

# Перегляд подій SSH у реальному часі
journalctl -fu ssh
```

Ці команди утворюють базовий набір інструментів для швидкого ситуаційного аналізу. Перша команда дає загальне уявлення про масштаб проблеми; друга — визначає конкретних «порушників», яких можна внести до чорного списку вручну або перевірити в базах репутації IP-адрес (наприклад, AbuseIPDB); третя — виявляє, які імена користувачів найчастіше перебираються зловмисниками, що може свідчити про цільову атаку на конкретний акаунт або про використання стандартного словника імен. Регулярне виконання цих команд як частини адміністративної рутини формує у фахівця з безпеки стійку звичку тримати руку на пульсі системи і помічати відхилення від норми до того, як вони переростуть у реальний інцидент.

Ефективний аналіз журналів неможливий без розуміння нормальної базової поведінки системи — так званої «базової лінії» (baseline). Якщо адміністратор знає, що зазвичай до SSH-сервера підключається лише кілька відомих IP-адрес у робочий час і кількість невдалих спроб не перевищує 10–20

на добу, то поява 5 000 невдалих спроб за годину є очевидною аномалією, навіть якщо жодна з IP ще не перевищила поріг maxretry. Саме на принципі відхилення від норми базуються сучасні системи виявлення аномалій. Для зручності аналізу великих обсягів журналів використовують SIEM-рішення (Security Information and Event Management), які агрегують події з різних джерел, забезпечують їх кореляцію та надають засоби автоматичного оповіщення при виявленні характерних патернів атак. Хоча розгортання повноцінного SIEM виходить за рамки поточної теми, розуміння його ролі є важливим для формування цілісного уявлення про екосистему засобів виявлення вторгнень.

Система journald, що входить до складу systemd, є основним механізмом журналювання у більшості сучасних дистрибутивів Linux. На відміну від традиційних текстових логів, journald зберігає події у бінарному структурованому форматі, що забезпечує більш ефективне індексування і пошук. Команда journalctl дозволяє гнучко фільтрувати події: наприклад, journalctl -u ssh --since "1 hour ago" виведе всі події SSH-сервісу за останню годину, а journalctl -u ssh --since "2024-01-15 00:00:00" --until "2024-01-15 06:00:00" дозволяє проаналізувати конкретний часовий проміжок. Таке структуроване журналювання значно прискорює аналіз інцидентів і є важливим елементом системи виявлення атак. Крім того, journald підтримує пересилання подій до зовнішніх систем збору логів через протокол syslog, що дозволяє централізувати журнали з кількох серверів і аналізувати їх у єдиному місці — важлива властивість для корпоративних середовищ із великою кількістю хостів.

5.3. Системи виявлення та запобігання вторгненням і механізми автоматичного блокування

Ручний аналіз журналів є ефективним засобом розслідування інцидентів, але практично непридатним для реагування в режимі реального часу: зловмисник може здійснити тисячі спроб за хвилини, поки адміністратор навіть не встигне відкрити файл логу. Для автоматизованого реагування на атаки призначені системи виявлення вторгнень (Intrusion Detection System, IDS) та системи запобігання вторгненням (Intrusion Prevention System, IPS). Ці два класи систем тісно пов'язані між собою і часто розглядаються разом як IDS/IPS, хоча між ними є принципова відмінність: IDS лише виявляє та сигналізує про підозрілу активність, тоді як IPS здатна активно блокувати загрозу, не чекаючи дій адміністратора.

На рисунку 5.2 зображено загальну архітектуру IDS/IPS у мережевій інфраструктурі. Схема є горизонтальним ланцюжком компонентів зліва направо: «Зовнішній трафік» → «Маршрутизатор/Firewall» → «IDS/IPS» → «Внутрішня мережа». Від блоку «IDS/IPS» відходять дві стрілки: одна вгору позначена «Сповіщення адміністратора (IDS)» — вона вказує лише на функцію виявлення, друга стрілка вниз позначена «Блокування з'єднання (IPS)» і спрямована назад до компонента маршрутизатора, символізуючи активний захист. Паралельно від блоку «IDS/IPS» йде стрілка до компонента «SIEM /

Журнал подій», що збирає та зберігає дані для подальшого аналізу. Мережевий IDS/IPS аналізує трафік на рівні мережі, хостовий — безпосередньо на захищеному сервері.

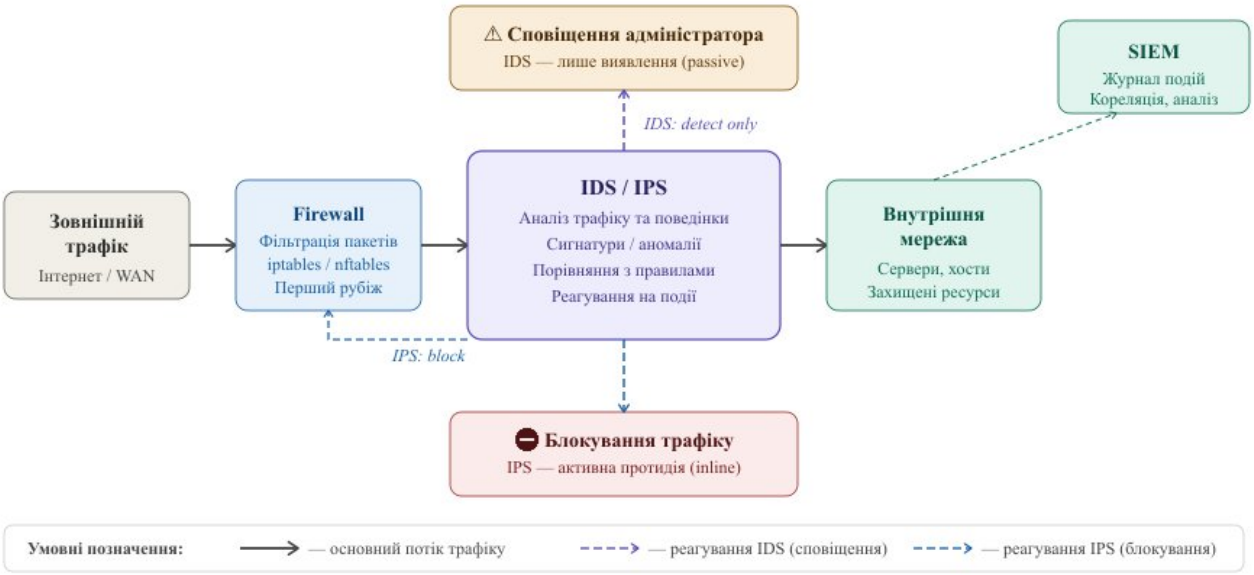

Рисунок 5.2 — Архітектура мережевого IDS/IPS та взаємодія компонентів

За принципом розгортання IDS/IPS поділяють на мережеві (NIDS/NIPS) та хостові (HIDS/HIPS). Мережеві системи аналізують трафік на рівні мережі, перехоплюючи пакети та порівнюючи їх із сигнатурами відомих атак або виявляючи аномалії в протоколах. Хостові системи працюють безпосередньо на сервері або робочій станції, що захищається, і моніторять системні виклики, зміни файлів, активність користувачів і локальні журнали. Для захисту від атак brute force та credential stuffing хостові IDS/IPS є більш доречними, адже вони мають прямий доступ до журналів автентифікації і можуть реагувати миттєво без затримок, пов'язаних із мережевим перехопленням.

Серед мережевих систем виявлення вторгнень найбільш відомими є Snort та Suricata. Snort — одна з перших і найбільш поширених систем класу NIDS з відкритим кодом, що використовує базу правил (сигнатур) для ідентифікації відомих атак у мережевому трафіку. Suricata є більш сучасним рішенням, що підтримує багатопоточну обробку і сумісна з правилами Snort, додатково пропонуючи можливості аналізу протоколів і виявлення аномалій. Обидві системи можуть працювати як у пасивному режимі IDS (лише фіксація і сповіщення), так і в активному режимі IPS (блокування трафіку через взаємодію з брандмауером). Для захисту від brute force на рівні мережі ці системи можуть виявляти характерні патерни, як-от велика кількість TCP-з'єднань до SSH-порту з різних адрес, і формувати відповідні правила блокування. Однак їх налаштування та підтримка потребують значно більшого часу і кваліфікації порівняно з Fail2ban, тому для невеликих інфраструктур останній залишається оптимальним вибором.

Одним із найпоширеніших і найефективніших інструментів автоматичного блокування для Linux-систем є Fail2ban. Це програмне

забезпечення моніторить вказані файли журналів за допомогою регулярних виразів і при виявленні заданої кількості підозрілих подій за певний проміжок часу автоматично додає правило до брандмауера, що блокує IP-адресу порушника. Fail2ban підтримує широкий набір попередньо налаштованих «в'язниць» (jails) для різних служб — SSH, FTP, Apache, Nginx, Postfix тощо, — що робить його універсальним рішенням для захисту різноманітних сервісів на одному сервері. Після закінчення часу блокування правило автоматично видаляється, якщо не налаштовано інше.

На рисунку 5.3 схематично зображено загальний принцип роботи Fail2ban. Схема складається з блоків, з'єднаних послідовними стрілками. Зліва розташований блок «Лог-файл (auth.log)», від якого стрілка веде до блоку «Монітор журналів (Log Monitor)» — компонента, що постійно зчитує нові рядки файлу. Далі стрілка веде до блоку «Фільтр (Filter / regex)», де кожен рядок порівнюється з шаблоном failregex. Якщо збіг є, подія надходить до блоку «Лічильник спроб (Counter)», що відстежує кількість збігів у межах часового вікна findtime. Коли лічильник перевищує значення maxretry, активується блок «Виконавець дій (Action)», що надсилає команду до «Брандмауера (iptables/nftables)» — той додає правило блокування для відповідної IP-адреси. Окремо показані блоки «Білий список (ignoreip)» з підключенням до лічильника (адреси зі списку не рахуються) та «Таймер bantime», після спрацювання якого правило видаляється. Жирним шрифтом у блоках підписано назви параметрів: findtime над лічильником, maxretry на переході від лічильника до виконавця, bantime над таймером.

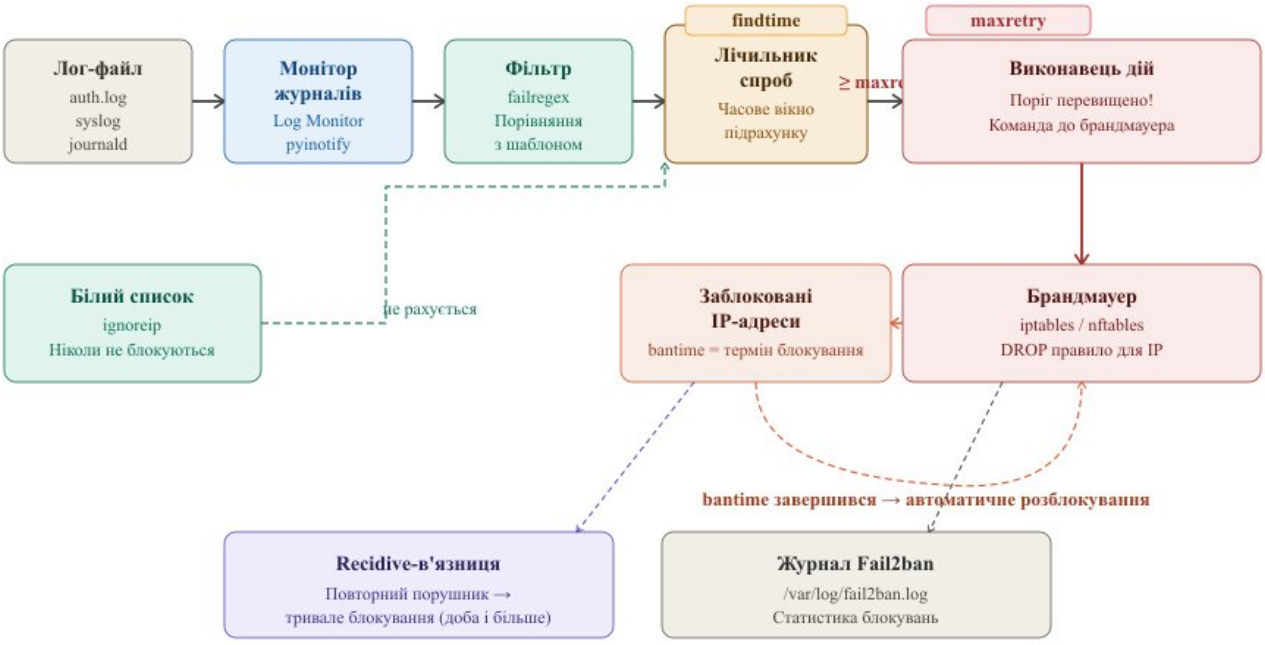

Рисунок 5.3 — Схема роботи Fail2ban: від появи події в журналі до блокування IP-адреси

Конфігурація Fail2ban здійснюється через файли у директорії /etc/fail2ban/. Головний файл jail.conf містить параметри за замовчуванням; усі зміни рекомендується вносити до файлу jail.local, щоб вони не

перезаписувалися при оновленні пакета. Кожна в'язниця описує один захищуваний сервіс і включає параметри enabled, port, filter, logpath, а також maxretry, findtime і bantime. Приклад конфігурації для захисту SSH виглядає так:

```
[sshd]
enabled  = true
port     = ssh
filter   = sshd
logpath  = /var/log/auth.log
maxretry = 3
findtime = 600
bantime  = 3600
ignoreip = 127.0.0.1/8 192.168.1.0/24
```

У цьому прикладі параметри налаштовані на захист SSH-порту: дозволено лише 3 невдалі спроби протягом 10 хвилин, і при їх перевищенні IP блокується на 1 годину. Мережа 192.168.1.0/24 додана до білого списку, щоб адміністратори локальної мережі не могли бути заблоковані. У таблиці 5.3 систематизовано основні параметри конфігурації Fail2ban та їх призначення, а у таблиці 5.4 наведено приклади регулярних виразів, що використовуються для розпізнавання подій у журналах.

Таблиця 5.3 — Основні параметри конфігурації Fail2ban

Параметр	Значення за замовч.	Призначення
maxretry	5	Максимальна кількість невдалих спроб перед блокуванням
findtime	600 с (10 хв)	Часове вікно, в якому рахуються спроби
bantime	600 с (10 хв)	Тривалість блокування IP-адреси
ignoreip	127.0.0.1/8	Адреси, що ніколи не блокуються (білий список)
backend	auto	Метод моніторингу файлів журналів (pyinotify, gamin тощо)

Таблиця 5.4 — Приклади регулярних виразів у фільтрах Fail2ban для SSH

Патерн атаки	Регулярний вираз (failregex)
Невдалий вхід SSH	Failed password for .+ from <HOST>
Невірний користувач SSH	Invalid user .+ from <HOST>
Перевищення часу очікування	Connection closed by <HOST> \[preauth\]
Помилка ключа	error: maximum authentication attempts exceeded .+ from <HOST>

Fail2ban дозволяє також створювати власні фільтри для нестандартних сервісів або для уточнення стандартних. Файли фільтрів розташовані в /etc/fail2ban/filter.d/ і містять секцію [Definition] з параметром failregex. Нижче наведено структуру власного фільтра для SSH, що охоплює кілька типових шаблонів невдалої автентифікації:

