Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи забезпечення безпеки в операційних системах з відкритим кодом

(тема)

Виконав:

студент II курсу, групи КСМм-23-1

Цуканов Є.Є.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Комп’ютерні системи та мережі
(повна назва освітньої програми)

Керівник: доц. Сорокін А.Р.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерні системи та мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Цуканову Єгору Євгеновичу
(прізвище, ім’я, по батькові)

1. Тема роботи Методи забезпечення безпеки в операційних системах з відкритим кодом

затверджена наказом по університету від “ 22 ” листопада 2024 р. № 1237 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 20 січня 2025 р.

3. Вхідні дані до роботи 1) Тема роботи; 2) Операційна система Linux;
3) Методи контролю доступу

4. Перелік питань, що потрібно опрацювати у роботі 1) Аналіз предметної області
2) Вибір і обгрунтування програмних засобів
3) Побудова системи забезпечення безпеки та контролю доступу
4) Тестування оптимізованого методу
5) Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 14 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	26.11.24-01.12.24	
2	Вибір технології розробки та	02.12.24-05.12.24	
	інструментальних засобів		
3	Розробка алгоритмічного забезпечення	06.12.24-20.12.24	
4	Розробка програмних модулів	21.12.24-30.12.24	
5	Відлагодження програмних модулів	31.12.24-07.01.25	
6	Оформлення матеріалів кваліфікаційної роботи	08.01.25-15.01.25	
7	Подання кваліфікаційної роботи керівникові та	16.01.25-19.01.25	
	її попередній захист		
8	Подання кваліфікаційної роботи на	20.01.25	

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Сорокін А.Р.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 92 с., 11 рис., 6 табл., 1 дод., 35 джерел.

ОПЕРАЦІЙНІ СИСТЕМИ, ВІДКРИТИЙ КОД, БЕЗПЕКА, LINUX, КОНТРОЛЬ ДОСТУПУ, ОПТИМІЗАЦІЯ.

Метою цієї роботи є дослідження методів забезпечення безпеки в операційних системах з відкритим кодом, оцінка їх ефективності та пропозиція оптимізації одного з методів для підвищення рівня захисту. Для досягнення цієї мети в роботі буде здійснено огляд існуючих методів захисту, вивчено основні підходи до управління безпекою, проведено оптимізацію методу контролю доступу та представлено результати тестування реалізованих покращень.

У цій роботі розглядаються методи забезпечення безпеки в операційних системах з відкритим кодом, зокрема в Linux, а також запропоновано оптимізацію одного з методів для покращення захисту. У першому розділі представлено огляд основних понять безпеки та механізмів захисту в операційних системах із відкритим кодом. У другому розділі аналізуються сучасні методи забезпечення безпеки, включаючи контроль доступу, криптографічний захист, системи виявлення вторгнень та методи збереження цілісності даних. У третьому розділі пропонується оптимізація методу контролю доступу, оцінюється його ефективність, і розробляються рекомендації для підвищення захищеності системи. Робота пропонує новий підхід до управління безпекою в ОС з відкритим кодом та забезпечує практичні рекомендації для подальшого розвитку систем захисту на основі відкритого коду.

ABSTRACT

Master’s thesis: 92 pages, 11 figures, 6 tables, 1 appendices, 35 sources.

OPERATING SYSTEMS, OPEN SOURCE, SECURITY, LINUX, ACCESS CONTROL, OPTIMIZATION.

The purpose of this work is to study security methods in open source operating systems, evaluate their effectiveness, and propose optimization of one of the methods to increase the level of protection. To achieve this goal, the work will review existing security methods, study the main approaches to security management, optimize the access control method, and present the results of testing the implemented improvements.

This work considers security methods in open source operating systems, in particular in Linux, and also proposes optimization of one of the methods to improve protection. The first section presents an overview of the basic security concepts and protection mechanisms in open source operating systems. The second section analyzes modern security methods, including access control, cryptographic protection, intrusion detection systems, and methods for preserving data integrity. The third section proposes optimization of the access control method, evaluates its effectiveness, and develops recommendations for increasing system security. The work offers a new approach to security management in open source OSs and provides practical recommendations for the further development of open source-based security systems.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційна система

AES – вдосконалений стандарт шифрування (англ., Advanced Encryption Standard)

AIDE – розширене середовище виявлення вторгнень (англ., Advanced Intrusion Detection Environment)

APT – розширений інструмент пакетування (англ., Advanced Package Tool)

DAC – дискреційний контроль доступу (англ., Discretionary Access Control)

IDS – система виявлення вторгнень (англ., Intrusion Detection System)

IPS – система запобігання вторгненням (англ., Intrusion Prevention System)

LTS – довгострокова підтримка (англ., Long-Term Support)

LUKS – налаштування уніфікованого ключа Linux (англ., Linux Unified Key Setup)

MAC – обов'язковий контроль доступу (англ., Mandatory Access Control)

OSSEC – безпека відкритого коду (англ., Open Source Security)

RBAC – контроль доступу на основі ролей (англ., Role-Based Access Control)

SELinux – Linux із підвищеною безпекою (англ., Security-Enhanced Linux)

SHA-2 – алгоритм безпечного хешування 2 (англ., Secure Hash Algorithm 2)

ВСТУП

З розвитком інформаційних технологій і постійним зростанням кількості даних, які обробляються комп’ютерними системами, питання безпеки інформації набуває все більшої актуальності. Сучасні операційні системи (ОС) використовуються в різноманітних галузях – від корпоративних серверів до пристроїв інтернету речей (IoT) та особистих комп'ютерів. Операційні системи з відкритим кодом, такі як Linux, мають широке поширення завдяки їх гнучкості, можливостям налаштування та підтримці з боку глобальної спільноти розробників. Водночас відкритість коду робить такі системи вразливими до різних видів атак, що створює потребу в розробці ефективних методів забезпечення їхньої безпеки.

Основними перевагами операційних систем з відкритим кодом є доступність коду, висока адаптивність, можливість модифікації та підтримка численних розширень. Проте відкритість коду також відкриває певні ризики, адже зловмисники можуть детально вивчати структуру та механізми ОС, виявляти потенційні слабкі місця і використовувати їх у своїх інтересах. Оскільки безпека операційних систем є одним із ключових факторів у забезпеченні цілісності даних і захисті від несанкціонованого доступу, для ОС з відкритим кодом розробляються численні механізми захисту, такі як криптографія, системи контролю доступу, виявлення вторгнень, забезпечення цілісності даних та резервне копіювання.

Метою цієї роботи є дослідження методів забезпечення безпеки в операційних системах з відкритим кодом, оцінка їх ефективності та пропозиція оптимізації одного з методів для підвищення рівня захисту. Для досягнення цієї мети в роботі буде здійснено огляд існуючих методів захисту, вивчено основні підходи до управління безпекою, проведено оптимізацію методу контролю доступу та представлено результати тестування реалізованих покращень.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ В ОПЕРАЦІЙНИХ СИСТЕМАХ З ВІДКРИТИМ КОДОМ

1.1 Основи безпеки в операційних системах з відкритим кодом

Операційні системи з відкритим кодом (Open Source Operating Systems) відрізняються від закритих рішень тим, що їхній вихідний код доступний для перегляду, модифікації та розповсюдження. Відкритий код дозволяє широкому колу дослідників, розробників та користувачів брати участь у його вдосконаленні, а також забезпечує прозорість та можливість аудиту, що є важливими аспектами для безпеки. Безпека в таких системах базується на принципах прозорості, спільного контролю, постійного вдосконалення та адаптивності до сучасних загроз.

Однією з основних переваг відкритих операційних систем є прозорість вихідного коду. Це означає, що будь-хто може переглянути код, знайти потенційні вразливості та запропонувати виправлення. Цей принцип забезпечує:

Можливість незалежного аудиту – дослідники та компанії можуть перевіряти код на наявність помилок, закладок або інших потенційно небезпечних компонентів.

Швидке виявлення та усунення вразливостей – спільнота розробників і користувачів оперативно знаходить та виправляє проблеми, що робить систему більш надійною.

Високий рівень довіри – користувачі та організації більше довіряють системам, де є можливість перевірити та оцінити безпекові аспекти коду. [1]

Спільнота є важливим компонентом у розвитку та підтримці безпеки ОС з відкритим кодом. Ця модель, де багато фахівців можуть одночасно перевіряти код і робити внески, забезпечує:

Колективний підхід до виявлення проблем: багато очей можуть знайти більше вразливостей і запропонувати різні підходи до їх виправлення.

Підтримку вразливих компонентів: різні версії ОС, що більше не підтримуються комерційно, можуть продовжувати вдосконалюватися та використовуватися завдяки підтримці спільноти.

Прикладом тут є проекти, як-от Linux Foundation, яка об’єднує тисячі розробників для покращення ОС Linux. [2]

Через постійний доступ до вихідного коду системи з відкритим кодом здатні швидко адаптуватися до нових загроз. Замість того щоб чекати на офіційне оновлення від постачальника, користувачі або незалежні команди можуть швидко вносити зміни. Це включає:

Можливість термінового патчування: відкритий код дозволяє швидко усувати вразливості, що знижує ризик потенційних атак.

Адаптація до нових вимог безпеки: додавання нових модулів або налаштувань, щоб відповідати актуальним вимогам безпеки.

Контроль доступу (Access Control): система дозволяє чітко контролювати, які дії можуть виконувати користувачі та процеси в системі. Основні моделі контролю доступу:

- Discretionary Access Control (DAC) – контроль доступу, де власник об’єкта має змогу самостійно налаштувати доступ до нього. Це базовий підхід, реалізований у більшості ОС;
- Mandatory Access Control (MAC) – суворіший контроль доступу, що визначає політики безпеки незалежно від користувачів. Прикладом є SELinux, що використовує MAC для захисту від несанкціонованого доступу;
- Role-Based Access Control (RBAC) – доступ контролюється на основі ролей, що є особливо корисним у багатокористувацьких системах.

Ізоляція процесів (Process Isolation): цей механізм запобігає взаємодії процесів, які не мають привілеїв, що значно зменшує ризик перехоплення даних чи управління. Linux реалізує ізоляцію процесів через:

- контейнери (наприклад, Docker), що дозволяють запускати додатки в ізольованому середовищі;

- Jails у FreeBSD – механізм, що дозволяє створювати ізольовані "в’язниці" для процесів із чітко визначеними привілеями;

- керування правами доступу (Privilege Management): системи з відкритим кодом дозволяють контролювати привілеї, щоб обмежити доступ до критичних частин системи. Наприклад, Linux використовує Linux Capabilities для управління привілеями окремих процесів;

- шифрування і збереження даних (Data Encryption and Storage Security): шифрування даних у файлових системах допомагає захистити інформацію навіть у разі фізичного доступу до диска. Популярні інструменти для шифрування включають LUKS (Linux Unified Key Setup) для шифрування дисків у Linux та ZFS encryption у OpenZFS для шифрування даних;

- системи моніторингу та журналювання (Monitoring and Logging Systems): у відкритих системах є багато інструментів для моніторингу активності, наприклад, Auditd для Linux, що дозволяє створювати журнали дій у системі. Моніторинг дозволяє швидко реагувати на потенційні загрози та аналізувати інциденти безпеки.

Гнучкість і адаптивність: користувачі можуть модифікувати та налаштовувати ОС відповідно до конкретних вимог. Наприклад, у корпоративному середовищі можливо налаштувати контроль доступу за допомогою SELinux для різних підрозділів.

Економічна ефективність: ОС з відкритим кодом зазвичай не потребують витрат на ліцензії, що є вигідним для організацій, а також дозволяє спрямовувати ресурси на інші заходи безпеки.

Співпраця і підтримка з боку спільноти: завдяки підтримці спільноти, користувачі можуть отримати допомогу з налаштуванням безпеки, а також обмінюватися знаннями про нові загрози та способи захисту.

Водночас, відкриті операційні системи мають певні виклики:

- відсутність офіційної підтримки: деякі ОС з відкритим кодом можуть не мати офіційної технічної підтримки, що змушує користувачів покладатися на спільноту;

- потенційна складність налаштування: для належної безпеки необхідні глибокі знання та досвід, що може створювати труднощі для звичайних користувачів. [3]

1.2 Основні безпекові механізми в Linux

Операційна система Linux пропонує різноманітні механізми безпеки, які спрямовані на захист від несанкціонованого доступу, забезпечення цілісності даних та підтримку безпечної роботи. Серед основних інструментів безпеки Linux – SELinux, AppArmor, Linux Capabilities, а також різноманітні системи шифрування та моніторингу. Кожен з цих механізмів має свої особливості та переваги. Ілюстрація архітектури Linux та її основні елементи безпеки показана на рисунку 1.1.