```
     # /etc/fail2ban/filter.d/custom-ssh.conf
     [Definition]
     failregex = ^%(__prefix_line)sFailed password for .* from <HOST>
port \d+ ssh2$
                 ^%(__prefix_line)sInvalid user .* from <HOST> port \d+$
                 ^%(__prefix_line)smaximum authentication attempts .*
from <HOST>$
     ignoreregex =
```

Тег <HOST> у регулярних виразах є спеціальним позначенням Fail2ban: він автоматично замінюється на вираз для розпізнавання IPv4 та IPv6 адрес у рядках журналу. Параметр ignoreregex дозволяє задати шаблон рядків, що повинні ігноруватися навіть при збігу з failregex — наприклад, рядки від відомих сканерів безпеки або систем моніторингу.

Окрім Fail2ban, для захисту від автоматизованих атак застосовуються й інші підходи. Налаштування служби SSH для роботи на нестандартному порті суттєво знижує кількість автоматизованих атак від ботнетів, що сканують лише стандартні порти, хоча і не забезпечує реального захисту від цілеспрямованого зловмисника. Значно ефективнішим є перехід від парольної автентифікації до автентифікації на основі криптографічних ключів: навіть якщо зловмисник знає правильний пароль, без відповідного приватного ключа він не зможе увійти, адже перевіряється математично складне криптографічне твердження, а не рядок символів. Ці заходи доповнюють автоматичне блокування і разом формують ешелоновану систему захисту.

5.4. Налаштування параметрів реагування, оцінка ефективності та багаторівневий захист

Встановлення та увімкнення системи захисту від brute force — лише початок роботи: не менш важливим є правильне налаштування параметрів реагування і подальша оцінка того, наскільки ефективно захист справляється зі своїм завданням. Неправильно підібрані параметри здатні або знизити захищеність до неприйнятного рівня, або спровокувати численні помилкові спрацювання, що блокують легітимних користувачів. Баланс між чутливістю системи і кількістю хибних спрацювань є однією з центральних задач при конфігурації IPS-рішень і вимагає урахування специфіки конкретного середовища.

Розглянемо ключові параметри на прикладі Fail2ban. Значення maxretry визначає, скільки невдалих спроб допускається перед блокуванням. Надто мале значення (2–3) може призвести до блокування легітимних користувачів, що просто помилилися при введенні пароля. Надто велике (20–30) фактично дозволяє зловмисникові виконати значну частину атаки до блокування.

Типовим компромісом є значення 5. Параметр findtime задає часове вікно: якщо 5 невдалих спроб відбуваються протягом 600 секунд, IP блокується; якщо ж ті самі 5 спроб розподілені на годину — блокування не відбувається. Параметр bantime визначає тривалість блокування: стандартні 10 хвилин відлякують більшість ботнетів, але для критичних систем застосовують значно більші значення — до кількох діб. У таблиці 5.5 наведено рекомендовані значення параметрів для різних типів середовищ.

Таблиця 5.5 — Рекомендовані значення параметрів Fail2ban для різних середовищ

Параметр	Тестове середовище	Продукційне середовище	Критична система
maxretry	10	5	3
findtime	1200 с	600 с	300 с
bantime	300 с	3600 с	86400 с

Fail2ban підтримує функцію recidive (рецидивіст): якщо IP-адреса вже була заблокована раніше і знову атакує систему, вона блокується на значно триваліший термін. Ця функція реалізована як окрема в'язниця і дозволяє суттєво підвищити стійкість захисту проти наполегливих зловмисників. Конфігурація recidive-в'язниці включає власний logpath, що вказує на журнал самого Fail2ban (/var/log/fail2ban.log), і довші значення bantime — типово від кількох годин до тижня. Такий підхід формує «поступову ескалацію» відповіді: перше порушення — коротке блокування, повторне — значно триваліше.

Для оцінки ефективності захисту необхідно регулярно аналізувати журнали Fail2ban. Корисними є команди: fail2ban-client status виводить список активних в'язниць та загальну статистику, fail2ban-client status sshd — детальну статистику для SSH, включаючи список поточних заблокованих IP. Аналіз файлу /var/log/fail2ban.log за тиждень або місяць дає відповідь на питання про інтенсивність атак, їхню географію та динаміку. Важливим показником є відношення кількості виявлених атак до кількості хибних спрацювань. Хибне спрацювання можна визначити постфактум: якщо заблокована IP належить до корпоративної мережі і легітимний користувач скаржиться на неможливість входу — це типовий false positive, що вимагає додавання адреси до ignoreip.

На рисунку 5.4 зображено часову шкалу типового інциденту: від початку атаки до автоматичного розблокування. Схема є горизонтальною стрічкою часу з позначками подій. Відмітка «00:00» — «Початок атаки: перша невдала спроба SSH». Відмітка «00:02» — «3-тя невдала спроба, Fail2ban починає відлік у межах findtime». Відмітка «00:05» — «5-та спроба: maxretry досягнуто, формується команда для iptables». Відмітка «00:06» — «IP-адреса заблокована: iptables DROP правило додано, запис у /var/log/fail2ban.log». Відмітка «01:06» (якщо bantime=3600) — «Автоматичне розблокування: правило видаляється».

Під стрічкою часу додатковим кольором підписані параметри: між «00:00» і «00:06» позначено findtime=600 сек, між «00:06» і «01:06» — bantime=3600 сек.

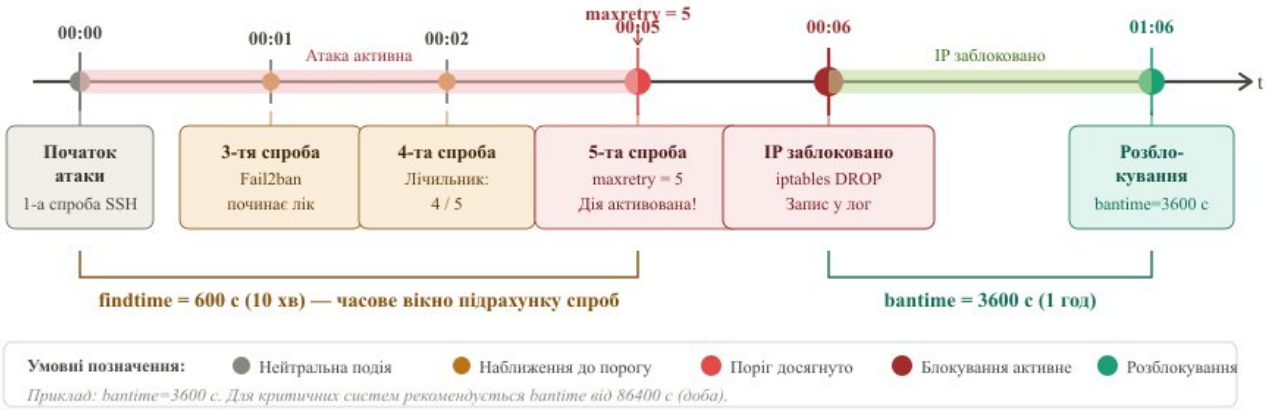

Рисунок 5.4 — Часова шкала інциденту: від початку атаки до блокування та розблокування

Важливо усвідомлювати, що жодна окрема технічна міра не є абсолютним захистом від усіх можливих атак. Зловмисник із доступом до великого ботнету може здійснювати credential stuffing, надсилаючи лише один запит з кожної адреси — у такому разі Fail2ban із maxretry=5 взагалі не відреагує, бо кожна адреса «відступає» після першої спроби. Протидія таким розподіленим атакам вимагає інших підходів: аналізу географії IP-адрес, обмеження кількості запитів по часу (rate limiting), використання CAPTCHA для веб-сервісів, а також впровадження багатофакторної автентифікації (MFA).

Концепція багаторівневого захисту (defense in depth) передбачає, що відмова одного рівня захисту не призводить до компрометації всієї системи — наступний рівень продовжує захищати актив. На рисунку 5.5 зображено піраміду рівнів захисту від автоматизованих атак. Схема є пірамідою з п'яти шарів, кожен з яких підписаний ліворуч від відповідного рівня: найнижчий і найширший рівень 1 — «SSH на нестандартному порті (зниження видимості)»; рівень 2 — «SSH-ключі замість паролів (криптографічна автентифікація)»; рівень 3 — «Fail2ban (автоматичне блокування за логами)»; рівень 4 — «Правила брандмауера (geo-blocking, rate limiting)»; рівень 5 — «Багатофакторна автентифікація (MFA)», що позначений як вершина піраміди. Стрілка збоку підписана: «Зловмисник долає кожен рівень з усе більшими зусиллями». Такий образ наочно пояснює, чому комбінація методів є набагато ефективнішою за будь-який один засіб.

Рисунок 5.5 — Піраміда багаторівневого захисту від атак на автентифікацію

Для загальної оцінки і вибору конкретних засобів захисту корисно порівняти їх ефективність і складність впровадження. У таблиці 5.6 наведено порівняльну характеристику п'яти основних методів захисту від атак на автентифікацію, що дозволяє сформувати обґрунтований комплекс заходів залежно від доступних ресурсів та рівня загрози.

Таблиця 5.6 — Порівняння методів захисту від атак на автентифікацію

Метод захисту	Ефективність проти Brute Force	Ефективність проти Credential Stuffing	Складність впровадження
Fail2ban	Висока	Середня	Низька
Багатофакторна автентифікація	Дуже висока	Дуже висока	Середня
CAPTCHA	Висока	Висока	Низька
Rate limiting	Середня	Висока	Середня
SSH ключі	Дуже висока	Не застосовується	Середня

Окремо слід розглянути практичну конфігурацію самого SSH-сервера як першого рубежу захисту. У файлі /etc/ssh/sshd_config рекомендується встановити такі параметри: PermitRootLogin no (заборона прямого входу під root, що позбавляє сенсу атаки на цей широко відомий обліковий запис), PasswordAuthentication no (вимкнення парольної автентифікації на користь ключів), MaxAuthTries 3 (обмеження кількості спроб на рівні самого протоколу, ще до втручання Fail2ban), LoginGraceTime 20 (скорочення часу очікування успішної автентифікації, що зменшує навантаження від «висячих» з'єднань). Додатково корисним є параметр AllowUsers, що явно перелічує дозволені імена користувачів — навіть якщо зловмисник знає правильний пароль облікового запису, якого немає в списку AllowUsers, сервер відмовить у з'єднанні. Також параметр UseDNS no прискорює встановлення з'єднань і усуває затримки,

пов'язані з оберненим DNS-розпізнаванням, що особливо важливо для сильно навантажених серверів. Для застосування змін необхідно перезавантажити службу командою systemctl restart sshd і переконатися у відсутності синтаксичних помилок командою sshd -t перед перезапуском — це запобігає ситуації, коли неправильна конфігурація блокує адміністратора ззовні. Ці параметри формують перший рубіж захисту, а Fail2ban — другий, що реагує на ті спроби, яким вдалося «дотягнутися» до механізму автентифікації.

Варто розглянути ще один нестандартний, але цікавий механізм захисту — port knocking. Це техніка, при якій мережевий порт служби (наприклад, SSH на порту 22) залишається закритим для зовнішніх з'єднань і відкривається лише після того, як клієнт у правильному порядку «постукає» — надішле TCP або UDP пакети до певної послідовності інших портів. Наприклад, послідовність може бути: з'єднання до порту 3000, потім 5000, потім 7000 — і лише після цього порт SSH відкривається для тієї IP-адреси на обмежений час. Зовні система виглядає як така, що взагалі не надає жодних послуг, і автоматизовані сканери ботнетів просто проходять повз неї. Недоліком port knocking є певне ускладнення адміністрування, тому він застосовується переважно у середовищах з підвищеними вимогами або у поєднанні з VPN-доступом для управлінських інтерфейсів. На практиці цей механізм реалізується за допомогою утиліти knockd у Linux, що стежить за мережевими подіями брандмауера і відкриває порт після фіксації правильної послідовності.

Загалом тема захисту від атак грубої сили та автоматизованих вторгнень є наочним прикладом того, як теоретичні принципи безпеки — зокрема принцип ешелонованого захисту і принцип найменших привілеїв — знаходять конкретне практичне втілення у налаштуванні операційної системи. Журналювання забезпечує видимість подій; IDS/IPS і Fail2ban — автоматизоване реагування; правильна конфігурація самих служб — мінімізацію поверхні атаки; MFA — додатковий криптографічний рубіж. Усі ці складові разом формують функціональний і практично реалізований захист, що студенти можуть безпосередньо налаштувати і перевірити під час лабораторних робіт у середовищі Linux, спостерігаючи в реальному часі, як система реагує на змодельовані атаки.

5.5. Практичні сценарії атак та захисту

Теоретичне розуміння типів атак і механізмів захисту набуває повної цінності лише тоді, коли воно підкріплене конкретними прикладами того, як атака розгортається у реальному середовищі і як система захисту реагує на неї. Розглянемо три характерних сценарії, що відображають різні моделі загроз і вимагають різних підходів до захисту. Кожен сценарій демонструє певний аспект взаємодії зловмисника із захищеною системою і допомагає зрозуміти, де саме система захисту спрацьовує, а де залишаються прогалини.

Сценарій 1: Класична атака brute force з одного IP. Зловмисник запускає автоматизований інструмент (наприклад, Hydra або Medusa), що надсилає по кілька сотень SSH-запитів на хвилину з однієї IP-адреси, перебираючи різні

імена користувачів і паролі. Протягом перших кількох секунд у /var/log/auth.log з'являються десятки рядків «Failed password for root from 203.0.113.42». Fail2ban, налаштований із maxretry=5 і findtime=600, виявляє перевищення порогу вже через 5–10 секунд від початку атаки і додає правило DROP для IP 203.0.113.42 до iptables. Усі подальші пакети від цієї адреси мовчки відкидаються на рівні брандмауера без залучення ресурсів SSH-сервера. Після закінчення bantime Fail2ban автоматично видаляє правило; якщо атака відновлюється — в'язниця recidive блокує адресу вже на значно довший термін.

Сценарій 2: Розподілена атака ботнету. Зловмисник координує ботнет із 10 000 скомпрометованих пристроїв (часто це звичайні домашні маршрутизатори або IoT-пристрої з незміненими стандартними паролями), кожен з яких надсилає лише по одній спробі входу на кілька хвилин — значно нижче порогу maxretry для будь-якого окремого IP. Fail2ban у цьому випадку не спрацьовує, адже жодна окрема адреса не перевищує порогу спроб. У журналах з'являються поодинокі «Failed password» від тисяч різних адрес, що статистично сприймається як «шум». Для виявлення такої атаки потрібен інший підхід: аналіз загальної кількості невдалих спроб у часі (значне зростання порівняно з базовою лінією), впровадження rate limiting на рівні брандмауера (обмеження загальної кількості нових SSH-з'єднань за секунду незалежно від джерела), а також геоблокування — якщо організація не очікує підключень з певних регіонів світу, їх можна заблокувати превентивно за допомогою баз даних геолокації IP-адрес. Додатковим індикатором розподіленої атаки може бути аналіз User Agent або інших характеристик з'єднань при атаці на веб-сервіси, проте для SSH цей метод непридатний через обмеження протоколу.

Сценарій 3: Credential stuffing через веб-додаток. Зловмисник використовує базу з 50 мільйонів пар логін/пароль, отриману з витоків різних сайтів, і автоматично перевіряє їх проти форми входу корпоративного порталу. Оскільки він використовує різні IP-адреси і коректні облікові дані, Fail2ban не виявляє атаку — з точки зору системи відбуваються «звичайні» входи (успішні) і «звичайні» невдалі спроби (від тих пар, що не підходять). Виявлення credential stuffing потребує аналізу поведінкових аномалій: незвична географія входу для конкретного користувача, вхід з нового пристрою, незвичний час активності. Сучасні веб-додатки використовують механізми fingerprinting браузера і поведінкового аналізу для відрізнення автоматизованих запитів від реальних користувачів. Ефективним протизаходом є впровадження MFA — навіть при правильному паролі зловмисник не матиме доступу до другого фактора, що повністю нейтралізує цей тип атаки. Також доречним є підключення до сервісів на кшталт HaveIBeenPwned API, що дозволяє під час реєстрації або зміни пароля перевіряти, чи не є він частиною відомих витоків, і блокувати використання скомпрометованих паролів превентивно.

Розглянуті три сценарії наочно демонструють, що не існує єдиного «срібного патрона» у захисті від атак на автентифікацію. Кожен тип атаки вимагає власного набору контрзаходів, а найбільш ефективна стратегія — це поєднання кількох незалежних механізмів захисту, що компенсують слабкі сторони один одного. Fail2ban відмінно справляється з класичним brute force,

але безсилий проти розподіленої атаки з унікальними IP; rate limiting допомагає проти ботнетів, але не захищає від credential stuffing; MFA є найбільш комплексним захистом, але вимагає залучення користувачів і відповідної інфраструктури. Тільки їх спільне застосування утворює достатньо міцний захисний периметр.

Тема 6. Мережеві фаєрволи та фільтрація трафіку в операційних системах

6.1. Поняття мережевого фаєрволу та архітектура фільтрації трафіку

Мережевий фаєрвол (від англ. firewall — вогненна стіна) є одним із ключових засобів забезпечення безпеки сучасних інформаційних систем. Ця назва має історичне походження: у будівельній справі вогнетривка стіна слугує бар'єром між відсіками будівлі та запобігає поширенню пожежі. У контексті мережевої безпеки фаєрвол виконує аналогічну функцію — він є бар'єром між мережами або між мережею й вузлом, крізь який проходить лише авторизований трафік, тоді як потенційно небезпечний блокується. Цей засіб контролює весь мережевий трафік, що входить до захищеної системи або виходить із неї, і приймає рішення про дозвіл або блокування окремих пакетів чи з'єднань на підставі наперед визначених правил. Фаєрвол є невід'ємним елементом будь-якої сучасної системи захисту інформації, а відсутність його або неправильна конфігурація відкриває зловмисникам широкі можливості для атаки.

Слід чітко розрізняти два основних різновиди фаєрволів за місцем розгортання: апаратні й програмні. Апаратний фаєрвол — це спеціалізований мережевий пристрій, зазвичай маршрутизатор із вбудованими функціями фільтрації, що встановлюється на периметрі мережі та захищає одразу всі хости всередині неї. Такий підхід типовий для корпоративних мереж, де потрібна централізована точка контролю. Програмний фаєрвол встановлюється безпосередньо на операційну систему конкретного вузла та захищає саме цей вузол. В операційних системах Лінукс програмний фаєрвол реалізований у вигляді підсистеми ядра Netfilter, інтерфейсом до якої для користувача слугує утиліта iptables або сучасніший nftables. Обидва підходи не є взаємовиключними: сучасна архітектура безпеки передбачає їх поєднання за принципом глибокого захисту (defense in depth), коли навіть після подолання периметрового захисту зловмисник стикається з додатковим бар'єром на рівні кожного хоста.

Архітектурно фільтрація трафіку в Лінукс побудована на концепції «гаків» (hooks) — точок у стеку мережевих протоколів ядра, у яких підсистема Netfilter може перехоплювати пакети і застосовувати до них правила. Весь мережевий трафік, що проходить через систему або породжується нею, обов'язково проходить крізь певну послідовність таких точок. Пакет, що надходить з мережі та адресований локальному процесу, проходить точки PREROUTING і INPUT. Пакет, що генерується локальним процесом і надсилається в мережу, проходить OUTPUT і POSTROUTING. Якщо ж система виконує функцію маршрутизатора і пакет лише транзитно проходить через неї, він обробляється в точках PREROUTING, FORWARD і POSTROUTING. Розуміння цього шляху є принциповим, оскільки правила потрібно встановлювати саме в тих точках, через які проходитиме трафік, що підлягає контролю.

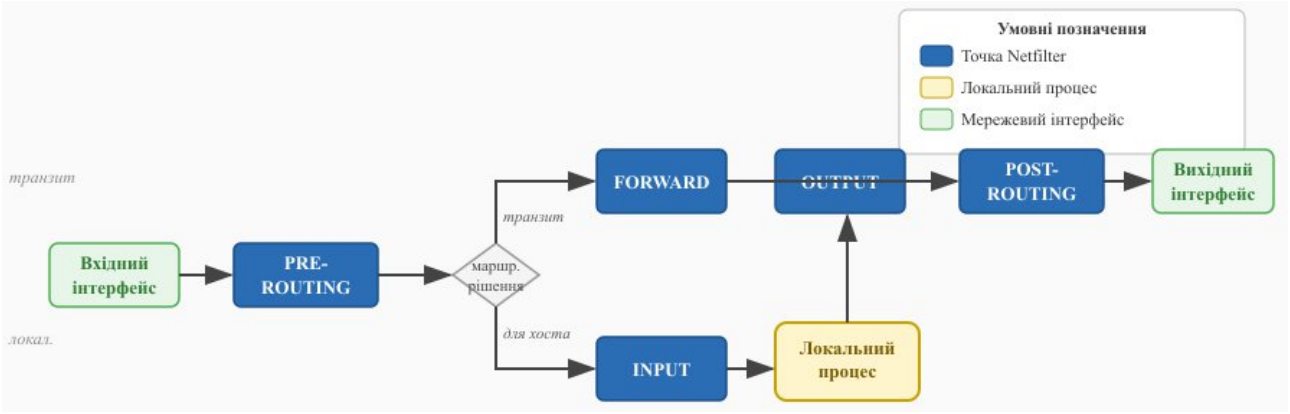

Рисунок 6.1 — Шлях мережевого пакета крізь підсистему Netfilter в Лінукс

На рисунку 6.1 зображена схема руху мережевого пакета у вигляді горизонтальної блок-схеми.

Для практичного розуміння важливо також усвідомити різницю між фаєрволом та суміжними поняттями. Система виявлення вторгнень (IDS) лише аналізує трафік і сигналізує про підозрілу активність, але сама нічого не блокує. Фаєрвол, навпаки, є активним засобом: він не лише виявляє, а й перешкоджає проходженню небажаного трафіку. Проксі-сервер діє на рівні додатку, розриває з'єднання між клієнтом і сервером та самостійно встановлює нове, тобто фактично замінює пряму комунікацію опосередкованою. Класичний мережевий фаєрвол натомість працює переважно на мережевому й транспортному рівнях моделі OSI, не перериваючи з'єднання, а лише вирішуючи долю кожного пакета. Сучасні рішення класу Next-Generation Firewall (NGFW) поєднують можливості традиційного фаєрволу з глибоким аналізом пакетів (DPI), інспекцією зашифрованого трафіку та функціями IDS/IPS, однак базова логіка фільтрації залишається незмінною і є фундаментом, на якому будується будь-яке розширення.

Окрему увагу варто приділити еволюції інструментів керування фаєрволом у Лінукс. Протягом тривалого часу стандартом де-факто залишається утиліта iptables, що з'явилася близько 2000 року і донині підтримується в усіх дистрибутивах. Утиліта nftables, яка була розроблена як її спадкоємець і почала активно впроваджуватися з 2014 року, пропонує уніфікований синтаксис, вищу продуктивність і зручніший механізм управління, однак iptables залишається широко поширеною і надалі підтримується навіть у нових системах через рівень сумісності. В корпоративному середовищі, де застосовуються дистрибутиви на основі Red Hat, поширеним є також інструмент firewalld, що надає зручний інтерфейс на основі зон безпеки і можна управляти як через командний рядок, так і через графічний інтерфейс. Для розуміння принципів роботи фаєрволу важливо вивчити саме iptables, оскільки її концепції — таблиці, ланцюги, правила і дії — є спільними для всіх сучасних реалізацій, а перехід від iptables до nftables або firewalld є значно простішим, ніж вивчення кожного інструменту з нуля. Нижче, в таблиці 6.1, наведено порівняльну характеристику трьох основних інструментів керування фаєрволом у Лінукс.

Таблиця 6.1 — Порівняльна характеристика інструментів керування фаєрволом у Лінукс

Характеристика	iptables	nftables	firewalld
Рік появи	~2000	2014	2011
Рівень складності синтаксису	Середній	Спрощений, уніфікований	Простий (зони)
Продуктивність	Середня	Висока	Середня
Підтримка IPv4 та IPv6	Окремі утиліти (iptables/ip6tables)	Єдина утиліта	Єдина утиліта
Підтримка дистрибутивів	Усі Лінукс	Сучасні Лінукс (5.x+)	Переважно RHEL/Fedora
Збереження правил	Вручну або iptables-save	Вручну або nft list	Автоматично через служби
Типове застосування	Загальна фільтрація, навчання	Нові сервери та маршрутизатори	Робочі станції, RHEL-сервери

6.2. Типи фільтрації пакетів. Таблиці та ланцюги правил

Фільтрація мережевого трафіку може здійснюватися на основі різних атрибутів пакетів, і від обраного типу фільтрації суттєво залежать як можливості захисту, так і витрати на його реалізацію. Найпростішою є безстанова (stateless) фільтрація, за якої кожен пакет аналізується незалежно від інших — лише на підставі інформації, що міститься в його власних заголовках. Правила безстанової фільтрації оперують такими атрибутами, як IP-адреса відправника, IP-адреса одержувача, протокол транспортного рівня (TCP, UDP, ICMP), номер порту відправника й одержувача, а також прапорці TCP (SYN, ACK, FIN тощо). Перевагою цього підходу є його простота й висока швидкість обробки, адже система не зберігає жодного внутрішнього стану між пакетами. Недоліком є неможливість відрізнити «перший» пакет нового з'єднання від відповіді на вже встановлене з'єднання, що ускладнює написання коректних правил і відкриває можливості для обходу захисту через маніпуляції з прапорцями TCP.

Більш потужним і безпечним є підхід на основі відстеження стану з'єднання — так звана stateful-фільтрація. У цьому випадку підсистема фаєрволу веде внутрішню таблицю відстеження з'єднань (conntrack в Лінукс), яка фіксує стан кожного активного з'єднання. Можливі стани: NEW (перший пакет нового з'єднання), ESTABLISHED (пакет належить вже встановленому з'єднанню), RELATED (пакет пов'язаний із наявним з'єднанням, наприклад, пакет помилки ICMP у відповідь на TCP-запит), INVALID (пакет, що не відповідає жодному відомому стану). Це дозволяє писати значно компактніші й

надійніші правила: наприклад, дозволити весь вхідний трафік зі станами ESTABLISHED та RELATED і блокувати всі нові вхідні з'єднання, які не були явно дозволені. Саме stateful-фільтрація є стандартом у сучасних конфігураціях. Таблиця 6.2 містить порівняння двох підходів.

Таблиця 6.2 — Порівняльна характеристика типів фільтрації мережевого трафіку

Характеристика	Безстанова (stateless)	Стан-орієнтована (stateful)
Збереження стану з'єднань	Ні	Так (таблиця conntrack)
Критерії фільтрації	IP, порт, протокол, прапорці	IP, порт, протокол, стан, час
Продуктивність	Висока (менше пам'яті)	Дещо нижча (витрати на conntrack)
Складність правил	Висока (потрібні парні правила)	Нижча (автоматичний облік відповідей)
Захист від спуфінгу	Обмежений	Кращий (перевірка стану)
Типове застосування	Маршрутизатори, прості ACL	Хостові фаєрволи, периметрові шлюзи

В утиліті iptables правила фільтрації організовані у вигляді ієрархічної структури з таблиць і ланцюгів. Таблиця (table) — це логічна група ланцюгів, призначених для певного типу обробки пакетів. Основних таблиць чотири: filter (фільтрація трафіку), nat (трансляція адрес), mangle (модифікація заголовків пакетів) і raw (особливі позначки, що дозволяють уникнути відстеження стану для певних пакетів). Для більшості завдань захисту використовується таблиця filter, яка є таблицею за замовчуванням. Ланцюг (chain) — це впорядкований список правил всередині таблиці, що застосовується до пакетів, які проходять через відповідну точку Netfilter. Таблиця filter містить три вбудовані ланцюги: INPUT, FORWARD і OUTPUT. Таблиця nat — ланцюги PREROUTING, OUTPUT і POSTROUTING. Взаємозв'язок таблиць і ланцюгів із точками Netfilter зображено на рисунку 6.2.

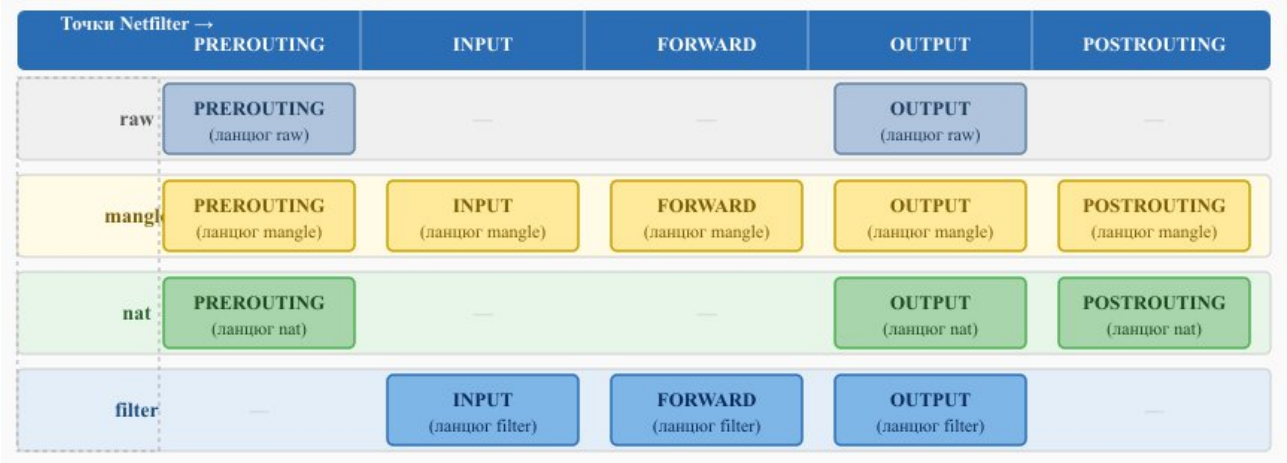

Рисунок 6.2 — Структура таблиць та ланцюгів iptables і їх відповідність точкам Netfilter

На рисунку 6.2 подана таблично-схематична ілюстрація. У верхньому рядку горизонтально розміщені п'ять точок Netfilter: PREROUTING, INPUT, FORWARD, OUTPUT, POSTROUTING — кожна у власній синій комірці. Нижче розміщено чотири горизонтальні смуги, що відповідають чотирьом таблицям iptables: raw (сіра), mangle (жовта), nat (зелена), filter (блакитна). У кожній смузі-рядку таблиці підсвічені лише ті стовпці (точки Netfilter), в яких дана таблиця має вбудований ланцюг. Наприклад, таблиця filter підсвічена тільки в стовпцях INPUT, FORWARD і OUTPUT. Таблиця nat — у PREROUTING, OUTPUT та POSTROUTING. Таблиця raw — у PREROUTING та OUTPUT. Таблиця mangle — у всіх п'яти точках. Під кожною підсвіченою коміркою у дужках вказано назву відповідного ланцюга. Схема дозволяє наочно бачити, в яких точках кожна таблиця «чекає» на пакет.

Механізм обробки пакетів в iptables є послідовним: пакет порівнюється з правилами ланцюга по черзі — від першого до останнього, і як тільки пакет відповідає умові правила, застосовується вказана дія (target), після чого перебір зазвичай зупиняється. Окрім вбудованих ланцюгів, в iptables можна створювати власні (користувацькі) ланцюги та переходити до них із правил вбудованих ланцюгів за допомогою дії JUMP (або скорочено -j із назвою ланцюга). Це суттєво спрощує організацію складних наборів правил: наприклад, можна створити окремі ланцюги для обробки вхідного HTTP-трафіку, SSH-трафіку та трафіку бази даних, і переходити до них із ланцюга INPUT залежно від порту призначення. Такий підхід полегшує підтримку конфігурації та зменшує кількість зайвих порівнянь. Якщо пакет пройшов весь користувацький ланцюг і жодне правило не спрацювало, виконується дія RETURN — пакет повертається до батьківського ланцюга і продовжує порівнюватися з наступними після JUMP правилами.

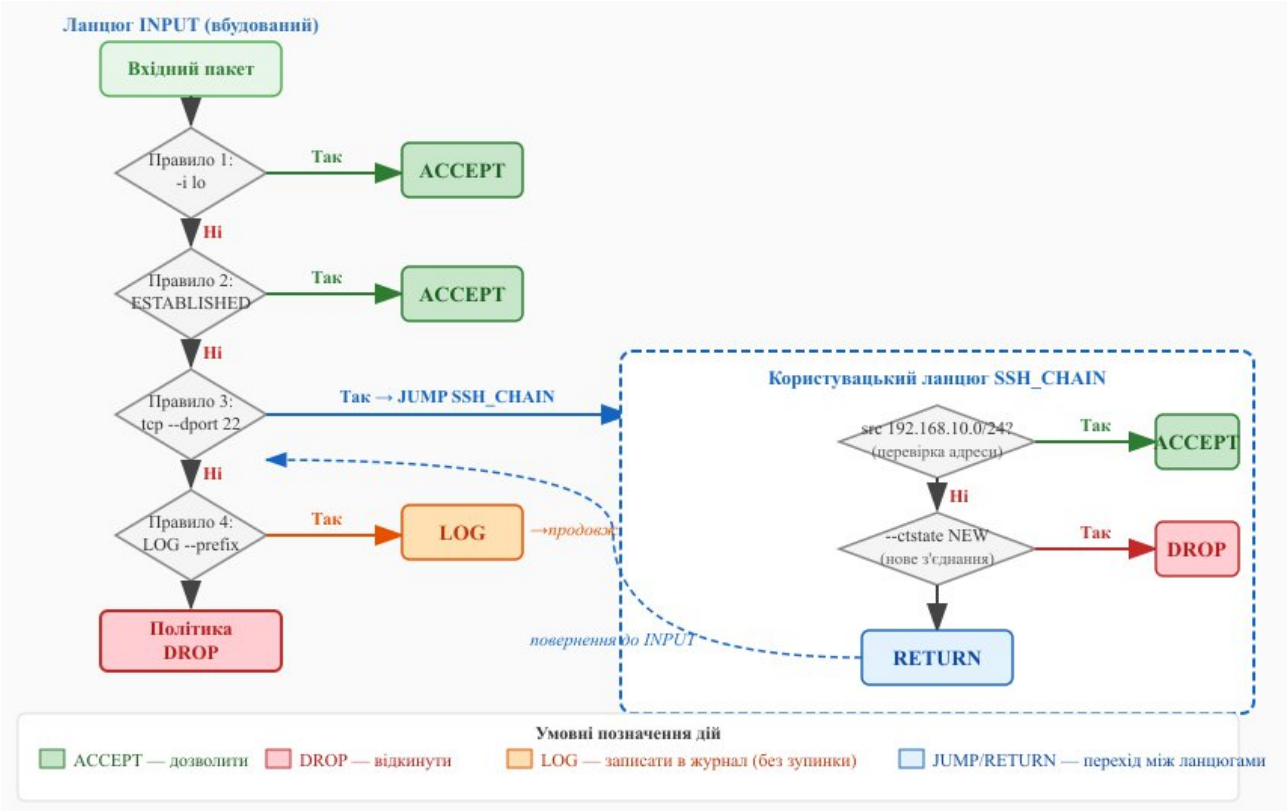

Рисунок 6.3 ілюструє механізм послідовного перебору правил у ланцюзі та варіанти переходу до користувацького ланцюга

На рисунку 6.3 зображена вертикальна блок-схема. Зверху вниз розміщені ромбоподібні блоки прийняття рішень: «Правило 1 відповідає?», «Правило 2 відповідає?» і так далі до «Правило N відповідає?». Від кожного ромба ліворуч виходить стрілка «Ні», що веде до наступного ромба нижче. Від кожного ромба праворуч виходить стрілка «Так», що веде до прямокутника з назвою дії: від «Правила 1» — до «ACCEPT», від «Правила 2» — до «JUMP → SSH_CHAIN» (позначено переходом у бік, де зображено вкладений підланцюг із власними правилами та результатом «ACCEPT / RETURN»), від «Правила 3» — до «LOG», від «Правила 4» — до «DROP». Унизу, після останнього ромба, розміщено прямокутник «Політика за замовчуванням (ACCEPT або DROP)», до якого веде стрілка «Ні» від останнього правила. Кольорове кодування: зелені блоки — ACCEPT, червоні — DROP/REJECT, жовті — LOG, блакитні — JUMP/RETURN.

У таблиці 6.3 систематизовано основні дії (target), які можна застосовувати в правилах iptables. Знання можливих дій і розуміння різниці між ними є необхідною умовою написання коректних і ефективних конфігурацій фаєрволу.

Таблиця 6.3 — Основні дії (target) в iptables та їх характеристики

Дія (target)	Що відбувається з пакетом	Чи зупиняється обробка	Типове застосування
ACCEPT	Пакет	Так (у поточній	Явний дозвіл

	пропускається далі	таблиці)	легітимного трафіку
DROP	Пакет відкидається мовчки	Так	Блокування небажаного трафіку без повідомлення відправника
REJECT	Пакет відкидається з надсиланням ICMP-повідомлення або TCP RST відправнику	Так	Відмова у доступі із поверненням помилки клієнту
LOG	Інформація про пакет записується до системного журналу	Ні (обробка продовжується)	Аудит, діагностика, фіксація підозрілого трафіку
RETURN	Пакет повертається до батьківського ланцюга	Ні (у поточному ланцюзі)	Завершення обробки в користувацькому ланцюзі
JUMP (назва ланцюга)	Обробка передається до зазначеного ланцюга	Ні	Структурування правил через користувацькі ланцюги
MASQUERADE / SNAT	IP-адреса джерела замінюється на адресу вихідного інтерфейсу	Так	NAT для виходу в Інтернет (таблиця nat)
DNAT	IP-адреса (та/або порт) призначення замінюється	Так	Перенаправлення портів / Port Forwarding (таблиця nat)

Загальноприйнятою практикою безпеки є встановлення для вхідного ланцюга INPUT політики DROP з подальшим явним дозволом лише необхідних видів трафіку. Такий підхід реалізує принцип «заборонено все, що не дозволено явно», що є значно безпечнішим за протилежний підхід «дозволено все, що не заборонено явно». Щоб проілюструвати практичну роботу з iptables, розглянемо типову базову конфігурацію хостового фаєрволу для сервера. Спочатку встановлюємо жорсткі політики за замовчуванням: iptables -P INPUT DROP і iptables -P FORWARD DROP забороняють весь вхідний і транзитний трафік, тоді як iptables -P OUTPUT ACCEPT дозволяє вихідний трафік. Далі дозволяємо трафік через петлевий інтерфейс lo, який необхідний для коректної роботи локальних служб: iptables -A INPUT -i lo -j ACCEPT. Потім дозволяємо увесь трафік, що належить вже встановленим і пов'язаним з'єднанням: iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT. Нарешті, дозволяємо нові вхідні підключення лише на потрібні порти, наприклад SSH: iptables -A INPUT -p tcp --dport 22 -m conntrack --ctstate NEW -j ACCEPT.

6.3. Трансляція мережевих адрес (NAT) та її роль у безпеці

Трансляція мережевих адрес, відома за абревіатурою NAT (Network Address Translation), є механізмом, за якого маршрутизатор або фаєрвол змінює адресну інформацію в заголовках IP-пакетів під час їх проходження. Найпоширенішим різновидом є масквардингова (masquerade) або PAT (Port Address Translation) форма NAT, за якої багато внутрішніх хостів приватної мережі виходять в Інтернет через одну публічну IP-адресу. При цьому маршрутизатор зберігає таблицю відповідностей між внутрішніми адресами й портами та зовнішніми портами, щоб правильно доставляти відповіді тому хосту, що ініціював з'єднання. Саме ця технологія є причиною того, що мільярди пристроїв домашніх і корпоративних мереж можуть одночасно мати доступ до Інтернету через обмежений пул публічних IPv4-адрес, і вона продовжує широко застосовуватися навіть в епоху IPv6, де теоретично вистачає адрес для кожного пристрою.

З точки зору безпеки NAT забезпечує важливий побічний захисний ефект: хости всередині приватної мережі стають недоступними для прямого звернення ззовні, оскільки мають лише приватні адреси (з діапазонів 10.0.0.0/8, 172.16.0.0/12 або 192.168.0.0/16), що не маршрутизуються в глобальній мережі. Зовнішній зловмисник, навіть знаючи, що за NAT-шлюзом є внутрішні хости, не може встановити з ними пряме з'єднання без попередньо налаштованого перенаправлення портів. Проте слід чітко розуміти, що NAT сам по собі не є засобом безпеки і не може замінити повноцінний фаєрвол: він не перевіряє вміст пакетів, не може заблокувати шкідливий трафік, ініційований зсередини мережі, і не захищає від атак через дозволені порти. NAT — це насамперед технологія економії адресного простору, і її захисний ефект є лише корисним побічним наслідком.

В iptables NAT налаштовується через окрему таблицю nat, яка містить ланцюги PREROUTING, OUTPUT і POSTROUTING. Для реалізації стандартного NAT типу «багато до одного» потрібно додати правило в ланцюг POSTROUTING таблиці nat: iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE, де eth0 — вихідний мережевий інтерфейс. Ця команда вказує системі замінювати адресу відправника у всіх пакетах, що виходять через eth0, на зовнішню адресу цього інтерфейсу. Для того щоб система могла пересилати пакети між інтерфейсами, також необхідно увімкнути пересилання пакетів на рівні ядра командою echo 1 > /proc/sys/net/ipv4/ip_forward, або встановити відповідний параметр у файлі /etc/sysctl.conf для збереження налаштування після перезавантаження.

Іншим важливим застосуванням NAT є перенаправлення портів (Port Forwarding, або DNAT — Destination NAT). Ця техніка дозволяє зробити певний сервіс, що працює на внутрішньому хості за NAT, доступним ззовні через певний порт на зовнішній адресі шлюзу. Наприклад, якщо веб-сервер працює на внутрішньому хості з адресою 192.168.1.10, можна налаштувати шлюз так, щоб усі вхідні з'єднання на порт 80 зовнішньої адреси перенаправлялися на цей внутрішній хост командою iptables -t nat -A

PREROUTING -p tcp --dport 80 -j DNAT --to-destination 192.168.1.10:80. У таблиці 6.4 систематизовано основні сценарії застосування NAT.

Таблиця 6.4 — Типи NAT та їх застосування в iptables

Тип NAT	Ланцюг / таблиця	Типовий сценарій застосування
SNAT / Masquerade	POSTROUTING / nat	Спільний вихід внутрішніх хостів в Інтернет через один публічний IP
DNAT (Port Forwarding)	PREROUTING / nat	Публікація внутрішнього сервісу назовні через певний порт шлюзу
Redirect	PREROUTING / nat	Перенаправлення трафіку на інший порт (наприклад, для прозорого проксі)
Hairpin NAT	PREROUTING + POSTROUTING / nat	Доступ до опублікованого сервісу з внутрішньої мережі через зовнішній IP шлюзу

Важливо розуміти взаємодію між правилами NAT і правилами фільтрації в iptables. Оскільки Netfilter застосовує таблиці в певному порядку, правила NAT у ланцюзі PREROUTING виконуються до того, як пакет потрапляє до ланцюга FORWARD таблиці filter. Це означає, що при написанні правил фільтрації для перенаправленого трафіку потрібно вказувати вже змінені (внутрішні) адреси, а не оригінальні зовнішні. Нехтування цим нюансом є поширеною помилкою: DNAT-правило спрацьовує, але пакет потім блокується фільтром — і в результаті сервіс виявляється недоступним попри, здавалося б, правильно налаштоване перенаправлення. Окрім технічних помилок налаштування, слід також усвідомлювати фундаментальні обмеження захисту, яке надає NAT. По-перше, NAT не захищає від атак, ініційованих зсередини мережі: якщо внутрішній хост заражений шкідливим програмним забезпеченням, воно може вільно надсилати трафік назовні через NAT-шлюз, і цей трафік ніяк не відрізнятиметься від легітимного. По-друге, технологія DNS rebinding дозволяє зловмиснику змусити браузер жертви звертатися до внутрішніх ресурсів через підроблені DNS-відповіді, обходячи тим самим захист периметра. По-третє, некоректно налаштоване перенаправлення портів може відкрити зовнішній доступ до внутрішніх служб, що не мали бути публічними. Усе це підкреслює необхідність комплексного підходу, де NAT доповнює фаєрвол, але не замінює його.

Нижче на рисунку 6.4 показана мережева діаграма типового розгортання фаєрволу з NAT для захисту внутрішнього сегмента мережі та публікації веб-сервера назовні.

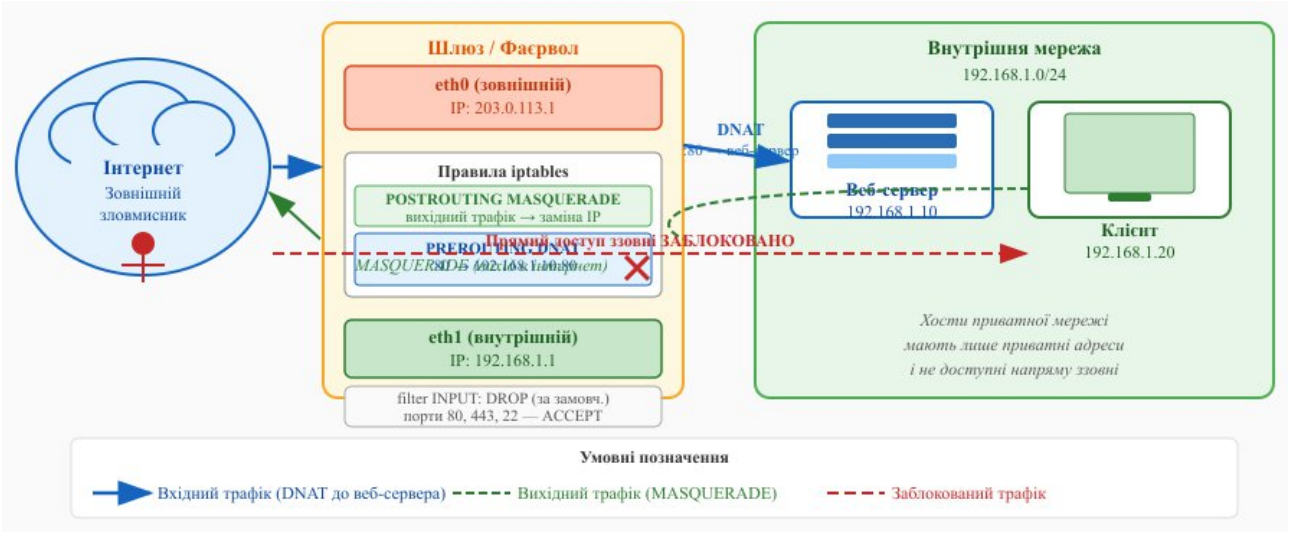

Рисунок 6.4 — Схема розгортання фаєрволу з NAT: захист внутрішньої мережі та публікація веб-сервера

На рисунку 6.4 зображена мережева діаграма з трьома зонами, розміщеними горизонтально зліва направо. Ліворуч — хмара «Інтернет» з позначеним зовнішнім IP-адресою (203.0.113.1). У центрі — прямокутник «Шлюз / Фаєрвол» із двома мережевими інтерфейсами: зовнішнім (eth0, 203.0.113.1) і внутрішнім (eth1, 192.168.1.1). Усередині блоку шлюзу схематично позначено дві групи правил: «POSTROUTING MASQUERADE» (для вихідного трафіку внутрішньої мережі) та «PREROUTING DNAT :80 → 192.168.1.10» (для перенаправлення вхідного веб-трафіку). Праворуч від шлюзу розміщено «Внутрішню мережу» (192.168.1.0/24), що містить два вузли: «Клієнт» (192.168.1.20) та «Веб-сервер» (192.168.1.10). Стрілки показують три потоки трафіку: (1) від Клієнта через MASQUERADE до Інтернету — зелена стрілка; (2) із Інтернету через DNAT до Веб-сервера — синя стрілка; (3) заблокований прямий вхідний трафік до Клієнта — червона стрілка з хрестиком.

6.4. Логування мережевих подій та підготовка до аналізу інцидентів

Навіть найдосконаліший фаєрвол буде неповноцінним засобом захисту без належної системи логування. Журнали мережевих подій виконують декілька критично важливих функцій. По-перше, вони є основою для виявлення спроб несанкціонованого доступу: аналіз журналів дозволяє помітити підозрілі патерни ще до того, як зловмисник досягне своєї мети. По-друге, журнали є необхідним джерелом доказів при розслідуванні інцидентів: без них неможливо встановити, коли і звідки було здійснено атаку, які дії виконував зловмисник у системі і які дані могли бути скомпрометовані. По-третє, регулярний аналіз журналів дозволяє оцінювати ефективність правил фаєрволу й виявляти ситуації, коли правила або надмірно обмежують легітимний трафік, або пропускають щось зайве. Нарешті, в ряді регуляторних вимог і стандартів

(наприклад, PCI DSS або ISO 27001) наявність і цілісність журналів мережевих подій є обов'язковою вимогою, невиконання якої тягне за собою санкції.

В iptables логування реалізується через дію LOG, яку можна застосовувати в будь-якому ланцюзі та до будь-якого типу пакетів. На відміну від ACCEPT, DROP чи REJECT, дія LOG не завершує обробку пакета — після запису в журнал пакет продовжує порівнюватися з наступними правилами ланцюга. Тому правило з дією LOG зазвичай розміщується безпосередньо перед правилом, що реально дозволяє або забороняє трафік. Щоб зафіксувати всі заблоковані вхідні пакети, можна додати перед загальним DROP правило: iptables -A INPUT -j LOG --log-prefix "[IPTABLES DROP] " --log-level 4. Параметр --log-prefix задає текстовий префікс, що виводитиметься перед кожним рядком у журналі і суттєво спрощує пошук і фільтрацію потрібних записів. Параметр --log-level визначає пріоритет повідомлення за стандартом syslog (4 відповідає рівню warning). Записи iptables потрапляють до системного журналу — у системах на основі systemd їх можна переглянути командою journalctl -k | grep IPTABLES або через файл /var/log/kern.log.

Кожен рядок журналу iptables містить багату інформацію: час події, мережевий інтерфейс, MAC-адреси, IP-адреси джерела й призначення, TTL, розмір пакета, протокол транспортного рівня, порти джерела й призначення, прапорці TCP та додаткові параметри. Ця сукупність даних дозволяє повністю відтворити картину мережевої активності та є цінним матеріалом для криміналістичного аналізу. Важливо пам'ятати, що надмірне логування може суттєво навантажити систему і швидко заповнити дисковий простір, тому рекомендується логувати лише заблокований або підозрілий трафік, а також застосовувати обмеження частоти записів за допомогою модуля --limit. Наприклад, правило iptables -A INPUT -m limit --limit 5/min -j LOG --log-prefix "[RATE-LIMITED] " дозволить записувати не більше п'яти повідомлень на хвилину, запобігаючи перевантаженню журналу при масованих атаках на сканування портів.

Для ефективного аналізу журналів та своєчасного виявлення інцидентів просте читання файлів журналів є недостатнім. Серед відкритих рішень класу SIEM, що підтримують Лінукс, варто відзначити ELK Stack (Elasticsearch, Logstash, Kibana) та Graylog. ELK Stack працює наступним чином: Logstash збирає та нормалізує журнали з різних джерел, Elasticsearch індексує й зберігає їх у структурованому вигляді, а Kibana надає вебінтерфейс для побудови запитів, дашбордів і графіків. Навіть без повноцінного SIEM корисним є налаштування агрегації журналів на централізований syslog-сервер, що запобігає можливості зловмисника знищити сліди своєї діяльності, маніпулюючи локальними журналами. Рисунок 6.5 ілюструє типову схему організації такої системи.

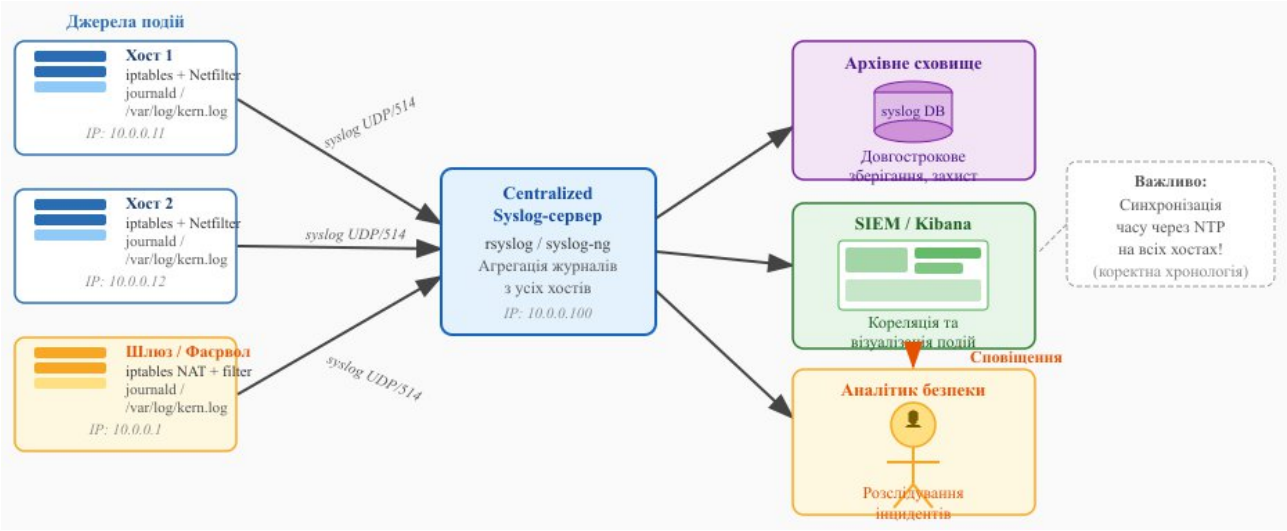

Рисунок 6.5 — Схема організації централізованого логування подій фаєрволу та їх обробки

На рисунку 6.5 зображена схема організації системи логування мережевих подій. У лівій частині схеми вертикально розміщені три хости з iptables (позначені іконками серверів), кожен з яких генерує записи в локальний журнал — journald або /var/log/kern.log. Від кожного хоста горизонтальна стрілка з підписом «syslog / UDP 514» веде до центрального блоку «Централізований syslog-сервер» у середній частині схеми. Від syslog-сервера стрілки розходяться до трьох блоків у правій частині: «Архівне сховище» (циліндр бази даних, підпис «довгострокове зберігання»), «SIEM / Kibana» (монітор із дашбордом, підпис «кореляція та візуалізація подій») і «Аналітик безпеки» (іконка людини). Від блоку «SIEM / Kibana» стрілка з підписом «Сповіщення / Alert» веде до «Аналітика безпеки». Усі з'єднання підписані протоколами або типами даних. Загальний напрямок потоку — зліва направо.

Важливим аспектом підготовки до аналізу інцидентів є не лише наявність журналів, а й забезпечення їх цілісності та незмінності. Якщо зловмисник отримав підвищені привілеї в системі, він може спробувати видалити або підробити журнали. Для протидії цьому варто відразу пересилати журнали на окремий, надійно захищений сервер, доступ до якого з скомпрометованої системи є обмеженим або повністю відсутнім. Також слід налаштувати ротацію журналів (logrotate) для автоматичного архівування і стиснення старих файлів із збереженням їх впродовж визначеного терміну. Критично важливою є також синхронізація системного часу через NTP, оскільки журнали з неправильними часовими мітками унеможливлюють коректне відновлення хронології інциденту.

6.5. Типові помилки конфігурації фаєрволу

Навіть добре розуміючи теоретичні основи фільтрації трафіку, адміністратори нерідко припускаються практичних помилок при налаштуванні фаєрволу, які суттєво знижують рівень захищеності системи. Розгляд типових

помилок є важливою частиною підготовки фахівця з кібербезпеки, адже знання того, «як не треба робити», часто не менш цінне, ніж знання правильного підходу. Перша і найбільш критична помилка — відсутність правила для петлевого інтерфейсу lo. Цей інтерфейс використовується локальними процесами для зв'язку між собою в межах тієї самої системи. Якщо встановлено жорстку політику DROP для ланцюга INPUT і не додано явний дозвіл для lo, багато системних служб перестануть нормально функціонувати: наприклад, може перестати відповідати локальна база даних або системний менеджер служб. Правило iptables -A INPUT -i lo -j ACCEPT є обов'язковим і має бути першим у ланцюзі INPUT.

Друга поширена помилка — неправильний порядок правил у ланцюзі. Оскільки iptables перевіряє правила послідовно і зупиняється на першому відповідному, більш специфічні правила завжди мають розміщуватися перед більш загальними. Класичний приклад: якщо спочатку додати правило, що блокує всі пакети з певної IP-адреси (DROP), а потім — правило, що дозволяє SSH-трафік (ACCEPT) для тієї самої адреси, то SSH-доступ ніколи не буде дозволений, адже пакети будуть відфільтровані ще першим правилом. Третя помилка — відсутність правила LOG перед блокувальними правилами, особливо на початку налаштування. Без логування адміністратор не може розрізнити ситуацію «фаєрвол правильно блокує атаку» від ситуації «фаєрвол блокує легітимний трафік через помилку в правилах». Логування блокованого трафіку під час відлагодження конфігурації є обов'язковим кроком.

Четверта, і дуже небезпечна помилка — блокування власного доступу по SSH при роботі на віддаленому сервері. Якщо адміністратор підключений до сервера через SSH і встановлює політику DROP для INPUT, не додавши спочатку правило, що дозволяє вже встановлені SSH-з'єднання та нові підключення на порт 22, він миттєво втрачає доступ до сервера. В хмарних і віртуальних середовищах відновлення такого доступу потребує використання консолі управління або відновлення з резервної копії. Щоб уникнути цього, слід дотримуватися правила: спочатку додаємо всі дозволяючі правила (особливо для SSH), і лише потім встановлюємо жорстку політику DROP. П'ята помилка — збереження правил лише в оперативній пам'яті без запису на диск. У Лінукс правила iptables за замовчуванням є тимчасовими і зникають після перезавантаження. Для їх збереження необхідно або скористатися утилітою iptables-save та iptables-restore, або встановити і налаштувати пакет iptables-persistent. Нехтування цим кроком призводить до того, що після планового або аварійного перезавантаження сервер залишається повністю незахищеним.

Тема 7. Управління оновленнями, вразливостями та конфігурацією безпеки

7.1. Вразливості програмного забезпечення: поняття, класифікація та системи оцінювання

Будь-яка досить складна програмна система містить помилки. Частина з них є лише функціональними недоліками — програма працює не так, як задумано, але це не несе прямої загрози безпеці. Однак інша частина помилок має принципово інший характер: вони дозволяють зловмисникові впливати на роботу системи у спосіб, який виходить за межі задуманого розробником. Такі помилки називають вразливостями (англ. vulnerabilities). Вразливість — це слабке місце в програмному або апаратному забезпеченні, у налаштуваннях системи чи в організаційних процесах, яке може бути використано для порушення конфіденційності, цілісності або доступності інформації. Важливо розуміти, що вразливість сама по собі не завдає шкоди — вона лише створює умову, за якої шкода стає можливою. Для реалізації загрози необхідне поєднання вразливості, наявності зловмисника та можливості отримати доступ до вразливого компонента.

Вразливості бувають різної природи. Одні виникають унаслідок помилок програмування — наприклад, переповнення буфера, некоректна обробка вхідних даних або відсутність перевірки прав доступу перед виконанням критичної операції. Переповнення буфера є класичним прикладом: якщо програма виділяє буфер фіксованого розміру для вхідних даних і не перевіряє, чи не перевищують вхідні дані цей розмір, зловмисник може надіслати рядок, що «перетікає» за межі буфера і перезаписує сусідні ділянки пам’яті — аж до підміни адреси повернення функції та отримання контролю над виконанням програми. Інші вразливості є наслідком архітектурних рішень: наприклад, протокол, спроєктований без урахування сучасних вимог до автентифікації. Треті — результат неправильного налаштування системи адміністратором. Усі ці типи вразливостей є реальними та поширеними в сучасних системах, і саме тому управління вразливостями є невід’ємною частиною будь-якої системи захисту інформації.

Щоб вразливості можна було відстежувати, аналізувати та усувати у скоординований спосіб, у 1999 році компанія MITRE Corporation запропонувала систему CVE — Common Vulnerabilities and Exposures. CVE є стандартизованим словником публічно відомих вразливостей, кожній з яких присвоюється унікальний ідентифікатор. Ідентифікатор має форму CVE-PPPP-HHHH, де PPPP — рік виявлення або публікації, а HHHH — порядковий номер. Наприклад, CVE-2021-44228 — це відома критична вразливість у бібліотеці Apache Log4j, яку у 2021 році використовували для масових атак на вебсервери по всьому світу. Завдяки CVE фахівці з безпеки, виробники програмного забезпечення та системні адміністратори мають спільну мову для обговорення конкретних проблем без неоднозначності у назвах. Бази даних CVE є загальнодоступними та регулярно поповнюються — їх можна переглянути на

ресурсах MITRE (cve.mitre.org) та NVD (nvd.nist.gov — National Vulnerability Database від NIST).

Однак просто знати про вразливість недостатньо — важливо розуміти, наскільки вона небезпечна, щоб правильно розставити пріоритети при її усуненні. Для цього використовується система CVSS — Common Vulnerability Scoring System. CVSS — це відкрита методологія оцінювання критичності вразливостей за числовою шкалою від 0 до 10. Чим вище значення, тим небезпечніша вразливість: значення від 9,0 до 10,0 відповідають критичному рівню, від 7,0 до 8,9 — високому, від 4,0 до 6,9 — середньому, від 0,1 до 3,9 — низькому. Нульове значення означає відсутність загрози. Поточна актуальна версія CVSS 3.1 враховує кілька груп показників, і саме їх розуміння дозволяє не просто отримати числову оцінку, а й зрозуміти, чому та чи інша вразливість є небезпечною. Таблиця 7.1 ілюструє відповідність числових значень рівням критичності та загальну інтерпретацію.

Таблиця 7.1 — Рівні критичності вразливостей за шкалою CVSS 3.1

Числовий діапазон CVSS	Рівень критичності	Рекомендована пріоритетність реагування
0,0	Відсутній	Не потребує дій
0,1 – 3,9	Низький	Планове усунення
4,0 – 6,9	Середній	Усунення у плановому циклі оновлень
7,0 – 8,9	Високий	Прискорене усунення, тимчасові заходи
9,0 – 10,0	Критичний	Негайне реагування

Базова оцінка CVSS формується на основі характеристик, які не змінюються залежно від середовища. До них належать: вектор атаки (AV — Attack Vector), що описує, звідки може бути здійснена атака — з мережі, з локального сегмента або за умови фізичного доступу до пристрою; складність атаки (AC — Attack Complexity), що відображає, наскільки важко відтворити умови для успішної атаки; необхідність привілеїв (PR — Privileges Required) та взаємодії з користувачем (UI — User Interaction); а також вплив на конфіденційність (C), цілісність (I) та доступність (A) — кожен з параметрів може набувати значень «немає», «частковий» або «повний». Усі ці значення кодуються у компактному векторному рядку. Наприклад, рядок CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H означає: атака можлива з мережі (AV:N), складність низька (AC:L), привілеї не потрібні (PR:N), взаємодія з користувачем не потрібна (UI:N), вплив на конфіденційність, цілісність і доступність — повний (C:H/I:H/A:H). Саме такий вектор має CVE-2021-44228 (Log4Shell), що відповідає оцінці 10,0 — максимально можливій. Поверх базової оцінки CVSS дозволяє враховувати тимчасові фактори — наприклад, чи існує публічно доступний код для експлуатації, — а також контекст конкретної організації: наскільки критичним є уражений компонент у її інфраструктурі.

Щоб краще зрозуміти, як CVSS застосовується на практиці, розглянемо кілька реальних прикладів. Таблиця 7.2 наводить відомі CVE-вразливості різного рівня критичності. Аналізуючи ці приклади, можна помітити закономірність: найвищі оцінки отримують вразливості, які дозволяють виконати довільний код без будь-якої аутентифікації, з мережі, без взаємодії з користувачем — тобто вразливості, атаку на які можна повністю автоматизувати і масово застосовувати по всьому інтернету.

Таблиця 7.2 — Приклади реальних CVE-вразливостей та їх оцінки за CVSS 3.1

Ідентифікатор CVE	Назва / Уражений компонент	Оцінка CVSS	Суть вразливості
CVE-2021-44228	Log4Shell / Apache Log4j	10,0	Виконання довільного коду через підроблений рядок у журналі
CVE-2019-0708	BlueKeep / Windows RDP	9,8	Виконання коду без аутентифікації через протокол RDP
CVE-2021-34527	PrintNightmare / Windows Print Spooler	8,8	Виконання коду з привілеями SYSTEM через службу друку
CVE-2022-0847	Dirty Pipe / Linux kernel	7,8	Запис до файлів, доступних лише для читання, підвищення привілеїв
CVE-2023-44487	HTTP/2 Rapid Reset / вебсервери	7,5	DoS-атака через масові скидання HTTP/2-запитів

Практичне значення CVSS важко переоцінити. Уявімо, що адміністратор отримав сповіщення про 50 вразливостей у системах організації. Без пріоритетизації він не знає, з чого починати. Завдяки оцінкам CVSS стає зрозуміло, що 3 з них є критичними (CVSS 9+) і потребують негайної уваги, 10 — з високим рівнем, які слід усунути протягом кількох днів, а решта можуть зачекати до наступного планового циклу технічного обслуговування. Водночас CVSS відображає теоретичну небезпеку вразливості, але не враховує автоматично специфіку конкретної організації: наскільки критичним є уражений компонент, чи є він взагалі доступним ззовні, які дані він обробляє. Саме тому досвідчені фахівці використовують оцінку CVSS як відправну точку, але коригують пріоритети з урахуванням контексту.

Важливим поняттям, яке варто розрізняти разом із CVE, є поняття «exploit» (засіб експлуатації). Exploit — це конкретний код або послідовність дій, яка використовує вразливість для отримання несанкціонованого доступу або виконання довільного коду. Наявність публічно доступного exploit суттєво підвищує реальну небезпеку вразливості: якщо exploit доступний, будь-хто з мінімальними технічними знаннями може здійснити атаку, не розуміючи

деталей вразливості. Саме тому при прийнятті рішень про пріоритетність усунення вразливостей фахівці орієнтуються не лише на базовий бал CVSS, а й на інформацію про наявність exploit у відкритих джерелах — наприклад, у базі Exploit-DB або на платформі GitHub.

7.2. Процес управління вразливостями та роль оновлень програмного забезпечення

Управління вразливостями — це не одноразова дія, а безперервний процес, що складається з кількох послідовних етапів: виявлення, оцінювання, пріоритетизації, усунення та верифікації. Такий процес необхідний тому, що вразливості з'являються постійно: виробники програмного забезпечення регулярно публікують бюлетені безпеки, незалежні дослідники знаходять нові слабкі місця, а зловмисники постійно вивчають уже відомі. Організація, яка не управляє вразливостями системно, неминуче накопичує так званий «борг безпеки» — набір невиправлених проблем, які рано чи пізно можуть бути використані проти неї. Рисунок 7.1 ілюструє загальну часову послідовність подій, пов'язаних із вразливістю, від її виникнення до усунення.

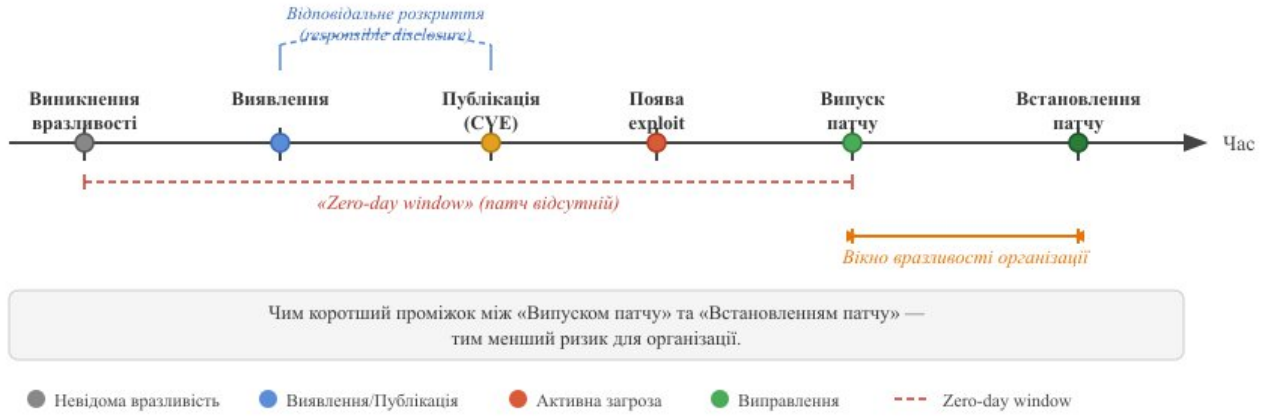

Рисунок 7.1 — Життєвий цикл вразливості від виникнення до усунення

Опис рисунка. Рисунок являє собою горизонтальну часову вісь зліва направо, на якій відзначено шість ключових точок у хронологічному порядку. Перша точка — «Виникнення вразливості»: помилка з'являється в коді або конфігурації, але ще нікому не відома; ця точка може передувати всім іншим на роки. Друга точка — «Виявлення»: дослідник або зловмисник знаходить вразливість. Третя точка — «Публікація (CVE)»: вразливість отримує ідентифікатор і стає публічно відомою; між «Виявленням» і «Публікацією» позначений проміжок відповідального розкриття (responsible disclosure), коли дослідник спочатку повідомляє виробника і дає час підготувати патч. Четверта точка — «Поява exploit»: з'являється публічний код для атаки; відстань від публікації CVE до появи exploit буває дуже малою — іноді кілька годин. П'ята точка — «Випуск патчу»: виробник публікує виправлення. Шоста точка — «Встановлення патчу організацією». Між п'ятою і шостою точками позначено подвійною стрілкою «вікно вразливості організації». Окремо над початком осі

стрілкою виділено «zero-day window» — відрізок від виникнення вразливості до моменту, коли патч ще не існує. У нижній частині рисунка поясне посилання: «Чим коротший проміжок між Випуском патчу та Встановленням — тим менший ризик для організації».

Першим етапом є інвентаризація та виявлення вразливостей. Щоб знати, що захищати, потрібно спочатку знати, що взагалі є в системі: які операційні системи встановлені, яке прикладне програмне забезпечення, які конкретні версії пакетів і бібліотек. Після цього виконується порівняння встановленого ПЗ із базами відомих вразливостей — це може робитися вручну або за допомогою спеціалізованих інструментів сканування. У середовищі Linux таку функцію виконують утиліти на зразок debsecan (для Debian-сумісних систем) або загальніші сканери на зразок OpenVAS. Сканер порівнює версії встановлених пакетів із базою CVE і формує звіт, у якому зазначено, які компоненти є вразливими та який їх бал CVSS. Таблиця 7.3 порівнює основні інструменти, що використовуються на різних етапах управління вразливостями і конфігурацією.

Таблиця 7.3 — Основні інструменти управління вразливостями та конфігурацією у Linux-середовищі

Інструмент	Призначення	Ключові переваги	Обмеження
debsecan	Сканування вразливостей встановлених пакетів	Тісна інтеграція з APT, мінімальне навантаження	Лише для Debian/Ubuntu-сумісних систем
OpenVAS / Greenbone	Комплексне мережеве сканування вразливостей	Велика база перевірок, детальні звіти	Ресурсоємний, потребує окремого сервера
AIDE	Моніторинг цілісності файлової системи	Легкий, швидкий, не потребує мережі	Тільки файлова система, не виявляє вразливості в коді
OpenSCAP	Перевірка відповідності стандартам безпеки (CIS, STIG)	Автоматичні HTML-звіти відповідності	Потребує профілів (XCCDF-файлів)
Ansible	Автоматизація конфігурації та оновлень	Масштабованість, ідемпотентність, без агента на вузлах	Потребує навчання, YAML-синтаксис
unattended-upgrades	Автоматичне встановлення оновлень безпеки	Вбудований у Debian/Ubuntu, простий у налаштуванні	Тільки пакети з репозиторіїв дистрибутива

Після виявлення настає етап оцінювання і пріоритетизації. Не кожна знайдена вразливість є рівнозначно небезпечною для конкретної організації — важливо враховувати контекст. Вразливість у вебсервері, який відкритий в інтернет, є значно більш критичною, ніж аналогічна вразливість у програмі, яка запускається лише адміністратором на ізольованій машині. Тому при пріоритетизації враховуються числовий бал CVSS, наявність відомого exploit, доступність уразливого компонента з мережі та критичність даних, якими цей

компонент оперує. Таблиця 7.4 показує матрицю пріоритетів, якою зручно користуватися під час прийняття рішень про терміни усунення: чим вищий бал CVSS і чим більше сприятливих для зловмисника умов, тим коротший допустимий термін реагування.

Таблиця 7.4 — Матриця пріоритетизації усунення вразливостей

Оцінка CVSS	Є публічний exploit	Компонент доступний з мережі	Пріоритет	Рекомендований термін усунення
9,0 – 10,0	Так	Так	Критичний	До 24 годин, екстрений патч
9,0 – 10,0	Ні	Так	Критичний	До 72 годин
7,0 – 8,9	Так	Так	Високий	До 7 днів
7,0 – 8,9	Ні	Ні	Середній	До 30 днів
4,0 – 6,9	Ні	Ні	Середній	Плановий цикл (30–90 днів)
0,1 – 3,9	Будь-яке	Будь-яке	Низький	Планове усунення або прийняття ризику

Основним і найефективнішим способом усунення вразливостей є встановлення оновлень програмного забезпечення. Коли розробник виявляє вразливість (самостійно або отримавши повідомлення від дослідника безпеки), він розробляє виправлення — patch — і випускає оновлену версію компонента або окремий патч безпеки. Встановлення цього патчу закриває вразливість: зловмисник більше не може скористатися відповідним слабким місцем. Саме тому своєчасне встановлення оновлень є найпростішою та найефективнішою мірою захисту у більшості випадків. У середовищі Linux механізм оновлень тісно пов’язаний із системою пакетного менеджера: команда sudo apt update && sudo apt upgrade спочатку оновлює список доступних версій пакетів у локальному кеші, а потім встановлює виправлені версії. Важливо підтримувати в актуальному стані не лише компоненти операційної системи, а й усі прикладні програми та бібліотеки — адже значна частина реальних атак спрямована саме на вразливості у прикладному ПЗ, а не в ядрі ОС.

Однак встановлення оновлень не завжди можна виконати миттєво. У виробничих середовищах оновлення потребує тестування, оскільки нова версія може порушити роботу залежних компонентів. На цей час між появою вразливості та встановленням патчу організація залишається вразливою. Для зниження ризику в цей проміжний період застосовуються компенсуючі заходи (workarounds або mitigations): вимкнення вразливої функціональності, обмеження мережевого доступу до уразливого компонента за допомогою фаєрвола, тимчасове переведення сервісу в офлайн тощо. Такі заходи не закривають саму вразливість, але суттєво ускладнюють або унеможливлюють її експлуатацію і є стандартною практикою в проміжний час між появою вразливості і встановленням патчу.

Важливою складовою процесу управління оновленнями є перевірка цілісності програмних пакетів перед їх встановленням. Встановлення пакету з ненадійного або скомпрометованого джерела може бути навіть небезпечнішим, ніж відсутність оновлення, — якщо зловмисник підмінив пакет на шкідливу версію. Тому сучасні пакетні менеджери перевіряють цифровий підпис кожного пакету перед його встановленням. Рисунок 7.2 ілюструє, як саме відбувається ця перевірка в ланцюжку між розробником та системним адміністратором.

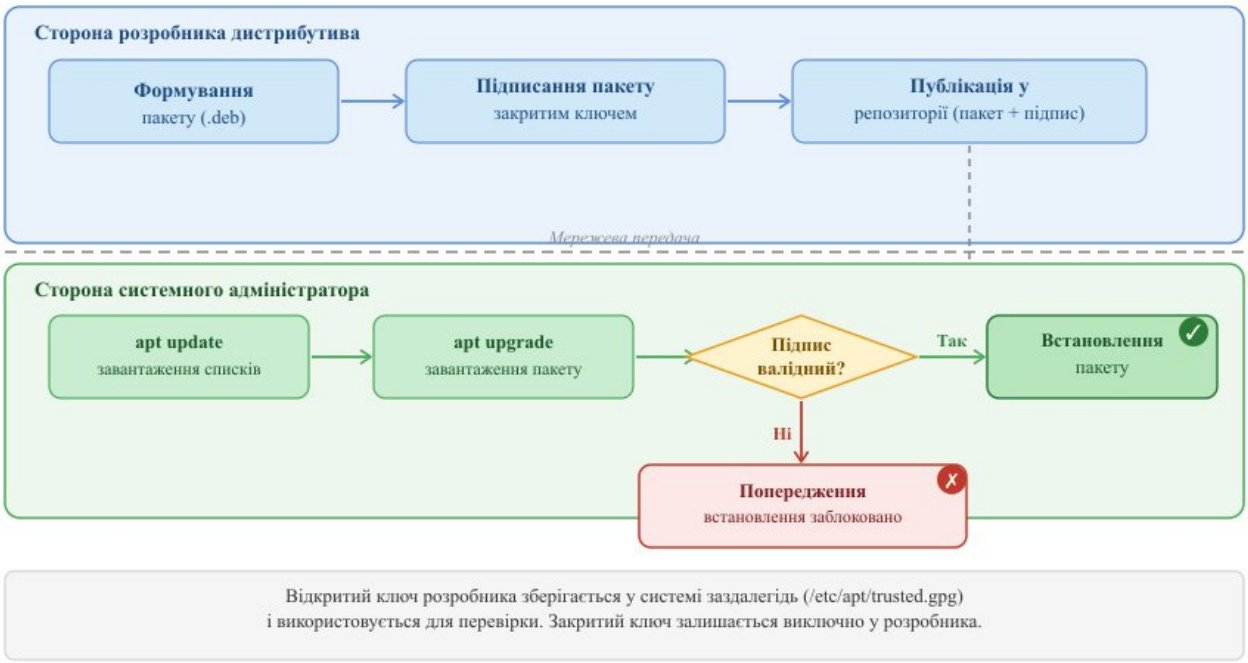

Рисунок 7.2 — Схема перевірки цифрового підпису програмного пакету

Опис рисунка. Рисунок розділений горизонтальною пунктирною лінією на два рівні. Верхній рівень підписаний «Сторона розробника дистрибутива», нижній — «Сторона адміністратора». У верхньому рівні три блоки зі стрілками між ними зліва направо: «Формування пакету (.deb)» → «Підписання пакету закритим ключем» → «Публікація у репозиторії (пакет + підпис)». У нижньому рівні чотири блоки: «apt update: завантаження списків пакетів» → «apt upgrade: завантаження пакету» → ромб «Підпис валідний?». Від ромба ідуть дві гілки: стрілка «Так» веде до блоку «Встановлення пакету (зелений колір)», стрілка «Ні» веде до блоку «Попередження адміністратора, встановлення заблоковано (червоний колір)». Між верхнім і нижнім рівнями є вертикальна двонаправлена стрілка між блоком «Публікація у репозиторії» та блоком «завантаження пакету», підписана «Мережева передача». Окремо внизу підпис: «Відкритий ключ розробника заздалегідь збережений у системі (/etc/apt/trusted.gpg) і використовується для перевірки. Закритий ключ залишається виключно у розробника».

У Linux-системах на базі APT пакети підписуються ключем розробника дистрибутива, і менеджер пакетів перевіряє цей підпис автоматично. Якщо підпис невалідний або відсутній, система попереджає адміністратора і відмовляється встановлювати пакет. Переглянути список довірених ключів

можна командою gpg --list-keys, а перевірити підпис конкретного файлу пакету — командою dpkg-sig --verify пакет.deb. Такий механізм гарантує, що встановлений пакет справді є тим, чим він представляється, і не був модифікований у ланцюжку постачання — від сервера розробника до комп'ютера адміністратора.

На практиці управління оновленнями в масштабах організації вимагає автоматизації. Відстежувати вручну наявність патчів для десятків або сотень систем нереально. У Linux-середовищах для автоматичного встановлення оновлень безпеки використовується пакет unattended-upgrades, який налаштовується так, щоб встановлювати лише оновлення з репозиторію безпеки дистрибутива, не чіпаючи функціональних оновлень. Для більш складних сценаріїв — централізованого керування конфігурацією та розгортання оновлень на багатьох серверах одночасно — застосовуються інструменти на зразок Ansible. Наприклад, простий Ansible-сценарій для встановлення оновлень безпеки матиме такий вигляд:

```
- name: Встановити оновлення безпеки
  hosts: all
  become: yes
  tasks:
    - name: Оновити список пакетів
      apt:
        update_cache: yes

    - name: Встановити лише оновлення безпеки
      apt:
        upgrade: yes
        default_release: "{{ ansible_distribution_release }}-security"
```

Цей сценарій застосовується до всіх серверів організації одночасно, виконує оновлення безпечним способом і повертає звіт про результати виконання. Головна перевага такого підходу — повна відтворюваність і централізованість: замість того щоб підключатися до кожного сервера окремо, адміністратор запускає один сценарій і бачить зведений результат для всієї інфраструктури.

Особливим випадком є так звані вразливості нульового дня (zero-day). Це вразливості, для яких на момент виявлення ще не існує виправлення з боку виробника. Назва «нульовий день» означає, що у виробника є нуль днів, щоб підготувати захист, — а зловмисник вже активно використовує вразливість. Такі вразливості є **особливо** небезпечними, оскільки традиційний підхід «встановіть патч» у цьому випадку неможливий. Захист від zero-day будується на компенсуючих заходах, системах виявлення аномалій та принципах глибокого ешелонованого захисту, які дозволяють мінімізувати збиток навіть у разі успішної експлуатації вразливості. Після встановлення оновлень необхідним кроком є верифікація: перевірити, що патч справді застосований, можна командами dpkg -l назва_пакету або apt-cache policy назва_пакету. Без верифікації існує ризик, що оновлення не застосувалося через технічну

помилку або конфлікт залежностей і адміністратор буде помилково вважати систему захищеною.

Практичний приклад: інцидент Log4Shell (CVE-2021-44228)

Для кращого розуміння того, наскільки швидко реальна вразливість може перетворитися на масову загрозу, розглянемо конкретний приклад — одну з найбільш резонансних вразливостей останніх років. Apache Log4j — це надзвичайно поширена бібліотека журналювання для Java-застосунків, що використовується в тисячах продуктів: від вебсерверів та ігрових платформ до корпоративного програмного забезпечення. У грудні 2021 року дослідники виявили, що бібліотека виконує спеціальні команди, вбудовані у рядки, які вона записує до журналу. Це означало: якщо зловмисник міг змусити застосунок записати певний рядок до журналу — наприклад, через поле імені користувача або заголовок HTTP-запиту — Log4j автоматично завантажував і виконував довільний Java-клас з підконтрольного зловмисникові сервера.

Вразливість отримала оцінку CVSS 10,0 і була публічно розкрита 9 грудня 2021 року. Вже протягом кількох годин після публікації з'явилися перші публічні exploit-коди, а через добу зафіксовано мільйони спроб експлуатації по всьому світу — зловмисники масово сканували інтернет у пошуку вразливих систем і відразу намагалися встановити на них шкідливе програмне забезпечення. Організації, що реагували на вразливість, мали надзвичайно малий часовий проміжок. Правильна реакція полягала у негайному застосуванні компенсуючих заходів ще до виходу офіційного патчу: додавання спеціального параметру запуску JVM (-Dlog4j2.formatMsgNoLookups=true), оновлення правил вебзастосункового фаєрвола для блокування відповідних рядків або тимчасове вимкнення вразливих сервісів. Перші офіційні виправлення від Apache Foundation вийшли протягом двох діб. Цей приклад наочно демонструє, чому комплексний процес управління вразливостями є критично важливим: організації, що мали чіткий план реагування, компенсуючі заходи та налагоджений моніторинг, змогли суттєво скоротити вікно вразливості порівняно з тими, хто не мав жодного формалізованого процесу.

7.3. Управління конфігурацією як складова підтримки безпечного стану системи

Навіть якщо у системі встановлені всі наявні патчі безпеки, це ще не гарантує її захищеності. Значна частина реальних зломів відбувається не через невиправлені вразливості, а через неправильне налаштування системи — так звані «небезпечні конфігурації» (misconfigurations). Наприклад, сервер баз даних, доступний з інтернету без аутентифікації; вебсервер, де увімкнений режим відображення переліку файлів у директоріях; служба SSH, що дозволяє вхід від імені суперкористувача root через мережу — все це є типовими проблемами конфігурації, які жодний патч не виправить, оскільки вони є результатом свідомих (нехай і необережних) рішень адміністратора. Управління конфігурацією безпеки — це систематичний процес приведення та

підтримання налаштувань систем у відповідності до встановлених стандартів безпеки.

Відправною точкою для управління конфігурацією є базові лінії безпеки (security baselines) — задокументовані набори налаштувань, яким повинна відповідати система. Для більшості поширених операційних систем і програмних платформ такі базові лінії вже розроблені й опубліковані авторитетними організаціями. Найвідомішим джерелом є CIS Benchmarks — детальні рекомендації з конфігурації, що видаються Центром безпеки в Інтернеті (Center for Internet Security). CIS Benchmarks існують для Linux, Windows, різних вебсерверів, СУБД, хмарних платформ тощо, і кожна рекомендація супроводжується поясненням ризику та конкретною інструкцією із застосування. Паралельно NIST публікує стандарти серії SP 800, зокрема SP 800-123 «Guide to General Server Security», які охоплюють загальні принципи безпечного налаштування серверів. В Україні у сфері захисту інформації застосовуються ДСТУ, що відповідають або гармонізовані з міжнародними стандартами, зокрема ДСТУ ISO/IEC 27001 щодо систем управління інформаційною безпекою та пов’язані з ним стандарти серії ISO/IEC 27000.

Практика зміцнення системи (hardening) полягає у послідовному застосуванні рекомендацій базових ліній: вимкненні непотрібних служб і протоколів, налаштуванні коректних дозволів на файли та директорії, закритті зайвих мережевих портів, виключенні дефолтних облікових записів або зміні їхніх паролів, обмеженні прав виконання для звичайних користувачів. Кожен з цих кроків є конкретним і перевіряємим: можна зробити знімок стану системи до та після hardening і порівняти їх. Таблиця 7.5 є чек-листом базового зміцнення Linux-сервера, який можна використовувати як відправну точку при введенні нової системи в експлуатацію.

Таблиця 7.5 — Чек-лист базового зміцнення Linux-сервера

№	Дія	Команда або файл	Пріоритет
1	Встановити всі доступні оновлення безпеки	sudo apt update && sudo apt upgrade	Критичний
2	Заборонити вхід через SSH від імені root	PermitRootLogin no у /etc/ssh/sshd_config	Критичний
3	Вимкнути аутентифікацію за паролем у SSH (лише ключі)	PasswordAuthentication no у /etc/ssh/sshd_config	Критичний
4	Налаштувати фаєрвол та заблокувати зайві порти	sudo ufw enable, sudo ufw allow для потрібних портів	Високий
5	Вимкнути непотрібні служби	sudo systemctl disable назва_служби	Високий
6	Встановити суворі права на конфігураційні файли	sudo chmod 600 /etc/ssh/sshd_config	Високий
7	Перевірити та обмежити облікові записи з sudo-правами	sudo cat /etc/sudoers	Середній
8	Налаштувати автоматичне встановлення оновлень безпеки	sudo apt install unattended-upgrades	Середній

№	Дія	Команда або файл	Пріоритет
9	Встановити та налаштувати AIDE для моніторингу змін	sudo apt install aide && sudo aideinit	Середній
10	Перевірити відповідність CIS Benchmark за допомогою OpenSCAP	sudo oscap xccdf eval --profile cis profile.xml	Низький

Важливою концепцією є принцип мінімальної поверхні атаки, який безпосередньо пов’язаний з управлінням конфігурацією. Поверхня атаки — це сукупність усіх точок, через які зловмисник потенційно може взаємодіяти із системою: відкриті мережеві порти, доступні служби, облікові записи, API-ендпоінти, поля вхідних даних. Що менша поверхня атаки, то менше потенційних шляхів для злому. Тому управління конфігурацією завжди передбачає принцип «вимкни все, що не потрібно» — навіть якщо вимкнений сервіс не має відомих вразливостей сьогодні, він може отримати їх завтра, і якщо він не потрібний, краще його просто не запускати. Таблиця 7.6 наводить приклади типових небезпечних конфігурацій та рекомендованих контрзаходів.

Таблиця 7.6 — Типові небезпечні конфігурації та рекомендовані контрзаходи

Небезпечна конфігурація	Пов’язаний ризик	Рекомендований контрзахід
Вхід через SSH дозволений для root	Підбір пароля адміністратора, відсутність персонального журналу дій	PermitRootLogin no у /etc/ssh/sshd_config
Відкриті зайві мережеві порти	Розширена поверхня атаки	Вимкнути непотрібні служби, налаштувати фаєрвол
Дефолтні паролі адміністраторів	Легкий злом відомими словниками	Змінити всі дефолтні облікові дані після встановлення
Слабкі дозволи на конфігураційні файли	Читання секретів іншими користувачами	Встановити права 640 або 600 на критичні файли
Увімкнені непотрібні мережеві служби	Можливість атаки через невиправлені вразливості	Відключити або видалити невикористовувані служби
Відсутність автоматичного оновлення безпеки	Накопичення невиправлених вразливостей	Налаштувати unattended-upgrades або аналог

Управління конфігурацією у великих середовищах неможливо здійснювати вручну — кількість систем просто не дозволяє цього. Тому застосовується підхід «інфраструктура як код» (Infrastructure as Code, IaC): конфігурація системи описується у вигляді декларативних файлів, які зберігаються в системі контролю версій і автоматично застосовуються. Інструменти на зразок Ansible дозволяють описати бажаний стан сервера у вигляді playbook-файлу та застосовувати його до десятків або сотень серверів одночасно. Головна перевага такого підходу полягає в тому, що конфігурація є повторюваною, задокументованою та перевіряється системою контролю версій — тобто будь-яка зміна фіксується з позначкою часу та ім’ям автора. Це суттєво спрощує розслідування інцидентів: якщо налаштування безпеки були

послаблені, у системі контролю версій залишається відповідний запис із зазначенням, хто, коли і що змінив.

Перевірка відповідності конфігурації встановленим стандартам виконується за допомогою автоматизованих інструментів аудиту конфігурації. Один із найпоширеніших — OpenSCAP (Security Content Automation Protocol), який дозволяє перевіряти Linux-систему на відповідність CIS Benchmarks або профілям DISA STIG та формувати детальний HTML-звіт з результатами та рекомендаціями. Наприклад, типовий звіт OpenSCAP може показати, що система відповідає 74 % вимог CIS Benchmark і містить перелік 23 пунктів, що потребують виправлення, — з конкретними інструкціями для кожного. Такий звіт є зручною відправною точкою для подальших заходів hardening та може бути наданий керівництву як підтвердження поточного стану захищеності.

Окремої уваги заслуговує управління конфігурацією у контексті управління змінами. Будь-яке налаштування системи — це потенційна зміна її рівня захищеності: встановлення нового програмного забезпечення, зміна конфігурації служби, відкриття нового мережевого порту — усі ці дії можуть як покращити, так і погіршити захищеність. Тому в зрілих організаціях діє формальний процес управління змінами, в рамках якого будь-яка зміна проходить перевірку з точки зору безпеки перед впровадженням. Крім того, виконується моніторинг конфігурації: спеціалізовані системи (наприклад, AIDE — Advanced Intrusion Detection Environment у Linux) відстежують зміни у файловій системі та сповіщають адміністратора про несанкціоновані модифікації. AIDE порівнює поточний стан файлів із раніше збереженою еталонною базою, і якщо критичний конфігураційний файл або виконуваний бінарний файл змінився без відповідного запису у системі управління змінами — це є сигналом до розслідування.

Завершуючи розгляд управління конфігурацією, важливо підкреслити, що воно є не одноразовою акцією, а безперервним процесом. Конфігурація системи змінюється з часом: встановлюються нові компоненти, змінюються вимоги до функціональності, стандарти безпеки оновлюються у відповідь на нові загрози. Тому необхідно регулярно проводити повторні перевірки відповідності та повертати системи до встановленої базової лінії, якщо були виявлені відхилення. Такий підхід забезпечує стабільно безпечний стан системи, незважаючи на природну «ентропію», яка накопичується в будь-якій живій системі з часом.

.

7.4. Комплексний підхід до підтримки захищеного стану системи

Управління оновленнями, вразливостями та конфігурацією не є незалежними процесами — вони тісно пов’язані між собою та з іншими елементами системи захисту інформації. Оновлення закривають вразливості у програмному коді, але не усувають проблеми конфігурації. Управління конфігурацією зменшує поверхню атаки, але не компенсує відсутні патчі. Аудит вразливостей виявляє проблеми обох типів, але його результати мають сенс лише тоді, коли існують процеси їх усунення. Тільки у поєднанні ці три процеси утворюють ефективний захист, а їх безперервна взаємодія формує

операційну основу безпеки будь-якої організації. Рисунок 7.3 відображає узагальнену схему безперервного циклу підтримки захищеного стану системи.

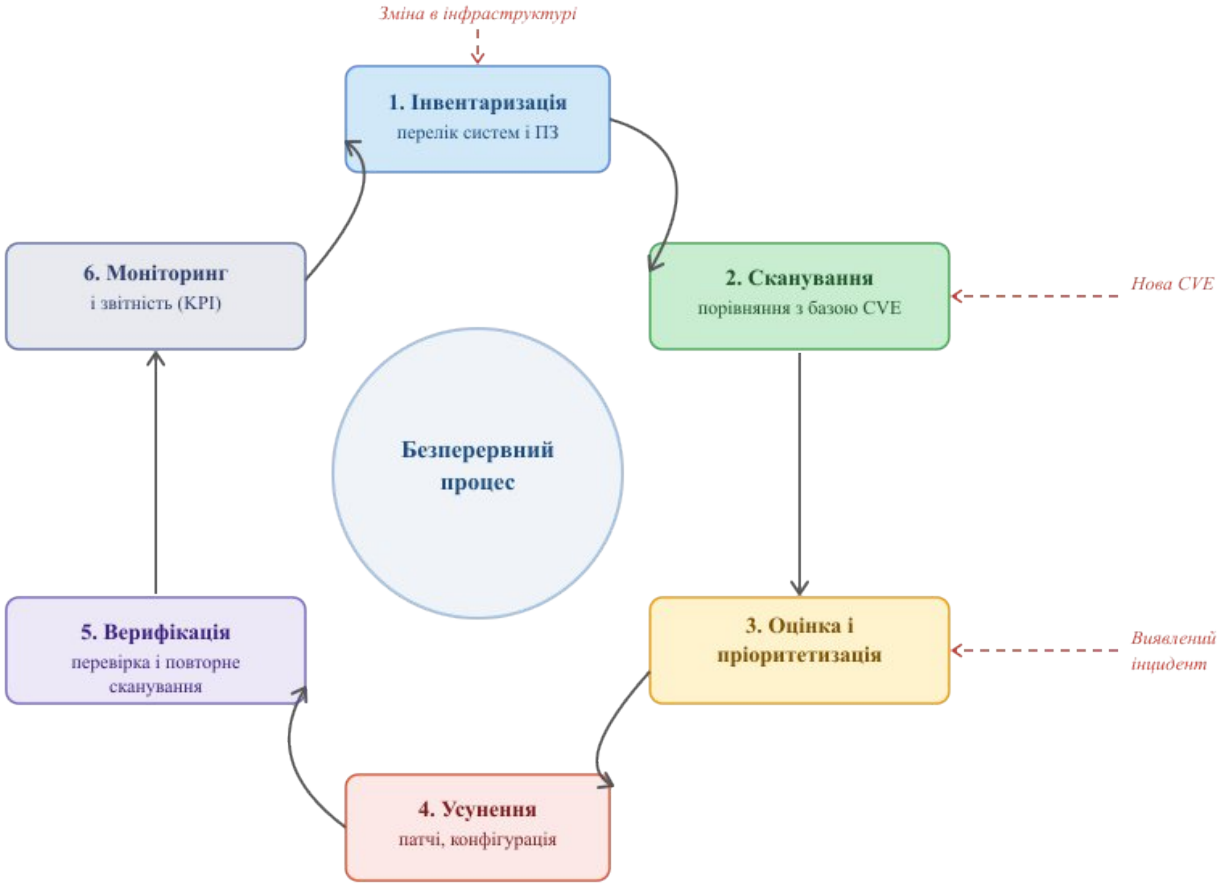

Рисунок 7.3 — Цикл підтримки захищеного стану системи

Опис рисунка. Рисунок являє собою кругову діаграму, що складається з шести прямокутних блоків, рівномірно розміщених по колу зі стрілками між ними за годинниковою стрілкою. Перший блок (вгорі) — «Інвентаризація»: визначення переліку систем і встановленого ПЗ. Другий блок (праворуч угорі) — «Сканування вразливостей»: порівняння із базами CVE, формування звіту. Третій блок (праворуч унизу) — «Оцінка і пріоритетизація»: CVSS, матриця ризиків, контекст організації. Четвертий блок (внизу) — «Усунення»: встановлення патчів, зміна конфігурації, компенсуючі заходи. П’ятий блок (ліворуч унизу) — «Верифікація»: перевірка застосування оновлень, повторне сканування. Шостий блок (ліворуч угорі) — «Моніторинг і звітність»: відстеження KPI, сповіщення, звіти для керівництва. Від шостого блоку стрілка повертається до першого, підкреслюючи безперервність циклу. У центрі кола великий напис «Безперервний процес». По зовнішньому краю діаграми три пунктирні стрілки ззовні вказують на різні блоки циклу: «нова публікація CVE» → «Сканування», «зміна в інфраструктурі» → «Інвентаризація», «виявлений інцидент» → «Оцінка», — показуючи зовнішні тригери позапланового запуску циклу.

На операційному рівні підтримка захищеного стану вимагає встановлення чітких метрик і показників. Скільки часу проходить від публікації критичного

CVE до встановлення патчу? Який відсоток систем відповідає затвердженій базовій лінії конфігурації? Скільки систем мають невиправлені вразливості з рейтингом CVSS понад 7,0? Ці показники (KPI — Key Performance Indicators) дозволяють кількісно вимірювати стан захищеності і відстежувати його динаміку. Наприклад, якщо в поточному місяці середній час від публікації критичного CVE до встановлення патчу складає 4 дні, а у попередньому — 10 днів, то це є вимірюваним підтвердженням того, що процес реагування покращився. Без вимірювань неможливо керувати: організація не може знати, чи поліпшився її стан безпеки після впровадження тих чи інших заходів.

Невід'ємним елементом комплексного підходу є моніторинг і сповіщення. Системний адміністратор не може постійно вручну перевіряти стан усіх систем — йому потрібні автоматизовані механізми, які сповіщають його про появу нових критичних CVE щодо встановленого ПЗ, про відхилення конфігурації від базової лінії, про невдалі спроби оновлення. У Linux-середовищах для цього можна використовувати поштові сповіщення пакетного менеджера, підписки на списки розсилки безпеки дистрибутива (наприклад, ubuntu-security-announce) або інтеграцію із централізованими системами управління безпекою. Своєчасне сповіщення скорочує час реагування і, отже, зменшує вікно вразливості — що є безпосереднім покращенням рівня захищеності.

Важливою концепцією комплексного підходу є ешелонований захист (defense in depth). Цей принцип стверджує, що жоден окремий захисний механізм не є абсолютно надійним, тому необхідно будувати кілька незалежних рівнів захисту. Якщо зловмисник подолав один рівень — наприклад, знайшов невиправлену вразливість у вебзастосунку — наступний рівень (обмежені права процесу, налаштований фаєрвол, моніторинг файлової системи) має завадити йому нанести значну шкоду або хоча б вчасно виявити вторгнення. Управління оновленнями і конфігурацією є двома окремими рівнями у цій ешелонованій системі захисту: якщо зловмисник не може скористатися вразливістю в коді (бо встановлений патч), він спробує скористатися слабкою конфігурацією, і навпаки.

Практично корисним є також поняття «вікна обслуговування» (maintenance windows) — заздалегідь запланованих часових проміжків, протягом яких виконується встановлення оновлень і внесення змін у конфігурацію систем. Такий підхід дозволяє мінімізувати вплив на доступність сервісів для кінцевих користувачів, оскільки перезапуски служб і короткі перерви у роботі відбуваються у заздалегідь погоджений час. Водночас для критичних оновлень безпеки, що усувають активно експлуатовані вразливості, допускається відступ від графіка — встановлення «позапланового» екстреного патчу виконується якнайшвидше незалежно від розкладу, адже ризик від зволікання перевищує незручність від незапланованої перерви у роботі сервісу.

Окремо слід звернути увагу на людський фактор в управлінні конфігурацією та оновленнями. Більшість помилок конфігурації виникає не через зловмисний намір, а через поспіх, брак знань або відсутність формалізованих процедур. Адміністратор, що налаштовує новий сервер без чек-

листа та базової лінії, майже гарантовано припуститься певних помилок, навіть маючи достатній рівень знань. Саме тому важливим елементом зрілої системи управління безпекою є документовані процедури та чек-листи для типових операцій: розгортання нового сервера, налаштування нового сервісу, реагування на критичну вразливість. Технічні засоби автоматизації, такі як Ansible-сценарії, виводять цю ідею на новий рівень: замість того щоб виконувати кроки чек-листа вручну і ризикувати пропустити деякі з них, адміністратор запускає перевірений скрипт, результат якого є повністю відтворюваним і не залежить від втоми або уваги виконавця в конкретний момент.

Підсумовуючи матеріал теми, можна сформулювати три ключові принципи, що лежать в основі підтримки захищеного стану систем. По-перше, своєчасність: кожен день, протягом якого відома вразливість залишається невиправленою, збільшує ризик її експлуатації, і приклад Log4Shell наочно це демонструє — від публікації CVE до масових атак минули лічені години. По-друге, системність: управління оновленнями та конфігурацією мають бути не реактивними (усуваємо проблеми після їх виявлення), а проактивними — систематично знижувати імовірність виникнення проблем через регулярні перевірки, автоматизацію та дотримання стандартів. По-третє, вимірюваність: захищеність системи має бути кількісно виміряна за допомогою конкретних KPI, що дозволяє керувати нею та демонструвати результати керівництву і замовникам. Ці принципи є універсальними і застосовуються незалежно від масштабу системи — від окремого особистого комп’ютера до корпоративної інфраструктури з тисячами вузлів.

Тема 8. Криптографічні механізми захисту даних у програмних системах

8.1. Роль криптографії у забезпеченні безпеки даних

Криптографія є одним із фундаментальних інструментів захисту інформації в сучасних програмних системах. Її завдання полягає у перетворенні відкритих даних на форму, недоступну для сторонніх осіб, а також у підтвердженні автентичності та цілісності цих даних. У контексті тріади безпеки CIA — конфіденційність (Confidentiality), цілісність (Integrity) та доступність (Availability) — криптографія безпосередньо забезпечує перші два елементи й опосередковано впливає на третій через захищену автентифікацію доступу. Без криптографічних механізмів реалізація захищених систем передачі даних, безпечного зберігання файлів або надійної автентифікації користувачів була б практично неможливою — або вимагала б фізичного захисту всіх каналів зв'язку, що нереалістично у сучасному розподіленому середовищі.

Зрозуміти роль криптографії можна на простому прикладі: якщо ви надсилаєте повідомлення через незахищену мережу, будь-хто, хто контролює мережевий трафік, може прочитати його вміст. Шифрування перетворює це повідомлення на набір символів, позбавлений сенсу без знання ключа. Саме тому навіть якщо зловмисник перехопить зашифровані дані, він не зможе ними скористатися. Крім захисту від прочитання, криптографія також дозволяє переконатися, що дані не були змінені в процесі передачі і що відправник є саме тим, за кого себе видає. Ці три властивості — конфіденційність, цілісність та автентичність — є ключовими цілями, які ставить перед собою криптографія як дисципліна.

Основними поняттями криптографії є відкритий текст (plaintext) — вихідні дані, що підлягають захисту; шифртекст (ciphertext) — результат перетворення відкритого тексту за допомогою алгоритму шифрування; ключ — секретна величина, яка керує процесом шифрування та дешифрування; алгоритм шифрування — математична функція, що описує порядок перетворення даних. Безпека сучасних криптографічних систем ґрунтується не на секретності самого алгоритму, а на секретності ключа — цей принцип, сформульований у XIX столітті бельгійським криптографом Огюстом Керкгоффсом, є базовим правилом сучасної криптографії і відомий як принцип Керкгоффса. Це означає, що навіть якщо алгоритм шифрування є загальновідомим — що, як правило, і є — дані залишаються захищеними доти, поки ключ зберігається в таємниці.

Криптографія бере участь у захисті даних на трьох рівнях: захист даних у стані спокою (data at rest), захист даних під час передачі (data in transit) та захист даних під час обробки (data in use). Дані у стані спокою — це файли на жорсткому диску або записи у базі даних; їх захищають шляхом шифрування диска або окремих файлів. Дані під час передачі захищаються за допомогою захищених протоколів, таких як TLS або SSH, детальніше про які йдеться в наступних темах курсу. Захист даних під час обробки є найскладнішим завданням, і сучасні технології, зокрема гомоморфне шифрування та довірені

середовища виконання (Trusted Execution Environments), знаходяться ще на стадії активного розвитку та поступового впровадження.

З точки зору практичного застосування криптографія реалізується через стандартизовані алгоритми та протоколи, що пройшли ретельну наукову перевірку. Самостійна розробка криптографічних алгоритмів є вкрай ризикованою практикою: навіть незначні помилки в реалізації можуть зробити захист повністю неефективним, при цьому зовні система виглядатиме захищеною. Тому фахівці з безпеки завжди орієнтуються на перевірені стандарти та бібліотеки, такі як OpenSSL або libsodium, уникаючи спокуси написати власну реалізацію шифрування «з нуля». Сучасні операційні системи, зокрема Linux, надають вбудовані криптографічні примітиви на рівні ядра та стандартні програмні інтерфейси для їх використання в прикладних програмах.

Важливо розуміти, що криптографія є необхідним, але недостатнім засобом захисту. Навіть найсильніший алгоритм шифрування не допоможе, якщо ключ зберігається незахищено, якщо алгоритм застосовано некоректно або якщо система має вразливості на інших рівнях. Безпека системи визначається найслабшою її ланкою, а тому криптографія завжди розглядається в комплексі з іншими засобами захисту: контролем доступу, аудитом, захистом від мережевих атак, управлінням оновленнями тощо. Саме тому в рамках курсу «Безпека програм та даних» криптографія розглядається як складова частина комплексного підходу до захисту інформаційних систем, а не як самодостатнє рішення.

8.2. Симетричне та асиметричне шифрування

Усі алгоритми шифрування поділяються на дві великі категорії залежно від способу використання ключів: симетричні та асиметричні. Розуміння принципів роботи кожної категорії є необхідною умовою для грамотного вибору криптографічних механізмів захисту в конкретній ситуації. Обидві категорії мають свої сильні та слабкі сторони, і на практиці вони, як правило, використовуються разом у так званих гібридних схемах шифрування — саме такий підхід реалізовано у переважній більшості сучасних захищених протоколів. На рисунку 8.2 схематично зображено принципову відмінність між двома підходами до шифрування.

Рисунок 8.2 – Порівняльна схема симетричного та асиметричного шифрування

Симетричне шифрування — це підхід, при якому для шифрування та дешифрування використовується один і той самий ключ. Уявіть собі скриньку з замком: і відправник, і одержувач мають однаковий ключ, яким можна як замкнути, так і відімкнути скриньку. Перевагою такого підходу є висока швидкість роботи: симетричні алгоритми здатні обробляти гігабайти даних за секунди навіть на апаратному забезпеченні середнього класу. Найпоширенішим сучасним алгоритмом симетричного шифрування є AES (Advanced Encryption Standard) з довжиною ключа 128 або 256 біт, прийнятий NIST як федеральний стандарт США у 2001 році і широко використовуваний у всьому світі — від шифрування дисків до захисту мережевого трафіку. На Linux для роботи з AES можна скористатися командою openssl enc -aes-256-cbc -in файл.txt -out файл.enc для шифрування файлу.

Головною проблемою симетричного шифрування є так звана проблема розподілу ключів. Щоб захищено спілкуватися, обидві сторони повинні мати один і той самий ключ, але як передати цей ключ безпечно, якщо захищений канал ще не встановлено? Це замкнуте коло вирішується або за допомогою особистої зустрічі для обміну ключем — що непрактично на відстані — або за допомогою асиметричного шифрування. Саме тому у сучасних захищених протоколах симетричне шифрування відповідає за захист основного потоку даних, а асиметричне — за безпечну передачу симетричних ключів. Інший підхід до вирішення проблеми розподілу ключів — алгоритм Діффі-Геллмана, який дозволяє двом сторонам виробити спільний секретний ключ через відкритий канал зв'язку без передачі самого ключа.

Асиметричне шифрування, яке також називають шифруванням з відкритим ключем (public-key cryptography), використовує математично пов'язану пару ключів: відкритий (public key) та закритий (private key). Відкритий ключ можна вільно поширювати — він публікується і доступний будь-кому. Закритий ключ зберігається виключно у власника. Математична особливість цієї пари полягає в тому, що дані, зашифровані відкритим ключем, можна розшифрувати лише відповідним закритим ключем, а обчислити закритий ключ із відкритого практично неможливо за розумний час — саме в цьому й полягає математична основа безпеки асиметричної криптографії.

Відомими прикладами асиметричних алгоритмів є RSA (на основі складності факторизації великих чисел), алгоритми на основі еліптичних кривих (ECC), а також ECDSA, який широко використовується для цифрових підписів.

На Linux для генерації пари ключів RSA використовується команда ssh-keygen -t rsa -b 4096, яка створює відкритий і закритий ключі, що широко застосовуються для автентифікації при підключенні до серверів по протоколу SSH. Відкритий ключ (~/.ssh/id_rsa.pub) розміщується на сервері у файлі ~/.ssh/authorized_keys, а закритий (~/.ssh/id_rsa) залишається на клієнтській машині. Коли користувач намагається підключитися, сервер надсилає зашифроване повідомлення, яке може розшифрувати лише той, хто має відповідний закритий ключ, — таким чином автентифікація відбувається без передачі пароля по мережі. Для підвищення безпеки закритий ключ рекомендується захищати додатковою парольною фразою (passphrase), яка шифрує файл ключа на диску.

RSA зазвичай використовує ключі довжиною 2048 або 4096 біт, тоді як алгоритми на основі еліптичних кривих (ECC) забезпечують порівнянний або вищий рівень захисту при значно коротших ключах: наприклад, 256-бітний ключ ECDSA забезпечує захист, еквівалентний 3072-бітному ключу RSA. Це робить ECC особливо привабливим для ресурсообмежених середовищ: мобільних пристроїв, IoT-систем та інших платформ, де обчислювальні ресурси та пам'ять є дефіцитними. На Linux для генерації ключа на основі еліптичних кривих можна скористатися командою ssh-keygen -t ed25519, яка використовує криву Edwards25519 — один із найбезпечніших і найшвидших варіантів ECC, рекомендований для нових систем.

Таблиця 8.1 – Порівняльна характеристика симетричного та асиметричного шифрування

Характеристика	Симетричне шифрування	Асиметричне шифрування
Кількість ключів	Один спільний ключ для шифрування та дешифрування	Пара ключів: відкритий (public) та закритий (private)
Швидкість роботи	Дуже висока — у 100–1000 разів швидше за асиметричне	Низька через математичну складність операцій
Приклади алгоритмів	AES-128, AES-256, ChaCha20	RSA-2048, RSA-4096, ECDSA (еліптичні криві)
Управління ключами	Складне: ключ необхідно заздалегідь безпечно передати одержувачу	Спрощене: відкритий ключ можна публікувати відкрито
Типове застосування	Шифрування великих обсягів даних, файлів, дисків, резервних копій	Безпечний обмін ключами, цифрові підписи, автентифікація
Основна проблема	Проблема безпечного	Низька продуктивність при

Характеристика	Симетричне шифрування	Асиметричне шифрування
	розподілу ключів між сторонами	великих обсягах даних

Як видно з таблиці 8.1, жодна з категорій шифрування не є універсальним рішенням для всіх задач. Саме тому в реальних системах застосовується гібридна схема. На рисунку 8.3 зображено принцип роботи гібридного шифрування на прикладі протоколу TLS, що захищає більшість сучасного веб-трафіку та передачу даних: спочатку за допомогою асиметричного шифрування узгоджується симетричний сеансовий ключ (session key), а потім весь подальший обмін даними відбувається вже за допомогою симетричного алгоритму AES з цим ключем. Це поєднання дає змогу отримати переваги обох підходів: зручність управління ключами від асиметричного шифрування та високу продуктивність від симетричного.

Рисунок 8.3 – Гібридна схема шифрування (на прикладі принципу TLS)

Варто зазначити, що вибір між різними алгоритмами шифрування не є статичним: з часом алгоритми, які вважалися надійними, можуть виявитися вразливими через зростання обчислювальних потужностей або відкриття нових криптоаналітичних методів. Так, DES (Data Encryption Standard) з 56-бітним ключем, що вважався стандартом у 1970-х роках, став практично зламаним вже наприкінці 1990-х через зростання швидкості перебору. Алгоритм 3DES, що замінив DES, також поступово виводиться з використання. AES-128 і AES-256, прийняті у 2001 році, наразі вважаються надійними, однак розвиток квантових обчислень у майбутньому може потребувати переходу до постквантових алгоритмів. Це підкреслює важливість регулярного перегляду застосовуваних криптографічних стандартів та своєчасного оновлення систем.

8.3. Криптографічні хеш-функції та цифровий підпис

Крім шифрування, яке забезпечує конфіденційність, криптографія пропонує ще два важливих механізми: хеш-функції для забезпечення цілісності та цифровий підпис для підтвердження автентичності. Ці механізми відповідають на різні запитання безпеки: шифрування гарантує, що сторонні не прочитають дані; хеш-функція гарантує, що дані не були змінені; цифровий

підпис гарантує, що дані дійсно походять від певного відправника і не були підмінені. Разом ці три механізми забезпечують повноцінний захист даних у сучасних програмних системах і саме в такому поєднанні реалізовані у переважній більшості захищених протоколів.

Криптографічна хеш-функція — це математична функція, яка перетворює вхідні дані довільної довжини на рядок фіксованого розміру, що називається дайджестом або хешем. Хеш-функції мають кілька ключових властивостей: вони є детерміністичними (одні й ті самі вхідні дані завжди дають один і той самий хеш), односпрямованими (маючи лише хеш, неможливо відновити вихідні дані), стійкими до колізій (надзвичайно малоймовірно знайти два різних набори вхідних даних з однаковим хешем) та чутливими до змін (навіть зміна одного символу у вхідних даних призводить до повністю іншого хешу — ця властивість називається ефектом лавини). На Linux команда sha256sum файл.txt виведе 256-бітний хеш вмісту файлу у вигляді 64 шістнадцяткових символів; будь-яка зміна навіть одного байту файлу призведе до принципово іншого результату.

Хеш-функції широко застосовуються для перевірки цілісності файлів. Коли ви завантажуєте програму з інтернету, на сайті розробника зазвичай публікується хеш файлу. Після завантаження ви обчислюєте хеш отриманого файлу і порівнюєте його з опублікованим значенням: якщо вони збігаються, файл не був пошкоджений або підмінений. Саме так менеджери пакетів у Linux, зокрема apt, перевіряють цілісність завантажених пакетів перед встановленням. Крім того, хеш-функції є базовим механізмом зберігання паролів: замість самого пароля в базі зберігається його хеш, і при автентифікації система порівнює хеш введеного пароля зі збереженим. Якщо база даних паролів потрапить до зловмисника, він отримає лише хеші, а не самі паролі.

Таблиця 8.2 – Порівняльна характеристика криптографічних хеш-функцій

Алгоритм	Довжина хешу	Стійкість до атак	Рекомендація
MD5	128 біт (16 байт)	Зламано: відомі практичні колізії, непридатний для криптографії	Не використовувати
SHA-1	160 біт (20 байт)	Слабка: атака SHAttered (2017) довела практичність колізій	Застарілий
SHA-256	256 біт (32 байти)	Висока: стійкий до відомих атак, стандарт NIST FIPS 180-4	Рекомендується
SHA-3 (Keccak)	224–512 біт (змінна)	Дуже висока: незалежна архітектура від SHA-2, резервний стандарт NIST	Перспективний
bcrypt / Argon2	Фіксована + сіль	Дуже висока: навмисно повільний алгоритм, стійкий	Тільки для паролів

Алгоритм	Довжина хешу	Стійкість до атак	Рекомендація
		до перебору паролів	

Як видно з таблиці 8.2, вибір хеш-функції суттєво впливає на рівень безпеки системи. MD5 та SHA-1 нині вважаються криптографічно зламаними та непридатними для захисту. Атака SHAttered, продемонстрована у 2017 році дослідниками Google і CWI Amsterdam, показала можливість створення двох різних PDF-файлів з однаковим SHA-1 хешем, що повністю дискредитує цей алгоритм для підписів і перевірки цілісності. Для більшості завдань рекомендується SHA-256, але для зберігання паролів звичайні хеш-функції не підходять з принципово іншої причини: вони занадто швидко обчислюються. Сучасний комп'ютер здатний обчислити мільярди SHA-256 хешів за секунду, що дозволяє зловмисникові перебирати паролі з величезною швидкістю. Спеціалізовані функції bcrypt, scrypt або Argon2 виконуються значно повільніше за задумом — вони є «стійкими до паролів» (password hashing functions) і автоматично додають сіль для захисту від райдужних таблиць.

Цифровий підпис — це криптографічний механізм, що дозволяє підтвердити автентичність і цілісність документа або повідомлення, а також забезпечити неспростовність (non-repudiation) — властивість, за якої відправник не може відмовитися від факту підписання. Цифровий підпис будується на основі асиметричної криптографії: відправник підписує дані своїм закритим ключем, а будь-хто, хто має відповідний відкритий ключ, може перевірити підпис. На практиці підписується не сам документ, а його хеш, що значно прискорює процес і дозволяє підписувати документи будь-якого розміру, оскільки операції з асиметричними ключами над великими блоками даних були б надто повільними. Схема формування та перевірки цифрового підпису детально зображена на рисунку 8.1.

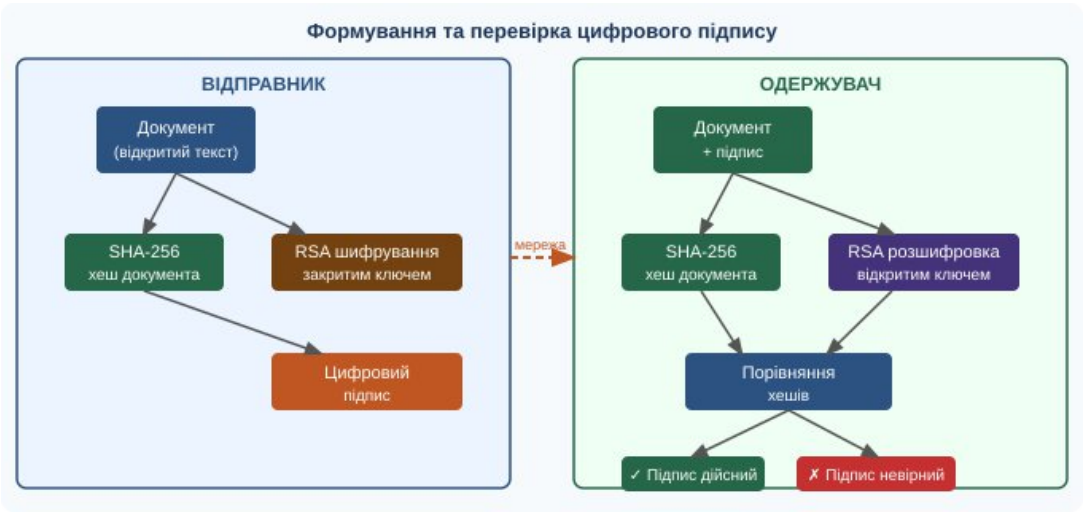

Рисунок 8.1 – Схема формування та перевірки цифрового підпису

На рисунку 8.1 зображено дві сторони: відправника (ліворуч, синій блок) та одержувача (праворуч, зелений блок). Відправник виконує два паралельних

кроки: обчислює хеш документа за допомогою SHA-256, а потім шифрує цей хеш своїм закритим ключем RSA — результатом є цифровий підпис. Документ разом із підписом передається одержувачу через мережу (показано помаранчевою стрілкою). Одержувач також обчислює хеш отриманого документа і паралельно розшифровує підпис відкритим ключем відправника, отримуючи оригінальний хеш. Якщо обидва хеші збігаються, підпис є дійсним (зелений блок); якщо ні — підпис невірний або документ був змінений після підписання (червоний блок).

Цифрові підписи широко використовуються у повсякденній роботі Linux-систем, хоча користувачі часто не помічають цього. Менеджери пакетів apt та yum автоматично перевіряють цифровий підпис кожного пакету перед встановленням, використовуючи відкриті ключі розробників зі системного сховища. Якщо пакет був підмінений або пошкоджений, перевірка підпису виявить це і встановлення буде відхилено з повідомленням про помилку автентифікації. Аналогічно цифровий підпис застосовується для підтвердження автентичності електронної пошти (стандарти DKIM та S/MIME), юридично значущого підпису документів у системах електронного документообігу, а також у протоколах оновлення програмного забезпечення для забезпечення автентичності файлів оновлень.

8.4. Практичне застосування та типові помилки при впровадженні криптографії

Теоретичні знання про криптографічні механізми набувають практичного сенсу тільки тоді, коли фахівець розуміє, як їх правильно застосовувати для захисту реальних систем. Практичне застосування криптографії охоплює кілька основних сценаріїв: захист файлів і дисків, захист повідомлень під час передачі, захист резервних копій, а також автентифікація у програмних системах. Кожен сценарій має свою специфіку та вимагає обдуманого вибору алгоритмів і схем роботи з ключами. У таблиці 8.3 узагальнено рекомендовані підходи для найбільш поширених практичних сценаріїв із зазначенням відповідних інструментів для Linux.

Таблиця 8.3 – Вибір криптографічного інструменту за сценарієм використання

Сценарій захисту	Рекомендований алгоритм	Причина вибору	Приклад на Linux
Шифрування файлів і дисків	AES-256-GCM або AES-256-CBC	Висока швидкість, перевірений стандарт	LUKS, gpg --symmetric
Безпечний обмін ключами	RSA-4096 або ECDH (Curve25519)	Передача без спільного секрету заздалегідь	ssh-keygen, openssl genpkey
Зберігання паролів	Argon2id або bcrypt	Стійкість до	/etc/shadow у

Сценарій захисту	Рекомендований алгоритм	Причина вибору	Приклад на Linux
		перебору, вбудована сіль	Linux
Перевірка цілісності файлів	SHA-256 або SHA-3	Стійкість до колізій, швидке обчислення	sha256sum файл.txt
Цифровий підпис документів	ECDSA (P-256) або RSA-PSS	Неспростовність, автентичність відправника	gpg --sign, openssl dgst
Шифрування повідомлень між особами	OpenPGP (GPG), гібридна схема	RSA/ECC + AES, стандарт міжособистісного захисту	gpg --encrypt --recipient

Захист файлів на рівні операційної системи Linux реалізується кількома способами. Найбільш поширеним є шифрування всього розділу диска за допомогою LUKS (Linux Unified Key Setup) у поєднанні з dm-crypt — підсистемою ядра Linux для прозорого шифрування блочних пристроїв. При такому підході весь вміст розділу зберігається зашифрованим і стає доступним лише після введення пароля або надання ключового файлу при завантаженні системи. Це особливо важливо для захисту даних у разі фізичної крадіжки пристрою: навіть якщо зловмисник вийме жорсткий диск і підключить його до іншого комп'ютера, він не зможе прочитати дані без ключа. Для шифрування окремих файлів або архівів застосовуються утиліти GPG або openssl.

Захист резервних копій є окремим і надзвичайно важливим завданням, яке часто недооцінюється. Резервні копії за своєю природою містять ті самі дані, що й основні системи, але часто зберігаються у менш захищених місцях — на зовнішніх дисках, у хмарних сховищах або на стрічках. Якщо резервна копія не зашифрована, її витік може стати катастрофою навіть за умови надійного захисту основної системи. Правильна практика полягає у шифруванні резервних копій перед їх переміщенням за межі захищеного середовища. На Linux це можна реалізувати командою gpg --symmetric --cipher-algo AES256 архів.tar, яка зашифрує архів симетричним ключем. При цьому пасфразу від зашифрованого архіву необхідно зберігати окремо від самого архіву і в надійному місці — у разі її втрати дані стануть безповоротно недоступними.

Поширеним практичним завданням є також захищений обмін файлами або повідомленнями між двома сторонами. Стандартом де-факто тут є GNU Privacy Guard (GPG) — реалізація стандарту OpenPGP, яка дозволяє шифрувати та підписувати файли і повідомлення за гібридною схемою. Принцип роботи такий: кожен учасник генерує пару ключів GPG командою gpg --full-generate-key, відкриті ключі обмінюються між учасниками або публікуються на сервері ключів, після чого відправник шифрує повідомлення відкритим ключем

одержувача командою gpg --encrypt --recipient user@example.com файл.txt. Розшифрувати таке повідомлення може лише власник відповідного закритого ключа командою gpg --decrypt файл.txt.gpg. Варто зазначити, що GPG використовує гібридну схему: генерується випадковий сеансовий ключ AES, яким шифруються дані, а сам сеансовий ключ шифрується відкритим ключем RSA або ECC одержувача.

Окремого розгляду заслуговує небезпека режиму ECB (Electronic Codebook) при використанні блокового шифру AES. Алгоритм AES шифрує блоки даних по 128 біт незалежно один від одного, і в режимі ECB однакові блоки відкритого тексту завжди перетворюються на однакові блоки шифртексту. Це означає, що структура та повторювані фрагменти у вихідних даних залишаються видимими у шифртексті, що робить такий підхід криптографічно слабким. На рисунку 8.4 наведено ілюстративний приклад: зображення, зашифроване в режимі ECB, зберігає видимі контури, тоді як те саме зображення, зашифроване в режимі GCM, виглядає як цілковитий випадковий шум.

Рисунок 8.4 – Порівняння результатів шифрування у режимах ECB та GCM

Правильними сучасними режимами є GCM (Galois/Counter Mode) та CBC (Cipher Block Chaining) з додатковою автентифікацією. Режим GCM є особливо рекомендованим, оскільки він є автентифікованим режимом шифрування (AEAD — Authenticated Encryption with Associated Data): він не лише шифрує дані, але й обчислює автентифікаційний тег, який дозволяє виявити будь-яку модифікацію шифртексту. Якщо зловмисник змінить навіть один біт зашифрованих даних, перевірка автентифікаційного тегу виявить це при дешифруванні. Схожим чином працює алгоритм ChaCha20-Poly1305, який є альтернативою AES-GCM і широко використовується в мобільних додатках та протоколі TLS 1.3 через свою ефективність на процесорах без апаратного прискорення AES.

Таблиця 8.4 – Типові помилки при впровадженні криптографії та шляхи їх усунення

Помилка	Наслідок	Правильний підхід
Використання застарілих алгоритмів (MD5, SHA-1, DES, 3DES)	Можливість злому через відомі криптоаналітичні атаки та колізії	Застосовувати AES-256, SHA-256 або SHA-3, RSA-2048 або вище

Помилка	Наслідок	Правильний підхід
Самостійна реалізація криптоалгоритмів	Помилки реалізації, вразливості до атак через побічні канали	Використовувати перевірені бібліотеки: OpenSSL, libsodium
Зберігання ключів у відкритому вигляді (у коді або репозиторії)	Компрометація всіх даних, захищених цим ключем	Зберігати ключі у захищених сховищах, змінних середовища або HSM
Використання режиму ECB для блокового шифру	Збереження структури відкритого тексту у шифртексті	Застосовувати AES-GCM або AES-CBC з MAC
Повторне використання ключа чи вектора ініціалізації (IV/nonce)	Можливість перехресного аналізу та відновлення відкритого тексту	Генерувати новий унікальний IV для кожної операції шифрування
Ігнорування автентифікації шифртексту (без MAC або AEAD)	Атаки на цілісність: непомітна підміна або модифікація шифртексту	Застосовувати автентифіковане шифрування: AES-GCM, ChaCha20-Poly1305

Аналіз типових помилок у таблиці 8.4 показує, що більшість з них пов'язана не з вибором неправильного алгоритму, а з некоректним способом його застосування. Особливої уваги заслуговує помилка, пов'язана зі зберіганням ключів у відкритому вигляді у вихідному коді або репозиторіях. Це надзвичайно поширена проблема: за даними досліджень, щодня на публічних платформах (наприклад, GitHub) з'являються тисячі комітів, що містять API-ключі, паролі або криптографічні ключі у відкритому вигляді. Зловмисники використовують автоматизовані сканери для пошуку таких витоків і можуть скомпрометувати систему протягом хвилин після публікації. На Linux для безпечного зберігання секретів рекомендується використовувати змінні середовища (export SECRET_KEY=...) або спеціалізовані менеджери секретів, а права доступу до файлів з чутливими даними мають бути обмежені командою chmod 600 ключ.pem.

Не менш поширеною помилкою є повторне використання вектора ініціалізації (IV) або nonce при симетричному шифруванні. Вектор ініціалізації — це випадкові дані, що додаються до ключа перед шифруванням для забезпечення унікальності кожної операції: навіть якщо один і той самий відкритий текст шифрується одним і тим самим ключем двічі, результат повинен бути різним завдяки різним IV. Якщо IV повторюється (що часто відбувається через помилки в реалізації лічильників або генераторів випадкових чисел), виникає серйозна вразливість: зловмисник може отримати інформацію про відкритий текст шляхом XOR-операцій над двома шифртекстами з однаковим IV. Ця атака, відома як «two-time pad», зруйнувала безпеку ряду реальних систем, зокрема ранніх версій WEP (протокол захисту Wi-Fi).

Розглянуті у цій темі принципи та механізми криптографічного захисту утворюють фундамент для розуміння більш складних тем курсу. Управління криптографічними ключами, яке коротко згадувалося в контексті типових помилок, детально розглядається в темі 9 — включаючи повний життєвий цикл ключів, апаратні та програмні засоби їх захисту та основи інфраструктури відкритих ключів (PKI). Захищені протоколи передачі даних SSH та TLS, принцип роботи яких ґрунтується на описаних гібридних схемах шифрування, розглядаються в темі 10. Таким чином, глибоке розуміння криптографічних примітивів, здобуте в рамках поточної теми, є необхідним підґрунтям для всього наступного матеріалу курсу, присвяченого захисту даних та безпечній комунікації.

ТЕМА 9 УПРАВЛІННЯ КРИПТОГРАФІЧНИМИ КЛЮЧАМИ ТА ДОВІРЧОЮ ІНФРАСТРУКТУРОЮ

9.1. Криптографічний ключ як об'єкт захисту

Криптографічний ключ є центральним поняттям сучасної інформаційної безпеки, і саме від його належного захисту залежить надійність усієї системи шифрування. Алгоритм шифрування, як правило, є відкритим і добре вивченим стандартом — наприклад, AES або RSA — тоді як секретність забезпечується виключно ключем. Цей принцип сформулював ще у 1883 році нідерландський криптограф Огюст Керкгоффс, і він залишається наріжним каменем сучасної криптографії: стійкість системи не має залежати від секретності алгоритму, вся таємниця повинна бути зосереджена у ключі. Компрометація ключа рівнозначна повному зламу системи захисту, тоді як знання алгоритму без ключа не дає зловмиснику жодної суттєвої переваги. Саме тому управління ключами виокремлюється в окрему дисципліну та є одним із найскладніших практичних аспектів побудови захищених систем.

Існує кілька типів криптографічних ключів, кожен із яких виконує певну роль у системі захисту. Симетричні ключі використовуються в алгоритмах AES, ChaCha20 для шифрування та дешифрування одним і тим самим секретом, що вимагає попереднього захищеного обміну цим секретом між сторонами. Асиметричні ключові пари, що застосовуються в RSA або алгоритмах на еліптичних кривих (ECDSA, ECDH), складаються з відкритого та закритого ключів: відкритий можна розповсюджувати вільно, тоді як закритий зберігається суворо в таємниці. Сеансові ключі є тимчасовими симетричними ключами, що генеруються для одного сеансу зв'язку та знищуються після його завершення. Ключі шифрування ключів (KEK) призначені виключно для захисту інших ключів — вони утворюють ієрархію, де верхній рівень захищає нижній. Ключі підпису застосовуються лише для створення та перевірки цифрового підпису і не повинні використовуватись для шифрування — змішування призначень є типовою помилкою проєктування.

Сеансові ключі забезпечують властивість, яку називають **прямою секретністю (Perfect Forward Secrecy, PFS)**: навіть якщо довгостроковий закритий ключ сервера буде скомпрометований у майбутньому, зловмисник не зможе розшифрувати раніше перехоплені сесії, оскільки сеансові ключі вже знищено. Ця властивість принципово відрізняє сучасні протоколи (TLS 1.3 з ECDH) від застарілих схем (RSA Key Transport у TLS 1.0/1.1), де розкриття закритого ключа дозволяло дешифрувати весь архів перехоплених з'єднань. Для досягнення PFS кожен новий сеанс використовує унікальну ефемерну пару ключів Діффі-Гелмана, що генерується «на льоту» і не зберігається після завершення сесії. У Лінукс перевірити, чи підтримує сервер PFS, можна командою: openssl s_client -connect example.com:443 — у виводі слід шукати рядок Cipher, що містить ECDHE або DHE.

Ризики, пов'язані з ключами, є різноплановими і не зводяться лише до технічних атак. Крадіжка ключа з незахищеного сховища, підбір слабкого

ключа через недостатню довжину або неякісний генератор випадковості, витік через недбале поводження — це реальні загрози, з якими регулярно стикаються організації. Показовим прикладом є інцидент 2011 року, коли зловмисники зламали компанію RSA Security та отримали доступ до ключів для генерації токенів двофакторної автентифікації SecurID. Незважаючи на те, що алгоритм TOTP залишається стійким, компрометація ключового матеріалу дозволила клонувати токени і здійснити подальші атаки на клієнтів компанії. Цей випадок наочно демонструє, що захист ключів є не менш важливим, ніж вибір криптографічного алгоритму.

9.2. Життєвий цикл криптографічних ключів

Криптографічний ключ не існує вічно: він проходить через визначений набір етапів від появи до знищення, що у сукупності називають життєвим циклом ключа. Стандарти NIST SP 800-57 та ISO/IEC 11770 детально регламентують ці етапи і визначають вимоги до кожного з них. Чітке слідування встановленому циклу дозволяє мінімізувати ризик компрометації та забезпечити відповідність нормативним вимогам. На рисунку 9.1 схематично зображено послідовність етапів, а у таблиці 9.1 наведено детальний опис кожного етапу та заходів безпеки.

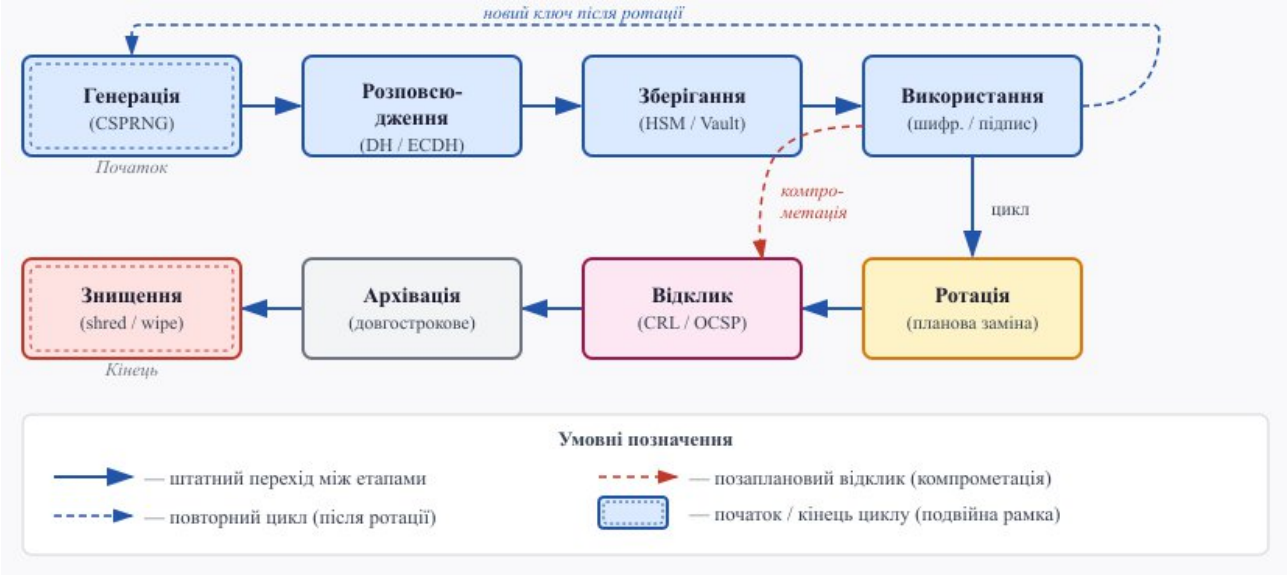

Рисунок 9.1 — Діаграма життєвого циклу криптографічного ключа

Таблиця 9.1 — Етапи життєвого циклу криптографічного ключа та заходи безпеки

Етап	Зміст етапу	Ключові заходи безпеки
Генерація	Створення ключового матеріалу з використанням криптографічно стійкого генератора випадкових чисел	Апаратний генератор ентропії; заборона псевдовипадкових функцій загального призначення
Розповсюдження	Передача ключів між учасниками	Захищені канали, шифрування під

	обміну	час передачі, перевірка автентичності одержувача
Зберігання	Розміщення ключів у захищеному сховищі на весь час їх дії	Шифрування «в стані спокою», розмежування доступу, резервне копіювання
Використання	Безпосереднє застосування ключів у криптографічних операціях	Обмеження кола операцій для кожного ключа, журналювання
Ротація	Планова або позапланова заміна ключа на новий	Дотримання термінів дії, автоматизована ротація через KMS
Відклик	Виведення ключа з обігу до закінчення терміну дії	Публікація у CRL / OCSP, оповіщення всіх учасників, негайна заміна
Архівація і знищення	Довгострокове збереження та гарантоване знищення після завершення потреби	Фізична ізоляція архіву, утиліти перезапису (shred), документування

Генерація ключів є першим і фундаментальним етапом, від якого залежить стійкість усієї системи захисту. Ключ повинен генеруватися за допомогою криптографічно стійкого генератора псевдовипадкових чисел (CSPRNG), що отримує достатній обсяг ентропії від апаратних джерел: теплового шуму процесора, мікрофона, рухів миші, мережевої активності. У Лінукс для цього можна використати системний генератор через стандартні утиліти:

```
# Генерація 256-бітного симетричного ключа (AES-256)
openssl rand -base64 32

# Генерація RSA-ключової пари 4096 біт із захистом парольною
фразою
openssl genrsa -aes256 -out private.pem 4096

# Генерація пари ключів на еліптичній кривій P-256
openssl ecparam -name prime256v1 -genkey -noout -out
ec_private.pem
```

Самостійна реалізація генератора ключів без глибокого розуміння криптографічних принципів є небезпечною помилкою. Навіть незначне зміщення у розподілі випадкових чисел може зробити ключ вразливим до математичного аналізу. Відомий приклад — вразливість ROCA (2017 рік), що стосувалася бібліотеки RSALib від Infineon: через помилку в генераторі ключів RSA-ключі з довжиною до 2048 біт могли бути відновлені за лічені години на звичайному комп'ютерному кластері. Мільйони смарт-карт, токенів і TPM-мікросхем виявилися скомпрометованими. Тому на практиці рекомендується завжди використовувати перевірені бібліотеки та стандартні механізми ОС, уникаючи «власних» рішень.

Розповсюдження ключів — один із найскладніших етапів, адже вимагає вирішення класичної задачі: як передати секрет безпечним каналом, якщо захищений канал ще не встановлений? Протокол обміну ключами Діффі-Гелмана (1976 р.) вирішує цю проблему математично: дві сторони обмінюються відкритими значеннями і незалежно обчислюють спільний секрет, який ніколи не передається по мережі. Перехопивши трафік, зловмисник бачить лише відкриті значення, з яких обчислити секрет є обчислювально складним завданням. Сучасні реалізації на еліптичних кривих (ECDH) застосовуються у TLS 1.3, SSH та мессенджері Signal. У корпоративних системах для симетричних ключів нерідко застосовується схема KEK/DEK: сам ключ шифрування даних (DEK) захищається «ключем ключів» (KEK), що зберігається в HSM, а DEK передається вже в зашифрованому вигляді. Таблиця 9.2 узагальнює основні алгоритми обміну ключами.

Таблиця 9.2 — Порівняльна характеристика алгоритмів обміну ключами

Алгоритм	Тип	Переваги	Недоліки	PFS
Diffie-Hellman (DH)	Асиметр.	Обмін без попереднього секрету	Вразливий до MitM без автентифікації; великі ключі (2048+)	Ні
ECDH	Асиметр.	Коротші ключі, вища швидкість	Складніша реалізація; залежність від параметрів кривої	Так*
RSA Key Transport	Асиметр.	Широка підтримка, проста концепція	Немає PFS; великий розмір ключів	Ні
PSK (Pre-Shared Key)	Симетр.	Не потребує PKI	Проблема початкового розподілу секрету	Ні

* PFS для ECDH досягається за умови використання ефемерних (одноразових) ключових пар (ECDHE).

Ротація ключів — це планова заміна діючого ключа на новий із перешифруванням даних або переукладенням захищених сесій. Необхідність ротації зумовлена кількома факторами: зменшенням обсягу даних, зашифрованих одним ключем (що знижує ризик криптоаналітичних атак), обмеженням вікна можливої компрометації та виконанням нормативних вимог. Таблиця 9.3 наводить рекомендовані терміни ротації для різних типів ключів згідно з провідними стандартами.

Таблиця 9.3 — Рекомендовані терміни ротації криптографічних ключів

Тип ключа	Рекомендований термін	Стандарт	Примітки
Симетричний сеансовий ключ	1 сеанс зв'язку	NIST SP 800-57	Знищується після завершення сесії (PFS)

Симетричний ключ шифрування даних (DEK)	До 1 року або 1 ГБ даних	PCI DSS, NIST SP 800-57	Залежить від обсягу шифрованих даних
Ключ шифрування ключів (KEK)	1–3 роки	NIST SP 800-57	Зміна потребує перешифрування всіх DEK
Асиметричний ключ підпису (CA Intermediate)	2–5 років	ISO 11770, CA/Browser Forum	Термін дії сертифіката обмежує строк використання
Ключ Root CA	10–20 років	WebTrust, RFC 5280	Зберігається офлайн у HSM; зміна потребує оновлення якорів довіри

Практична ротація симетричного ключа шифрування файлів у Лінукс може виглядати так: спочатку генерується новий ключ, потім файли розшифровуються старим ключем і перешифровуються новим, після чого старий ключ знищується. У середовищах із великою кількістю ключів (наприклад, мікросервісна архітектура з десятками сервісів) ручна ротація стає нездійсненною — тут на допомогу приходять автоматизовані KMS-рішення, що виконують ротацію прозоро для застосунку. Наприклад, HashiCorp Vault підтримує автоматичну ротацію Transit Engine-ключів за розкладом, а AWS KMS дозволяє налаштувати щорічну ротацію одним прапорцем. Знищення ключа по завершенні його використання вимагає гарантованого видалення всіх копій, включно з резервними: у Лінукс для безпечного перезапису файлу ключа застосовується команда `shred -u -z private.key`, яка перезаписує вміст файлу кілька разів перед видаленням.

9.3. Апаратні та програмні засоби захисту ключів

Для захисту криптографічних ключів від несанкціонованого доступу та витоку використовується широкий спектр засобів — від спеціалізованих апаратних пристроїв до програмних сховищ і хмарних сервісів. Вибір конкретного засобу залежить від рівня чутливості даних, вимог нормативних стандартів, моделі розгортання (on-premise, хмара, гібрид) та наявного бюджету. У таблиці 9.4 наведено порівняльну характеристику основних типів засобів захисту ключів, включаючи орієнтовну вартість.

Таблиця 9.4 — Порівняльна характеристика засобів захисту криптографічних ключів

Засіб	Рівень захисту	Орієнтовна вартість	Типові сфери застосування
HSM (мережевий)	FIPS 140-2 Level 3–4	$10 000 – $100 000	Кореневі CA, банківські HSM, хмарні KMS
HSM (PCI-карта)	FIPS 140-2 Level 3	$2 000 – $15 000	Сервери шифрування БД,

			підпис коду
TPM 2.0 (вбудований)	FIPS 140-2 Level 2	Вбудований у пристрій	Шифрування диска (LUKS+TPM), Secure Boot, автентифікація пристрою
USB-токен / смарт-карта	Середній–Високий	$20 – $200 за пристрій	2FA для користувачів, підпис документів
HashiCorp Vault (програмний)	Середній (залежить від конфігурації)	Open Source / $0+	DevOps, мікросервіси, Kubernetes Secrets
Хмарний KMS (AWS/GCP/Azure)	Середній–Високий	За запит / ~$1/ключ/міс.	Хмарні застосунки, шифрування сховищ S3/GCS

Апаратний модуль безпеки (Hardware Security Module, HSM) є найнадійнішим засобом захисту криптографічних ключів і являє собою спеціалізований пристрій — внутрішня PCI-карта або зовнішній мережевий блок — що виконує всі криптографічні операції всередині захищеного фізичного середовища. Ключовою властивістю HSM є неможливість вилучення ключів у відкритому вигляді: пристрій фізично захищений від несанкціонованого розтину (tamper-evident або tamper-resistant) і знищує ключовий матеріал при виявленні спроби фізичного вторгнення. HSM сертифікуються відповідно до стандарту FIPS 140-2 (або його наступника FIPS 140-3), причому рівень 3 і вище передбачає активний захист від фізичних атак — вихід із ладу при спробі розкриття корпусу. Такі пристрої застосовуються в банківських системах для захисту PIN-кодів, у центрах сертифікації для захисту кореневих ключів підпису, а також хмарними провайдерами в рамках послуги KMS.

Модуль довіреної платформи (Trusted Platform Module, TPM) — це мікросхема, інтегрована безпосередньо в материнську плату комп'ютера, що надає апаратну підтримку криптографічних операцій та захищеного зберігання ключів на рівні пристрою. На відміну від HSM, TPM є масово розповсюдженим рішенням — стандарт TPM 2.0 присутній у більшості ноутбуків і настільних комп'ютерів корпоративного класу. TPM використовується для захищеного завантаження ОС (Secure Boot), шифрування дисків за схемою LUKS+TPM у Лінукс, а також для автентифікації пристрою в корпоративній мережі. Важливою особливістю TPM є прив'язка ключа до виміряного стану системи (Platform Configuration Registers, PCR): якщо хтось спробує перенести зашифрований диск на інший комп'ютер або підмінити завантажувальне ПЗ, TPM відмовить у видачі ключа, оскільки виміряний стан системи не збігатиметься з еталонним. У Лінукс перевірити наявність TPM можна командою `ls /dev/tpm*` або `tpm2_getcap properties-fixed`.

USB-токени та смарт-карти є портативними апаратними засобами, що зберігають закриті ключі та сертифікати і виконують криптографічні операції безпосередньо всередині мікроконтролера пристрою, не дозволяючи ключу

покидати захищене середовище. Підключення токена надає можливість підписати документ або пройти автентифікацію, але без фізичного пристрою жодна операція неможлива — це забезпечує другий фактор автентифікації. Стандарти PKCS#11 та OpenSC дозволяють використовувати токени в Лінукс: пакет opensc забезпечує підтримку більшості пристроїв, а утиліта pkcs11-tool дозволяє керувати ключами безпосередньо з командного рядка. Щоб переглянути вміст підключеного токена, достатньо виконати `pkcs11-tool --module /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so -l --list-objects`.

Програмні сховища ключів є доступнішим, проте менш захищеним варіантом. У Лінукс широко використовуються формати PEM та PKCS#12 для зберігання ключів у файловій системі, захищених парольною фразою та правами доступу ОС (chmod 600). Утиліта GnuPG (gpg) надає повноцінне середовище для управління ключами і зберігає їх у директорії ~/.gnupg. Для серверних застосунків та мікросервісів усе більшого поширення набувають спеціалізовані менеджери секретів — HashiCorp Vault, AWS Secrets Manager, Azure Key Vault. Ці системи надають централізоване API для отримання ключів, ведуть детальний журнал доступу, підтримують автоматичну ротацію та інтегруються з Kubernetes через механізм Secrets Store CSI Driver. Рисунок 9.2 ілюструє загальну архітектуру трирівневого управління ключами.

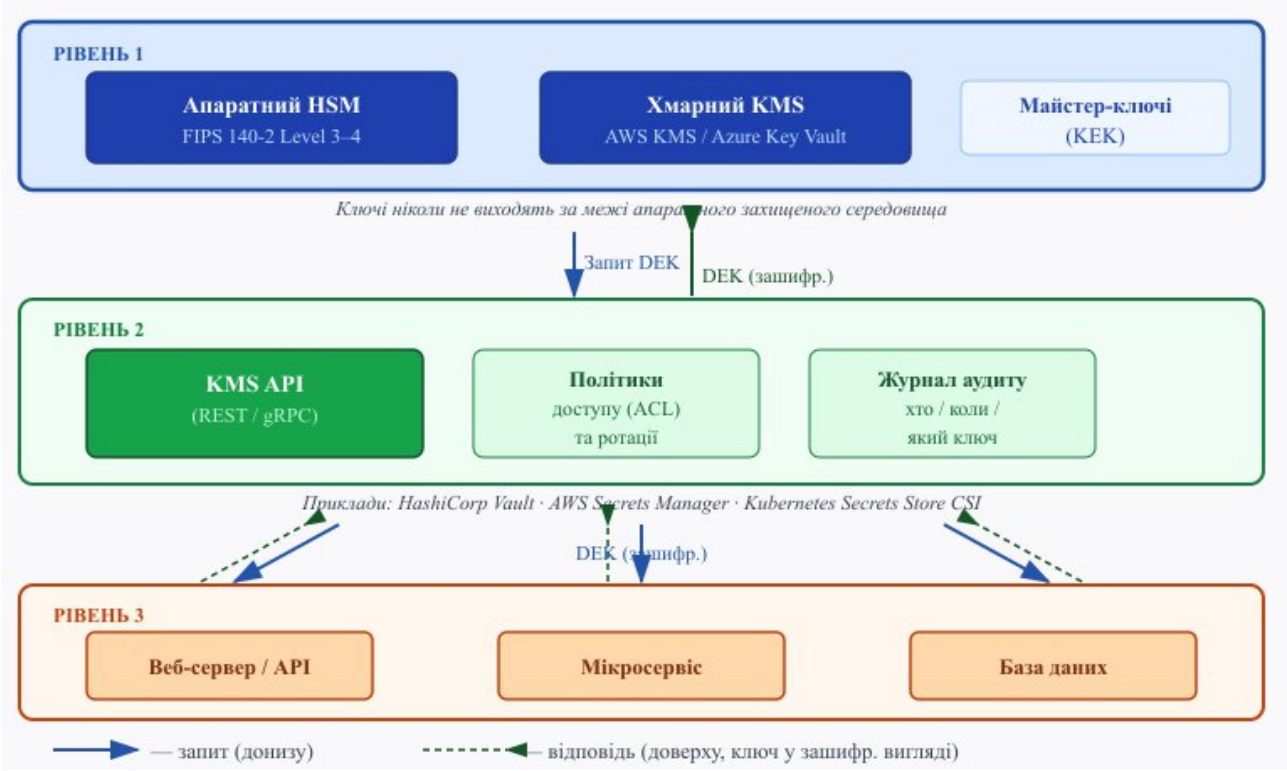

Рисунок 9.2 — Трирівнева архітектура централізованого управління ключами (KMS)

HashiCorp Vault є одним із найпоширеніших програмних рішень для централізованого управління секретами у хмарних та контейнерних середовищах. Vault підтримує різні «бекенди» для зберігання секретів (Key-Value, Transit для шифрування «як сервіс», PKI для управління сертифікатами)

та різні методи автентифікації (LDAP, AWS IAM, Kubernetes ServiceAccount). Принцип роботи Vault базується на динамічних секретах: замість того, щоб зберігати статичний пароль до бази даних, Vault генерує тимчасові облікові дані для кожного запиту з обмеженим часом дії. Наприклад, для отримання ключа через Vault CLI у Лінукс достатньо виконати `vault kv get secret/myapp/db_password` — за умови попередньої автентифікації та наявності відповідних прав. Після закінчення TTL (Time To Live) Vault автоматично відкликає виданий секрет, що суттєво зменшує наслідки його можливого витоку.

9.4. Інфраструктура відкритих ключів (PKI) та стандарти

Інфраструктура відкритих ключів (Public Key Infrastructure, PKI) — це сукупність технологій, процедур і організаційних заходів, що забезпечують управління відкритими ключами та цифровими сертифікатами в масштабах організації або всього Інтернету. Основна проблема, яку вирішує PKI — це проблема автентичності відкритого ключа: як переконатися, що отриманий відкритий ключ справді належить стороні, за яку вона себе видає, а не зловмиснику, що виконує атаку «людина посередині»? Відповідь PKI полягає у введенні довіреної третьої сторони — Центру сертифікації (Certification Authority, CA), — який своїм електронним підписом засвідчує відповідність між відкритим ключем та ідентифікатором його власника. Таблиця 9.5 описує основні компоненти PKI та їх функції.

Таблиця 9.5 — Основні компоненти інфраструктури відкритих ключів (PKI)

Компонент PKI	Функція	Аналогія
Root CA (кореневий)	Вершина ієрархії; самопідписаний сертифікат; підписує Intermediate CA	Міністерство, що засновує реєстр нотаріусів
Intermediate CA (підпорядкований)	Видає сертифікати кінцевим сутностям; ізолює Root CA від мережі	Нотаріальна контора, уповноважена міністерством
RA (реєстраційний орган)	Перевіряє особу заявника перед тим, як CA видасть сертифікат	Відділ прийому документів перед нотаріусом
Сертифікат X.509	Зв'язує відкритий ключ з ідентифікатором власника; підписаний CA	Паспорт, виданий державою
CRL	Список відкликаних сертифікатів; оновлюється CA за розкладом	Чорний список недійсних паспортів
OCSP	Протокол онлайн-перевірки статусу сертифіката в реальному	Запит до онлайн-бази паспортного контролю

	часі	

Цифровий сертифікат стандарту X.509 є основним документом PKI і містить відкритий ключ суб'єкта, відомості про власника (Common Name, Organization, Country), серійний номер, терміни дії (Not Before / Not After), а також цифровий підпис видаючого CA. Структура X.509 v3 також містить поля розширень (Extensions): наприклад, Subject Alternative Names (SAN) визначає DNS-імена та IP-адреси, для яких дійсний сертифікат; KeyUsage обмежує допустимі операції (підпис, шифрування); ExtendedKeyUsage уточнює призначення (TLS-автентифікація сервера, підпис коду). Щоб переглянути повний вміст сертифіката у Лінукс, достатньо виконати наступну команду:

```
openssl x509 -in server.crt -text -noout
```

Процес отримання сертифіката починається зі створення запиту на підписання — Certificate Signing Request (CSR). Власник генерує пару ключів і формує CSR, що містить відкритий ключ та відомості про заявника, підписані власним закритим ключем. Рисунок 9.3 ілюструє процес видачі сертифіката від генерації ключів до отримання готового сертифіката.

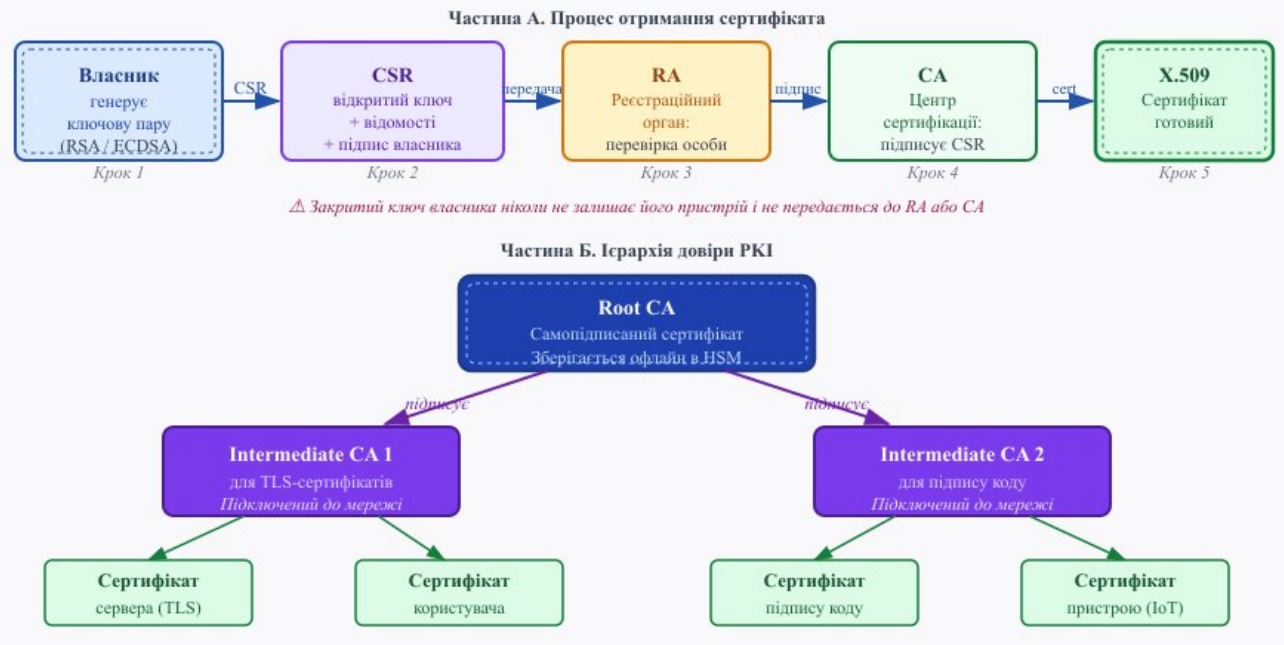

Рисунок 9.3 — Процес видачі сертифіката X.509

Конкретна послідовність команд для генерації ключової пари, формування CSR та перевірки отриманого сертифіката в Лінукс виглядає так. Спочатку генерується закритий ключ і CSR в один крок:

```
openssl req -new -newkey rsa:4096 -nodes \
  -keyout server.key -out server.csr \
  -subj "/C=UA/ST=Odesa/O=University/CN=example.com"
```

Після отримання підписаного сертифіката від CA можна перевірити ланцюжок довіри:

```
# Перевірка ланцюжка: root-ca.pem → intermediate-ca.pem → server.crt
openssl verify -CAfile root-ca.pem -untrusted intermediate-ca.pem server.crt