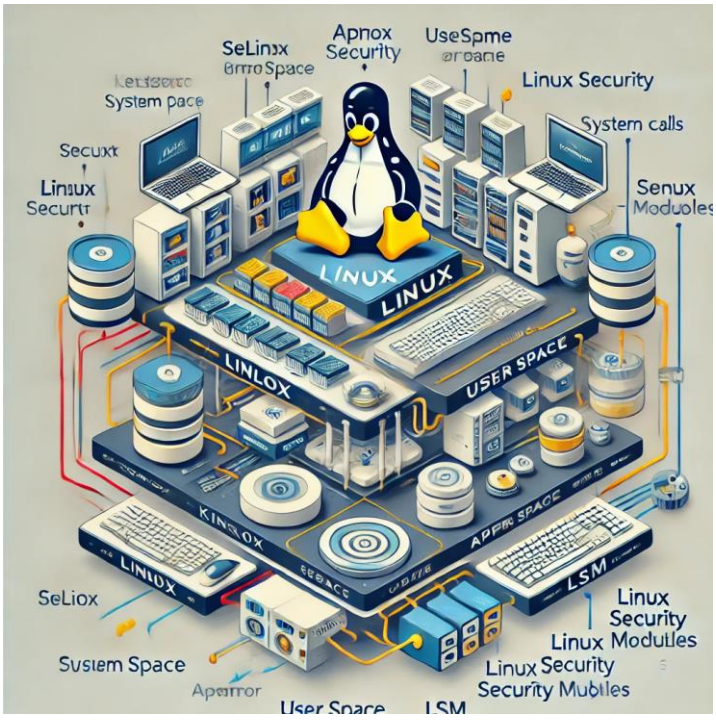

Рисунок 1.1 – Архітектура Linux та основні елементи безпеки

SELinux – це розширення ядра Linux, розроблене Агентством національної безпеки США, що реалізує систему політик безпеки на основі обов'язкового контролю доступу (Mandatory Access Control, MAC). Основною метою SELinux є захист системи шляхом обмеження привілеїв користувачів і процесів.

Політики безпеки: SELinux використовує політики безпеки, що визначають, до яких файлів, каталогів і процесів має доступ кожен користувач чи процес. Політики можуть бути строгими (strict policy) або гнучкими (targeted policy).

Режими роботи SELinux:

- Enforcing – SELinux активно застосовує політики безпеки та блокує всі дії, що порушують ці політики;
- Permissive – SELinux лише реєструє порушення політик без обмеження доступу, що дозволяє адміністраторам виявляти потенційні проблеми;
- Disabled – SELinux повністю вимкнено, і політики безпеки не діють.

Переваги та недоліки:

- переваги: детальний контроль доступу, надійний захист від ескалації привілеїв і забезпечення ізоляції процесів;
- недоліки: складність налаштування, що вимагає глибоких знань, а також можливі проблеми сумісності з деякими додатками. [4]

AppArmor – це альтернативна до SELinux система контролю доступу, що надає можливість обмежувати дії окремих додатків, базуючись на моделі профілів.

Профілі безпеки: кожна програма, захищена AppArmor, має профіль, що визначає її доступ до ресурсів системи. Профілі можуть бути налаштовані як у вигляді дозволених дій, так і в вигляді обмежень для певного додатку.

Простота налаштування: AppArmor відомий своєю простотою у налаштуванні, оскільки використовує менше складних правил, ніж SELinux, і надає інструменти для автоматичного генерування профілів.

Режими роботи:

- Enforce mode – застосовуються обмеження відповідно до профілів безпеки;
- Complain mode – профілі працюють у режимі контролю, реєструючи потенційні порушення без обмеження дій.

Переваги та недоліки:

- переваги: простота налаштування, менше навантаження на систему;
- недоліки: менш детальний контроль порівняно з SELinux, обмежений захист на рівні додатків, а не системи в цілому.

Linux Capabilities дозволяють розподілити адміністративні права (що зазвичай надаються користувачу root) між окремими процесами або користувачами, тим самим зменшуючи ризик ескалації привілеїв і підвищуючи безпеку.

Основні можливості: у Linux права адміністратора можна розділити на менші capabilities (наприклад, CAP_NET_ADMIN для адміністрування мережі або CAP_SYS_MODULE для управління модулями ядра).

Контроль привілеїв процесів: за допомогою capabilities можна надати окремим процесам права, необхідні для виконання конкретних завдань, не надаючи їм повних привілеїв root.

Переваги та недоліки:

- переваги: зниження ризику компрометації системи через надання мінімально необхідних прав процесам;
- недоліки: складність управління, потреба в детальному налаштуванні привілеїв. [5]

Control Groups (cgroups) – це механізм, що дозволяє обмежувати ресурси, що споживаються групами процесів. Cgroups активно використовується для контейнеризації та ізоляції додатків у Linux.

Обмеження ресурсів: cgroups дозволяють контролювати використання ЦП, оперативної пам'яті, мережі та інших ресурсів для окремих процесів або груп процесів.

Ізоляція процесів: cgroups сприяють безпечному виконанню процесів, забезпечуючи контроль над їхньою взаємодією з іншими частинами системи.

Переваги та недоліки:

- переваги: можливість обмеження ресурсів для забезпечення стабільної роботи, підтримка контейнеризації;

- недоліки: складність конфігурації, потреба в глибокому розумінні системних ресурсів.

Linux пропонує засоби для ізоляції додатків за допомогою контейнеризації, що дозволяє запускати додатки в окремих середовищах без впливу на решту системи.

LXC (Linux Containers): забезпечують легку віртуалізацію на рівні операційної системи, дозволяючи ізолювати додатки в окремих контейнерах.

Docker: один із найпоширеніших інструментів контейнеризації, що дозволяє запускати ізольовані додатки з обмеженим доступом до ресурсів.

Kubernetes: система управління контейнерами, яка забезпечує масштабованість, управління ресурсами та оркестрацію контейнерів у великих середовищах.

Переваги та недоліки:

- переваги: ізоляція додатків, покращена безпека за рахунок обмеження доступу до системних ресурсів;

- недоліки: складність управління контейнерами в масштабних середовищах, потреба у підтримці.

Шифрування даних в Linux є важливим засобом захисту конфіденційності та цілісності інформації. Linux підтримує різні методи шифрування для захисту даних на рівні диска, файлової системи або окремих файлів.

LUKS (Linux Unified Key Setup): стандарт для шифрування дисків, який дозволяє захистити дані на фізичних носіях за допомогою симетричного шифрування.

EncFS та eCryptfs: інструменти для шифрування на рівні файлової системи, які дозволяють шифрувати окремі каталоги або файли.

dm-crypt: модуль шифрування на рівні блочних пристроїв, що інтегрується з LUKS для створення зашифрованих розділів.

Переваги та недоліки:

- переваги: захист від фізичного доступу до даних, зниження ризику крадіжки інформації;
- недоліки: можливі затримки у роботі системи, потреба в управлінні ключами.

Моніторинг системи та журналювання подій дозволяють адміністраторам відстежувати активність, виявляти підозрілі дії та аналізувати інциденти.

Auditd: служба журналювання подій безпеки, що дозволяє фіксувати всі важливі дії в системі, включаючи доступ до файлів, зміну привілеїв та інші важливі події.

Syslog: система журналювання, що збирає й зберігає різноманітну інформацію про роботу сервісів і додатків.

Fail2Ban: інструмент, що аналізує журнали на предмет підозрілих дій (наприклад, неуспішні спроби входу) і автоматично блокує IP-адреси, з яких здійснюються підозрілі запити.

Переваги та недоліки:

- переваги: покращене реагування на інциденти безпеки, можливість виявлення аномальної поведінки;
- недоліки: необхідність налаштування та постійного моніторингу, велика кількість даних.

Оновлення та патчі безпеки відіграють важливу роль у підтримці захищеності операційної системи, оскільки вони закривають вразливості, виправляють помилки в програмному коді та забезпечують стійкість до нових загроз. Своєчасне управління оновленнями є критичним аспектом для захисту даних і зниження ризику експлуатації вразливостей. У Linux цей

процес має свої особливості, пов'язані з відкритим вихідним кодом системи та великою кількістю дистрибутивів. [6]

1.3 Управління оновленнями та патчами безпеки

У Linux існує низка підходів і інструментів для управління оновленнями, які допомагають адміністраторам систем і користувачам забезпечувати захист своїх комп’ютерів та мереж. Ключовими аспектами цього процесу є управління репозиторіями, автоматизація оновлень та підтримка політики оновлення.

1.3.1 Типи оновлень у Linux

Оновлення в Linux можна розділити на кілька категорій:

Патчі безпеки – усувають вразливості в системі. Ці патчі є найважливішими, оскільки дозволяють оперативно закривати відомі вразливості до того, як вони будуть експлуатовані зловмисниками.

Оновлення програмного забезпечення – додають нові функції або покращують продуктивність існуючих програм, але можуть також включати виправлення безпеки.

Оновлення ядра – оновлюють ядро Linux, яке відповідає за управління системними ресурсами та взаємодію з апаратним забезпеченням. Оновлення ядра часто включають виправлення безпеки та оптимізації.

Патчі стабільності – спрямовані на виправлення багів і поліпшення стабільності роботи системи. [7]

1.3.2 Управління репозиторіями та джерелами оновлень

У Linux велике значення мають репозиторії – центральні сховища програмного забезпечення, з яких користувачі та адміністратори можуть

завантажувати оновлення. Репозиторії зазвичай контролюються та підтримуються розробниками дистрибутиву, що забезпечує їхню безпеку та актуальність.

Офіційні репозиторії: забезпечують оновлення, що перевірені та схвалені для конкретного дистрибутиву. Це найбільш надійне джерело, оскільки кожне оновлення проходить контроль якості та тестування.

Репозиторії сторонніх розробників: можуть включати пакети, які недоступні в офіційних репозиторіях, проте вони можуть містити потенційно небезпечний код. Адміністратори повинні ретельно перевіряти джерела перед використанням сторонніх репозиторіїв.

1.3.3 Інструменти для управління оновленнями

У різних дистрибутивах Linux використовуються різні інструменти для управління оновленнями та патчами безпеки. Деякі з основних інструментів включають:

APT (Advanced Package Tool): використовується в дистрибутивах, базованих на Debian (Ubuntu, Linux Mint). APT дозволяє легко оновлювати пакети, встановлювати нові версії програм та видаляти застарілі пакети.

Основні команди: apt update для оновлення списку доступних пакетів, apt upgrade для встановлення оновлень та apt full-upgrade для оновлення з видаленням застарілих пакетів, якщо потрібно.

YUM/DNF: використовується в дистрибутивах на основі RPM (Fedora, CentOS). YUM дозволяє здійснювати пошук, встановлення, видалення та оновлення програмного забезпечення.

DNF – це вдосконалена версія YUM, яка покращує роботу з залежностями та продуктивність. Команди dnf update або yum update дозволяють оновити пакети та безпекові патчі.

Zypper використовується в дистрибутивах openSUSE та SUSE Linux Enterprise, пропонуючи можливості для зручного управління пакетами та репозиторіями. Основна команда для оновлення – zypper update.

Pacman – це менеджер пакетів Arch Linux, що дозволяє швидко оновлювати програмне забезпечення. Команда pacman -Syu оновлює всі пакети до найновіших версій. [8]

1.3.4 Автоматизація оновлень

Автоматичне встановлення оновлень – це ефективний спосіб забезпечення безпеки, особливо в великих інфраструктурах, де кількість систем і додатків може бути значною. Автоматизація дозволяє знизити людський фактор та зменшити ймовірність пропущення критичних оновлень.

Unattended-upgrades: у дистрибутивах на основі Debian є утиліта, що дозволяє автоматично встановлювати важливі оновлення без втручання користувача. Вона особливо корисна для серверів, де важливо звести до мінімуму час простою через оновлення.

DNF-Automatic: цей інструмент, який підтримується в дистрибутивах на основі Fedora, дозволяє автоматично застосовувати оновлення для забезпечення безпеки.

Landscape – це сервіс управління оновленнями від Ubuntu, що пропонує можливість централізованого керування кількома системами. Він дозволяє системним адміністраторам автоматизувати оновлення для масштабних інфраструктур, забезпечуючи своєчасне встановлення патчів.