# Перевірка строку дії сертифіката
openssl x509 -in server.crt -noout -dates
```

Ієрархія довіри в PKI будується від кореневого CA (Root CA) — найвищого рівня, чий сертифікат є самопідписаним і заздалегідь встановленим у браузерах та операційних системах як «якір довіри». Оскільки Root CA є найбільш критичним компонентом, його закритий ключ зберігається в HSM, а сам Root CA не підключається до мережі постійно — він активується лише для підписання сертифікатів Intermediate CA. Якщо Intermediate CA буде скомпрометовано, Root CA може відкликати його сертифікат, не змінюючи власного ключа. Механізм перевірки відкликання реалізується через CRL (Certificate Revocation List) — список серійних номерів відкликаних сертифікатів, що публікується CA за розкладом — та через протокол OCSP (Online Certificate Status Protocol), що дозволяє перевірити статус сертифіката в реальному часі без необхідності завантаження повного списку CRL. OCSP є кращим вибором для динамічних середовищ, тоді як CRL підходить для сценаріїв без постійного підключення.

Сучасним доповненням до PKI є механізм Certificate Transparency (CT), запроваджений Google у 2013 році. CT вимагає, щоб усі публічно видані сертифікати реєструвалися у загальнодоступних журналах (CT Logs) до того, як браузери почнуть їм довіряти. Це дозволяє власникам доменів та спеціалізованим сервісам (наприклад, crt.sh) автоматично виявляти несанкціоновано видані сертифікати — наприклад, якщо хтось спробує отримати сертифікат для вашого домену в іншому CA. Починаючи з Chrome 68 (2018 р.), браузер відмовляється довіряти TLS-сертифікатам, що не присутні принаймні у двох незалежних CT-журналах. Для корпоративних PKI, що не видають публічних сертифікатів, CT не є обов'язковим, проте аналогічний принцип журналювання варто реалізовувати внутрішньо для аудиту.

Нормативне регулювання управління криптографічними ключами визначається низкою міжнародних та національних стандартів. Американський стандарт NIST SP 800-57 «Recommendation for Key Management» є одним із найбільш детальних і широко застосовуваних — він охоплює всі аспекти від вибору алгоритмів і довжини ключів до управління їх повним життєвим циклом. Стандарт ISO/IEC 11770 «Key Management» визначає загальні принципи та конкретні протоколи встановлення ключів. FIPS 140-2/140-3 регламентує вимоги до криптографічних модулів, у тому числі до реалізації захисту ключів. В Україні питання криптографічного захисту регулюються Законом «Про електронні довірчі послуги», підзаконними актами ДСТСЗІ та стандартом ДСТУ 4145, що визначають вимоги до електронного підпису на

основі еліптичних кривих, акредитованих центрів сертифікації ключів (АЦСК) та застосування засобів ЕЦП у державних системах. У таблиці 9.6 систематизовано типові помилки управління ключами, їх наслідки та способи запобігання.

Таблиця 9.6 — Типові помилки управління ключами та способи їх запобігання

Типова помилка	Можливий наслідок	Спосіб запобігання
Hardcoded keys у вихідному коді	Витік ключа через публічний Git-репозиторій	Змінні середовища, HashiCorp Vault, AWS Secrets Manager
Зберігання ключа разом із зашифрованими даними	Повна втрата захисту при компрометації носія	Розділене зберігання: дані окремо, ключі — у KMS або HSM
Відсутність ротації ключів	Довгострокова компрометація після витоку	Автоматизована ротація через KMS за розкладом
Використання ключа для різних цілей (підпис + шифрування)	Атаки крос-використання; складніший аудит	Окремі ключові пари для кожного призначення
Ігнорування прострочених сертифікатів	Відмова сервісу; попередження безпеки у браузерах	Моніторинг строків дії (cert-manager, Vault PKI)
Слабкий генератор ключів / мала довжина	Підбір ключа за розумний час	CSPRNG ОС; RSA ≥ 2048 біт, AES ≥ 128 біт, ECDSA P-256+

У 2019 році дослідники виявили, що тисячі приватних ключів RSA та SSH були випадково опубліковані у публічних GitHub-репозиторіях. Автоматичний сканер truffleHog за кілька хвилин знаходив їх через пошук характерних рядків (-----BEGIN RSA PRIVATE KEY-----). Частина ключів продовжувала використовуватися роками після публікації, що давало зловмисникам постійний доступ до відповідних систем. Цей випадок підкреслює важливість як технічних засобів (pre-commit hooks для сканування секретів), так і організаційних процедур (обов'язкова перевірка репозиторіїв перед публікацією).

ТЕМА 10. ЗАХИЩЕНІ ПРОТОКОЛИ ПЕРЕДАЧІ ДАНИХ ТА ЇХ РЕАЛІЗАЦІЯ

10.1. Принципи побудови захищених протоколів передачі даних

Будь-яка передача даних комп'ютерною мережею несе в собі потенційні ризики: зловмисник може перехопити дані в дорозі, підмінити їх або видати себе за законного відправника. Захищені протоколи передачі даних — це стандартизовані правила обміну інформацією між двома або більше сторонами, які технічними засобами гарантують конфіденційність, цілісність та автентичність переданих даних. Їх поява стала відповіддю на фундаментальну проблему відкритих мереж: стек протоколів TCP/IP, розроблений у 1970-х роках, не передбачав засобів захисту від зловмисних учасників. Оскільки повністю замінити базові мережеві протоколи неможливо, захист було реалізовано на вищих рівнях моделі OSI у вигляді окремого «захисного шару», що огортає вже існуючі протоколи. Саме цей підхід дав початок усім сучасним захищеним протоколам.

В основі будь-якого захищеного протоколу лежать три ключові завдання, тісно пов'язані між собою. Перше — забезпечення конфіденційності: шифрування даних таким чином, щоб лише отримувач міг їх розшифрувати. Друге — забезпечення цілісності: гарантування того, що дані не були змінені під час передачі. Третє — автентифікація сторін зв'язку, яка підтверджує справжність учасників обміну та унеможливлює атаку підміни (Man-in-the-Middle, MitM). Кожне з цих завдань вирішується за допомогою різних криптографічних примітивів, проте захищений протокол поєднує їх в єдину систему так, щоб одна властивість підкріплювала іншу. Атака Man-in-the-Middle — один із класичних прикладів того, що може статися, якщо хоча б одне з цих трьох завдань не вирішено.

На рисунку 10.1 зображена схема атаки Man-in-the-Middle (MitM). Схема складається з трьох прямокутників, розташованих горизонтально: «Аліса» зліва, «Зловмисник (MitM)» по центру та «Боб» праворуч. Від Аліси до Зловмисника спрямована стрілка з підписом «TLS-з'єднання з підробленим сертифікатом зловмисника». Від Зловмисника до Боба — окрема стрілка з підписом «окреме TLS-з'єднання з легітимним сертифікатом Боба». По центру під фігурою Зловмисника написано «перехоплює, читає та пересилає всі дані». Нижче схеми розміщено підпис: «Умова атаки: Аліса не перевіряє сертифікат, або він є самопідписаним».

Рисунок 10.1 — Схема атаки Man-in-the-Middle при відсутності перевірки сертифіката

Процес встановлення захищеного з'єднання завжди починається з «рукостискання» (handshake) — початкового обміну повідомленнями, під час якого сторони домовляються про спільні параметри безпеки. На цьому етапі вирішуються питання: які алгоритми шифрування та хешування будуть використовуватися, як буде проведено взаємну автентифікацію, і яким чином сторони отримають спільний секретний ключ для шифрування подальшого трафіку. Окремої уваги заслуговує завдання погодження ключів: обидві сторони мають отримати однаковий ключ шифрування, не передаючи його відкрито по мережі. Для цього використовується алгоритм обміну ключами Діффі-Геллмана (DH) та його еліптично-криптографічний варіант ECDH, який дозволяє двом сторонам обчислити однакове спільне значення навіть у разі, якщо весь їхній обмін спостерігається третьою стороною.

На рисунку 10.2 зображена спрощена схема принципу обміну ключами Діффі-Геллмана. Схема складається з двох колонок — «Аліса» і «Боб» — та горизонтальних стрілок між ними. Спочатку обидві сторони публічно домовляються про відкриті параметри: велике просте число p та генератор g. Далі Аліса обирає таємне число a і обчислює A = g^a mod p; Боб обирає b і обчислює B = g^b mod p. Аліса надсилає A Бобу (стрілка вправо), Боб надсилає B Алісі (стрілка вліво) — обидва значення передаються відкритим каналом. Нарешті обидві сторони самостійно обчислюють спільний секрет: Аліса — B^a mod p, Боб — A^b mod p. Обидва результати однакові й дорівнюють g^(ab) mod p. Внизу схеми підпис: «Зловмисник бачить A і B, але не може відновити g^ab без знання a або b».

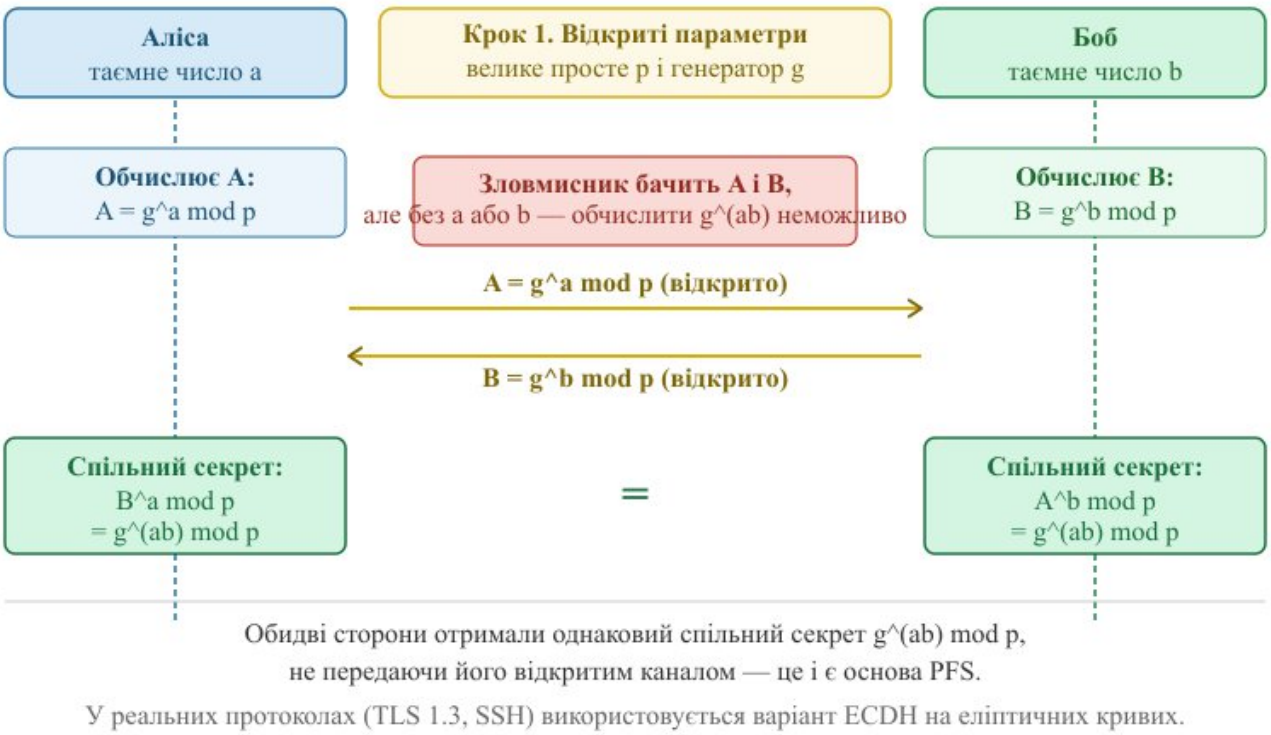

Рисунок 10.2 — Принцип обміну ключами Діффі-Геллмана

Важливою концепцією сучасних захищених протоколів є властивість прямої секретності (Perfect Forward Secrecy, PFS). Вона означає, що навіть якщо довгостроковий приватний ключ сервера буде скомпрометований у майбутньому, зловмисник не зможе розшифрувати раніше записаний трафік. Це досягається тим, що для кожного нового сеансу зв'язку генерується унікальна пара тимчасових ключів, а після завершення сеансу вони знищуються. Протоколи, що реалізують PFS, використовують набори шифрів на основі DHE або ECDHE (ephemeral Diffie-Hellman), де «E» означає «тимчасовий». Ця властивість стала обов'язковою вимогою у TLS 1.3 і є бажаною при налаштуванні SSH-сервера.

Для розуміння різниці між протоколами варто звернути увагу на рівень моделі OSI, на якому вони працюють, оскільки це визначає сферу їх застосування та спосіб взаємодії з іншими компонентами системи. Протоколи прикладного рівня, такі як SSH, захищають конкретний тип трафіку і є повністю прозорими для мережі. Протоколи транспортного рівня, такі як TLS, захищають потік байтів між двома процесами і можуть застосовуватися до будь-якого протоколу прикладного рівня (HTTP→HTTPS, SMTP→SMTPS тощо). Протоколи мережевого рівня, такі як IPsec, захищають IP-пакети і тому є прозорими для всіх вищих рівнів — застосунок навіть не знає, що його трафік зашифрований. У таблиці 10.1 наведено узагальнене порівняння основних захищених протоколів.

Таблиця 10.1 — Порівняння основних захищених протоколів передачі даних

Протокол	Рівень OSI	Призначення	Автентифікація	Шифрування

Протокол	Рівень OSI	Призначення	Автентифікація	Шифрування
SSH	Прикладний (7)	Захищене дист. управління, тунелювання, SCP/SFTP	Пароль, ключова пара, сертифікат	AES-GCM, ChaCha20-Poly1305
TLS 1.3	Транспортний (4)	Захист веб-трафіку, API, пошти, будь-якого TCP-застосунку	Сертифікати X.509, PSK	AES-GCM, ChaCha20-Poly1305 (лише AEAD)
TLS 1.2	Транспортний (4)	Те саме; широко підтримується, але потребує ретельного налаш.	Сертифікати X.509	AES-GCM, AES-CBC та ін. (залежить від шифронабору)
IPsec	Мережевий (3)	VPN, захист IP-пакетів між мережами та хостами	IKE (сертифікати, PSK)	AES-CBC, AES-GCM, ChaCha20
DTLS	Транспортний (4)	Захист UDP-потоків: VoIP, відеоконференції, VPN	Сертифікати X.509	AES-GCM (аналог TLS для UDP)

10.2. Протоколи SSH та TLS/SSL: архітектура та механізми захисту

Протокол SSH (Secure Shell) був розроблений у 1995 році фінським дослідником Тату Юленем як пряма відповідь на виявлену атаку на мережу Хельсінського університету, де зловмисники перехоплювали паролі, передані у відкритому вигляді через протоколи telnet і rsh. Нині SSH є де-факто стандартом для захищеного дистанційного управління сервером у системах Linux. Протокол складається з трьох рівнів: транспортний рівень SSH відповідає за встановлення захищеного з'єднання та автентифікацію сервера за відкритим ключем; рівень автентифікації користувача підтверджує особу клієнта; рівень з'єднання мультиплексує один фізичний канал на декілька логічних потоків — наприклад, окремий термінал і сесію передачі файлів водночас.

Автентифікація на основі ключової пари є найбезпечнішим методом входу у SSH і настійно рекомендується замість парольної автентифікації. Принцип роботи такий: адміністратор генерує асиметричну ключову пару — відкритий ключ розміщується на сервері у файлі ~/.ssh/authorized_keys, тоді як закритий ключ зберігається лише у клієнта. При підключенні сервер надсилає клієнту випадковий виклик (challenge), підписаний клієнтом своїм закритим ключем; сервер перевіряє підпис відкритим ключем і дозволяє доступ. Закритий ключ ніколи не покидає машину клієнта, тому навіть якщо зловмисник отримає копію authorized_keys — автентифікуватися він не зможе.

В Linux генерація ключової пари виконується командою ssh-keygen -t ed25519, де ed25519 — сучасний алгоритм підпису на основі еліптичних кривих.

Протокол TLS (Transport Layer Security) є еволюцією SSL (Secure Sockets Layer), розробленого компанією Netscape у 1994 році. Усі версії SSL та TLS 1.0 і 1.1 вважаються застарілими та вразливими; у сучасних системах використовуються виключно TLS 1.2 і TLS 1.3. TLS розташовується між транспортним рівнем (TCP) та рівнем застосунку: він отримує від застосунку байтовий потік, шифрує та передає через TCP. З точки зору застосунку, TLS виглядає як звичайний TCP-з'єднок. Саме тому протоколи, що раніше передавали дані відкрито (HTTP, SMTP, IMAP), отримали захищені варіанти (HTTPS, SMTPS, IMAPS) шляхом простого «загортання» у TLS, без зміни самих протоколів вищого рівня.

Рукостискання TLS 1.3, на відміну від TLS 1.2, відбувається за один повний оберт (1 RTT) замість двох (2 RTT), що суттєво зменшує затримку при встановленні з'єднання. Клієнт вже в першому повідомленні (ClientHello) надсилає параметри обміну ключами, тому шифрований трафік може починатися значно раніше. На рисунку 10.3 зображена послідовна схема рукостискання TLS 1.3. Схема складається з двох вертикальних ліній — «Клієнт» зліва і «Сервер» справа — та горизонтальних стрілок між ними. Перша стрілка від клієнта до сервера підписана «ClientHello (версія, підтримувані шифри, ключ ECDH)». Від сервера до клієнта йдуть три стрілки поспіль: «ServerHello (обраний шифр, ключ ECDH сервера)», «EncryptedExtensions + Certificate + CertificateVerify» та «Finished». Остання стрілка від клієнта до сервера підписана «Finished». Після цього позначено двосторонню стрілку з написом «Зашифровані дані застосунку». Праворуч від схеми підписи: між першою парою стрілок — «1 RTT», а нижче — «Дані починаються тут».

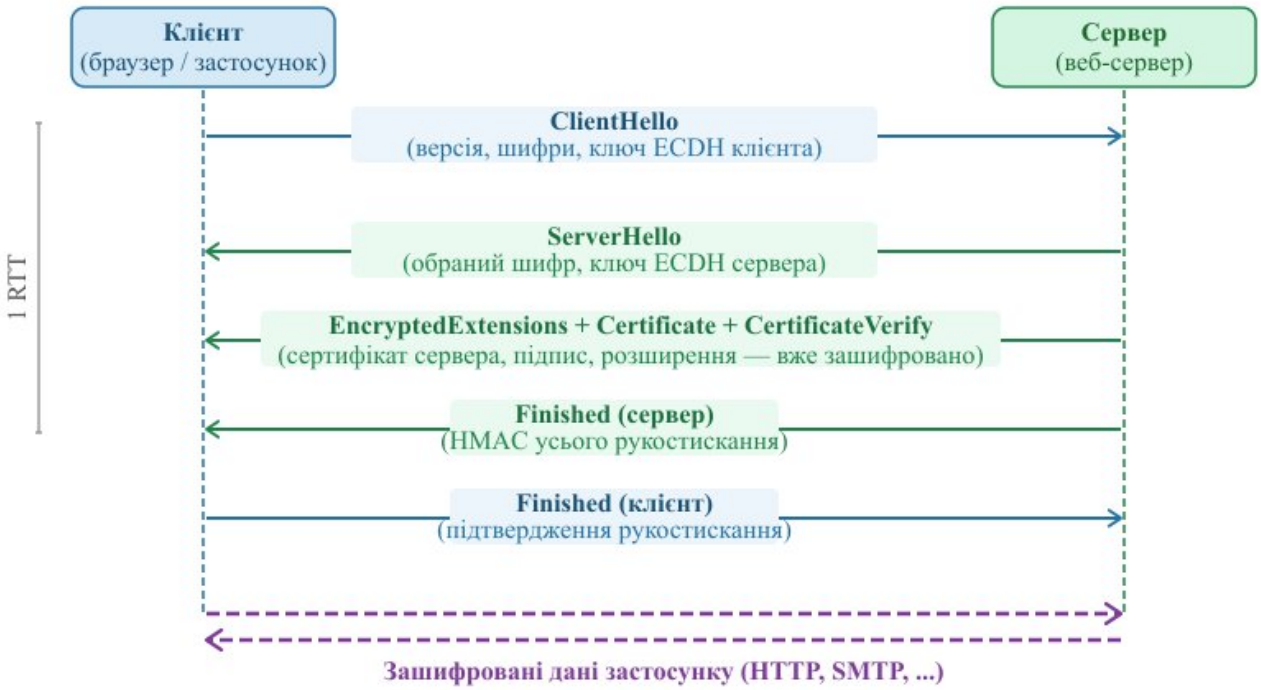

Рисунок 10.3 — Схема рукостискання TLS 1.3 (1 RTT)

Ще однією радикальною відмінністю TLS 1.3 є видалення підтримки слабких і застарілих алгоритмів: RC4, MD5, SHA-1, DES, 3DES, RSA-шифрування (без PFS) більше не допускаються. У TLS 1.3 дозволено лише шифронабори з прямою секретністю та AEAD-шифрами (Authenticated Encryption with Associated Data), де шифрування та перевірка цілісності виконуються одночасно в межах одного алгоритму. Це значно спрощує безпечне налаштування сервера, адже вибір шифронаборів по суті зводиться до вибору між AES-GCM і ChaCha20-Poly1305. Порівняння версій протоколів TLS/SSL з точки зору безпеки та статусу наведено в таблиці 10.2.

Таблиця 10.2 — Хронологія версій TLS/SSL: статус та ключові характеристики

Версія	Рік	Статус	RFC	Ключові проблеми / особливості
SSL 2.0	1995	Заборонена (RFC 6176)	RFC 6176	Слабкі шифри, відсутність захисту рукостискання, немає PFS
SSL 3.0	1996	Заборонена (RFC 7568)	RFC 6101, 7568	Атака POODLE: розшифрування даних через padding oracle
TLS 1.0	1999	Застаріла (RFC 8996)	RFC 2246, 8996	Атаки BEAST, CRIME; слабкий IV для CBC; немає AEAD
TLS 1.1	2006	Застаріла (RFC 8996)	RFC 4346, 8996	Виправлено BEAST, але немає AEAD; підтримка RC4 та MD5
TLS 1.2	2008	Рекомендована (за умов правил. налаш.)	RFC 5246	Підтримка AEAD (AES-GCM), SHA-256+; PFS — не обов'язковий
TLS 1.3	2018	Найкраща — рекомендована	RFC 8446	1-RTT handshake; лише AEAD; PFS обов'язковий; видалено слабкі алгоритми

Поняття шифронабору (cipher suite) є важливим при налаштуванні TLS 1.2, де вибір широкий, а частина варіантів є небезпечними. Назва шифронабору, наприклад TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, читається так: ECDHE — алгоритм обміну ключами з прямою секретністю, RSA — алгоритм підпису для автентифікації сервера, AES_256_GCM — симетричний AEAD-шифр, SHA384 — хеш для HMAC і псевдовипадкових функцій. Адміністратор може обмежити список дозволених шифронаборів у конфігурації сервера; в Linux для nginx це директива ssl_ciphers у файлі /etc/nginx/nginx.conf, а для Apache — SSLCipherSuite у /etc/apache2/mods-enabled/ssl.conf. Рекомендовані шаблони конфігурацій для різних серверів публікує Mozilla у проєкті «Mozilla SSL Configuration Generator».

Практична перевірка конфігурації TLS-сервера виконується за допомогою утиліт командного рядка, доступних у Linux. Команда openssl s_client -connect example.com:443 -tls1_3 встановлює TLS-з'єднання та виводить детальну інформацію — узгоджену версію протоколу, обраний шифронабір та ланцюг сертифікатів. Для розгорнутого аудиту всіх підтримуваних версій та шифронаборів використовується утиліта nmap зі скриптом ssl-enum-ciphers: команда nmap --script ssl-enum-ciphers -p 443 example.com виводить повний перелік підтримуваних шифрів із оцінкою їх надійності. Утиліта testssl.sh є більш спеціалізованим інструментом аудиту: вона перевіряє сервер на наявність відомих вразливостей (POODLE, BEAST, Heartbleed та ін.) і формує детальний звіт.

10.3. Протокол IPsec, сертифікати X.509 та ланцюги довіри

Протокол IPsec (Internet Protocol Security) відрізняється від SSH та TLS тим, що захищає трафік на мережевому рівні моделі OSI, тобто безпосередньо на рівні IP-пакетів. Це означає, що будь-який протокол вищого рівня — TCP, UDP, ICMP тощо — автоматично отримує захист без будь-яких змін у застосунку. IPsec є набором стандартів, що включає: протокол AH (Authentication Header) для забезпечення цілісності та автентичності пакетів без шифрування; протокол ESP (Encapsulating Security Payload) для шифрування та забезпечення цілісності одночасно; протокол IKE (Internet Key Exchange) для встановлення захищених асоціацій між сторонами та обміну ключами. У більшості практичних застосувань використовується ESP, оскільки він забезпечує і шифрування, і цілісність.

IPsec підтримує два режими роботи, що принципово відрізняються за принципом захисту та областю застосування. У транспортному режимі захищається лише корисне навантаження IP-пакета — тобто TCP/UDP-сегмент — тоді як оригінальний IP-заголовок залишається відкритим. Цей режим використовується для захисту з'єднань між двома конкретними хостами. У тунельному режимі весь оригінальний IP-пакет (разом із заголовком) шифрується і вміщується всередину нового IP-пакета — завдяки цьому внутрішні адреси сховані від зовнішньої мережі, і саме цей режим є основою VPN-тунелів. Порівняння двох режимів наведено в таблиці 10.3.

Таблиця 10.3 — Порівняння транспортного та тунельного режимів IPsec

Характеристика	Транспортний режим	Тунельний режим
Що захищається	Лише корисне навантаження (payload) пакета	Весь оригінальний IP-пакет (разом із заголовком)
IP-заголовок	Оригінальний — відкритий	Новий зовнішній заголовок; оригінальний приховано
Типове застосування	Захист між двома конкретними хостами (host-to-host)	VPN між мережами (site-to-site) або remote access VPN

Характеристика	Транспортний режим	Тунельний режим
Приховування топології	Ні — адреси відправника і одержувача відкриті	Так — внутрішні адреси не видно зовні
Накладні витрати	Менші — без додаткового IP-заголовка	Більші — додається новий IP-заголовок та ESP-трейлер

На рисунку 10.4 зображена схема інкапсуляції пакету у тунельному режимі IPsec ESP. Схема показує два рядки. Верхній рядок підписано «Оригінальний IP-пакет» і містить три блоки зліва направо: «IP-заголовок (src: 10.0.0.1)», «TCP-заголовок», «Дані». Нижній рядок підписано «Після IPsec ESP тунельний режим» і містить п'ять блоків: «Новий IP-заголовок (src: 203.0.113.1)», «ESP-заголовок», «[Зашифровано: оригінальний IP-заг. + TCP-заг. + Дані]», «ESP-трейлер», «ESP Auth». Фігурна дужка охоплює всі три зашифрованих блоки та підписана «ESP шифрування та цілісність». Стрілка між двома рядками підписана «Новий пакет, що передається через Інтернет».

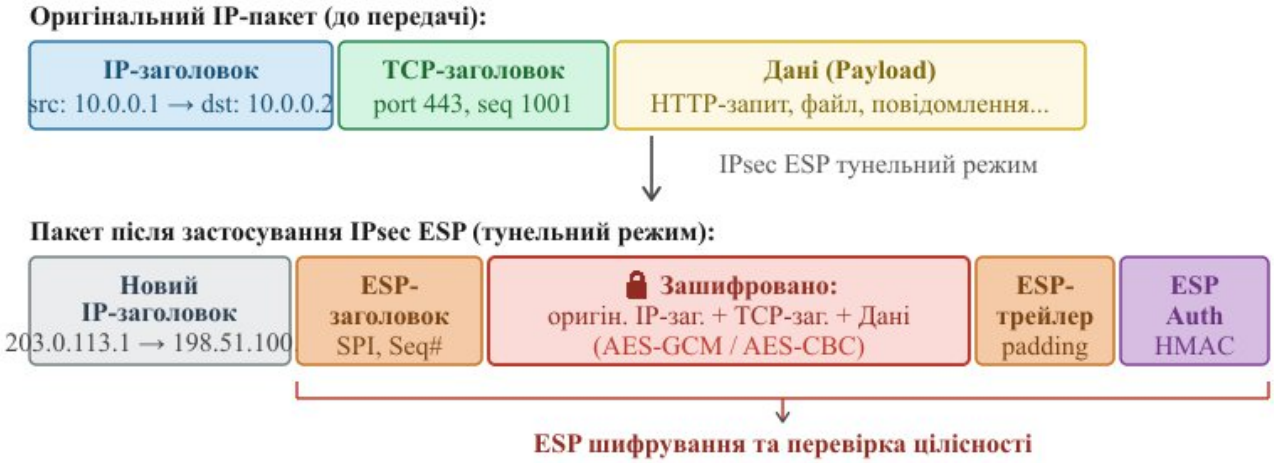

Рисунок 10.4 — Інкапсуляція IP-пакета в тунельному режимі IPsec ESP

Ключовим елементом автентифікації в TLS, IPsec та інших протоколах є цифровий сертифікат формату X.509. Сертифікат — це структурований електронний документ, підписаний цифровим підписом центру сертифікації (Certificate Authority, CA), який стверджує: «цей відкритий ключ дійсно належить суб'єкту з таким іменем». Структура сертифіката включає кілька важливих полів, кожне з яких відіграє свою роль у системі довіри. Детальний перелік полів із поясненнями та прикладами наведено в таблиці 10.4.

Таблиця 10.4 — Основні поля сертифіката X.509 v3

Поле сертифіката	Призначення	Приклад значення
Version	Версія формату X.509	v3 (найпоширеніша)
Serial Number	Унікальний номер у межах CA; використовується в CRL та OCSP	04:A1:2B:3C:D4:...
Issuer	Ім'я центру сертифікації, що	CN=R3, O=Let's Encrypt, C=US

Поле сертифіката	Призначення	Приклад значення
	видав сертифікат	
Validity (Not Before / Not After)	Термін дії: дата початку та дата закінчення	2024-01-01 — 2024-09-28
Subject	Ім'я власника сертифіката (домен, організація, особа)	CN=example.com
Subject Public Key Info	Відкритий ключ та алгоритм	RSA 2048 bit / EC P-256
Subject Alternative Name (SAN)	Перелік усіх доменів/IP, для яких дійсний сертифікат	DNS:example.com, DNS:www.example.com, IP:93.184.216.34
Signature Algorithm	Алгоритм цифрового підпису CA	SHA256withRSA / ecdsa-with-SHA256
Signature Value	Власне цифровий підпис CA — захищає всі вище поля	(двійкові дані)

На рисунку 10.5 зображена ієрархічна схема ланцюга довіри сертифікатів X.509. Схема має вигляд дерева з трьох рівнів. На вершині — прямокутник «Кореневий CA (Root CA)» з підписом «Самопідписаний, вбудований в ОС / браузер». Від нього вниз ідуть дві стрілки з підписом «підписує», що ведуть до двох прямокутників другого рівня: «Проміжний CA 1 (Intermediate CA)» та «Проміжний CA 2». Від кожного з них ідуть стрілки вниз до прямокутників третього рівня: «Сертифікат example.com», «Сертифікат mail.com» та «Сертифікат api.example.com». Праворуч від дерева розміщено блок зі стрілкою: «Клієнт перевіряє підписи знизу вгору, доки не дійде до довіреного кореневого CA». Знизу підпис: «Якщо Root CA є у сховищі довіри клієнта — ланцюг вважається дійсним».

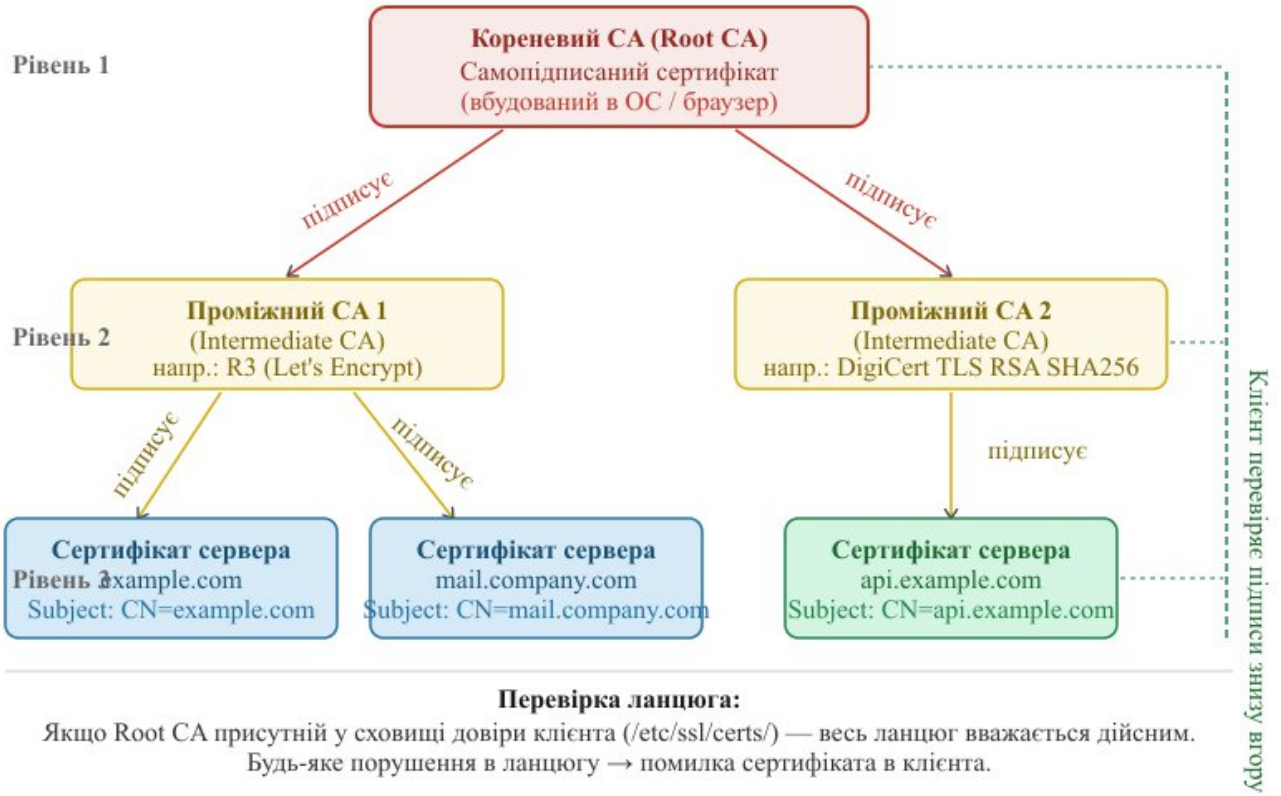

Рисунок 10.5 — Ієрархічна структура ланцюга довіри сертифікатів X.509

Концепція ланцюга довіри (chain of trust) є основою системи PKI (Public Key Infrastructure). Операційна система Linux та браузери містять вбудований список довірених кореневих CA. Кореневий CA підписує сертифікати проміжних CA, а ті, своєю чергою, видають сертифікати кінцевим суб'єктам. Коли клієнт отримує сертифікат сервера, він перевіряє підпис на ньому; далі перевіряє підпис на сертифікаті проміжного CA; якщо він виданий кореневим CA з довіреного сховища — ланцюг вважається дійсним. Якщо хоча б одне посилання недійсне — клієнт відображає попередження. Сховище довірених CA в Linux знаходиться у директорії /etc/ssl/certs/, а додавання корпоративного CA виконується командою update-ca-certificates.

Механізм відкликання сертифікатів є важливим елементом безпеки PKI. Якщо приватний ключ було скомпрометовано або сертифікат видано помилково, CA повинен відкликати його до закінчення терміну дії. Список відкликання сертифікатів (CRL) — це підписаний список серійних номерів відкликаних сертифікатів, який CA публікує регулярно. Протокол OCSP (Online Certificate Status Protocol) дозволяє клієнту перевірити статус конкретного сертифіката в реальному часі. Сучасним і ефективнішим підходом є OCSP Stapling: сервер сам завчасно отримує підписану OCSP-відповідь від CA і надає її клієнту разом із сертифікатом, що зменшує затримку та навантаження на CA. В Linux перегляд сертифіката виконується командою openssl x509 -in certificate.crt -text -noout, яка виводить усі поля у читабельному форматі.

10.4. Типові помилки конфігурації захищених протоколів та їх наслідки

Навіть бездоганно спроєктований протокол стає беззахисним у разі його неправильного налаштування. Практика інформаційної безпеки показує, що переважна більшість реальних інцидентів пов'язана не з фундаментальними вразливостями самих протоколів, а з помилками при їх розгортанні та конфігурації. Більшість помилок підпадає під кілька категорій: підтримка застарілих версій протоколів, слабкі або некоректно налаштовані сертифікати, відключена перевірка ланцюга довіри та небезпечні налаштування за замовчуванням. Вивчення цих помилок дозволяє як уникнути їх при самостійному налаштуванні, так і виявляти при аудиті чужої інфраструктури — обидва завдання є ключовими для фахівця з кібербезпеки.

Підтримка застарілих версій протоколів — найбільш поширена і небезпечна помилка. SSL 3.0 вразливий до атаки POODLE (Padding Oracle On Downgraded Legacy Encryption), яка дозволяє зловмиснику, що перехоплює трафік, поступово відновлювати вміст зашифрованих блоків. TLS 1.0 вразливий до атак BEAST (Browser Exploit Against SSL/TLS) і CRIME, пов'язаних зі слабкістю режиму CBC та алгоритму стиснення даних. Попри те, що сучасні браузери давно відмовились від цих версій, сервери нерідко продовжують їх підтримувати «для сумісності зі старими клієнтами» — і це відкриває вектор для атак типу downgrade, коли зловмисник змушує клієнт використати слабшу версію протоколу. В Linux перевірити підтримку застарілого протоколу можна командою openssl s_client -connect example.com:443 -ssl3: якщо з'єднання встановилося — сервер потребує негайного перегляду конфігурації.

Окремого розгляду заслуговує реальний інцидент, пов'язаний не з помилкою конфігурації, а з помилкою в реалізації криптографічної бібліотеки — вразливість Heartbleed (CVE-2014-0160), виявлена у квітні 2014 року в бібліотеці OpenSSL. Вразливість містилася в реалізації розширення TLS «heartbeat» (RFC 6520), призначеного для перевірки активності з'єднання. Через помилку перевірки меж буфера зловмисник міг надіслати спеціально сформований запит і отримати у відповідь до 64 кілобайт пам'яті сервера — з яких за певних умов можна було витягти приватний ключ SSL/TLS, паролі або сесійні токени. Проблема торкнулась сотень тисяч серверів по всьому світу і наочно продемонструвала, що навіть правильно налаштований протокол не захищений від вразливостей у базових бібліотеках. Урок Heartbleed — своєчасне оновлення криптографічних бібліотек є такою ж обов'язковою практикою, як і правильна конфігурація.

Найпоширеніші помилки конфігурації TLS, їх наслідки та заходи виправлення систематизовано у таблиці 10.5. Ця таблиця може використовуватись як практичний чек-лист при аудиті будь-якого сервера, що приймає HTTPS-з'єднання.

Таблиця 10.5 — Типові помилки конфігурації TLS та заходи їх усунення

Помилка конфігурації	Можливі наслідки	Рекомендований захід
Дозвіл на TLS 1.0/1.1	Downgrade-атаки, POODLE,	Підтримувати лише TLS 1.2 і TLS

Помилка конфігурації	Можливі наслідки	Рекомендований захід
або SSL 3.0	BEAST; розшифрування трафіку	1.3
Самопідписаний сертифікат у продакшні	Відсутність гарантій автентичності сервера; Man-in-the-Middle	Використовувати сертифікати від довірених CA (Let's Encrypt та ін.)
Дозвіл слабких шифронаборів (RC4, DES, NULL)	Розшифрування перехопленого трафіку; порушення конфіденційності	Дозволяти лише AEAD-шифри: AES-GCM, ChaCha20-Poly1305
Вимкнена перевірка відкликання (OCSP/CRL)	Прийняття скомпрометованих сертифікатів; ризик MitM	Увімкнути OCSP Stapling; перевіряти CRL при розгортанні
Відсутність заголовка HSTS	SSL Stripping — примусове переведення з HTTPS на HTTP	Додати Strict-Transport-Security: max-age=63072000; includeSubDomains
Прострочений сертифікат або невірні SAN	Помилки в клієнтів; ризик ігнорування перевірок користувачами	Автоматизувати поновлення (Certbot); моніторити терміни дії
Ігнорування перевірки сертифіката у клієнт. коді (verify=false)	Повна відсутність автентифікації сервера; тривіальний MitM	Ніколи не вимикати перевірку в продакшні; використовувати правильні CA

Для наочної демонстрації різниці між безпечною та небезпечною конфігурацією наведемо приклад. Конфігурація nginx з підтримкою всіх версій TLS та без HSTS виглядає так:

```
# НЕБЕЗПЕЧНА конфігурація (застарілі версії, слабкі шифри)
ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;
ssl_ciphers HIGH:!aNULL:!MD5;
```

Натомість рекомендована сучасна конфігурація, що відповідає вимогам безпеки, має такий вигляд:

```
# БЕЗПЕЧНА конфігурація (лише TLS 1.2 і 1.3, AEAD-шифри, HSTS, OCSP)
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256;
ssl_prefer_server_ciphers off;
add_header Strict-Transport-Security "max-age=63072000" always;
ssl_stapling on;
ssl_stapling_verify on;
```

Помилки конфігурації SSH-сервера є окремою важливою темою. Якщо SSH-порт доступний з інтернету і парольна автентифікація увімкнена, сервер отримуватиме тисячі спроб входу щодоби від автоматизованих сканерів — це легко перевірити командою grep 'Failed password' /var/log/auth.log. Вимкнення парольної автентифікації (PasswordAuthentication no у файлі

/etc/ssh/sshd_config) і перехід виключно на ключову автентифікацію є найефективнішим захисним заходом. Додатково рекомендується вимкнути прямий вхід для root (PermitRootLogin no), обмежити кількість спроб автентифікації (MaxAuthTries 3), встановити тайм-аут для незавершених рукостискань (LoginGraceTime 30), а також видалити підтримку застарілих алгоритмів. Типові помилки конфігурації SSH систематизовано в таблиці 10.6.

Таблиця 10.6 — Типові помилки конфігурації SSH-сервера та безпечні налаштування

Директива / помилка	Ризик	Безпечне налаштування
PasswordAuthentication yes	Brute force та словникові атаки через мережу	PasswordAuthentication no
PermitRootLogin yes	Пряма компрометація root-облікового запису	PermitRootLogin no (або prohibit-password)
Застарілі алгоритми (diffie-hellman-group1-sha1, arcfour, hmac-md5)	Злом обміну ключами; підробка MAC; розшифрування сесії	KexAlgorithms curve25519-sha256; Ciphers aes256-gcm; MACs hmac-sha2-512-etm
X11Forwarding yes (без потреби)	Перехоплення X11-сесії зловмисним сервером	X11Forwarding no (якщо графічний forwarding не потрібен)
Стандартний порт (22) без змін	Мільйони автоматизованих спроб входу від інтернет-сканерів	Зміна порту + fail2ban/UFW; або обмеження доступу по IP
MaxAuthTries не обмежено	Перебір паролів або ключів без обмеження кількості спроб	MaxAuthTries 3; LoginGraceTime 30

Автоматизація поновлення сертифікатів є обов'язковою практикою, яку варто розглянути окремо. З 2020 року для публічних TLS-сертифікатів максимальний термін дії обмежено 398 днями. Прострочений сертифікат негайно спричиняє помилки у всіх клієнтів і фактично робить сервіс недоступним. Стандартним рішенням є Let's Encrypt — безкоштовний CA, що видає сертифікати строком на 90 днів і підтримує повністю автоматизоване поновлення через клієнт Certbot. В Linux встановлення виконується командою apt install certbot python3-certbot-nginx, після чого systemd-таймер автоматично поновлює сертифікат за 30 днів до закінчення. Моніторинг термінів дії всіх сертифікатів у інфраструктурі є обов'язковою частиною системи оперативного моніторингу — це дозволяє попередити інцидент задовго до його настання.

Підсумовуючи, слід наголосити на комплексному характері безпеки захищених протоколів: вона не зводиться до вибору «правильного» протоколу, а є результатом сукупності рішень — підтримуваних версій, набору шифрів, коректних сертифікатів, перевірки відкликання, моніторингу та своєчасного оновлення. Фахівець із кібербезпеки повинен розуміти внутрішні механізми протоколів, вміти їх налаштовувати і перевіряти. Перевірку конфігурації TLS

будь-якого публічного сервера можна здійснити через онлайн-сервіс Qualys SSL Labs (ssllabs.com/ssltest), а інструмент testssl.sh дозволяє провести аналогічний аудит у середовищі Linux без доступу до інтернету.

Тема 11. ВІРТУАЛЬНІ ПРИВАТНІ МЕРЕЖІ ТА ОРГАНІЗАЦІЯ ЗАХИЩЕНИХ ТУНЕЛІВ

11.1 Поняття VPN, топології та сфери застосування

Сучасні інформаційні системи рідко є монолітними: їхні компоненти розподілені між різними фізичними локаціями, хмарними платформами та мобільними пристроями. Така архітектура породжує принципове запитання — як забезпечити безпечну передачу даних через відкриті публічні мережі, зокрема через Інтернет, який за своєю природою є незахищеним середовищем? Саме для вирішення цієї проблеми розроблені технології побудови віртуальних приватних мереж, відомих під абревіатурою VPN (Virtual Private Network). Концептуально VPN — це спосіб організації логічно ізольованого комунікаційного каналу поверх публічної мережевої інфраструктури таким чином, щоб для кінцевих учасників він поводився як виділена захищена лінія зв'язку. Фізично трафік все одно проходить через загальнодоступні маршрутизатори та канали, проте завдяки криптографічному захисту стає нечитабельним для сторонніх спостерігачів.

Ключова ідея VPN полягає у побудові тунелю — механізму інкапсуляції, за якого вихідні мережеві пакети повністю загортаються у нові пакети, що передаються між точками-учасниками захищеного каналу. Уявіть, що ви кладете конфіденційний лист у непрозорий конверт, а потім вкладаєте цей конверт у ще один звичайний поштовий конверт: кур'єр бачить лише зовнішній конверт і доставляє його за вказаною адресою, не маючи жодного уявлення про вміст. Саме так і працює тунелювання у VPN: зовнішній рівень забезпечує доставку, а внутрішній — конфіденційність і цілісність даних. Зашифрований тунель не просто приховує вміст трафіку, але й аутентифікує учасників з'єднання, що унеможливлює підключення зловмисника від імені легітимного вузла. Детальніша схема процесу інкапсуляції наведена у рисунку 11.1. Рисунок 11.1 ілюструє процес інкапсуляції пакетів у VPN-тунелі. На схемі зображений горизонтальний потік зліва направо через чотири послідовні блоки.

Рисунок 11.1 — Процес інкапсуляції пакетів у VPN-тунелі

Сфери застосування VPN надзвичайно широкі. Найбільш поширений сценарій — забезпечення безпечного віддаленого доступу співробітників до корпоративної мережі з домашніх комп'ютерів або мобільних пристроїв, особливо актуальний у часи масового переходу на дистанційну роботу. Інший типовий сценарій — об'єднання офісів компанії в різних містах у єдину приватну мережу без оренди дорогих виділених каналів зв'язку. VPN також активно використовується для захисту трафіку в публічних Wi-Fi мережах (аеропорти, кафе), де ризик перехоплення особливо високий, а також провайдерами хмарних послуг для безпечного з'єднання клієнтів із хмарною інфраструктурою. Крім класичних рішень на базі IPsec та OpenVPN, існує окремий клас — SSL/TLS VPN, що реалізується безпосередньо у браузері або через легкий клієнт і не потребує встановлення додаткових мережевих драйверів: таке рішення особливо зручне для надання обмеженого доступу зовнішнім партнерам або підрядникам до конкретних веб-ресурсів організації. Ще одним сучасним напрямком є SD-WAN (Software-Defined Wide Area Network) — технологія, яка використовує VPN-тунелі як транспортну основу, але додатково забезпечує централізоване управління трафіком і динамічний вибір маршрутів, що актуально для великих розподілених підприємств.

З точки зору топології розрізняють два основних типи VPN-з'єднань. Site-to-site VPN (мережа-мережа) призначений для постійного з'єднання двох або більше локальних мереж: VPN-шлюзи на краях мереж автоматично встановлюють і підтримують тунель між собою, а кінцеві пристрої всередині кожної мережі не знають про існування VPN і працюють так, ніби перебувають в одній локальній мережі. Remote access VPN (VPN із віддаленим доступом) дозволяє окремим користувачам підключатися до корпоративної мережі з будь-якої точки — VPN-клієнт встановлюється безпосередньо на пристрій користувача і встановлює тунель до корпоративного VPN-сервера. Обидва типи використовують однакові криптографічні механізми, але відрізняються архітектурою розгортання, масштабом та особливостями управління.

Таблиця 11.1 — Порівняння топологій VPN

Критерій	Site-to-site VPN	Remote access VPN
Учасники	Два або більше мережеві шлюзи	Шлюз + клієнтський пристрій
Тип з'єднання	Постійне або за розкладом	Ініціюється користувачем на вимогу
Налаштування	На рівні мережевого обладнання	Клієнтське ПЗ + серверна частина
Прозорість для кінцевих вузлів	Повністю прозоре	Потребує клієнтського ПЗ
Типові сценарії	Об'єднання офісів, хмарні підключення	Віддалена робота, BYOD

Таблиця 11.1 узагальнює ключові відмінності між двома основними топологіями. Важливо розуміти, що ці топології не є взаємовиключними: організація може одночасно використовувати site-to-site VPN для об'єднання своїх офісів і remote access VPN для підключення мобільних співробітників. Разом зі зростанням популярності хмарних сервісів поширюється й змішана топологія hub-and-spoke, де центральним вузлом (hub) є хмарний VPN-концентратор, а до нього підключаються численні філіали та мобільні користувачі — така модель дозволяє суттєво спростити управління мережею та знизити витрати на обладнання.

11.2 Протокол IPsec: архітектура, компоненти та процес встановлення з'єднання

IPsec (Internet Protocol Security) є найбільш стандартизованою технологією для побудови VPN-тунелів. Це не окремий протокол, а ціла архітектура, описана в серії стандартів RFC і являє собою набір взаємопов'язаних протоколів та механізмів, які разом забезпечують захист на мережевому рівні моделі OSI. Принципова перевага такого підходу полягає в тому, що захист реалізується на рівні IP-пакетів, а не на рівні прикладних програм, що робить його прозорим для вищих рівнів стеку і дозволяє захищати трафік будь-яких додатків без їхньої модифікації. Саме тому IPsec став де-факто стандартом для побудови корпоративних VPN, особливо у сегменті site-to-site з'єднань між обладнанням різних виробників.

Архітектура IPsec включає кілька ключових компонентів. Перший — протокол AH (Authentication Header) — забезпечує цілісність та автентифікацію IP-пакета, але не забезпечує шифрування: на стороні відправника обчислюється криптографічна контрольна сума (HMAC), яка додається до заголовка пакета і перевіряється на стороні отримувача, а якщо пакет був змінений у дорозі, перевірка не пройде і пакет буде відкинуто. Другий компонент — протокол ESP (Encapsulating Security Payload) — є більш функціональним: він забезпечує шифрування корисного навантаження пакета, а також автентифікацію, тому на

практиці ESP використовується значно частіше, ніж AH. Третій компонент — протокол IKE (Internet Key Exchange) — вирішує автоматичне узгодження параметрів безпеки та обмін ключами між двома вузлами. Важливими базами даних архітектури є SAD (Security Association Database), де зберігаються всі активні параметри захисту, та SPD (Security Policy Database), де визначається, який трафік і як саме слід захищати. Схему взаємодії цих компонентів ілюструє рисунок 11.2.

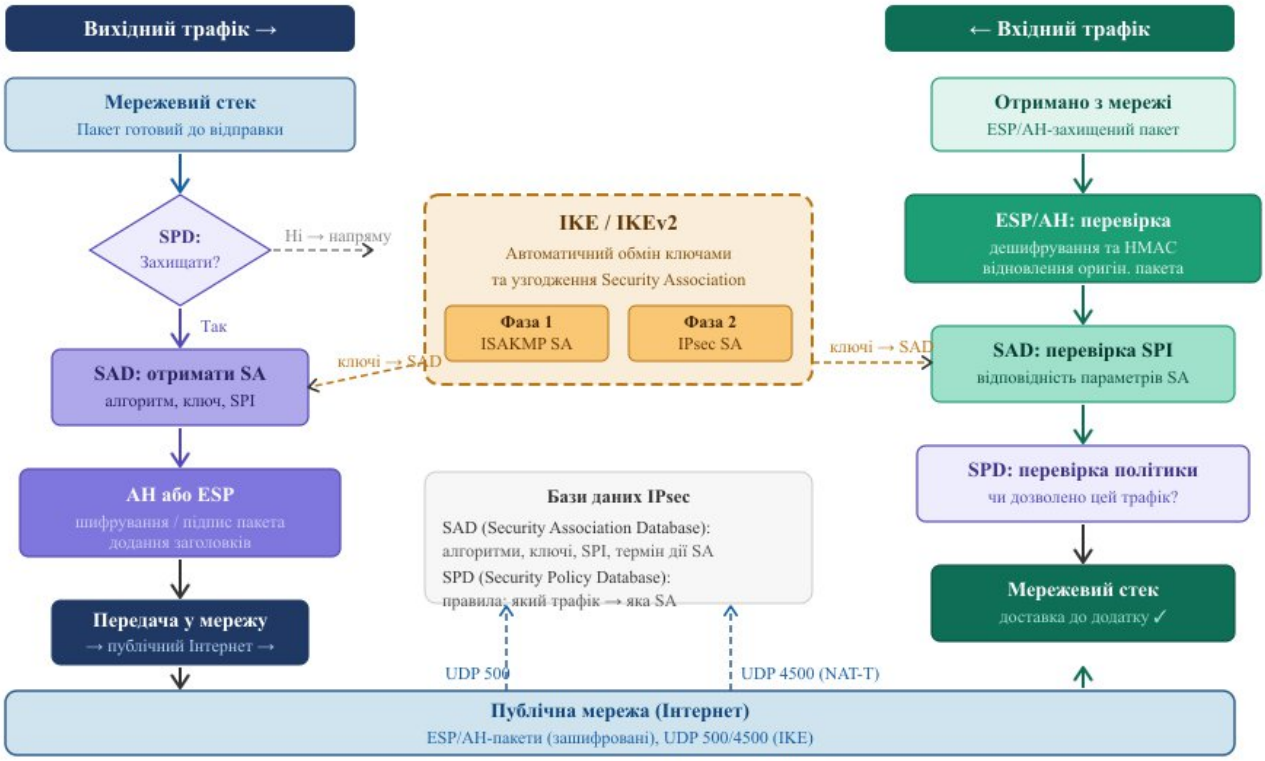

Рисунок 11.2 — Архітектура IPsec: взаємодія компонентів AH, ESP, IKE, SAD та SPD

Протокол IKE є серцем IPsec: саме він відповідає за автоматичне встановлення захищеного каналу перед тим, як почнеться передача корисних даних. Оригінальна версія IKEv1 працює у двох фазах. Фаза 1 (Main Mode або Aggressive Mode) встановлює захищений канал між самими вузлами IPsec — так зване ISAKMP SA: сторони обмінюються пропозиціями щодо алгоритмів, виконують обмін ключами Діффі-Хеллмана для вироблення спільного секрету, а потім аутентифікують одне одного за допомогою сертифікатів або попередньо розподілених ключів (PSK). Main Mode передбачає шість послідовних повідомлень і вважається більш безпечним, оскільки ідентифікатори вузлів передаються вже в зашифрованому вигляді. Aggressive Mode скорочує обмін до трьох повідомлень за рахунок того, що ідентифікатори передаються у відкритому вигляді ще до встановлення захищеного каналу, що робить його вразливим до ряду атак і не рекомендованим для нових розгортань. Фаза 2 (Quick Mode) встановлює вже безпосередні IPsec SA для захисту корисного трафіку, використовуючи захищений канал, утворений у фазі 1. Сучасна версія

IKEv2 суттєво спрощує і прискорює цей процес, скорочуючи кількість повідомлень і усуваючи багато недоліків IKEv1.

IPsec може працювати у двох режимах залежно від того, які частини IP-пакета захищаються. Транспортний режим (transport mode) захищає лише корисне навантаження оригінального пакета, залишаючи оригінальний IP-заголовок незміненим, і застосовується, коли захист потрібен безпосередньо між кінцевими вузлами комунікації. Тунельний режим (tunnel mode) є більш поширеним: він повністю загортає оригінальний IP-пакет, включаючи його заголовок, у новий пакет з новими зовнішніми заголовками, що повністю приховує топологію внутрішньої мережі від зовнішніх спостерігачів. Саме тунельний режим використовується у переважній більшості VPN-розгортань. Ключові відмінності між режимами наведені у таблиці 11.2.

Таблиця 11.2 — Порівняння режимів роботи IPsec

Параметр	Транспортний режим	Тунельний режим
Що захищається	Корисне навантаження пакета	Весь оригінальний IP-пакет
Оригінальний IP-заголовок	Залишається без змін	Зашифровується та прихована
Накладні витрати	Менші (без додаткового заголовка)	Більші (є зовнішній IP-заголовок)
Типове застосування	Захист між кінцевими вузлами (host-to-host)	VPN-тунелі між шлюзами (gw-to-gw)
Приховування топології	Ні — адреси вузлів видимі	Так — внутрішні адреси приховані

Ще одним важливим практичним аспектом IPsec є підтримка NAT Traversal (NAT-T). Класичний IPsec погано поєднується з мережевою трансляцією адрес (NAT), оскільки NAT змінює IP-заголовки пакетів, що руйнує перевірку цілісності AH, а ESP-пакети не мають портів, за якими NAT-пристрій міг би відстежувати з'єднання. NAT-T вирішує цю проблему, загортаючи ESP-пакети в UDP з портом 4500, що дозволяє їм проходити через NAT-пристрої так само, як звичайний UDP-трафік. В сучасних реалізаціях strongSwan та Libreswan NAT-T вмикається автоматично при виявленні NAT на шляху між вузлами, що суттєво спрощує розгортання у типових мережевих середовищах.

Концепція Perfect Forward Secrecy (PFS) є важливою властивістю безпечних VPN-з'єднань, яку варто розглянути окремо. Без PFS компрометація довгострокового ключа потенційно дозволяє розшифрувати весь записаний раніше трафік. З увімкненим PFS для кожної нової IPsec SA виконується окремий обмін Діффі-Хеллмана, внаслідок чого сесійні ключі не пов'язані між собою і з довгостроковими ключами автентифікації. Навіть якщо зловмисник якось отримає один сесійний ключ, він зможе розшифрувати лише ту конкретну сесію, але не попередні та наступні. У конфігурації strongSwan PFS

вмикається параметром pfs=yes у секції [ESP], а рекомендована група Діффі-Хеллмана — не нижче 14. Подібна функціональність доступна в OpenVPN через директиву tls-crypt та поновлення сесійних ключів за розкладом.

Таблиця 11.3 — Мережеві порти та протоколи VPN-технологій

Технологія / протокол	Порт	Транспорт	Призначення
IPsec ESP	50 (протокол IP)	IP	Зашифроване корисне навантаження
IPsec AH	51 (протокол IP)	IP	Автентифікація заголовка
IKE / IKEv2	500	UDP	Узгодження SA та обмін ключами
IPsec NAT-T	4500	UDP	ESP у режимі проходження NAT
OpenVPN (типово)	1194	UDP / TCP	VPN-тунель OpenVPN
WireGuard (типово)	51820	UDP	VPN-тунель WireGuard
SSL/TLS VPN	443	TCP	VPN через HTTPS (Cisco AnyConnect тощо)

Таблиця 11.3 містить порти та транспортні протоколи, знання яких необхідне як для налаштування правил міжмережевого екрана, так і для діагностики проблем з'єднання. Зверніть увагу, що ESP та AH є самостійними протоколами IP з номерами 50 та 51 відповідно — вони не є протоколами транспортного рівня і не мають «портів» у класичному розумінні. Саме через це класичний IPsec погано проходить через PAT/NAT і потребує NAT-T. OpenVPN зазвичай використовує UDP для кращої продуктивності, проте може бути перемкнений на TCP 443 для обходу рестриктивних брандмауерів. WireGuard працює виключно через UDP і не підтримує TCP, що є одним із його архітектурних обмежень.

11.3 Сучасні VPN-рішення: OpenVPN та WireGuard

Хоча IPsec є стандартним і функціонально потужним рішенням, його складність у налаштуванні та адмініструванні стала причиною появи альтернативних VPN-технологій. Серед них особливе місце займають OpenVPN та WireGuard — технології з принципово різними філософіями проектування, але однаковою метою. OpenVPN з'явився у 2001 році і швидко став популярним завдяки своїй гнучкості та крос-платформеності. Принципова відмінність OpenVPN від IPsec полягає в тому, що воно працює на транспортному рівні, використовуючи протоколи TCP або UDP, а не на мережевому рівні, і для шифрування застосовує бібліотеку OpenSSL та протокол TLS. OpenVPN здатен легко проходити крізь NAT і брандмауери, а при використанні TCP-порту 443 його трафік взагалі виглядає як звичайний HTTPS і може бути непомітний для

систем фільтрації. Для аутентифікації OpenVPN підтримує сертифікати X.509 на базі PKI (найбільш безпечний варіант), попередньо розподілені ключі та логін/пароль.

WireGuard — значно молодше рішення, що з'явилося у 2015 році і було включено до ядра Linux починаючи з версії 5.6 у 2020 році. Архітектурна філософія WireGuard кардинально відрізняється від попередників: якщо кодова база OpenVPN нараховує сотні тисяч рядків коду, то WireGuard реалізований приблизно у чотирьох тисячах рядків. Така лаконічність є принциповою перевагою з точки зору безпеки — менше коду означає меншу поверхню для потенційних вразливостей і спрощений аудит. WireGuard використовує лише сучасний набір криптографічних алгоритмів, відмовившись від підтримки застарілих протоколів: для шифрування застосовується ChaCha20 зі стрімером Poly1305 для автентифікації, для узгодження ключів — алгоритм на основі кривої Curve25519. На відміну від IPsec та OpenVPN, де є концепція «з'єднання» зі станом, WireGuard оперує поняттям «peer» — кожному вузлу відомі відкриті ключі інших вузлів, і будь-який пакет, підписаний відомим ключем, приймається без додаткових переговорів. Такий підхід значно спрощує налаштування і прискорює встановлення тунелю.

Важливо розуміти різницю між WireGuard, реалізованим у ядрі операційної системи, та WireGuard-go — реалізацією у просторі користувача (userspace) на мові Go. Ядерна реалізація забезпечує значно вищу продуктивність, оскільки обробка пакетів відбувається без переключення контексту між простором ядра та простором користувача, і саме вона доступна у Linux починаючи з версії 5.6. WireGuard-go, натомість, є портативним варіантом для систем, де ядерна реалізація недоступна (наприклад, старі версії Linux, macOS, FreeBSD або контейнерні середовища з обмеженнями), проте за продуктивністю він поступається ядерній реалізації. У практичному навчальному середовищі на Linux ми будемо використовувати саме ядерну реалізацію WireGuard, яка доступна «з коробки» і не потребує встановлення додаткових компонентів.

Окремої уваги заслуговує питання аутентифікації у різних VPN-рішеннях. OpenVPN і IPsec підтримують широкий спектр механізмів аутентифікації, які суттєво різняться за рівнем безпеки та зручністю використання. Найпростіший механізм — PSK (Pre-Shared Key, попередньо розподілений ключ) — зручний для тестових середовищ, але вимагає безпечного способу передачі ключа між сторонами та ускладнює масштабування. Аутентифікація на базі сертифікатів X.509 є більш надійною і добре масштабується: кожен клієнт отримує унікальний сертифікат, а відкликання доступу здійснюється через CRL або OCSP без необхідності змінювати ключі на всіх вузлах. EAP (Extensible Authentication Protocol) у поєднанні з IKEv2 дозволяє використовувати зовнішні системи аутентифікації (наприклад, RADIUS або Active Directory), що важливо для корпоративних середовищ із централізованим управлінням обліковими записами. WireGuard використовує виключно криптографію відкритих ключів (Curve25519) і не підтримує паролі або сертифікати X.509 — це одночасно є і перевагою

(простота, немає слабких паролів), і обмеженням (потребує власної системи управління ключами).

Таблиця 11.4 — Порівняльна характеристика VPN-технологій

Характеристика	IPsec/IKEv2	OpenVPN	WireGuard
Рівень OSI	Мережевий (L3)	Транспортний / L3	Мережевий (L3), ядро
Криптографія	AES, SHA-2, ECDH	OpenSSL / TLS 1.3	ChaCha20, Poly1305, Curve25519
Гнучкість алгоритмів	Висока	Висока	Фіксований набір
Кодова база	Велика, складна	~120k рядків	~4k рядків
Складність налаш.	Висока	Середня	Низька
Аутентифікація	PSK, сертифікати, EAP	PSK, сертифікати, пароль	Тільки відкриті ключі
Проходження NAT	NAT-T (UDP 4500)	Добре (UDP/TCP)	Добре (тільки UDP)
Продуктивність	Висока (у ядрі)	Середня	Дуже висока (у ядрі)
PFS підтримка	Так (параметр pfs=yes)	Так (tls-crypt + rekey)	Вбудовано за замовч.

Таблиця 11.4 наочно демонструє, що кожне з трьох рішень має власні переваги та недоліки, і вибір конкретної технології повинен ґрунтуватися на аналізі вимог до безпеки, функціональних потреб та наявної інфраструктури. IPsec залишається галузевим стандартом для корпоративних мережевих з'єднань і добре підходить для інтеграції з обладнанням різних виробників. OpenVPN зарекомендував себе як гнучке рішення для remote access VPN з підтримкою складних сценаріїв аутентифікації. WireGuard стає дедалі привабливішим там, де важливі простота, продуктивність і мінімальна поверхня атаки. Зверніть увагу, що WireGuard забезпечує Perfect Forward Secrecy вбудовано — кожні кілька хвилин ключі автоматично оновлюються через механізм HKDF без будь-яких додаткових налаштувань.

11.4 Практичні аспекти побудови та адміністрування захищених тунелів

Теоретичне розуміння VPN-технологій є необхідною, але недостатньою умовою для їхнього успішного практичного застосування. Побудова реально захищеного тунелю вимагає врахування конкретних деталей конфігурації: правильного вибору криптографічних алгоритмів, налаштування аутентифікації, управління ключами, вибору режимів маршрутизації трафіку, розв'язання проблем MTU та моніторингу стану з'єднань. Навіть добре спроектована VPN-технологія може бути скомпрометована через неправильне

розгортання — практичні навички налаштування та адміністрування не менш важливі, ніж розуміння теоретичних основ.

Одним із найважливіших практичних рішень є вибір криптографічних параметрів. Для IPsec у сучасних конфігураціях рекомендується AES-256 для шифрування, SHA-256 або SHA-384 для цілісності та групи Діффі-Хеллмана від 14 і вище (бажано групи 20 або 21 на основі еліптичних кривих). Застарілі алгоритми DES, 3DES, MD5 та групи Діффі-Хеллмана 1 і 2 слід вважати небезпечними і не використовувати в нових конфігураціях. Розглянемо мінімальний приклад конфігурації WireGuard у Linux, який демонструє типову структуру та ключові параметри. Файл конфігурації /etc/wireguard/wg0.conf для серверного вузла матиме такий вигляд:

```
[Interface]
Address = 10.10.0.1/24
ListenPort = 51820
PrivateKey = <приватний_ключ_сервера>

[Peer]
PublicKey = <публічний_ключ_клієнта>
AllowedIPs = 10.10.0.2/32
PersistentKeepalive = 25
```

У наведеному прикладі секція [Interface] описує параметри локального вузла: адресу інтерфейсу у VPN-мережі, порт прослуховування та приватний ключ. Секція [Peer] визначає параметри вузла-партнера: його публічний ключ і дозволені IP-адреси, тобто які адреси можуть бути джерелом або призначенням трафіку через цей тунель. Параметр PersistentKeepalive зі значенням 25 секунд змушує вузол регулярно надсилати keepalive-пакети, що важливо для підтримки активності з'єднання через NAT-пристрої. Пара ключів генерується простими командами: wg genkey видає приватний ключ у стандартний вивід, wg pubkey обчислює відповідний публічний ключ. Після створення файлу конфігурації інтерфейс запускається командою wg-quick up wg0, а для автоматичного запуску при старті системи — systemctl enable wg-quick@wg0.

Важливим практичним аспектом є розщеплення тунелю, або split tunneling — можливість направляти через VPN лише певний трафік, а не весь. У WireGuard для повного тунелю (весь трафік через VPN) у секції [Peer] вказується AllowedIPs = 0.0.0.0/0, 0::0/0, тоді як для split tunneling — лише конкретні підмережі, наприклад AllowedIPs = 10.0.0.0/8, 192.168.1.0/24. Рисунок 11.3 ілюструє різницю між цими підходами з точки зору потоків трафіку.

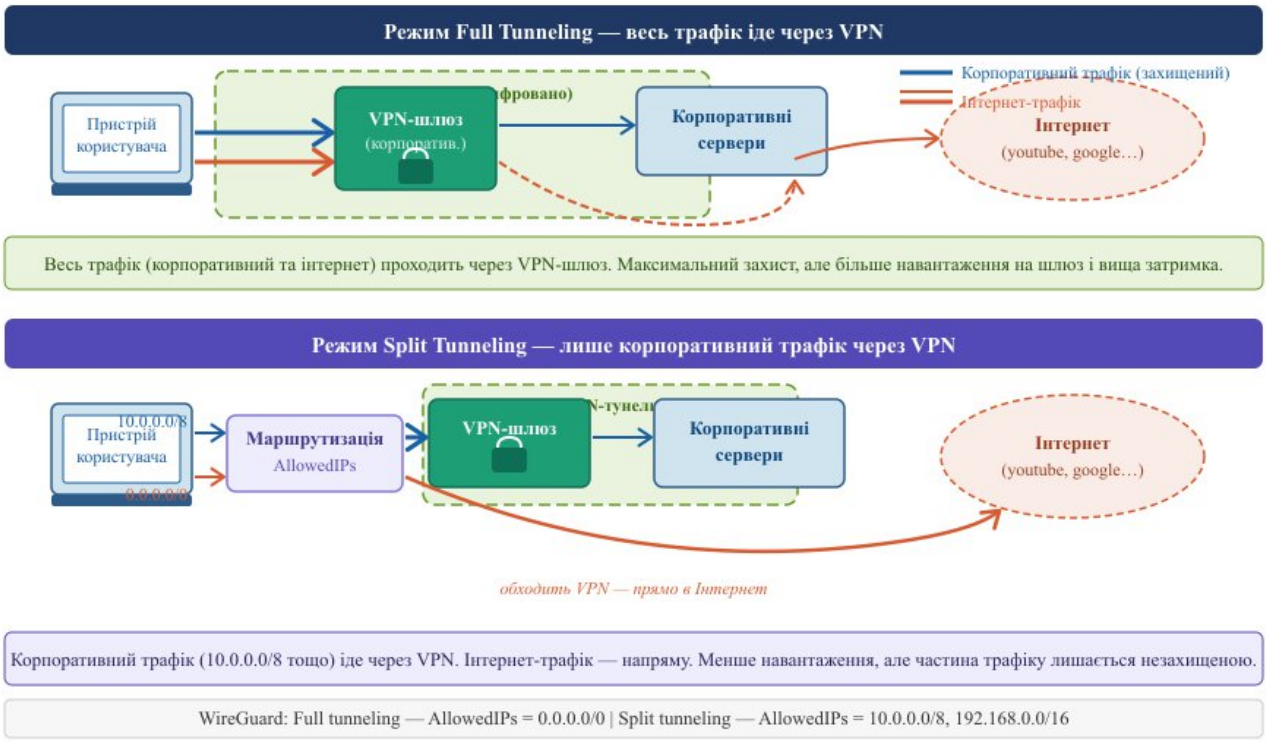

Рисунок 11.3 — Порівняння режимів Full Tunneling та Split Tunneling

Проблема MTU (Maximum Transmission Unit) є одним із найпоширеніших практичних ускладнень при розгортанні VPN. Оскільки VPN додає власні заголовки до пакетів, ефективний розмір корисних даних у пакеті зменшується. Для стандартного Ethernet MTU = 1500 байт, а WireGuard з його накладними витратами потребує приблизно 60 байт на заголовки, тому реальний MTU всередині тунелю слід встановлювати близько 1420 байт. Якщо MTU налаштований неправильно, великі пакети фрагментуються або відкидаються, що проявляється у вигляді дивної поведінки: невеликі запити проходять нормально (ping, DNS), а завантаження великих файлів або відкриття HTTPS-сторінок «зависає». Для діагностики такої проблеми у Linux корисна команда ping із встановленим прапором заборони фрагментації та зазначеним розміром пакета: ping -M do -s 1400 10.10.0.1. У файлі конфігурації WireGuard параметр MTU у секції [Interface] дозволяє задати правильне значення явно.

Управління ключами є критично важливим аспектом безпеки VPN. Для OpenVPN, що використовує PKI-інфраструктуру, необхідно правильно організувати Центр сертифікації: регулярно відкликати компрометовані сертифікати та підтримувати актуальний CRL або OCSP-сервер, встановлювати розумні терміни дії сертифікатів (наприклад, один рік для клієнтських і три-п'ять років для серверних) та зберігати приватний ключ CA в офлайн-режимі на захищеному носієві. Для WireGuard управління ключами простіше за структурою, але не менш важливе: кожен вузол має пару ключів, які можуть бути замінені в будь-який момент без складних процедур. Однак важливо зберігати приватні ключі в захищеному місці — файл конфігурації WireGuard, що містить приватний ключ, повинен мати права доступу 600 (тільки для власника) і зберігатися в директорії з правами 700. Також доцільно запровадити

регулярну ротацію ключів: компрометований ключ WireGuard замінюється генерацією нової пари та оновленням відповідних записів на всіх пірах.

Моніторинг та аудит VPN-з'єднань є невід'ємною частиною підтримки безпеки. Для WireGuard поточний стан усіх інтерфейсів та пірів відображається командою wg show: вона виводить публічний ключ кожного піра, його кінцеву точку, дозволені IP-адреси, час отримання останнього пакета і обсяг переданих даних. Ці дані корисні при виявленні аномалій: наприклад, якщо піер, що має підключатися зі статичної IP-адреси, раптово з'являється з іншої адреси, це привід для розслідування. Для IPsec strongSwan надає утиліту swanctl --list-sas, що відображає активні Security Association з використовуваними алгоритмами та лічильниками пакетів. Системні журнали (journalctl або /var/log/syslog) фіксують події встановлення та розриву VPN-з'єднань, помилки аутентифікації та інші важливі події безпеки. Регулярний перегляд таких журналів або налаштування автоматичних сповіщень про аномалії є обов'язковою практикою в продуктивних середовищах.

Діагностика VPN-з'єднань потребує систематичного підходу і знання відповідних інструментів. Найбільш цінним інструментом є tcpdump, що дозволяє перехоплювати та аналізувати трафік на різних мережевих інтерфейсах: виконавши tcpdump -i eth0 udp port 51820, можна переконатися, що WireGuard-пакети дійсно надходять на зовнішній інтерфейс, а tcpdump -i wg0 дозволить побачити вже розшифрований трафік всередині тунелю. Для перевірки маршрутизації корисна команда ip route get 10.0.0.1, яка показує, через який інтерфейс і шлюз буде відправлений пакет на вказану адресу. Команда ping через тунель дозволяє перевірити базову зв'язність, а traceroute допомагає з'ясувати, чи трафік дійсно іде через тунель, а не в обхід нього. Аналіз виводу wg show і логів системи у поєднанні з цими інструментами дозволяє діагностувати більшість типових проблем з VPN-з'єднаннями.

Таблиця 11.5 — Типові помилки конфігурації VPN та методи їх усунення

Помилка конфігурації	Можливі наслідки	Рекомендований захід
Використання DES, 3DES, MD5	Можливість розшифрування трафіку	Перейти на AES-256, SHA-256+
Приватний ключ з правами 644 або 666	Компрометація ключа та тунелю	chmod 600; директорія chmod 700
Відсутність перевірки сертифіката (OpenVPN)	Атаки «людина посередині»	Вмикати verify-x509-name або tls-verify
Надто широкі AllowedIPs у WireGuard	Надмірний доступ до мережі	Обмежити лише потрібними підмережами
Відсутність PFS (IPsec pfs=no)	Ретроспективне розшифрування при компрометації ключа	Увімкнути PFS (pfs=yes, DH group 14+)
Неправильне значення MTU	Зависання великих TCP-сесій	Встановити MTU 1420 або використати PMTUD

Відсутність моніторингу з'єднань	Непомічені несанкціоновані підключення	Налаштувати syslog, wg show, регулярний аудит
Незмінні ключі протягом тривалого часу	Зростання ризику компрометації	Запровадити регулярну ротацію ключів

Таблиця 11.5 узагальнює найбільш поширені помилки при налаштуванні VPN. Окремо слід зупинитися на темі атак, специфічних для VPN. Downgrade attack полягає у змушуванні обох сторін погодитися на більш слабкий криптографічний алгоритм — захист полягає у явному вказанні переліку допустимих алгоритмів у конфігурації та забороні будь-яких інших. IKE scanning дозволяє виявити наявність IPsec-шлюзу та отримати інформацію про підтримувані алгоритми, надсилаючи зонди на UDP-порт 500 — зменшення поверхні атаки досягається фільтрацією IKE-трафіку на брандмауері так, щоб він приймався лише з відомих IP-адрес партнерів. VPN fingerprinting — ідентифікація типу VPN-програмного забезпечення за характерними ознаками його трафіку — може бути ускладнена використанням нестандартних портів. Квантово-стійка криптографія є перспективним напрямком: сучасні алгоритми на основі еліптичних кривих (Curve25519, ECDH) теоретично можуть бути зламані квантовим комп'ютером за допомогою алгоритму Шора, тому вже зараз ведуться роботи зі стандартизації та впровадження постквантових алгоритмів (CRYSTALS-Kyber, ML-KEM) у VPN-стеки.

Підсумовуючи матеріал теми, варто підкреслити, що VPN є важливим, але не єдиним компонентом комплексної системи захисту. Правильно побудований і налаштований VPN-тунель забезпечує конфіденційність і цілісність переданих даних, аутентифікацію учасників з'єднання та захист від пасивного прослуховування і активних атак «людина посередині» на мережевому рівні. Разом із тим VPN не замінює засоби захисту на рівні додатків, систему виявлення вторгнень, правильне налаштування міжмережевих екранів та організаційні заходи безпеки. Комплексний підхід, що поєднує VPN з принципами defense in depth та Zero Trust, є єдиним способом забезпечити надійний захист у сучасних розподілених мережевих середовищах — і саме цим темам присвячені наступні розділи курсу.

Тема 12. Захист даних у розподілених, хмарних та контейнерних середовищах

12.1. Особливості зберігання та обробки даних у розподілених системах

Сучасні інформаційні системи дедалі рідше являють собою єдиний сервер із централізованою базою даних. Переважна більшість корпоративних і промислових рішень будується як розподілені системи, в яких обчислювальні ресурси, сховища даних і прикладні сервіси фізично або логічно розосереджені по різних вузлах, датацентрах або навіть географічних регіонах. Така архітектура дає суттєві переваги у масштабованості, відмовостійкості та продуктивності, проте водночас значно ускладнює забезпечення інформаційної безпеки, оскільки поверхня атаки зростає пропорційно кількості компонентів системи. Якщо в монолітній системі дані зберігаються в одному місці й доступ до них можна контролювати через єдину точку, то в розподіленій системі дані постійно переміщуються між вузлами, реплікуються, кешуються і агрегуються, що суттєво ускладнює їх відстеження та захист.

Ключовою особливістю безпеки розподілених систем є неможливість покладатися на «периметровий» підхід до захисту, при якому всі ресурси, що знаходяться всередині захищеного периметра мережі, вважаються довіреними за замовчуванням. У розподіленій системі вузли можуть знаходитися в різних мережах, деякі з них можуть бути хмарними сервісами поза корпоративним периметром, а з'єднання між ними пролягають через публічні мережі. Більше того, компрометація навіть одного вузла може надати зловмиснику плацдарм для горизонтального переміщення (lateral movement) по мережі та поступового отримання доступу до критичних ресурсів. Цей вид атаки полягає в тому, що, потрапивши до системи через один компонент, зловмисник поступово «переміщується» між вузлами, використовуючи облікові дані, вразливості або надлишкові привілеї, здобуваючи доступ до чим раз важливіших ресурсів.

Іншою принциповою відмінністю є складність управління ідентичностями в розподіленому середовищі. Якщо в традиційних системах перевіряється особа людини-користувача, то в розподілених системах більша частина взаємодій відбувається між сервісами — мікросервісами, фоновими процесами, демонами, агентами. Кожен такий сервіс має власну ідентичність і потребує механізмів автентифікації при зверненні до інших сервісів або ресурсів. Управління такими «машинними» ідентичностями є нетривіальним завданням: їх може бути тисячі, вони можуть створюватися і знищуватися динамічно, і якщо облікові дані одного сервісу буде скомпрометовано, зловмисник може отримати доступ до всіх ресурсів, до яких цей сервіс мав право звертатися.

Розподілені системи також породжують специфічні загрози, пов'язані з реплікацією та синхронізацією даних. Коли одні й ті самі дані зберігаються одночасно на кількох вузлах для підвищення доступності та продуктивності, виникає ризик порушення узгодженості: якщо зловмисник отримає доступ до

одного вузла та модифікує дані, то без належних механізмів перевірки цілісності ця підроблена версія може поширитися по всій системі шляхом реплікації. Крім того, логи, метадані та службова інформація, що генеруються розподіленою системою, можуть містити конфіденційні дані й потребують окремого захисту. Нарешті, необхідно враховувати вимоги законодавства щодо розташування даних: деякі категорії персональних або державних даних не можуть зберігатися за межами певної юрисдикції, що накладає додаткові архітектурні обмеження на систему.

12.2. Захист даних «at rest» та «in transit»

Будь-які дані в інформаційній системі в кожний момент часу перебувають в одному з двох основних станів: або вони зберігаються на носії (диску, стрічці, флеш-накопичувачі) — такий стан називають «at rest» (у спокої), або вони передаються каналом зв'язку між компонентами системи — такий стан називають «in transit» (у русі). Деякі фахівці також виділяють третій стан — «in use» (у використанні), коли дані завантажені у оперативну пам'ять і активно обробляються процесором, — однак захист даних у цьому стані є значно складнішим і, як правило, реалізується апаратними методами (наприклад, Intel SGX або AMD SEV). Рисунок 12.1 ілюструє ці три стани даних та відповідні механізми захисту. Захист в обох основних станах є обов'язковою умовою комплексної безпеки системи: недостатньо захистити лише передачу даних, якщо вони зберігаються у відкритому вигляді, і навпаки.

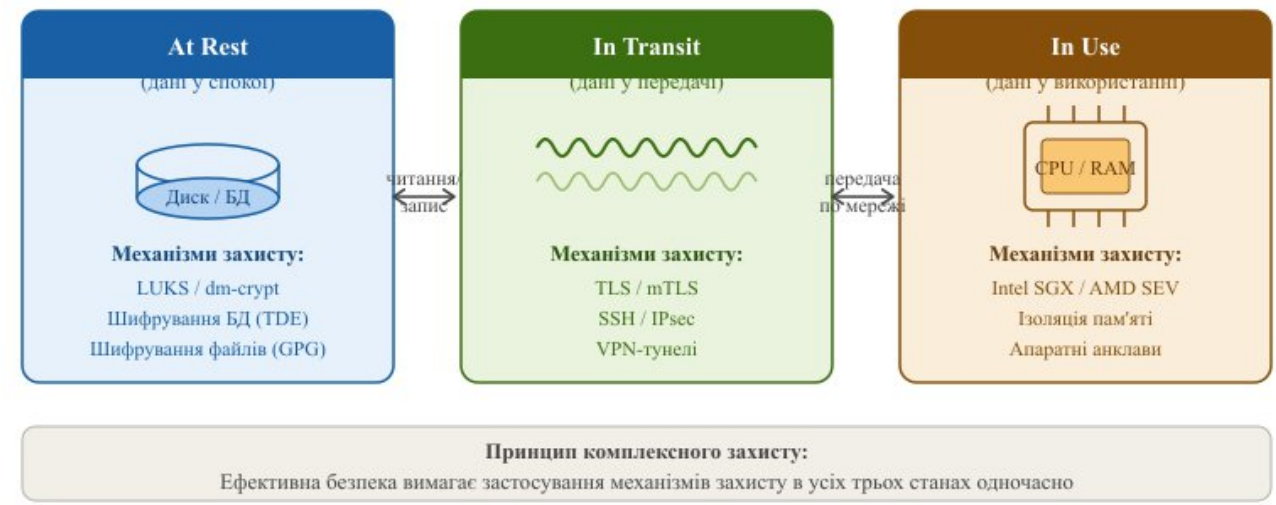

Рисунок 12.1 – Стани даних та відповідні механізми захисту

Захист даних «at rest» реалізується насамперед через шифрування сховищ і файлових систем. Існує кілька рівнів, на яких можна застосовувати таке шифрування: рівень всього диска (Full Disk Encryption, FDE), рівень розділу або логічного тому, рівень файлової системи та рівень окремих файлів або полів у базі даних. Шифрування на рівні диска є найбільш прозорим для додатків — воно виконується апаратним або програмним забезпеченням автоматично і не

потребує змін у прикладному ПЗ. У Linux для шифрування блокових пристроїв широко застосовується підсистема LUKS (Linux Unified Key Setup) разом із dm-crypt. Наприклад, за допомогою команди cryptsetup luksFormat /dev/sdb1 можна ініціалізувати зашифрований розділ, а cryptsetup luksOpen /dev/sdb1 secure_volume — відкрити його для використання. Після введення пароля або надання ключового файлу том підключається як звичайний блоковий пристрій; у разі фізичної крадіжки носія дані залишаються нечитабельними.

Важливою складовою захисту даних «at rest» є також шифрування на рівні бази даних або окремих полів. Якщо додаток зберігає особисті дані, паролі, платіжну інформацію або інші конфіденційні відомості у реляційній або NoSQL базі, їх доцільно шифрувати ще до запису — таким чином навіть компрометація облікових даних для підключення до бази не надасть зловмиснику доступу до фактичного вмісту. При цьому слід пам'ятати про розмежування між хешуванням і шифруванням: паролі зазвичай хешуються за допомогою спеціалізованих алгоритмів (bcrypt, scrypt, Argon2), тоді як дані, які необхідно відновити у вихідному вигляді, шифруються симетричним або асиметричним алгоритмом. Неправильний вибір між цими підходами є однією з найпоширеніших помилок при розробці систем зберігання даних.

Таблиця 12.1 – Порівняння підходів до захисту даних «at rest»

Рівень шифрування	Інструмент (Linux)	Переваги	Недоліки
Повне шифрування диска (FDE)	LUKS / dm-crypt	Захист при фізичному доступі; прозоро для ОС та додатків	Не захищає від зловмисника із доступом до запущеної системи
Шифрування файлової системи	eCryptfs, fscrypt	Шифрування окремих каталогів; різні ключі для різних користувачів	Метадані файлів (імена, розміри) можуть залишатися відкритими
Шифрування БД	PostgreSQL TDE, MySQL Keyring	Захист при доступі до файлів БД; шифрування окремих полів	Підвищене навантаження на CPU; ускладнює індексування
Шифрування в додатку	OpenSSL, libsodium, GPG	Максимальний контроль; незалежність від ОС та СУБД	Відповідальність за реалізацію лежить на розробнику

Як видно з таблиці 12.1, не існує «найкращого» рівня шифрування для всіх ситуацій. Оптимальним є поєднання кількох рівнів: типова продуктивна система може одночасно використовувати LUKS для шифрування всього тому зберігання даних, TLS для з'єднань із базою і шифрування на рівні додатку для окремих особливо чутливих полів.

Захист даних «in transit» спирається на використання захищених протоколів передачі, які забезпечують конфіденційність, цілісність та автентичність у каналі зв'язку. Детальному розгляду цих протоколів присвячені

теми 10 і 11 курсу, тому тут зупинимось лише на аспектах, специфічних для розподілених і хмарних середовищ. Основним протоколом захисту прикладного трафіку залишається TLS, і в розподілених системах важливо забезпечити його використання не лише на зовнішніх інтерфейсах, доступних клієнтам, але й у внутрішній комунікації між сервісами. Практика «TLS скрізь» (TLS everywhere або mTLS — mutual TLS) означає, що навіть з'єднання між мікросервісами всередині захищеної мережі здійснюються поверх TLS, що захищає від атак типу «людина посередині» на внутрішньому трафіку. При взаємній автентифікації (mTLS) обидві сторони — і клієнт, і сервер — пред'являють сертифікати, підтверджуючи свою ідентичність один одному.

12.3. Управління секретами, обліковими даними сервісів та модель Zero Trust

У будь-якій нетривіальній інформаційній системі є секрети — паролі до баз даних, API-ключі зовнішніх сервісів, токени автентифікації, сертифікати та приватні криптографічні ключі. Управління цими секретами є одним із найбільш недооцінених аспектів безпеки: за статистикою, значна частина реальних витоків даних стала можливою через те, що секрети зберігалися у небезпечних місцях — у коді у відкритому вигляді, у файлах конфігурації без шифрування або у системах контролю версій. Навіть якщо розробник пам'ятає видалити секрет із коду перед публікацією у репозиторій, він може залишитися в історії комітів і бути легко виявлений зловмисником за допомогою автоматизованих інструментів сканування. Управління секретами як окрема дисципліна покликана вирішити цю проблему системно, забезпечивши безпечне зберігання, ротацію та аудит доступу до всіх чутливих облікових даних.

Сучасний підхід до управління секретами передбачає використання спеціалізованих сховищ (secrets management systems). Найбільш відомим відкритим рішенням є HashiCorp Vault, яке надає API для безпечного зберігання секретів, їх динамічної генерації та автоматичної ротації. Принципова перевага Vault полягає в концепції динамічних облікових даних: замість того, щоб один і той самий пароль до бази даних зберігався роками і копіювався між сервісами, Vault може динамічно генерувати тимчасові облікові дані з обмеженим терміном дії для кожного конкретного запиту. Якщо такі облікові дані потрапляють до зловмисника, їхній термін дії вже закінчився або буде відкликано після виявлення інциденту. Хмарні провайдери пропонують власні сервіси: AWS Secrets Manager, Azure Key Vault, Google Cloud Secret Manager — усі вони інтегровані з IAM-системами і забезпечують детальне логування кожного звернення до секрету.

Не менш важливим є питання про те, як сервіс отримує початковий доступ до сховища секретів. Це відоме як «проблема нульового секрету» (bootstrapping problem): щоб отримати секрет, сервіс сам повинен автентифікуватися, але для автентифікації йому потрібен секрет. Рішення цієї проблеми в сучасних системах ґрунтується на апаратних або платформних

механізмах ідентифікації: в хмарних середовищах кожен обчислювальний ресурс (віртуальна машина, контейнер, serverless-функція) отримує криптографічно підтверджену ідентичність від платформи — Instance Metadata Service в AWS, Workload Identity у Google Kubernetes Engine — яку він пред'являє сховищу секретів без необхідності зберігати статичний пароль. Наступний рисунок 12.2 ілюструє цей процес.

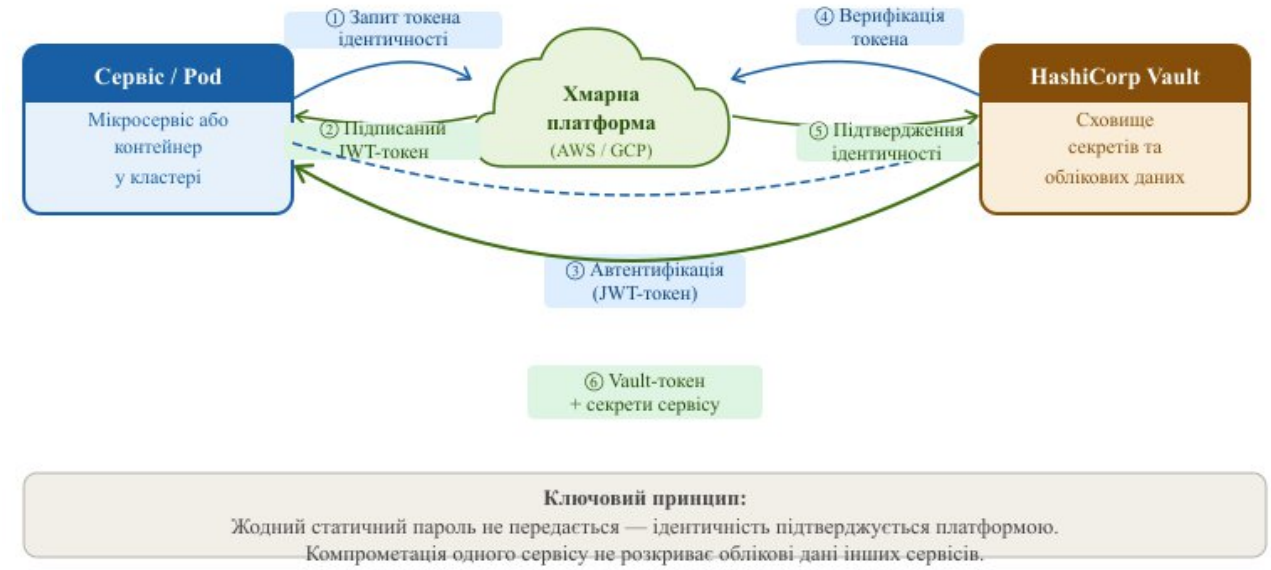

Рисунок 12.2 – Процес автентифікації сервісу у HashiCorp Vault з використанням платформної ідентичності

На тлі зростання складності розподілених систем і недостатності периметрового підходу до безпеки у 2010 році аналітик Forrester Research Джон Кіндерваг запропонував концепцію Zero Trust (нульова довіра). Основний принцип Zero Trust формулюється як «ніколи не довіряй, завжди перевіряй» (never trust, always verify): жоден запит — ні від зовнішнього користувача, ні від внутрішнього сервісу, ні від адміністратора — не отримує автоматичної довіри лише на підставі свого походження чи мережевого розташування. Кожне звернення до ресурсу має супроводжуватися явною автентифікацією та авторизацією, а надані права мають бути мінімально необхідними для виконання конкретного завдання.

Щоб краще зрозуміти Zero Trust, корисно порівняти її з традиційною моделлю безпеки. Рисунок 12.3 демонструє принципову різницю між цими двома підходами. У традиційній моделі існує «довірений» внутрішній периметр: будь-який ресурс у внутрішній мережі вважається безпечним, і основна увага приділяється захисту зовнішньої межі (фаєрвол, DMZ). У моделі Zero Trust поняття внутрішнього периметра зникає — кожен запит перевіряється незалежно від його джерела, мережа розбивається на мікросегменти, а комунікація між сервісами захищається шифруванням навіть всередині корпоративної мережі.

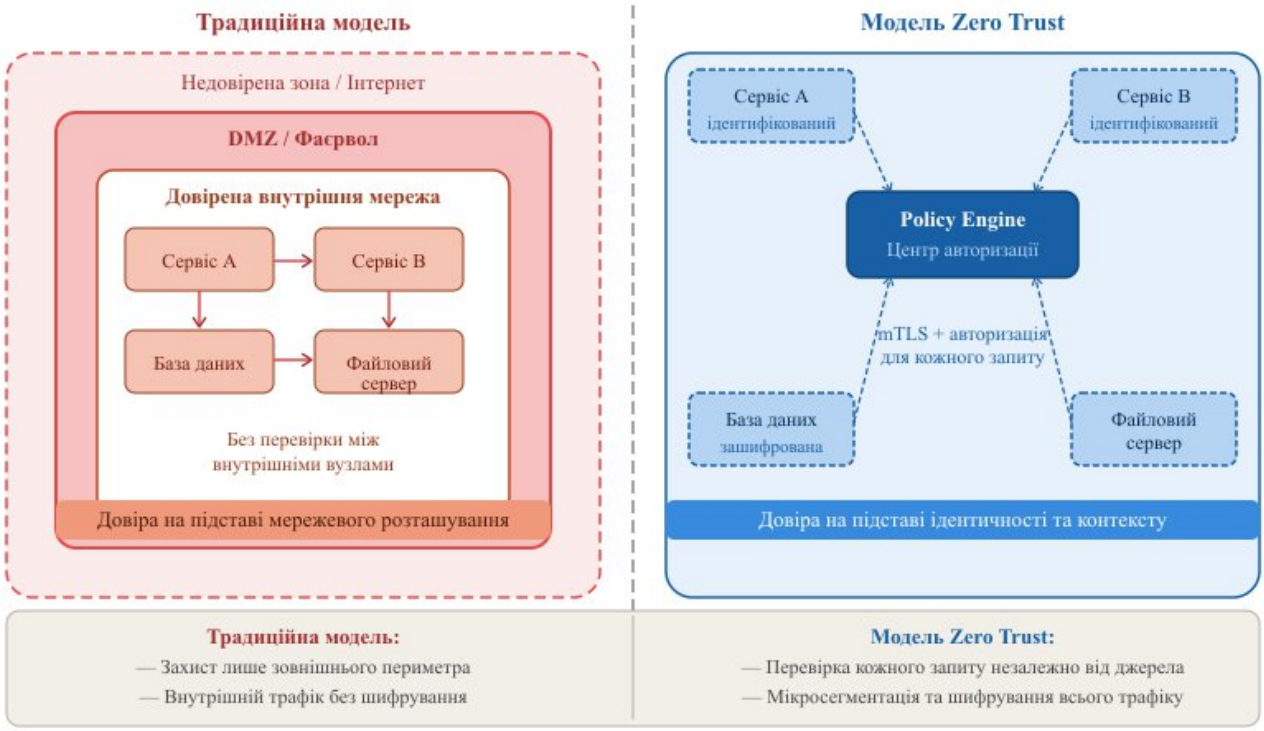

Рисунок 12.3 – Порівняння традиційної моделі безпеки та моделі Zero Trust

Таблиця 12.2 – Основні принципи моделі Zero Trust та їх практична реалізація

Принцип Zero Trust	Зміст	Приклад реалізації
Явна перевірка	Кожен запит автентифікується та авторизується незалежно від джерела	mTLS між мікросервісами; JWT-токени з коротким терміном дії
Найменші привілеї	Доступ надається лише до мінімально необхідних ресурсів	RBAC/ABAC; тимчасові облікові дані з обмеженим набором прав
Передбачення порушення	Система проектується з урахуванням неминучості компрометації будь-якого компонента	Мікросегментація; шифрування внутрішнього трафіку; детальне логування
Безперервний моніторинг	Постійний аналіз активності для виявлення аномалій	SIEM; поведінковий аналіз; автоматичне блокування підозрілих дій

Практична реалізація Zero Trust у розподіленій системі охоплює кілька ключових компонентів. По-перше, це мікросегментація мережі: замість єдиної «довіреної» внутрішньої мережі трафік між сервісами суворо контролюється мережевими політиками, і кожен сервіс може спілкуватися лише з тими, яким це явно дозволено. По-друге, це використання service mesh — інфраструктурного шару, який автоматично забезпечує взаємну автентифікацію (mTLS) між усіма сервісами, управляє авторизацією запитів, збирає метрики та логи. Відомими реалізаціями service mesh є Istio, Linkerd, Consul Connect. По-

третє, це політика найменших привілеїв для всіх ідентичностей — як людей, так і сервісів: кожна роль отримує рівно ті права, які необхідні для виконання її функцій, і нічого понад це.

12.4. Ізоляція ресурсів та мінімізація привілеїв у хмарних і контейнерних середовищах

Хмарні обчислення радикально змінили підхід організацій до розгортання та управління інформаційними системами. Замість того, щоб купувати та обслуговувати фізичне обладнання, компанії орендують обчислювальні ресурси у хмарних провайдерів — Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP) або інших. Така модель надає унікальні переваги з точки зору гнучкості та масштабованості, але водночас вносить нові безпекові виклики, пов'язані зі спільним використанням фізичної інфраструктури та складністю конфігурації численних хмарних сервісів. Концепція розподілу відповідальності (shared responsibility model) є фундаментальною для розуміння безпеки в хмарних середовищах.

Таблиця 12.3 – Модель розподілу відповідальності за безпеку між провайдером і клієнтом

Модель хмарних послуг	Відповідальність провайдера	Відповідальність клієнта
IaaS (Infrastructure as a Service) наприклад, AWS EC2	Фізична інфраструктура, мережеве обладнання, гіпервізор, фізична безпека датацентру	Операційна система, дані, додатки, IAM, мережеві правила (Security Groups)
PaaS (Platform as a Service) наприклад, AWS RDS, Heroku	Усе з IaaS + операційна система, runtime, середовище бази даних	Дані, конфігурація платформи, облікові записи, доступ до сервісу
SaaS (Software as a Service) наприклад, Gmail, Salesforce	Усе вище + прикладне програмне забезпечення, патчі, доступність	Дані, управління доступом користувачів, конфіденційність та відповідність вимогам

З таблиці 12.3 видно, що зі зростанням рівня абстракції (від IaaS до SaaS) провайдер бере на себе більше відповідальності, але клієнт при цьому не звільняється від обов'язку захищати свої дані та ідентичності — ця відповідальність залишається за клієнтом незалежно від моделі. Найбільш поширеними причинами хмарних інцидентів безпеки є саме некоректна конфігурація на стороні клієнта: публічно відкриті хмарні сховища, надмірно широкі IAM-ролі, відсутність шифрування, незахищені API. Хмарні провайдери надають інструменти для автоматичної перевірки конфігурацій — AWS Config, Azure Security Center, GCP Security Command Center — і їх регулярне використання є обов'язковою умовою підтримки безпечного стану хмарного середовища.

Контейнеризація на основі технології Docker і оркестратора Kubernetes стала домінуючим підходом до упаковки та розгортання прикладного ПЗ. Контейнер — це ізольоване середовище виконання, яке містить додаток разом із усіма залежностями та виконується в межах ядра операційної системи хоста. На відміну від повноцінних віртуальних машин, контейнери не мають власного ядра, що робить їх значно легшими за вагою, але означає, що межа ізоляції між контейнером і хостом тонша, ніж між гіпервізором і гостьовою ОС. У Linux ізоляція контейнерів реалізується через механізми ядра namespaces (простори імен) і cgroups (контрольні групи). Таблиця 12.4 наводить порівняння ключових механізмів ізоляції контейнерів.

Таблиця 12.4 – Механізми ізоляції контейнерів у Linux

Механізм	Призначення	Що ізолює / обмежує	Приклад
namespaces	Ізоляція ресурсів ядра	PID, мережа (net), файлова система (mnt), IPC, UTS, user	Контейнер бачить лише власні процеси, не може бачити процеси хоста
cgroups	Обмеження споживання ресурсів	CPU, оперативна пам'ять, дискове введення/виведення, мережева пропускна здатність	Контейнер не може використати більше 512 МБ RAM та 50% CPU
Linux Capabilities	Гранулярне управління привілеями	Замість повного root надаються окремі привілеї	CAP_NET_BIND_SERVICE — прослуховування портів < 1024 без root
Seccomp	Фільтрація системних викликів	Блокує небезпечні syscall, недоступні контейнеру	Заборона ptrace, mount, reboot для контейнерних процесів
AppArmor / SELinux	Мандатний контроль доступу	Обмежує файловий та мережевий доступ процесу за профілем	Контейнер може читати /etc, але не може змінювати /bin

Безпека контейнерних середовищ включає кілька рівнів, кожен з яких є важливим. На рівні образів контейнерів необхідно використовувати лише перевірені базові образи з офіційних репозиторіїв, регулярно оновлювати їх для усунення відомих вразливостей і сканувати образи на наявність вразливостей перед розгортанням. Рисунок 12.4 ілюструє архітектуру захисту контейнерного середовища у вигляді концентричних рівнів безпеки, де кожен рівень доповнює і підсилює попередній.

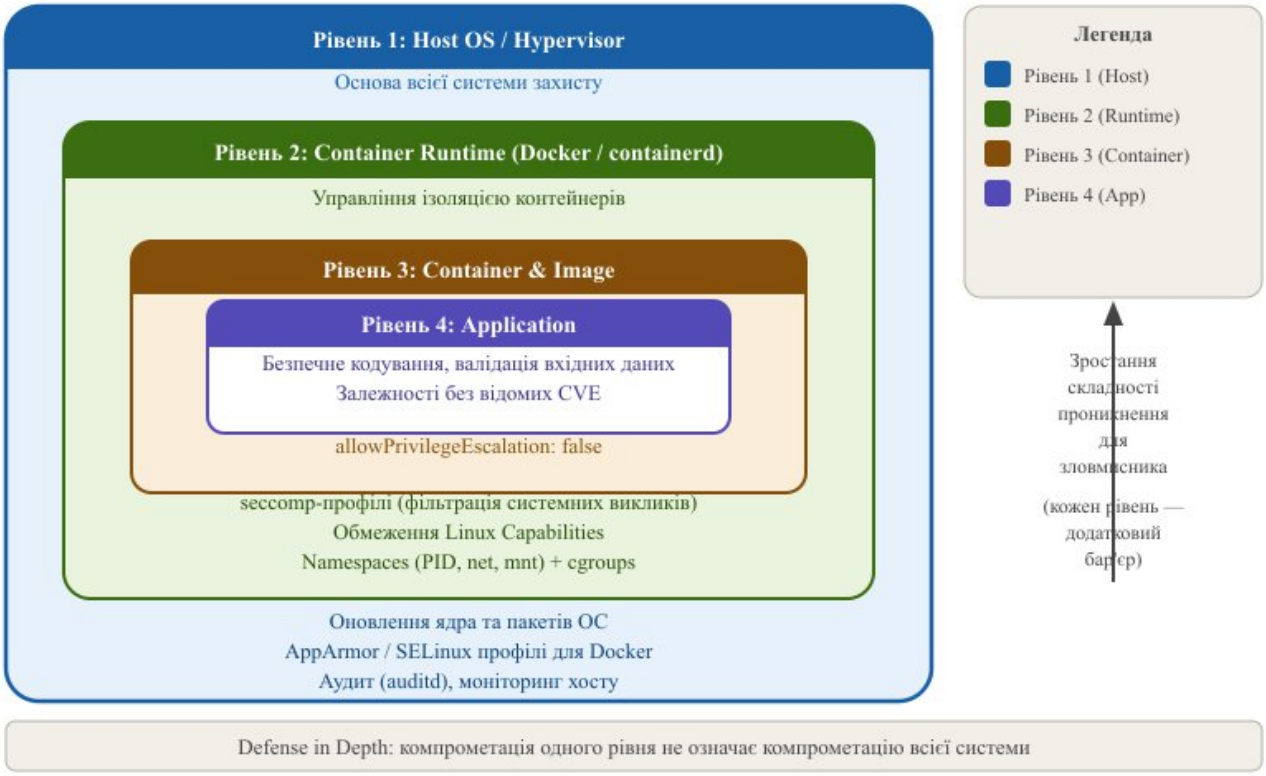

Рисунок 12.4 – Архітектура захисту контейнерного середовища: концентричні рівні ізоляції

Образи мають будуватися за принципом мінімальності: до них включається лише те програмне забезпечення, яке безпосередньо потрібне для роботи додатку, без зайвих утиліт, компіляторів та оболонок, які могли б бути використані зловмисником у разі компрометації контейнера. Для автоматичного сканування образів використовуються спеціалізовані інструменти. Таблиця 12.5 порівнює найбільш поширені з них.

Таблиця 12.5 – Порівняння інструментів сканування вразливостей контейнерних образів

Інструмент	Що сканує	Інтеграція	Особливості
Trivy	Образи Docker, файлові системи, репозиторії Git, конфігурації Kubernetes	CLI, CI/CD (GitHub Actions, GitLab), Kubernetes operator	Простий у використанні; виявляє вразливості, проблеми конфігурацій, секрети у коді
Clair	Образи контейнерів (шари образу)	Quay.io, Harbor registry, API	Акцент на вразливостях пакетів; добре підходить для самостійного хостингу реєстру
Docker Scout	Образи Docker	Docker Desktop, Docker Hub, CLI	Вбудований у Docker; надає рекомендації щодо базових образів
Grype	Образи контейнерів, файлові системи, SBOM	CLI, CI/CD, інтеграція з Syft	Висока точність; підтримка Software Bill

Інструмент	Що сканує	Інтеграція	Особливості
			of Materials (SBOM)

Kubernetes — найпопулярніший оркестратор контейнерів — надає широкий набір механізмів безпеки, правильне налаштування яких є критично важливим. Об'єкт NetworkPolicy дозволяє визначити, яким подам (pods) дозволено спілкуватися між собою і з зовнішнім світом, реалізуючи мікросегментацію на рівні оркестратора. SecurityContext дозволяє задати параметри безпеки для окремого контейнера або всього поду. Наприклад, типова безпечна конфігурація SecurityContext у маніфесті Kubernetes виглядає так:

```
securityContext:
  runAsNonRoot: true
  runAsUser: 1000
  readOnlyRootFilesystem: true
  allowPrivilegeEscalation: false
  capabilities:
    drop:
      - "ALL"
    add:
      - "NET_BIND_SERVICE"
```

Цей приклад YAML-конфігурації задає такі параметри: runAsNonRoot та runAsUser: 1000 забороняють запуск контейнера від імені суперкористувача root (UID 0) і встановлюють конкретний UID для процесу; readOnlyRootFilesystem робить кореневу файлову систему контейнера доступною лише для читання, що унеможливлює запис шкідливого коду у файлову систему; allowPrivilegeEscalation: false забороняє процесу всередині контейнера набувати більше привілеїв, ніж він має при запуску; нарешті, блок capabilities скидає всі стандартні Linux Capabilities (drop ALL) і додає лише одну необхідну — NET_BIND_SERVICE для прослуховування мережевих портів нижче 1024.

Секрети в Kubernetes (об'єкти типу Secret) за замовчуванням зберігаються в etcd у кодуванні base64 без шифрування, тому для продуктивних середовищ обов'язково необхідно налаштувати шифрування etcd at rest або використовувати зовнішнє сховище секретів (HashiCorp Vault, AWS Secrets Manager). Важливо розуміти, що base64 — це кодування, а не шифрування: будь-хто, хто має доступ до etcd або до об'єкта Secret у Kubernetes, може негайно декодувати значення без жодного ключа. PodSecurityAdmission дозволяє визначити на рівні простору імен (namespace) мінімальні вимоги безпеки до всіх подів у ньому — це системна «сітка безпеки», яка запобігає запуску ненавмисно небезпечно налаштованих контейнерів.

Управління мережевою безпекою у хмарних середовищах також суттєво відрізняється від традиційного підходу. У хмарних середовищах мережа є програмно-визначеною (Software-Defined Networking), і для контролю трафіку

використовуються Security Groups (AWS, GCP) або Network Security Groups (Azure) — по суті, це міжмережеві екрани, налаштування яких задається через API або веб-консоль. Критично важливо дотримуватися принципу найменших привілеїв при налаштуванні мережевих правил: правила повинні дозволяти лише необхідний трафік на конкретні порти від конкретних джерел, а не відкривати широкі діапазони адрес або весь Інтернет. Типовою помилкою є відкриття адміністративних сервісів (SSH, порт бази даних) у публічний Інтернет замість використання бастіонних хостів або VPN.

Підсумовуючи, захист даних у розподілених, хмарних та контейнерних середовищах потребує комплексного підходу, що охоплює шифрування даних у спокої та в русі, систематичне управління секретами, впровадження архітектурної концепції Zero Trust, багаторівневу ізоляцію контейнерів і суворе дотримання принципу найменших привілеїв на кожному рівні системи. Жоден із цих елементів не є достатнім сам по собі — лише їх поєднання формує стійкий захисний рубіж. Безпека не є одноразовим заходом: нові вразливості виявляються постійно, конфігурації змінюються, системи оновлюються — тому підтримка безпеки вимагає постійного моніторингу, перевірок конфігурацій, навчання персоналу та готовності реагувати на виявлені проблеми. Саме така система безперервного управління безпекою є метою, до якої має прагнути кожна організація, що серйозно ставиться до захисту своїх даних і ресурсів.

Тема 13. Моніторинг безпеки, реагування на інциденти та цифрові докази

13.1. Моніторинг безпеки програм та даних

Забезпечення безпеки інформаційної системи неможливе без постійного спостереження за її станом. Моніторинг безпеки — це безперервний процес збору, аналізу та інтерпретації інформації про події, що відбуваються в системі, з метою своєчасного виявлення загроз, аномалій і порушень встановленої політики безпеки. На відміну від одноразових перевірок, моніторинг є постійним фоновим процесом, що охоплює всі компоненти інформаційної системи — від ядра операційної системи до застосунків, що виконуються в ній. Саме завдяки моніторингу організація здатна дізнатися про інцидент не постфактум — після виявлення очевидних наслідків, — а в момент його розгортання або навіть на стадії підготовки до атаки. Таким чином, моніторинг є першою і необхідною умовою для ефективного реагування на загрози.

Щоб зрозуміти, де і що саме фіксується, слід розглянути поняття джерел подій безпеки. У Linux-системах основним механізмом журналювання є системний журнал (syslog), який збирає повідомлення від ядра, служб і застосунків у текстових файлах директорії /var/log. Наприклад, файл /var/log/auth.log містить детальну інформацію про всі спроби автентифікації — успішні входи через SSH, невдалі спроби введення пароля, підвищення привілеїв за допомогою команди sudo. Паралельно з syslog у сучасних дистрибутивах активно використовується systemd-journald, який зберігає журнали у двійковому форматі та дозволяє здійснювати зручну фільтрацію подій за часом, сервісом, пріоритетом тощо за допомогою команди journalctl. Така різноманітність інструментів журналювання є нормою для сучасних операційних систем і відображає потребу в різних типах даних для різних задач аналізу.

Окремим і дуже важливим механізмом є підсистема аудиту ядра Linux — auditd. Вона дозволяє фіксувати події на значно нижчому рівні, ніж звичайне журналювання: системні виклики, операції з файлами, зміни в мережевих з'єднаннях, дії конкретних користувачів або процесів. Правила аудиту задаються за допомогою утиліти auditctl, а зібрані події зберігаються у файлі /var/log/audit/audit.log. Наприклад, можна налаштувати аудит таким чином, щоб фіксувалося кожне відкриття файлу /etc/passwd. Якщо якийсь процес або користувач намагатиметься прочитати цей файл у незвичний час або з незвичного місця, відповідний запис з'явиться в журналі аудиту і стане відправною точкою для розслідування. Перетворення результатів аудиту у зручний формат виконується утилітами ausearch та aureport, які надають зручні способи фільтрації та побудови зведених звітів.

Окрім подій операційної системи, надзвичайно важливим джерелом інформації є мережеві журнали. У Linux-системах журналювання мережевого трафіку може виконуватися на рівні фаєрволу — засобами iptables або nftables, — а для глибокого аналізу пакетів застосовуються системи виявлення

вторгнень, зокрема Suricata або Snort. Ці системи аналізують трафік у реальному часі, порівнюючи його зі списком відомих сигнатур атак. Для прикладу, сигнатура Suricata може виглядати так: alert tcp any any -> $HOME_NET 22 (msg:"SSH брутфорс"; threshold: type threshold, track by_src, count 10, seconds 60; sid:1000001;) — це правило генерує сповіщення, якщо одна IP-адреса встановлює більше 10 TCP-з'єднань до порту 22 за 60 секунд. Мережеві журнали дозволяють виявляти підозрілі патерни трафіку та є ключовим джерелом для розслідування інцидентів після їхнього завершення.

Для систематичного уявлення про різні категорії джерел подій безпеки та їхню роль у моніторингу наведено таблицю 13.1.

Таблиця 13.1 — Джерела подій безпеки та їх характеристика

Джерело подій	Тип інформації	Приклади у Linux
Системні журнали ОС	Вхід/вихід користувачів, дії з файлами, зміни конфігурації	/var/log/syslog, /var/log/auth.log
Журнали застосунків	Помилки, виняткові ситуації, дії користувачів у застосунку	/var/log/apache2/access.log, /var/log/mysql/error.log
Мережеві журнали	Мережевий трафік, заблоковані з'єднання, сканування портів	Журнали iptables/nftables, tcpdump, Suricata
Журнали автентифікації	Успішні та невдалі спроби входу, зміни паролів	/var/log/auth.log, /var/log/faillog
Журнали аудиту ядра	Системні виклики, зміни файлів, мережеві з'єднання на рівні ядра	/var/log/audit/audit.log (auditd)

Практичне завдання моніторингу не зводиться лише до накопичення журналів — необхідно забезпечити їхній аналіз. У невеликих системах адміністратор може вручну переглядати журнали, використовуючи утиліти grep, awk, sed. Наприклад, команда grep "Failed password" /var/log/auth.log | awk '{print $11}' | sort | uniq -c | sort -rn дозволяє швидко отримати відсортований перелік IP-адрес, з яких надходили невдалі спроби входу через SSH, та підрахувати їхню кількість. Проте в реальних організаціях, де кількість систем обчислюється десятками або сотнями, ручний аналіз стає неможливим. Для вирішення цієї проблеми застосовуються системи централізованого збору та кореляції журналів — SIEM (Security Information and Event Management). SIEM-система збирає журнали з усіх джерел, нормалізує їх до єдиного формату, виявляє кореляції між подіями з різних систем і генерує сповіщення при виявленні підозрілих патернів.

Одним із ключових механізмів SIEM є правила кореляції подій — логічні умови, що поєднують декілька окремих подій з різних джерел і виявляють складні патерни атак. Наприклад, жодна з таких подій окремо не є сигналом тривоги: невдала спроба входу на один сервер, підключення до рідко

використовуваного порту на іншому сервері, нетиповий запит до бази даних. Але якщо всі три події відбулися з однієї IP-адреси протягом кількох хвилин, правило кореляції може класифікувати це як розвідувальну активність або спробу вторгнення і надіслати відповідне сповіщення аналітику. Саме ця здатність виявляти складні, багатоступеневі атаки відрізняє SIEM від простого журналювання. Архітектуру такої системи зображено на рисунку 13.2.

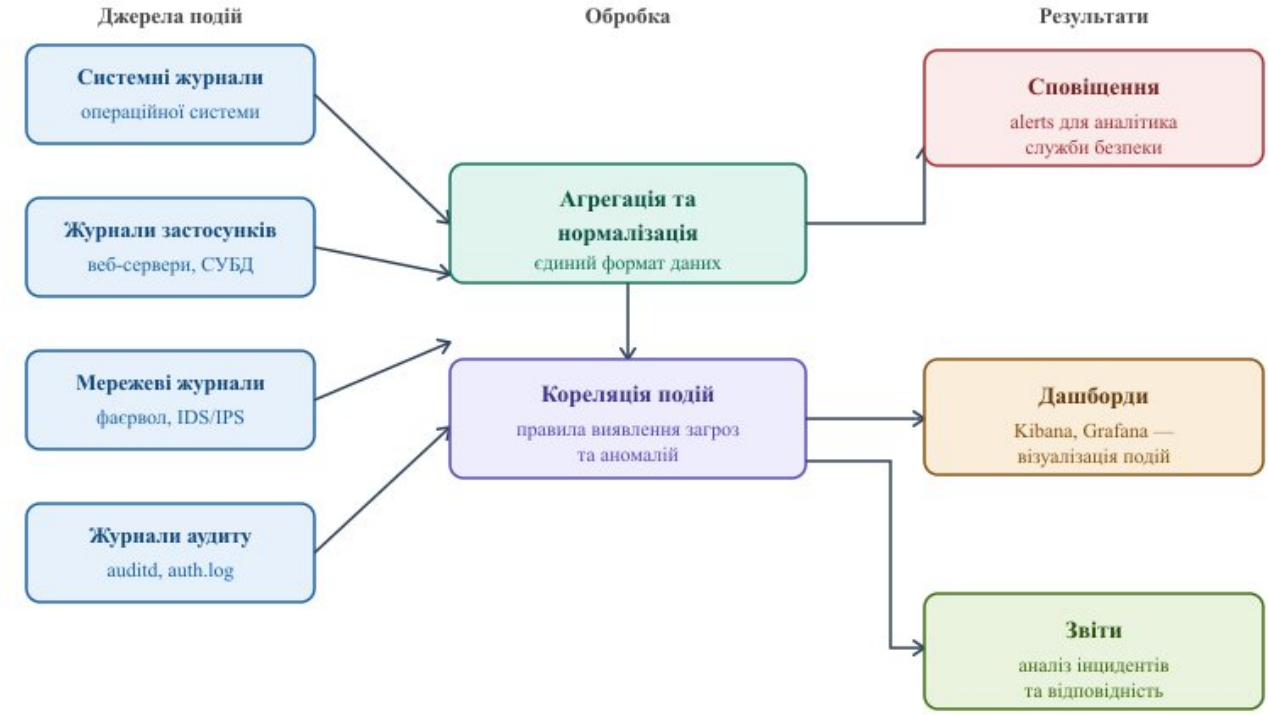

Рисунок 13.2 — Архітектура SIEM-системи

Рисунок 13.2 ілюструє потік даних у SIEM-системі від джерел до результатів. Зліва розташовані джерела подій: системні журнали операційної системи, мережеві журнали та журнали застосунків. Потоки даних від них надходять до блоку агрегації та нормалізації, де різнорідні записи приводяться до єдиного внутрішнього формату. Після нормалізації дані потрапляють до модуля кореляції подій, який застосовує правила для виявлення підозрілих патернів. Виходами системи є сповіщення для аналітика безпеки (alerts), наочні дашборди у веб-інтерфейсі (наприклад, Kibana) та зведені звіти. Такий конвеєрний підхід забезпечує масштабованість рішення незалежно від кількості підключених джерел.

Важливою складовою ефективного моніторингу є встановлення базової лінії нормальної поведінки системи. Якщо відомо, яким є звичайний рівень вхідних з'єднань до веб-сервера, яка кількість невдалих спроб автентифікації є нормою для конкретної системи, в які часові інтервали зазвичай активна та чи інша служба, — тоді будь-яке відхилення від цієї норми є сигналом до перевірки. Без цього навіть найпотужніша SIEM-система буде генерувати велику кількість помилкових спрацьовувань, що знизить довіру до системи моніторингу та збільшить навантаження на аналітиків. Порівняльну

характеристику основних інструментів моніторингу, що широко застосовуються в Linux-середовищах, наведено в таблиці 13.4.

Таблиця 13.4 — Порівняльна характеристика інструментів моніторингу безпеки

Інструмент	Тип	Переваги	Недоліки
syslog / rsyslog	Текстові журнали	Проста конфігурація, підтримка більшістю ОС, зручне пересилання на сервер	Слабка структурованість, складний пошук у великих обсягах
journalctl (systemd)	Двійкові журнали з індексом	Швидкий пошук і фільтрація, збереження метаданих, зручний синтаксис запитів	Прив'язка до systemd, складний аналіз без спеціальних інструментів
auditd	Аудит ядра	Деталізована фіксація на рівні ядра, гнучкі правила, висока надійність	Значне навантаження при розширених правилах, великий обсяг даних
Suricata / Snort	IDS/IPS (мережевий)	Виявлення мережевих атак, підтримка сигнатур, режим inline (блокування)	Потребує налаштування та оновлення сигнатур, ресурсоємний
Elastic Stack (ELK)	SIEM / агрегація	Централізація, пошук по всіх джерелах, наочні дашборди у Kibana	Складне розгортання, значні вимоги до ресурсів

13.2. Реагування на інциденти інформаційної безпеки

Навіть найдосконаліший захист не гарантує абсолютної невразливості системи, тому кожна організація повинна бути готова до виникнення інцидентів інформаційної безпеки та мати чітку процедуру реагування на них. Інцидент інформаційної безпеки — це подія або серія подій, що вказують на можливе порушення політики безпеки, несанкціонований доступ до ресурсів або компрометацію даних. Реагування на інциденти є структурованим процесом, що охоплює виявлення, локалізацію загрози, видалення її наслідків, відновлення нормальної роботи системи та аналіз отриманих уроків. Ця діяльність вимагає не лише технічних знань, але й чіткої організаційної підготовки — завчасно розроблених процедур, розподілених ролей і налагоджених каналів комунікації між учасниками команди реагування. Відповідно до стандарту NIST SP 800-61 та ДСТУ ISO/IEC 27035, процес реагування складається з п'яти основних фаз, що утворюють замкнений цикл.

Циклічну природу процесу реагування ілюструє рисунок 13.1. На рисунку зображено п'ять фаз, що послідовно змінюють одна одну і утворюють

замкнений цикл. Починаючи з фази виявлення та аналізу, процес рухається через стримування, викорінення і відновлення до фінального аналізу після інциденту, результати якого повертаються до початку циклу, вдосконалюючи здатність системи виявляти майбутні загрози. Такий цикл відображає ключову ідею сучасного підходу до безпеки: кожен інцидент є не лише загрозою, але й можливістю для вдосконалення.

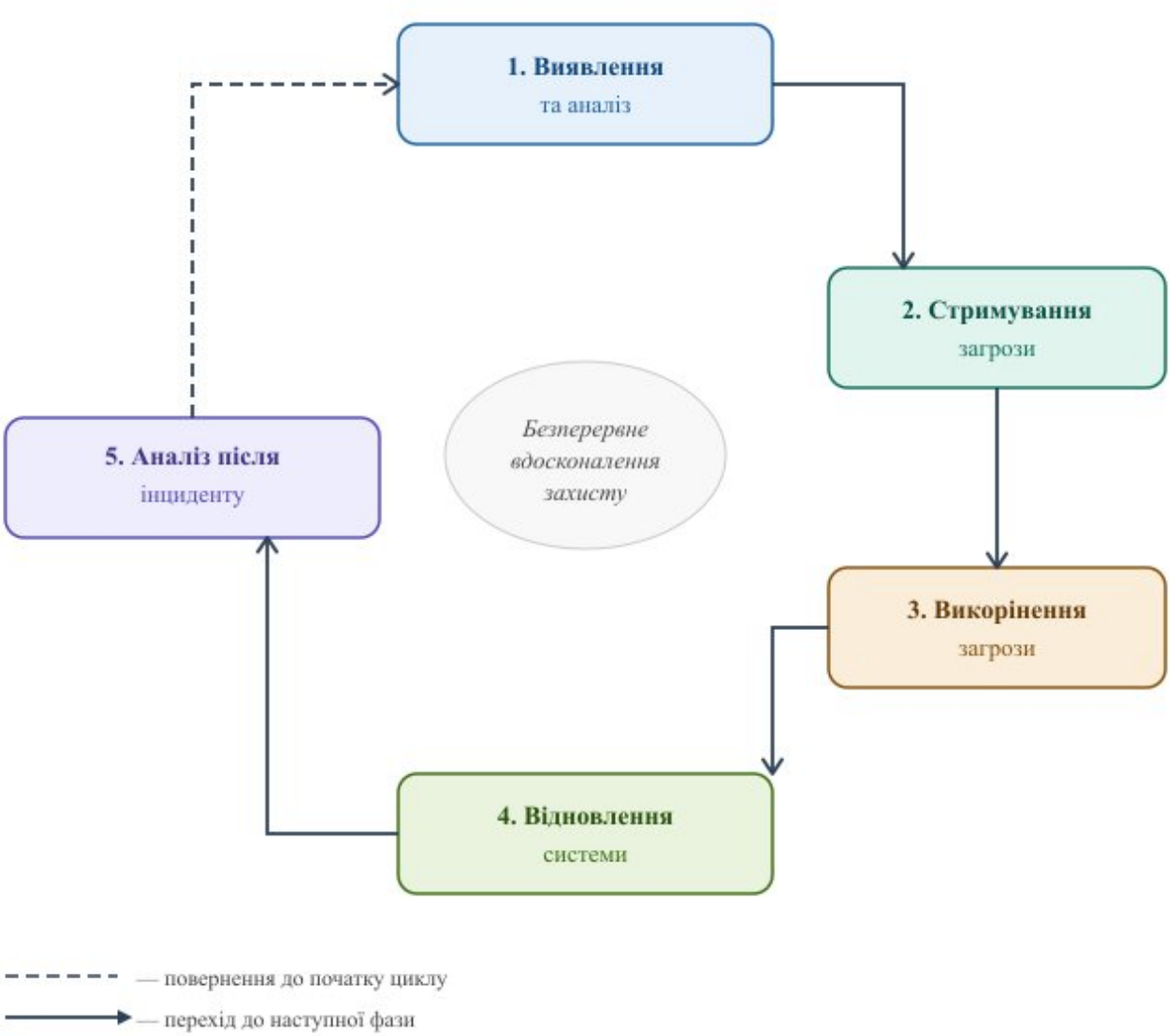

Рисунок 13.1 — Цикл реагування на інциденти інформаційної безпеки

Перша фаза — виявлення та аналіз інциденту — починається з надходження сигналу про підозрілу активність: це може бути автоматичне сповіщення від SIEM-системи, звернення користувача або результат планової перевірки журналів. Завдання аналітика на цьому етапі полягає в тому, щоб підтвердити або спростувати факт реального інциденту, відрізнивши його від помилкового спрацьовування. Для цього необхідно зібрати максимум доступних даних: переглянути відповідні фрагменти журналів, перевірити стан запущених процесів та мережевих з'єднань, оцінити цілісність критичних файлів. Наприклад, у Linux команди ps aux, ss -tulpn та ls -la /etc/ дозволяють оперативно отримати картину поточного стану системи і виявити аномалії.

Класичним прикладом підозрілої активності, яку можна виявити вже на цьому етапі, є SSH brute force — серія невдалих спроб входу, яку добре видно у файлі /var/log/auth.log.

Для наочного розуміння того, як виглядає реальна атака в журналах системи, розглянемо детальний приклад SSH brute force атаки. Зловмисник з IP-адреси 192.168.55.10 о 02:14 починає автоматичний підбір паролю, перебираючи різні імена користувачів. Протягом чотирьох хвилин у /var/log/auth.log фіксується більше 120 рядків виду: "Failed password for root from 192.168.55.10 port XXXXX ssh2". О 02:18 фіксується успішний вхід для облікового запису sysadmin — зловмисник підібрав пароль. Далі він намагається підвищити привілеї через sudo, а потім створює прихований обліковий запис backdoor_user для подальшого доступу. Саме такий зв'язаний ланцюжок подій у журналах, зображений у хронологічному вигляді на рисунку 13.4, є класичним об'єктом таймлайн-аналізу.

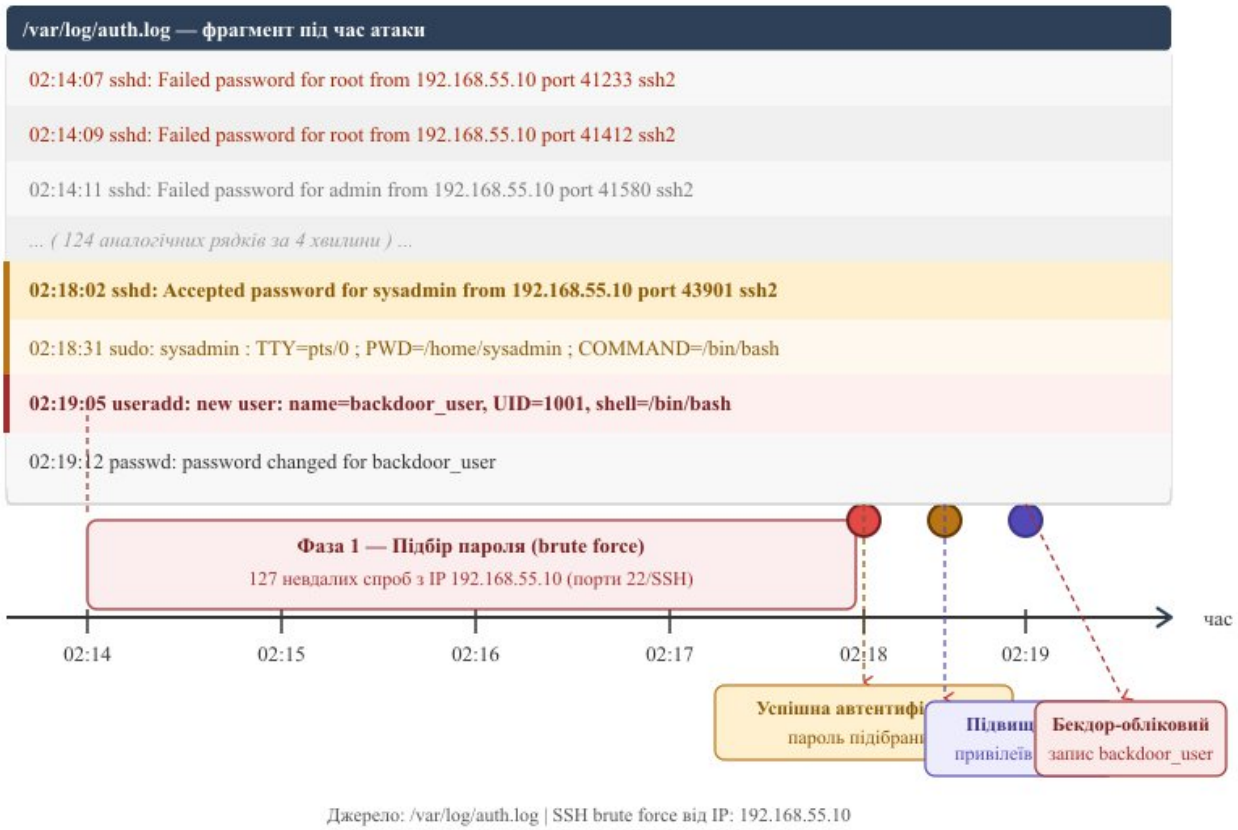

Рисунок 13.4 — Приклад таймлайн-аналізу: фрагмент журналу SSH brute force атаки

Рисунок 13.4 демонструє реальний вигляд записів у /var/log/auth.log під час SSH brute force атаки. Часова шкала в нижній частині рисунку показує, як за менш ніж шість хвилин послідовно розгорталися чотири ключові фази: масовий підбір пароля, успішна автентифікація, спроба підвищення привілеїв через sudo та створення прихованого облікового запису backdoor_user. Кожен рядок журналу містить мітку часу, ім'я сервісу, дію та IP-адресу джерела — саме ці

поля використовуються для побудови хронології та встановлення атрибуції атаки.

Після підтвердження інциденту настає фаза стримування, мета якої — зупинити поширення загрози. Залежно від характеру інциденту стримування може включати ізоляцію ураженого хоста від мережі, блокування підозрілих облікових записів, обмеження мережевого трафіку. У Linux ізоляцію мережі можна виконати, додавши правила до iptables або вимкнувши мережевий інтерфейс командою ip link set eth0 down. Важливо при цьому пам'ятати про баланс між стримуванням загрози та необхідністю зберегти леткі дані для розслідування — вимкнення системи призведе до втрати вмісту оперативної пам'яті. Фазу стримування та всі наступні фази процесу систематизовано в таблиці 13.2.

Таблиця 13.2 — Фази реагування на інцидент інформаційної безпеки

Фаза реагування	Основні дії	Очікуваний результат
1. Виявлення та аналіз	Збір даних із журналів, підтвердження інциденту, попередня класифікація	Підтверджений факт інциденту, первинний опис події
2. Стримування	Ізоляція уражених систем, блокування підозрілих облікових записів, обмеження мережевого доступу	Зупинено поширення інциденту, збережено можливість розслідування
3. Викорінення	Видалення шкідливого ПЗ, усунення вразливостей, відновлення скомпрометованих облікових даних	Усунено причину та наслідки інциденту в системі
4. Відновлення	Введення систем в експлуатацію, перевірка функціональності, посилення захисту	Система повернена до штатного режиму роботи
5. Аналіз після інциденту	Складання звіту, аналіз причин, вдосконалення процедур	Задокументований досвід, покращені процеси захисту

Ефективне реагування на інциденти неможливе без завчасно підготовлених планів і процедур. Організації розробляють плани реагування на інциденти (Incident Response Plan, IRP), у яких детально описані дії для типових сценаріїв: зараження шкідливим ПЗ, несанкціонований доступ до системи, витік даних, атака типу DoS/DDoS. Стандарт NIST SP 800-61 є одним із найбільш поширених міжнародних документів, що регламентує організацію процесів реагування, а ДСТУ ISO/IEC 27035 є його українським аналогом.

Практика показує, що організації, які регулярно проводять навчання та симуляції інцидентів, реагують значно ефективніше, ніж ті, що стикаються з реальними подіями без відповідної підготовки. Для кількісної оцінки ефективності реагування застосовуються спеціальні метрики, наведені в таблиці 13.5.

Таблиця 13.5 — Метрики ефективності реагування на інциденти

Метрика	Опис	Цільове значення
MTTD — Mean Time to Detect	Середній час від початку інциденту до його виявлення службою безпеки. Визначає ефективність моніторингу.	< 1 год (ідеал: < 15 хв)
MTTR — Mean Time to Respond	Середній час від виявлення інциденту до початку активних дій команди реагування.	< 4 год
MTTC — Mean Time to Contain	Середній час від виявлення до локалізації загрози та зупинки її поширення.	< 2 год
MTTRS — Mean Time to Restore	Середній час від інциденту до повного відновлення штатного функціонування систем.	< 24 год

13.3. Збір та аналіз цифрових доказів

У ході реагування на інцидент або розслідування кіберзлочину виникає необхідність збору цифрових доказів — електронних даних, що підтверджують факт події та дозволяють встановити її обставини. Наука, що займається збором, збереженням та аналізом цифрових доказів, називається цифровою форензикою (digital forensics) або комп'ютерно-технічною експертизою. Принципи цієї дисципліни є обов'язковими до дотримання навіть у тих випадках, коли розслідування проводиться не в судовому, а в корпоративному контексті, оскільки дозволяють гарантувати достовірність і відтворюваність отриманих результатів. Цифрові докази суттєво відрізняються від фізичних: вони можуть бути легко знищені або змінені, частина з них є нестійкою в часі, а їхня інтерпретація вимагає спеціальних технічних знань.

Першим і найважливішим принципом роботи з цифровими доказами є забезпечення їхньої цілісності. Будь-які дані, зібрані в ході розслідування, повинні залишатися незмінними після їх вилучення. Для підтвердження цього факту застосовуються криптографічні хеш-функції: відразу після збору даних обчислюється їхня хеш-сума (наприклад, за алгоритмом SHA-256), яка фіксується в документах розслідування. У Linux це виконується командами sha256sum ім'я_файлу або md5sum ім'я_файлу. Практичний приклад: після створення образу диска командою dd if=/dev/sda of=disk_image.img виконується команда sha256sum disk_image.img, яка повертає рядок виду: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855 disk_image.img. Цей хеш записується до протоколу і надалі використовується для підтвердження того, що образ не зазнав змін. Якщо в будь-який наступний

момент хеш, обчислений повторно, відрізнятиметься від зафіксованого, це є незаперечним свідченням модифікації доказів.

Принципово важливим аспектом збору доказів є пріоритизація відповідно до їхньої летючості. Порядок летючості (Order of Volatility), визначений у стандарті RFC 3227, вказує, які дані необхідно збирати в першу чергу, виходячи з того, наскільки швидко вони можуть зникнути. Рисунок 13.3 наочно ілюструє цей порядок у вигляді піраміди летючості.

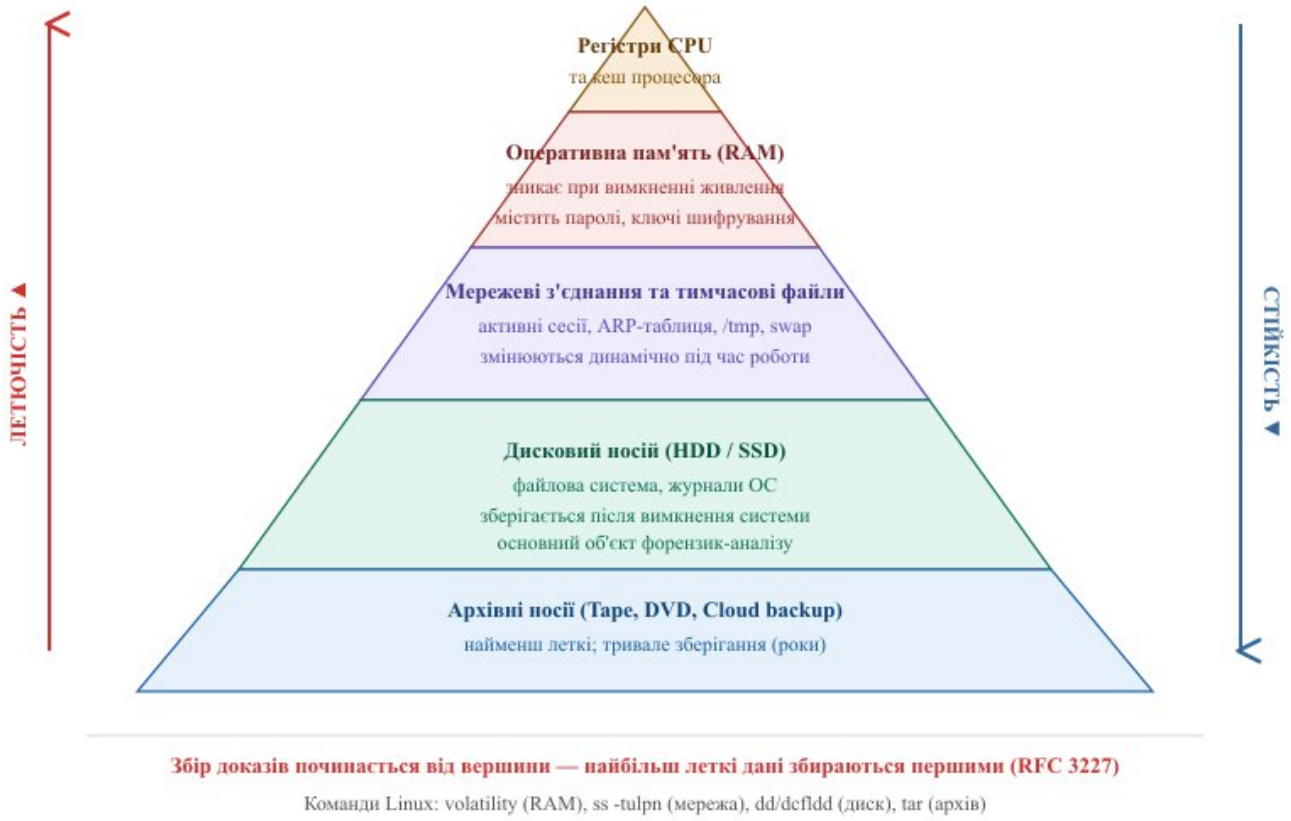

Рисунок 13.3 — Порядок летючості даних (піраміда летючості)

Піраміда летючості на рисунку 13.3 демонструє ієрархію даних від найбільш летючих (вершина) до найбільш стійких (основа). На самій вершині знаходяться регістри процесора та кеш CPU — ці дані існують протягом мікросекунд і зникають при найменшому переключенні контексту. Нижче розташована оперативна пам'ять (RAM), яка зберігає свій вміст лише до вимкнення живлення і може містити критично важливі для розслідування дані: фрагменти шкідливого коду в розшифрованому вигляді, сесійні ключі шифрування, паролі, що були введені. Далі йдуть мережеві з'єднання та тимчасові файли, що змінюються динамічно. Основу піраміди складають дані на дискових носіях та архівні носії — найменш леткі, що зберігаються роками. Зрозуміння цієї ієрархії є обов'язковою основою для правильної розстановки пріоритетів при зборі доказів: слідчий завжди починає від вершини піраміди.

Після збору летких даних переходять до зняття образів носіїв інформації. Правильним підходом є не безпосередня робота з оригінальним носієм, а створення його побітової копії — форензик-образу. Це дозволяє зберегти

оригінальний носій незміненим і проводити всі дослідження з копією. У Linux для цих цілей широко застосовується утиліта dd, а також більш спеціалізований інструмент dcfldd, що є форензик-версією dd з підтримкою автоматичного хешування та верифікації. Наприклад, команда dcfldd if=/dev/sda of=/mnt/backup/disk_image.img hash=sha256 hashwindow=1G hashlog=hashes.txt одночасно знімає образ і обчислює SHA-256 хеш кожного гігабайту даних, що суттєво спрощує наступну верифікацію. Для аналізу образів застосовуються спеціалізовані форензик-платформи Autopsy або The Sleuth Kit (TSK), які дозволяють переглядати файлові системи, відновлювати видалені файли, аналізувати метадані.

Для систематизації принципів роботи з цифровими доказами наведено таблицю 13.3, яка відображає ключові вимоги та їхнє практичне значення.

Таблиця 13.3 — Принципи роботи з цифровими доказами

Принцип	Зміст та практичне значення
Цілісність доказів	Зібрані дані не повинні бути змінені після вилучення. Перевіряється через хеш-суми (MD5, SHA-256), обчислені одразу після збору.
Ланцюг зберігання (chain of custody)	Документування кожного факту доступу до доказів: хто, коли, з якою метою. Забезпечує юридичну прийнятність матеріалів.
Мінімальний вплив на оригінальне джерело	Збір даних виконується з носіїв лише для читання або через форензик-образи. Оригінальні носії зберігаються незмінними.
Документування процесу	Всі дії слідчого фіксуються: команди, час виконання, отримані результати. Дозволяє відтворити процес незалежним фахівцем.
Пріоритет летких даних	Першочергово збираються дані, що зникають при вимкненні: вміст RAM, активні мережеві з'єднання, запущені процеси.

Аналіз зібраних цифрових доказів — це процес відновлення хронології подій та встановлення фактів інциденту. Одним із ключових аналітичних методів є таймлайн-аналіз: на основі часових міток у журналах, файлах і метаданих відновлюється послідовність дій зловмисника. У Linux кожен файл має три часові мітки, які можна переглянути командою stat ім'я_файлу: час останнього доступу (atime), час останньої модифікації вмісту (mtime) та час останньої зміни метаданих або атрибутів inode (ctime). Команда stat /etc/passwd, наприклад, може показати, що mtime файлу — три роки тому (адміністратор останній раз змінював облікові записи), але ctime — вчора вночі: це може означати, що хтось змінив права доступу або атрибути файлу, що є підозрілим сигналом. Однак важливо пам'ятати, що зловмисники можуть навмисно змінювати часові мітки (техніка timestomping), тому таймлайн-аналіз слід завжди проводити в поєднанні з іншими методами.

13.4. Відновлення систем, аналіз після інцидентів та питання для самоперевірки

Відновлення систем після інциденту є критично важливим етапом, що визначає, наскільки швидко і повно організація повертається до нормальної роботи. Основним інструментом відновлення є резервні копії (backup), і саме їхня наявність, актуальність та перевірена придатність до відновлення є тим фактором, що принципово визначає час простою після інциденту. Якщо регулярні резервні копії зберігаються в захищеному місці, ізольованому від основної системи, то навіть після повної компрометації або зашифрування системи зловмисником (як у випадку з ransomware) відновлення займає контрольований і прийнятний час. Резервні копії, що не існують або не перевірялися, є однією з найпоширеніших причин катастрофічних наслідків інцидентів для організацій.

Процес відновлення починається лише після того, як система пройшла фазу викорінення — тобто загроза повністю усунута. Відновлення виконується або шляхом завантаження чистого образу системи та відновлення даних з резервних копій, або через поступове виведення сервісів у робочий режим із одночасним посиленим моніторингом. Після відновлення система повинна перебувати під підвищеним наглядом протягом певного часу — зазвичай декілька тижнів — щоб переконатися, що загроза дійсно ліквідована і не залишилося прихованих backdoor-механізмів повторного доступу. Підвищена увага в цей період виправдана, адже досвідчені зловмисники нерідко залишають резервні шляхи проникнення, як це і показав приклад із рисунку 13.4, де зловмисник створив обліковий запис backdoor_user відразу після успішного входу.

Після завершення активної фази реагування настає час для ретельного аналізу обставин і наслідків інциденту — post-incident review або «post-mortem». Цей аналіз має відповісти на кілька ключових питань: який вектор атаки був використаний для початкового проникнення, чому наявні засоби захисту не зупинили або не виявили атаку своєчасно, скільки часу зловмисник перебував у системі до виявлення (показник MTTD з таблиці 13.5), які дані були скомпрометовані або втрачені. Відповіді на ці питання мають стати основою для конкретних дій: усунення виявлених вразливостей, вдосконалення правил кореляції у SIEM-системі, навчання персоналу, оновлення процедур реагування. Регулярна підготовка звітів про інциденти дозволяє організації накопичувати цінний практичний досвід і поступово підвищувати свою захищеність.

Підсумовуючи матеріал теми, варто підкреслити логічний взаємозв'язок розглянутих процесів. Моніторинг безпеки є основою для своєчасного виявлення інцидентів; виявлення інциденту запускає процес реагування; реагування включає збір і аналіз цифрових доказів; а після відновлення системи проводиться аналіз, результати якого вдосконалюють усі попередні процеси. Ця циклічна модель, зображена на рисунку 13.1, є основою концепції безперервного вдосконалення безпеки і перетворює окремі технічні заходи на єдину, живу систему захисту, здатну адаптуватися до нових загроз і викликів.

Тема 14. Стандарти та комплексний підхід до захисту програм і даних

14.1. Нормативно-правова база та міжнародні стандарти захисту інформації

Ефективний захист програм і даних неможливо побудувати виключно на технічних заходах, якими б досконалими вони не були. Будь-яка реальна система захисту функціонує в певному правовому та організаційному контексті, де вимоги законодавства, галузеві стандарти та корпоративні політики визначають, що саме і яким чином слід захищати. Саме тому розуміння нормативно-правової бази є невід'ємною частиною підготовки фахівця з кібербезпеки, адже незнання відповідних норм може призвести до юридичної відповідальності організації або до проектування системи захисту, яка формально існує, але не відповідає жодним реальним вимогам.

Нормативні документи у сфері інформаційної безпеки утворюють ієрархічну структуру, яка складається з кількох рівнів. На найвищому щаблі знаходяться міжнародні стандарти — документи, розроблені такими організаціями, як ISO, IEC, IETF або ITU-T, що визнаються і застосовуються в більшості країн світу. Нижче знаходяться національні стандарти — адаптації міжнародних документів або самостійно розроблені нормативи конкретної країни (наприклад, ДСТУ в Україні або NIST SP в США). Далі йдуть галузеві стандарти, що регулюють специфічні сфери — наприклад, стандарт PCI DSS для платіжної індустрії або HIPAA для охорони здоров'я у США. На нижньому рівні розташовані корпоративні політики та регламенти — внутрішні документи конкретної організації, що конкретизують вимоги вищих рівнів із урахуванням особливостей підприємства. Опис рівнів ієрархії стандартів наведено на рисунку 14.1: схема зображує чотири прямокутні блоки, розташовані зверху вниз у вигляді піраміди, де кожен нижній блок є ширшим і підписаний відповідно — «Міжнародні стандарти (ISO, IEC, IETF)», «Національні стандарти (ДСТУ, NIST SP)», «Галузеві стандарти (PCI DSS, HIPAA)», «Корпоративні політики та регламенти»; між блоками проведено стрілки, що показують напрямок конкретизації вимог — зверху вниз.

Рисунок 14.1 — Ієрархія нормативних документів у сфері інформаційної безпеки: від міжнародних стандартів до корпоративних політик

Найбільш відомою та широко застосовуваною є серія стандартів ISO/IEC 27000, розроблена Міжнародною організацією зі стандартизації спільно з Міжнародною електротехнічною комісією. Центральним документом серії є ISO/IEC 27001, який встановлює вимоги до систем управління інформаційною безпекою (СУІБ) і є основою для сертифікаційного аудиту. Цей стандарт побудований за циклічною моделлю PDCA (Plan–Do–Check–Act), що передбачає безперервне планування, впровадження, перевірку та вдосконалення заходів безпеки. На рисунку 14.2 зображено цикл PDCA: чотири сектори кола, позначені відповідно «Plan», «Do», «Check», «Act», пов'язані стрілками за годинниковою стрілкою; кожен сектор має коротку анотацію — «Plan: визначення контексту, цілей, ризиків», «Do: впровадження заходів, навчання, документування», «Check: внутрішній аудит, моніторинг метрик», «Act: коригувальні дії, перегляд СУІБ»; у центрі кола розміщено підпис «СУІБ ISO/IEC 27001».

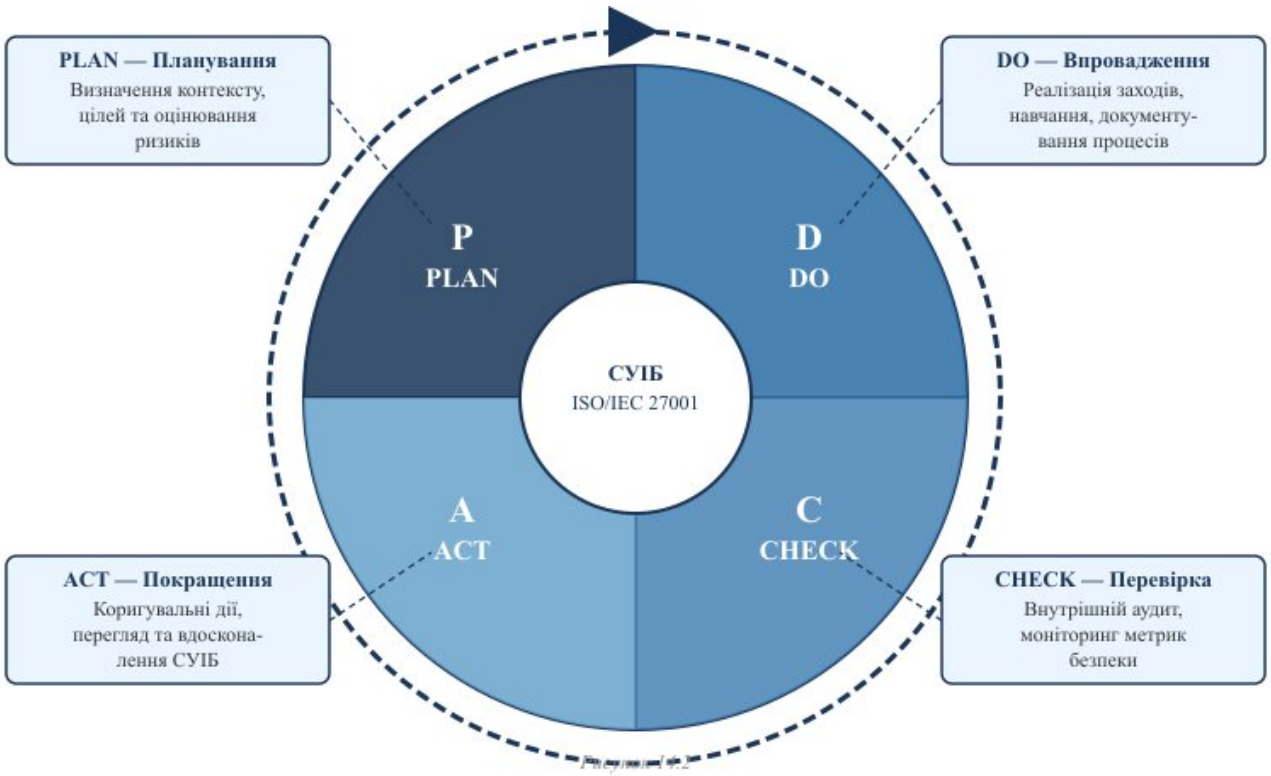

Рисунок 14.2 — Цикл PDCA як основа системи управління інформаційною безпекою за ISO/IEC 27001

Важливо розуміти, що ISO/IEC 27001 не диктує конкретних технічних рішень, а встановлює процесний підхід і вимоги до управління ризиками, надаючи організаціям гнучкість у виборі технологій. Практичним доповненням є стандарт ISO/IEC 27002, що містить детальні рекомендації щодо впровадження конкретних заходів безпеки, згрупованих у тематичні домени. У редакції 2022 року стандарт включає розділи щодо організаційних, кадрових, фізичних і технологічних заходів. Для захисту програм та даних особливо актуальні домени A.9 (контроль доступу), A.10 (криптографія), A.12 (операційна безпека) та A.16 (управління інцидентами). Ключові домени стандарту наведено в таблиці 14.1.

Таблиця 14.1 — Ключові домени стандарту ISO/IEC 27001/27002 та їх зміст

Розділ	Назва домену	Зміст вимог
A.5	Політики інформаційної безпеки	Розроблення, затвердження та регулярний перегляд політик безпеки керівництвом
A.6	Організація інформаційної безпеки	Розподіл відповідальності, взаємодія з органами влади, дистанційна робота
A.8	Управління активами	Інвентаризація, класифікація, обробка та знищення активів
A.9	Контроль доступу	Управління правами користувачів, автентифікація, привілейований доступ

Розділ	Назва домену	Зміст вимог
A.10	Криптографія	Політика використання криптографічних засобів, управління ключами
A.12	Операційна безпека	Захист від шкідливого ПЗ, резервне копіювання, журналювання подій
A.16	Управління інцидентами	Звітування про інциденти, реагування, аналіз та вилучення уроків

Сертифікація за стандартом ISO/IEC 27001 є добровільною, проте для багатьох організацій вона стає конкурентною перевагою або обов'язковою вимогою партнерів і замовників. Процес сертифікації складається з кількох етапів: спочатку організація самостійно впроваджує СУІБ відповідно до вимог стандарту, потім проводиться внутрішній аудит для оцінки готовності, і лише після цього запрошується акредитований зовнішній орган сертифікації для проведення офіційного аудиту у два етапи — документальна перевірка (Stage 1) та перевірка фактичного впровадження (Stage 2). При успішному проходженні видається сертифікат терміном на три роки з обов'язковими щорічними наглядовими аудитами. Вартість і тривалість процесу суттєво залежать від розміру організації та складності її інформаційної інфраструктури.

В Україні нормативно-правова база захисту інформації формується на декількох рівнях. На законодавчому рівні діють Закони «Про інформацію», «Про захист інформації в інформаційно-телекомунікаційних системах», «Про захист персональних даних», «Про основні засади забезпечення кібербезпеки України». Технічний захист інформації в державних організаціях регулюється Державною службою спеціального зв'язку та захисту інформації (ДССЗЗІ) через систему нормативних документів НД ТЗІ. У Сполучених Штатах провідну роль відіграє NIST зі своєю серією SP 800, а також Cybersecurity Framework (CSF), що структурує заходи безпеки за п'ятьма функціями: Identify, Protect, Detect, Respond та Recover. Регуляторні вимоги ЄС — зокрема, GDPR та Директива NIS2 — є обов'язковими для будь-якої організації, що обробляє персональні дані громадян Євросоюзу, незалежно від географічного розташування компанії. Порівняльну характеристику ключових нормативних документів наведено в таблиці 14.2.

Таблиця 14.2 — Порівняльна характеристика нормативних документів у сфері захисту інформації

Документ / стандарт	Область застосування	Ключові вимоги	Тип документа
ISO/IEC 27001:2022	Системи управління ІБ (СУІБ)	Планування, впровадження, моніторинг, покращення СУІБ за моделлю PDCA	Міжнародний стандарт (сертифікація)

Документ / стандарт	Область застосування	Ключові вимоги	Тип документа
NIST SP 800-53 Rev.5	Федеральні ІС США	Каталог заходів безпеки та конфіденційності (понад 1000 заходів)	Національний стандарт (США)
ДСТУ ISO/IEC 27001:2015	Організації України	Адаптація ISO 27001 для вітчизняного регулювання	Національний стандарт (UA)
НД ТЗІ 3.7-003-05	Технічний захист інформації в Україні	Порядок проведення атестації КСЗІ	Нормативний документ ДССЗЗІ
GDPR (2016/679)	Персональні дані громадян ЄС	Обробка, зберігання, захист ПД; штрафи до 4% річного обігу	Регламент ЄС (обов'язковий)

14.2. Вимоги до криптографічного захисту та аудит відповідності

Криптографічний захист займає особливе місце в нормативних документах з інформаційної безпеки, оскільки він є технічним фундаментом для реалізації конфіденційності, цілісності та автентичності даних. Різні стандарти висувають конкретні вимоги щодо алгоритмів шифрування, довжини ключів і організації управління ключами. Ці вимоги не є статичними: з розвитком обчислювальних потужностей і появою нових методів криптоаналізу стандарти регулярно переглядаються, а алгоритми, що раніше вважалися надійними, переходять до категорії застарілих. Саме тому фахівець з кібербезпеки повинен розуміти не лише поточні рекомендації, а й причини, з яких ті чи інші алгоритми визнаються недостатньо надійними.

Американський федеральний стандарт FIPS 140-3 (Federal Information Processing Standard) є обов'язковим для державних установ США і широко використовується як орієнтир надійності у комерційному секторі. Він визначає чотири рівні безпеки для криптографічних модулів — від першого (базова програмна реалізація) до четвертого (апаратний модуль із повним захистом від фізичного зламування). Сертифікація FIPS 140-3 є об'єктивним підтвердженням коректності реалізації криптографічного модуля: під час сертифікації незалежна лабораторія перевіряє правильність алгоритмів, механізмів самотестування та управління ключами. Порівняння рівнів стандарту наведено в таблиці 14.3, а рекомендовані та застарілі алгоритми — в таблиці 14.4.

Таблиця 14.3 — Рівні безпеки стандарту FIPS 140-3 та їх характеристики

Рівень	Характеристика захисту	Типові вимоги	Область застосування
Рівень 1	Базова програмна реалізація без спеціальних фізичних заходів захисту	Коректна реалізація алгоритмів, схвалених NIST	Загальне комерційне ПЗ, браузери

Рівень	Характеристика захисту	Типові вимоги	Область застосування
Рівень 2	Наявність апаратних механізмів ідентифікації та захист від несанкціонованого доступу	Запечатані корпуси, ролі оператора та адміністратора	Корпоративні сервери, HSM початкового рівня
Рівень 3	Фізичний захист з виявленням і реагуванням на спроби зламування	Обнулення ключів при спробі злому, антизондуючі покриття	Критичні корпоративні системи, банківські HSM
Рівень 4	Повний захист від будь-якого фізичного зламування з активним реагуванням	Захист від зміни напруги, температури, електромагнітного впливу	Військові та урядові системи найвищого рівня секретності

Таблиця 14.4 — Рекомендовані та застарілі криптографічні алгоритми

Категорія	Рекомендовані алгоритми	Застарілі (не використовувати)	Причина відмови від застарілих
Симетричне шифрування	AES-128, AES-256 (режими GCM, CTR, CBC з HMAC)	DES, 3DES, RC4	Короткий ключ (DES), слабкі режими (RC4 з відомими атаками зміщення)
Асиметричне шифрування	RSA-2048/4096, ECDH з кривими P-256/P-384	RSA-512, RSA-1024	Практично зламані сучасними методами факторизації
Цифровий підпис	ECDSA P-256/P-384, EdDSA (Ed25519), RSA-PSS 2048+	RSA-PKCS#1v1.5 з малим ключем	Вразливість до атак Bleichenbacher, малий ключ
Хеш-функції	SHA-256, SHA-384, SHA-512, SHA-3	MD5, SHA-1	Практично знайдені колізії (MD5 — з 2004р., SHA-1 — з 2017р.)
Обмін ключами	ECDHE, X25519, DHE (DH-групи ≥ 2048 біт)	Статичний DH, DHE < 1024 біт	Відсутність прямої секретності (PFS) або слабкі групи

Переходу до застарілих алгоритмів слід уникати навіть у тих випадках, коли їх підтримка реалізована у стандартних бібліотеках і програмне забезпечення за замовчуванням їх пропонує. Наприклад, хеш-функція MD5 практично зламана ще з 2004 року — тоді було вперше продемонстровано практичне знаходження колізій. Незважаючи на це, її досі можна зустріти в системах, що не оновлювалися. Аналогічна ситуація склалася з SHA-1, для якого у 2017 році команда Google і CWI Amsterdam здійснила першу практичну атаку зіткнення (SHAttered). Хеш-функції SHA-2 (SHA-256, SHA-384, SHA-512) залишаються надійними, а стандартизація нового сімейства SHA-3 надає додаткову альтернативу у разі виявлення вразливостей у SHA-2. Окрему увагу

заслуговує постквантова криптографія: NIST у 2024 році опублікував перші постквантові стандарти (FIPS 203 на базі CRYSTALS-Kyber, FIPS 204 на базі CRYSTALS-Dilithium), і організації, що зберігають дані з тривалим строком конфіденційності, вже повинні розпочинати планування міграції.

Управління криптографічними ключами є не менш важливим, ніж вибір алгоритму: навіть найнадійніший алгоритм не захистить дані, якщо ключі зберігаються у незахищеному місці. Стандарт NIST SP 800-57 регламентує повний життєвий цикл ключів — від генерації до знищення — і є практичним керівництвом для побудови системи управління ключами. Основні вимоги зводяться до наступного: ключі повинні генеруватися за допомогою криптографічно стійких генераторів псевдовипадкових чисел (CSPRNG), зберігатися у зашифрованому вигляді або в спеціалізованих апаратних сховищах (HSM), мати визначений строк дії та процедуру ротації, а відкликані або прострочені ключі — знищуватися за затвердженою процедурою.

Аудит відповідності (compliance audit) — це систематичний процес перевірки того, наскільки реальний стан системи захисту відповідає встановленим нормативним вимогам. На відміну від технічного аудиту безпеки, що зосереджений на пошуку вразливостей, аудит відповідності перевіряє документальне оформлення процесів, наявність і коректність внутрішніх регламентів, а також виконання визначених процедур персоналом. Класичний приклад: система шифрування технічно налаштована правильно, але якщо відсутня задокументована процедура управління ключами й відповідний персонал не проходив навчання — аудит відповідності ISO/IEC 27001 зафіксує невідповідність, навіть за відсутності технічних вразливостей. Процес аудиту зображено на рисунку 14.3 у вигляді блок-схеми: прямокутні блоки позначають послідовні кроки — «Визначення стандарту та об'єктів аудиту», «Збір документальних доказів», «Технічна перевірка конфігурацій», «Інтерв'ю з відповідальними особами», «Аналіз відповідності вимогам», «Формування звіту з аудиту (статуси: виконується / частково / не виконується)», «Формування плану коригувальних дій»; між першим і останнім блоком проведено стрілку зворотного зв'язку, що демонструє циклічність процесу.

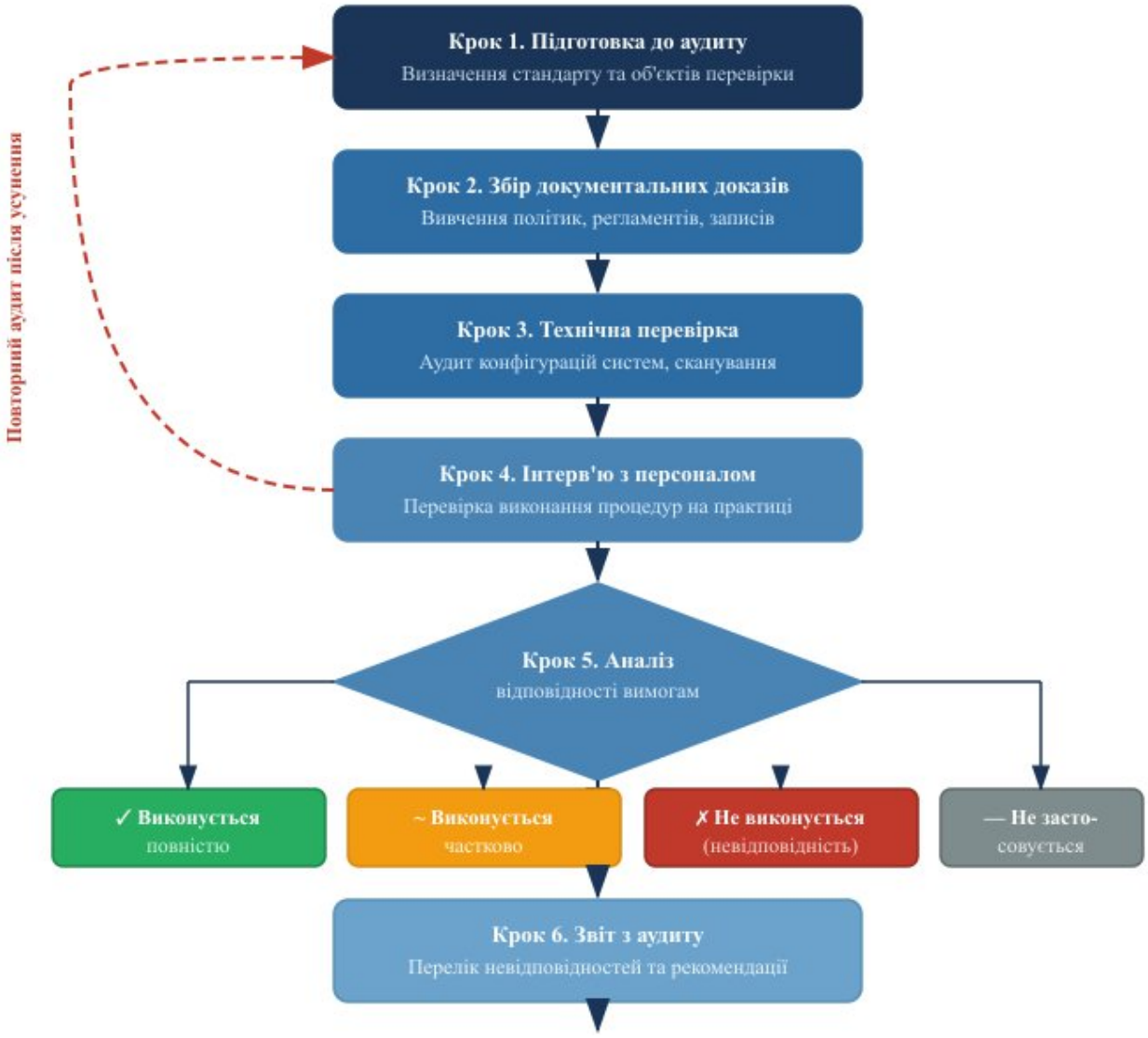

Рисунок 14.3 — Процес аудиту відповідності: від визначення стандарту до плану коригувальних дій

Для практичної перевірки відповідності стандарт CIS (Center for Internet Security) розробив детальні контрольні списки — CIS Benchmarks — для найпоширеніших операційних систем, хмарних платформ і прикладного програмного забезпечення. Наприклад, CIS Benchmark для Linux містить сотні конкретних перевірок: чи активовано брандмауер, чи заборонений вхід під root через SSH, чи налаштовано автоматичне оновлення безпеки. Перевірка може здійснюватися як вручну, так і за допомогою автоматизованих інструментів. У наведеному нижче прикладі 14.1 показано кілька команд Linux, якими перевіряють базові параметри захищеності відповідно до вимог CIS Benchmark.

Приклад 14.1 — Перевірка відповідності CIS Benchmark для Linux (вибірка команд)