1.3.5 Політика оновлень і контроль версій

Управління оновленнями в Linux передбачає також встановлення певної політики оновлення, яка регулює періодичність і типи встановлюваних оновлень. Основними підходами є:

- стабільні оновлення: фокусуються на встановленні лише перевірених оновлень, які не впливають на стабільність системи. Це підхід, який часто використовується на серверах для мінімізації ризиків;
- оновлення до найновіших версій: цей підхід дозволяє отримувати останні версії програмного забезпечення, що включають нові функції та патчі безпеки. Використовується в середовищах, де важлива актуальність функціоналу, наприклад, в розробці;
- LTS-версії (Long-Term Support): деякі дистрибутиви, такі як Ubuntu, пропонують підтримку версій із довготривалим обслуговуванням, де оновлення безпеки забезпечуються протягом кількох років. Це популярний вибір для серверних систем, де стабільність є ключовим пріоритетом.

1.3.6 Моніторинг та відстеження оновлень безпеки

Підтримка актуальності системи також включає моніторинг оновлень та відстеження вразливостей у встановлених програмах. Інструменти для цього включають:

Canonical Livepatch: сервіс від Canonical для оновлення ядра Ubuntu в реальному часі без необхідності перезавантаження. Це дозволяє знижувати час простою серверів.

Security Advisories: більшість основних дистрибутивів Linux (наприклад, Red Hat, Debian) випускають повідомлення про безпекові оновлення та рекомендації щодо патчів. Це дозволяє адміністраторам швидко дізнаватися про критичні оновлення.

Моніторинг вразливостей: засоби, такі як OSSEC і OpenSCAP, допомагають сканувати систему на предмет наявності вразливостей та перевіряти відповідність політикам безпеки. Таким чином, управління оновленнями та патчами в Linux є комплексним процесом, який передбачає вибір відповідних репозиторіїв, автоматизацію встановлення оновлень та моніторинг критичних виправлень. [9]

2 ОГЛЯД МЕТОДІВ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ В ОПЕРАЦІЙНИХ СИСТЕМАХ З ВІДКРИТИМ КОДОМ

2.1 Класифікація методів безпеки

Методи забезпечення безпеки в операційних системах з відкритим кодом, зокрема Linux, можна розділити на кілька категорій за їхньою метою та функціональними можливостями. Основні групи методів включають превентивні, детекційні та реакційні механізми. Ця класифікація дозволяє адміністраторам вибудувати багаторівневу стратегію захисту, що покриває всі аспекти безпеки від запобігання атакам до виявлення та реагування на інциденти.

2.1.1 Превентивні методи безпеки

Превентивні методи спрямовані на запобігання загрозам до їх виникнення. Вони створюють захисний бар'єр, що ускладнює або робить неможливим доступ зловмисника до критично важливих компонентів системи. Основні превентивні методи включають:

Контроль доступу: обмежує доступ користувачів до ресурсів системи, таких як файли, мережеві служби та пристрої. В Linux використовується декілька моделей управління доступом:

DAC (Discretionary Access Control) – кожен об’єкт (файл або папка) має власника, який визначає доступ до нього. Це базова модель безпеки, яка використовується в більшості Linux-систем.

MAC (Mandatory Access Control) – правила контролю доступу задаються централізовано та не можуть бути змінені користувачами. Наприклад, SELinux та AppArmor є реалізаціями MAC для Linux, які дозволяють обмежувати доступ до системних файлів та процесів.

Брандмауери: забезпечують захист на рівні мережі, фільтруючи вхідні та вихідні з'єднання. Linux має потужний брандмауер iptables (або nftables у новіших версіях), який дозволяє адміністраторам визначати правила для блокування або дозволу трафіку на основі IP-адрес, портів та інших параметрів. Для зручнішого управління брандмауером використовуються такі інструменти, як ufw (Uncomplicated Firewall), що особливо популярний в Ubuntu.

Антивірусні та антишпигунські інструменти: хоча Linux вважається менш вразливим до вірусів, існують рішення для захисту від шкідливого ПЗ, зокрема ClamAV, який використовується для сканування системи на наявність шкідливих програм. [10]

2.1.2 Детекційні методи безпеки

Детекційні методи використовуються для виявлення загроз у реальному часі або з невеликою затримкою. Ці методи дозволяють адміністраторам отримувати інформацію про підозрілі дії та швидко реагувати на них. Системи виявлення вторгнень (IDS): спеціалізовані інструменти, що відслідковують мережевий трафік або системні події для виявлення ознак вторгнення. Linux підтримує такі IDS, як Snort (мережевий IDS), який аналізує мережевий трафік на предмет підозрілих активностей, та AIDE (Host-based IDS), який моніторить зміни в системних файлах.

Моніторинг журналів та подій: журналювання системних подій дозволяє адміністраторам зберігати інформацію про дії в системі. У Linux для цього використовується syslog або journald, а також спеціалізовані інструменти, такі як Auditd для аудиту дій користувачів та подій безпеки.

Виявлення аномальної поведінки: аналіз аномалій у поведінці системи або користувачів за допомогою спеціалізованого ПЗ, як-от OSSEC, яке дозволяє виявляти підозрілу активність на основі попередньо визначених шаблонів. [11]

Таблиця 2.1 – Класифікація методів безпеки

Рівень безпеки	Превентивні механізми	Детекційні механізми	Реакційні механізми
Фізична безпека	- Контроль доступу (замки, картки доступу, біометрія) - Обмеження фізичного доступу - Протипожежне обладнання	- Камери відеоспостереження - Датчики руху - Сигналізація	- Активізація сигналізації - Виклик охорони або служб безпеки - Локалізація небезпечних зон
Логічна безпека	- Сильні паролі - Двофакторна аутентифікація - SELinux, AppArmor	- Журнали доступу (auditd, syslog) - Моніторинг підозрілої активності	- Блокування облікового запису при виявленні підозрілої активності - Зміна паролів
Безпека даних	- Шифрування даних (AES, RSA) - Резервне копіювання - Обмеження доступу до файлів	- Контроль хешів файлів - Моніторинг доступу до файлів (inotify)	- Відновлення даних із резервної копії - Оновлення ключів шифрування
Мережева безпека	- Брандмауери (iptables, ufw) - VPN - Використання TLS/SSL	- Виявлення атак (Snort, Suricata) - Логи мережевої активності	- Блокування підозрілих IP-адрес - Ізоляція скомпрометованих сегментів мережі
Програмна безпека	- Аналіз коду перед впровадженням - Регулярне оновлення ПЗ - Використання надійних бібліотек	- Виявлення вразливостей (OpenVAS, Nessus) - Моніторинг аномальної поведінки програм	- Патчинг виявлених вразливостей - Оновлення або видалення скомпрометованого ПЗ

2.1.3 Реакційні методи безпеки

Реакційні методи спрямовані на зниження шкоди, завданої атакою, та зменшення ризику подальшого проникнення. Це дозволяє швидко

локалізувати проблему і мінімізувати наслідки інциденту. До таких методів належать:

Автоматичне блокування підозрілих IP-адрес: інструменти, як-от Fail2Ban, сканують журнали та виявляють підозрілу активність, як-от багаторазові невдалі спроби входу. При виявленні підозрілих дій Fail2Ban автоматично блокує IP-адресу, з якої надходили атаки, на певний період.

Системи управління інцидентами: включають засоби для координації дій після інциденту, документування процесу та зберігання інформації для подальшого аналізу. OSSEC – популярний засіб для управління інцидентами, що дозволяє фіксувати та реагувати на різні події в реальному часі.

Резервне копіювання та відновлення: регулярне створення резервних копій є одним з найефективніших способів забезпечення стійкості до атак. Rsync і Duplicity – це інструменти, які дозволяють налаштувати автоматизоване резервне копіювання важливих даних. Також рекомендується зберігати копії в ізольованих середовищах або у хмарних сховищах для захисту від шкідливого ПЗ. [12]

2.2 Методи контролю доступу

Методи контролю доступу є важливими інструментами захисту операційних систем, оскільки вони обмежують можливість небажаного доступу до системних ресурсів, файлів і додатків. У відкритих операційних системах, таких як Linux, існує кілька ключових моделей контролю доступу: дискреційне управління доступом (DAC), мандатне управління доступом (MAC) і рольове управління доступом (RBAC). Розглянемо їх у деталях, а також специфічні інструменти для їх реалізації, як-от SELinux і AppArmor.

Дискреційне управління доступом (DAC) є однією з базових моделей контролю доступу, яка використовується за замовчуванням у багатьох операційних системах, зокрема в Linux. DAC передбачає, що власник кожного об’єкта (файлу, папки чи процесу) може самостійно визначати права

доступу до нього. Ця модель є досить гнучкою, однак може бути вразливою до певних типів атак, наприклад, до атаки шляхом компрометації прав користувача.

Приклад використання DAC у Linux: у Linux кожен файл або папка має власника, групу та дозволи доступу, які включають права читання, запису і виконання для власника, групи та інших користувачів. Наприклад, команда chmod дозволяє змінювати права доступу до файлів. Система також підтримує спеціальні дозволи, як-от setuid, setgid та sticky bit, які дають додатковий рівень контролю. [13]

Мандатне управління доступом (MAC) обмежує доступ користувачів до ресурсів на основі політик, які задаються адміністраторами системи і не можуть бути змінені користувачами. MAC є значно жорсткішим, ніж DAC, і широко використовується в середовищах, де необхідно запобігти будь-яким можливим компрометаціям доступу.

Інструменти MAC у Linux: SELinux (Security-Enhanced Linux): це система мандатного управління доступом, розроблена для надання високого рівня безпеки. SELinux використовує концепції політик та контекстів для обмеження доступу до файлів, процесів та мережевих ресурсів. Наприклад, SELinux дозволяє визначати, які процеси можуть взаємодіяти з певними файлами або службами. SELinux підтримує суворий і дозволяючий режими роботи.

AppArmor: це ще одна система MAC, яка дозволяє обмежувати доступ до ресурсів на основі профілів. Кожен процес у системі обмежений профілем, який визначає, які файли або ресурси може використовувати процес. AppArmor є менш складним у налаштуванні порівняно з SELinux і часто використовується в Ubuntu.

Приклад використання MAC: у SELinux файли та процеси мають різні "контексти", що дозволяє чітко визначати, які дії дозволені кожному об'єкту. Для налаштування SELinux потрібні політики безпеки, які можна редагувати та адаптувати відповідно до конкретних потреб системи.

Рольове управління доступом (RBAC) дозволяє призначати права доступу на основі ролей, що спрощує управління доступом у великих системах з великою кількістю користувачів. Кожна роль відповідає певним обов'язкам або функціям користувача, і привілеї визначаються на основі цієї ролі.

Приклад використання RBAC: у Linux рольове управління може бути реалізовано за допомогою груп користувачів. Наприклад, групі "адміністратори" можна надати доступ до конфігураційних файлів і системних ресурсів, тоді як група "користувачі" матиме обмежений доступ [14].

SELinux і AppArmor – це два провідні механізми контролю доступу на основі MAC, які допомагають забезпечити високий рівень безпеки системи. Їхнє налаштування та управління дозволами є відмінним інструментом для захисту як персональних, так і корпоративних систем.

SELinux: Використовує політики, які чітко визначають, які процеси можуть взаємодіяти з ресурсами системи. SELinux рекомендується використовувати на серверах або в корпоративних середовищах, де високий рівень безпеки є критично важливим. Наприклад, в CentOS SELinux є активованим за замовчуванням, і його політики можуть налаштовуватися для захисту певних серверних ролей, таких як веб-сервер або база даних.

AppArmor: Працює на основі профілів, які можуть обмежувати дії процесів, але без складних політик, характерних для SELinux. Наприклад, на Ubuntu за допомогою AppArmor можна створити профіль для певної програми, щоб обмежити її доступ до системних файлів або мережевих ресурсів.

Порівняння: SELinux надає більш складний, але гнучкий контроль доступу порівняно з AppArmor, однак він вимагає більшого розуміння принципів безпеки.