```
# Перевірка статусу брандмауера (UFW або firewalld)
sudo systemctl status ufw


# Перевірка ключових параметрів конфігурації SSH
```

```
    sudo grep -E '^(PermitRootLogin|PasswordAuthentication|MaxAuthTries)' /etc/ssh/sshd_config

    # Перевірка наявності доступних оновлень безпеки
    sudo apt list --upgradable 2>/dev/null | grep -i security

    # Перевірка прав доступу до критичних файлів
    ls -l /etc/passwd /etc/shadow /etc/sudoers

    # Запуск автоматизованого аудиту безпеки системи інструментом Lynis
    sudo lynis audit system
```

Інструмент Lynis формує детальний звіт із загальним балом захищеності (Hardening Index), переліком знайдених попереджень і рекомендацій, згрупованих за категоріями. Результати такого автоматизованого аудиту є зручною відправною точкою для Gap Analysis — аналізу розриву між поточним станом і вимогами обраного стандарту. Важливо розуміти, що автоматизовані інструменти виявляють технічні невідповідності, але не можуть перевірити наявність документації, проведення навчання персоналу чи коректність організаційних процесів — ці аспекти вимагають ручної перевірки у формі інтерв'ю та документального аудиту.

14.3. Інтеграція організаційних, програмних і технічних засобів захисту

Концепція комплексного захисту інформації виходить із принципу, що жоден окремо взятий засіб безпеки не здатний забезпечити прийнятний рівень захисту сучасної інформаційної системи. Реальна система захисту являє собою узгоджену взаємодію організаційних, програмних і технічних заходів, кожен з яких перекриває ті ризики, з якими інші заходи справляються гірше. Цей підхід відомий під назвою «глибинний захист» (defense in depth) і є одним із фундаментальних принципів кібербезпеки. Його можна порівняти зі стародавньою фортецею, де є і зовнішній рів, і стіна, і внутрішні укріплення, і стража — жодна з ліній захисту окремо не є достатньою, але разом вони створюють надійну систему, вимушуючи зловмисника послідовно долати кілька незалежних перепон.

Організаційні засоби захисту формують основу всієї системи безпеки, оскільки визначають правила і процедури, яких мають дотримуватися всі учасники інформаційних процесів. До організаційних засобів відносяться: розроблення та затвердження політик безпеки (загальної політики, парольної політики, політики використання мобільних пристроїв), регламентів реагування на інциденти, інструкцій з обробки конфіденційної інформації. Особливе місце займає парольна політика: типові мінімальні вимоги передбачають довжину не

менше 12 символів з обов'язковим включенням великих і малих літер, цифр та спеціальних символів, термін дії не більше 90 днів і заборону повторного використання останніх 10 паролів. Навчання та підвищення обізнаності персоналу є критично важливим організаційним заходом, оскільки більшість успішних атак використовує людський фактор — фішинг, соціальна інженерія, ненавмисне розкриття облікових даних.

Програмні засоби захисту реалізують технічні вимоги безпосередньо у програмному середовищі й охоплюють надзвичайно широкий спектр засобів. Механізми автентифікації та авторизації забезпечують контроль доступу до ресурсів; брандмауери й системи виявлення вторгнень захищають мережевий периметр; антивірусне та антишкідливе програмне забезпечення виявляє відомі загрози; системи шифрування захищають дані на рівні файлів, розділів диска або баз даних; SIEM-системи забезпечують централізоване управління журналами подій. У середовищі Linux конкретними прикладами є: обов'язкові моделі контролю доступу SELinux та AppArmor, фаєрвол nftables, система виявлення вторгнень OSSEC, утиліта аудиту auditd для детального журналювання, інструмент Fail2ban для захисту від атак підбору паролів. Кожен із цих програмних засобів вирішує конкретну задачу захисту і разом вони утворюють багаторівневу систему.

Технічні (апаратні) засоби захисту включають фізичні компоненти, що забезпечують безпеку на апаратному рівні. Апаратні модулі безпеки (HSM) зберігають криптографічні ключі в ізольованому захищеному середовищі, недоступному для операційної системи безпосередньо. Мікросхеми TPM (Trusted Platform Module) забезпечують цілісність завантаження системи й зберігання чутливих даних: наприклад, повне шифрування диска з використанням BitLocker або LUKS може прив'язуватися до конкретної конфігурації апаратного забезпечення через TPM, що унеможливлює розшифрування диска при його фізичному вилученні з комп'ютера. Апаратні токени та смарт-картки забезпечують двофакторну автентифікацію, стійку до перехоплення паролів.

Взаємодія рівнів захисту в концепції defense in depth можна уявити у вигляді концентричних кіл, де зовнішнє коло відповідає організаційним заходам, а у центрі знаходяться самі дані. На рисунку 14.4 показано схему рівнів захисту у вигляді концентричних кіл: зовнішнє коло підписане «Організаційні заходи» та забарвлене в блакитний колір; наступне коло — «Фізична безпека» (зелений); далі — «Мережевий захист» (жовтий); потім — «Безпека ОС» (помаранчевий); «Безпека додатків» (світло-червоний); у центрі знаходяться «Дані» (темно-червоний). Ліворуч від схеми розміщена стрілка з підписом «Вектор атаки», що вказує у бік центру, ілюструючи послідовне подолання рівнів захисту зловмисником. Таблицю 14.4 наведено нижче — в ній перелічено рівні, відповідні засоби і типові атаки, що спрямовані проти кожного рівня.

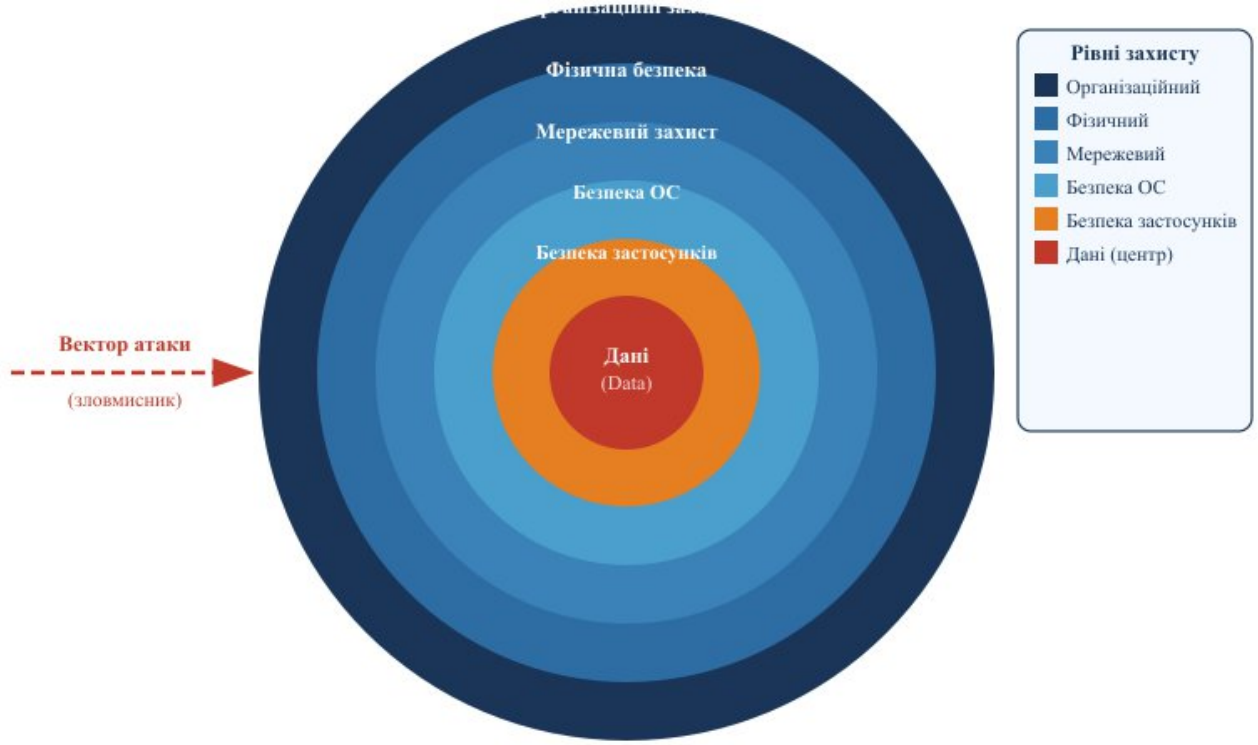

Рисунок 14.4 — Модель глибинного захисту (Defense in Depth): концентричні рівні від організаційних заходів до даних із вектором атаки

Таблиця 14.4 — Рівні захисту в моделі Defense in Depth, відповідні засоби та типові атаки

Рівень	Складова захисту	Приклади засобів	Типові атаки на цей рівень	Відповідні стандарти
1	Організаційний	Політики, регламенти, навчання, NDA	Соціальна інженерія, фішинг	ISO 27001, NIST 800-53
2	Фізичний	СКУД, відеонагляд, замки серверних	Крадіжка обладнання, фізичний доступ	ISO 27001 A.11
3	Мережевий	Фаєрвол, IDS/IPS, VPN, DMZ	MitM, DDoS, сканування портів	ISO 27001 A.13
4	Системний (ОС)	Оновлення, аудит конфігурацій, SELinux	Ескалація привілеїв, руткіти	CIS Benchmarks
5	Прикладний	Автентифікація, WAF, валідація вводу	SQL-ін'єкції, XSS, CSRF	OWASP, ISO 27034
6	Даних	Шифрування at rest/in transit, DLP	Витік даних, несанкціонований доступ	ISO 27001 A.10, FIPS 140-3

Особливо важливою є злагодженість між рівнями: слабка ланка на будь-якому рівні може нівелювати зусилля на інших. Класичним прикладом є ситуація, коли організація впровадила складну технічну інфраструктуру безпеки, але не проводила навчання персоналу — і зловмисник досяг мети за допомогою фішингового листа, навіть не намагаючись подолати жоден технічний засіб захисту. Принцип розподілу відповідальності (Separation of Duties) і матриця RACI (Responsible, Accountable, Consulted, Informed) є організаційними інструментами, що допомагають чітко визначити, хто відповідає за кожен захід безпеки, уникаючи ситуацій, коли відповідальність розмита і жоден конкретний виконавець не відчуває особистої відповідальності за певний аспект захисту.

14.4. Практичні аспекти побудови та оцінювання комплексної системи захисту

Розуміння стандартів і принципів є необхідною, але недостатньою умовою для побудови ефективної системи захисту. Необхідно також знати, як перейти від абстрактних вимог нормативних документів до конкретних технічних та організаційних рішень, як оцінити поточний стан захищеності організації та визначити пріоритети для вдосконалення. Відправною точкою є оцінювання ризиків відповідно до методології ISO/IEC 27005 або NIST RMF, що передбачає ідентифікацію активів, визначення загроз і вразливостей, оцінку ймовірності та потенційного впливу їх реалізації. Ризики можуть оброблятися чотирма способами: зниження (впровадження заходів захисту), прийняття, передача (страхування) або уникнення. Вибір стратегії є управлінським рішенням, що ґрунтується на технічному аналізі фахівців з кібербезпеки.

Gap Analysis — аналіз розриву між поточним станом захищеності і вимогами обраного стандарту — є одним із ключових практичних інструментів фахівця з кібербезпеки. Він надає структурований перелік того, що вже виконується, що виконується частково і що потребує впровадження з нуля. На основі результатів Gap Analysis формується план впровадження (roadmap), у якому заходи пріоритизуються з урахуванням рівня ризику, вартості впровадження та наявних ресурсів. Нижче наведено приклад 14.2 — скорочений перелік ключових питань, що перевіряються під час Gap Analysis щодо відповідності базовим вимогам ISO/IEC 27001.

Приклад 14.2 — Орієнтовний перелік пунктів Gap Analysis щодо ISO/IEC 27001 (вибірка)

```
[ ] Затверджена та актуальна Політика інформаційної безпеки (A.5)
[ ] Проведена інвентаризація та класифікація інформаційних активів (A.8)
[ ] Визначені власники активів та відповідальні за захист (A.6)
[ ] Реалізовано управління правами доступу (A.9): призначення, перегляд, відкликання
```

```
[ ] Задокументована процедура управління криптографічними ключами (A.10)
[ ] Налаштоване журналювання подій безпеки та зберігання логів (A.12)
[ ] Існує задокументована процедура управління вразливостями (A.12)
[ ] Визначений процес реагування на інциденти зі строками та відповідальними (A.16)
[ ] Проводиться регулярне навчання персоналу з питань ІБ (A.7)
[ ] Виконані внутрішні аудити СУІБ протягом останніх 12 місяців
[ ] Актуальний Statement of Applicability (SoA) затверджений керівництвом
[ ] Результати оцінювання ризиків задокументовані та регулярно переглядаються
```

Документ Statement of Applicability (SoA) є одним із ключових документів СУІБ: у ньому для кожного заходу з Додатку А стандарту вказується, чи застосовується цей захід в організації, і якщо так — у якій формі він реалізований; якщо ні — надається обґрунтування виключення. SoA є водночас карткою відповідності та управлінським рішенням: його затвердження керівництвом означає, що менеджмент розуміє і свідомо приймає ті ризики, для яких заходи визнані неприйнятними або такими, що не застосовуються. Відсутність або відсутність актуалізації SoA є однією з найпоширеніших причин зауважень під час сертифікаційного аудиту.

Метрики безпеки дозволяють кількісно оцінювати ефективність впроваджених заходів і відстежувати динаміку рівня захищеності з часом. Вони є інструментом звітування перед керівництвом без надмірного заглиблення в технічні деталі та обґрунтування необхідності тих чи інших інвестицій у захист. Таблиця 14.5 містить приклади ключових метрик безпеки, їх опис, цільові значення та рекомендовану частоту збору. Важливо, що метрики мають бути реалістичними та вимірюваними: метрика, яку важко або неможливо виміряти на практиці, не принесе користі незалежно від своєї теоретичної цінності.

Таблиця 14.5 — Ключові метрики безпеки для оцінювання ефективності СУІБ

Метрика	Опис / спосіб вимірювання	Ціль	Частота збору
MTTR (Mean Time to Remediate)	Середній час від моменту виявлення вразливості до її повного усунення	< 72 год (критичні)	Щотижнево
Охоплення оновленнями	(Систем з актуальними оновленнями / Всього систем) × 100%	> 95%	Щомісяця
% навченого персоналу	(Співробітники, що пройшли навчання / Всього співробітників) × 100%	100%	Щорічно

Метрика	Опис / спосіб вимірювання	Ціль	Частота збору
Кількість відкритих інцидентів	Загальна кількість зареєстрованих і ще не закритих інцидентів ІБ	Мінімум	Щоденно
Час виявлення інциденту (MTTD)	Середній час від початку інциденту до його виявлення системами моніторингу	< 24 год	Щомісяця
Відсоток аудитів без критичних знахідок	(Аудити без критичних невідповідностей / Всього аудитів) × 100%	> 80%	Щоквартально

Зрілість системи захисту оцінюється за допомогою моделей зрілості (Maturity Models). Більшість таких моделей розрізняє п'ять рівнів: початковий (процеси хаотичні, реактивні, залежать від окремих осіб), повторюваний (базові процеси задокументовані й виконуються, але без повної стандартизації), визначений (процеси стандартизовані, задокументовані та навчені по всій організації), керований (процеси вимірюються й контролюються за допомогою метрик), оптимізований (безперервне покращення процесів на основі кількісного зворотного зв'язку). Схема рівнів зрілості зображена на рисунку 14.5: п'ять сходинок, розташованих знизу вгору і зліва направо, кожна підписана відповідним рівнем та коротким описом; стрілка вздовж сходинок вказує напрямок зростання зрілості; рівні 1-2 позначено жовтим кольором, рівні 3-4 — помаранчевим, рівень 5 — зеленим, що умовно відповідає категоріям «незрілий», «зростаючий» та «зрілий».

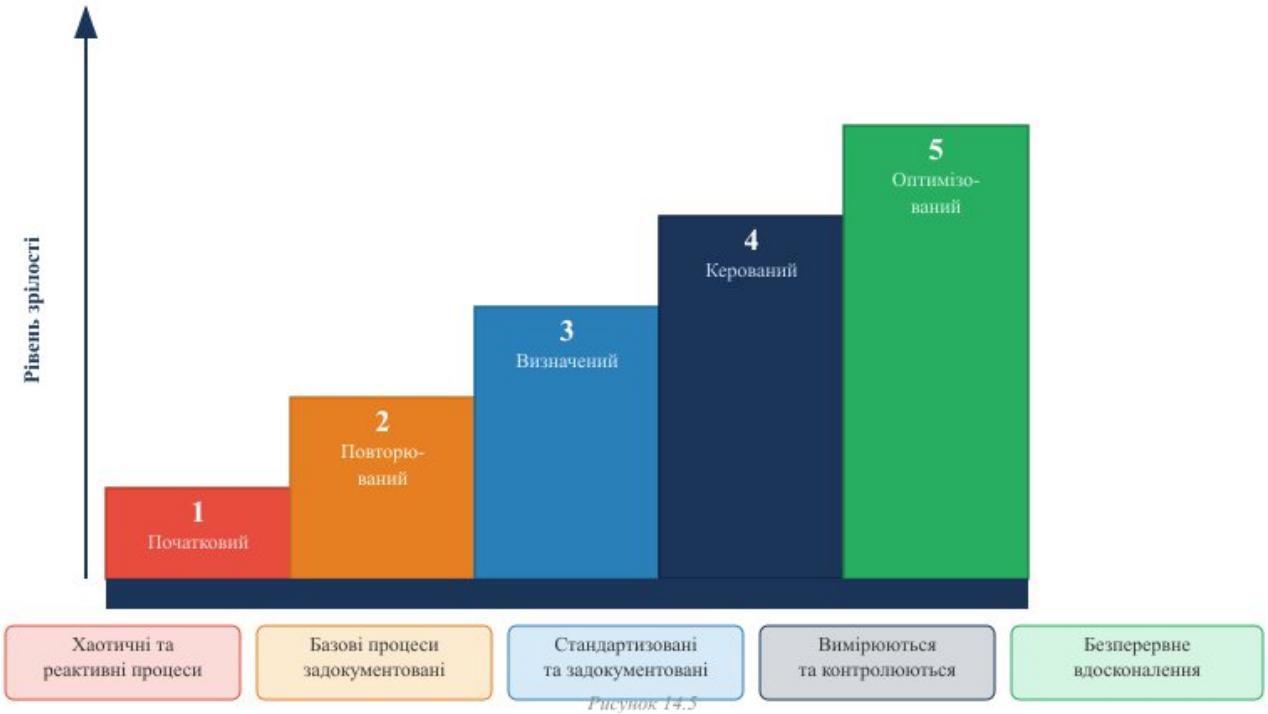

Рисунок 14.5

Рисунок 14.5 — Модель зрілості системи захисту: п'ять рівнів від хаотичних процесів до оптимізованої СУІБ

Важливим практичним аспектом є інтеграція вимог безпеки в процеси розробки програмного забезпечення. Концепція Security by Design передбачає, що питання безпеки розглядаються не як доповнення до готового продукту, а як невід'ємна частина процесу проектування від самого початку. Практичним втіленням є підхід DevSecOps, що інтегрує заходи безпеки в конвеєр безперервної інтеграції та доставки (CI/CD): статичний аналіз коду (наприклад, за допомогою SonarQube), перевірку вразливостей у залежностях (OWASP Dependency-Check), сканування образів контейнерів (Trivy). Такий підхід дозволяє виявляти та усувати вразливості на ранніх етапах розробки, коли їх виправлення є значно менш коштовним, ніж після розгортання у виробниче середовище.

Підсумовуючи розглянутий матеріал, можна сформулювати ключові принципи комплексного підходу до захисту програм і даних. По-перше, безпека є безперервним процесом, а не проектом із чітко визначеним завершенням: загрозове середовище постійно еволюціонує, і система захисту повинна адаптуватися до цієї динаміки. По-друге, відповідність стандартам є не самоціллю, а засобом систематизації та верифікації заходів безпеки, тому формальне виконання вимог без розуміння їх сенсу не гарантує реальної захищеності. По-третє, технічні засоби захисту ефективні лише тоді, коли вони спираються на відповідні організаційні процеси й компетентний персонал. По-четверте, комплексна система захисту вимагає балансу між усіма рівнями: надмірна концентрація на одному рівні при ігноруванні інших є типовою помилкою, що залишає значні вектори атаки незахищеними.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Горбенко В. І., Лісняк А. О. Безпека програм та даних : навчальний посібник для здобувачів ступеня вищої освіти бакалавра спеціальності 121 «Інженерія програмного забезпечення». — Запоріжжя : ЗНУ, 2022. — 72 с.
2. Сенів М. М., Яковина В. С. Безпека програм та даних : навчальний посібник. — Львів : Львівська політехніка, 2015. — 256 с.
3. Дем'яненко В. А., Кузнецова Ю. А. Безпека програм та даних [Електронний ресурс] : навчальний посібник до лабораторного практикуму. — Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «ХАІ», 2021. — 95 с.
4. Євсеєв С. П., Мілов О. В., Король О. Г. Лабораторний практикум з основ криптографічного захисту [Електронний ресурс] : навчальний посібник. — Харків : ХНЕУ ім. С. Кузнеця, 2020. — 222 с.
5. Технології захисту інформації в інформаційно-телекомунікаційних системах : навчальний посібник / А. В. Жилін, О. М. Шаповал, О. А. Успенський ; ІСЗЗІ КПІ ім. Ігоря Сікорського. — Київ : Вид-во «Політехніка» КПІ ім. Ігоря Сікорського, 2021. — 213 с.
6. Кузнецов О. О. Захист інформації в інформаційних системах : навчальний посібник. — Харків : ХНЕУ, 2018. — 510 с.
7. Мамченко С. М., Козюра В. Д., Бровко В. Д. Комплексні системи захисту інформації : навчальний посібник. — Київ : Національна академія СБУ, 2018. — 372 с.
8. Остапов С. Є., Євсеєв С. П., Король О. Г. Технології захисту інформації. — Чернівці : Видавничий дім «Родовід», 2014. — 428 с.
9. Згуровський М. Проблеми інформаційної безпеки в Україні, шляхи їх вирішення // Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні. — Київ, 2018. — С. 10–14.