AppArmor є простішим для налаштування і зручнішим для користувачів початкового рівня, що робить його кращим вибором для менш критичних середовищ. [15]

Таблиця 2.2 – Порівняння методів контролю доступу

Критерій	DAC (Discretionary Access Control)	MAC (Mandatory Access Control)	RBAC (Role-Based Access Control)
1	2	3	4
Безпека	- Менш надійний: доступ визначається власником ресурсу - Вразливий до зловживань користувачів	- Висока безпека: доступ базується на політиках, визначених адміністратором - Захищає від несанкціонованого доступу	- Середній рівень безпеки: залежить від правильного налаштування ролей - Мінімізує людський фактор
Ефективність	- Простий у налаштуванні для окремих ресурсів - Складно масштабувати в великих системах	- Менш ефективний у великих системах через суворі політики - Вимагає значних ресурсів для управління	- Висока ефективність у великих організаціях - Спрощує управління доступом через ролі

Продовження таблиці 2.2

1	2	3	4
Простота реалізації	- Легкий для реалізації в невеликих системах - Не потребує складних політик	- Складний у реалізації через сувору структуру - Вимагає глибоких знань безпеки	- Відносно простий: достатньо визначити ролі та права доступу - Легко інтегрується з існуючими системами
Гнучкість	- Висока: власник ресурсу може вільно керувати доступом	- Низька: правила доступу суворо визначені адміністраторами - Неможливо змінювати політики без дозволу	- Середня: зміни в доступі здійснюються через модифікацію ролей
Масшта бованість	- Обмежена: складно керувати великою кількістю користувачів і ресурсів	- Низька: управління політиками може стати надмірно складним у великих мережах	- Висока: дозволяє легко масштабувати доступ через управління ролями

2.3 Криптографічні методи захисту

Криптографічні методи є невід’ємною частиною сучасних стратегій безпеки для операційних систем, включаючи ОС з відкритим кодом, як-от

Linux. Використання криптографії дозволяє забезпечити конфіденційність, цілісність і аутентифікацію даних, що особливо важливо в умовах зростання обсягу обміну інформацією та необхідності захисту чутливих даних. Основними криптографічними методами захисту є шифрування (рисунок 2.1), хешування і цифровий підпис.

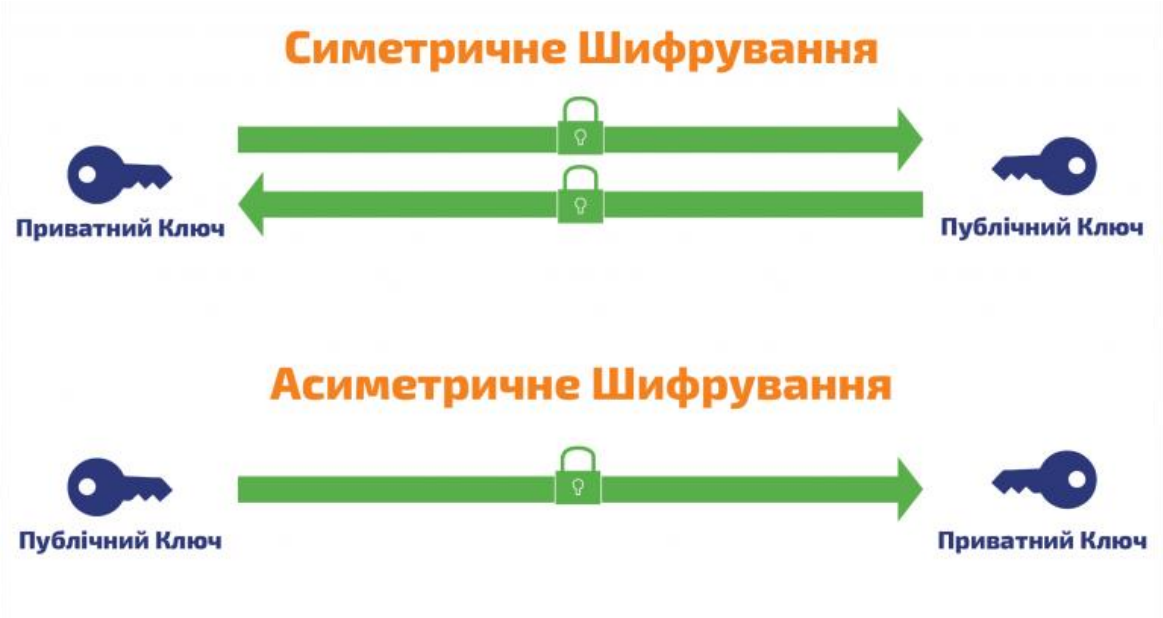

Рисунок 2.1 – Схема симетричного та асиметричного шифрування

Шифрування є одним із найважливіших методів захисту даних. Воно дозволяє перетворити дані у форму, що не може бути прочитана без відповідного ключа розшифрування. Це захищає інформацію від несанкціонованого доступу, навіть якщо зловмисник отримав фізичний доступ до пристрою. Схема симетричного та асиметричного шифрування показана на Рисунку 2.1.

Типи шифрування: симетричне шифрування: Використовує один і той самий ключ для шифрування і розшифрування даних. Воно забезпечує високу швидкість обробки даних, що підходить для великих обсягів інформації, однак має проблему із захистом ключа, оскільки його необхідно передавати між учасниками. Прикладом є AES (Advanced Encryption

Standard), який широко використовується в Linux для захисту файлів та дисків.

Асиметричне шифрування: Використовує пару ключів – публічний та приватний. Публічний ключ використовується для шифрування, тоді як приватний — для розшифрування. Це забезпечує безпечний обмін ключами і часто використовується для захисту мережевих з'єднань. Прикладом є RSA та ECDSA, які застосовуються в SSH-з'єднаннях для аутентифікації користувачів.

Приклад використання шифрування у Linux: LUKS (Linux Unified Key Setup): Це стандарт для шифрування дисків у Linux, який дозволяє шифрувати весь диск або окремі розділи, захищаючи таким чином дані від несанкціонованого доступу. LUKS використовує AES як стандартний алгоритм шифрування.

OpenSSL: Бібліотека, яка дозволяє шифрувати файли і з'єднання. OpenSSL підтримує як симетричні, так і асиметричні алгоритми, включаючи AES, RSA, та ECDSA. Вона також застосовується для захисту інтернет-з'єднань, наприклад, за допомогою протоколу TLS. [16]

Хешування є незамінним інструментом для забезпечення цілісності даних. Хеш-функція перетворює будь-яке повідомлення в унікальну зашифровану послідовність фіксованої довжини, яка не може бути розшифрована до оригінальних даних. Це дозволяє переконатися, що дані не були змінені під час передачі або зберігання.

Алгоритми хешування: SHA-2 (Secure Hash Algorithm 2) та SHA-3: Ці алгоритми є стандартами для хешування, які забезпечують високу стійкість до колізій та поширені у багатьох системах для зберігання паролів та перевірки цілісності файлів.

MD5: Старіший алгоритм, який використовується для перевірки цілісності, але більше не вважається надійним для безпекових цілей через вразливість до колізій.

Приклад використання хешування у Linux:

Tripwire: Інструмент, який використовує хешування для перевірки цілісності системних файлів. Він створює базу даних з хешів усіх важливих файлів і періодично перевіряє їх на відповідність. Якщо хеш не відповідає, це свідчить про потенційне втручання.

SHA-суми: Команди, як-от sha256sum, sha512sum у Linux, дозволяють користувачам створювати хеш-суми для файлів. Це корисно для перевірки цілісності файлів після їх завантаження або передачі. [17]

Цифровий підпис – це механізм, який дозволяє підтвердити справжність даних або повідомлення, використовуючи асиметричне шифрування. Він дозволяє одержувачу переконатися, що дані були відправлені певним відправником і не були змінені.

Механізм цифрового підпису: відправник створює хеш повідомлення і шифрує його своїм приватним ключем, що утворює цифровий підпис. Одержувач може розшифрувати підпис, використовуючи публічний ключ відправника, і порівняти його з хешем отриманого повідомлення.

Приклад використання цифрових підписів у Linux: GPG (GNU Privacy Guard): GPG дозволяє створювати цифрові підписи для файлів та електронних повідомлень. Він забезпечує аутентифікацію і захист від змін, що важливо для перевірки програмного забезпечення.

SSH-ключі: Під час підключення до віддаленого сервера через SSH, сервер використовує цифровий підпис для підтвердження своєї автентичності, що знижує ризик атак "людина посередині" (MitM). [18]

2.4 Системи виявлення та запобігання вторгненням (IDS та IPS)

Системи виявлення та запобігання вторгненням (IDS та IPS) є критично важливими компонентами безпеки для операційних систем, зокрема тих, що працюють на базі відкритого коду. IDS та IPS дозволяють виявляти, реагувати та запобігати спробам атак, забезпечуючи додатковий рівень захисту, особливо від внутрішніх загроз або складних мережевих атак.

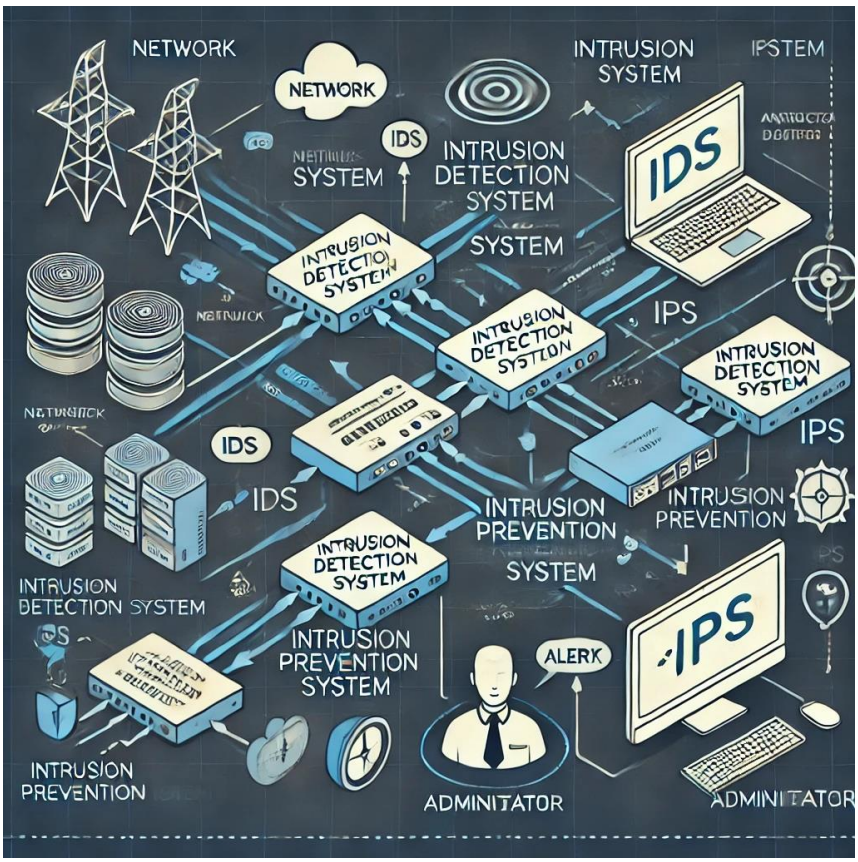

Рисунок 2.2 – Архітектура системи виявлення вторгнень

IDS (Intrusion Detection System) – система, яка лише виявляє спроби вторгнення або аномальну активність, але не запобігає їм. IDS зазвичай налаштовують для моніторингу трафіку та повідомлення адміністратора у разі підозрілих дій.

IPS (Intrusion Prevention System) – система, яка не лише виявляє, але й активно блокує підозрілу активність. Вона інтегрується з мережевим трафіком або вбудовується у систему, щоб запобігти атакам в режимі реального часу.

Вибір між IDS та IPS залежить від потреб та середовища, у якому використовується операційна система:

- IDS ідеально підходить для моніторингу з мінімальним втручанням, особливо у великих організаціях, де важливо фіксувати підозрілу активність;
- IPS доцільний у середовищах із високими вимогами до захисту,

наприклад, у фінансових установах або критичних інфраструктурах, де запобігання атакам має пріоритет. [19]

Системи IDS та IPS можна класифікувати на основі їхнього функціоналу та принципів роботи:

Сигнатурні системи (Signature-Based IDS/IPS)

Працюють на основі бази даних відомих атак або шаблонів підозрілої поведінки.

Ефективні проти відомих атак, але вразливі до нових загроз або модифікованих атак, для яких немає сигнатур.

Приклад: Snort – відкрите програмне забезпечення для мережевого моніторингу та аналізу трафіку. Snort може використовуватися як IDS для виявлення підозрілих дій, таких як атаки на вебсервери або спроби сканування портів. [20]

Аномальні системи (Anomaly-Based IDS/IPS)

Виявляють аномальну поведінку, порівнюючи трафік або активність із попередньо визначеним "нормальним" шаблоном. Відхилення від нормальної активності можуть розглядатися як потенційні загрози.

Ефективні проти невідомих атак, оскільки вони не залежать від сигнатур. Проте аномальні системи можуть генерувати велику кількість хибнопозитивних спрацьовувань.

Приклад: OSSEC (Open Source Security) – це HIDS (Host-based IDS), який використовує аналіз поведінки для виявлення аномалій на основі дій користувача або процесів у системі.

Гібридні системи поєднують підходи сигнатурного та аномального виявлення для підвищення ефективності виявлення атак.

Зазвичай забезпечують ширше покриття і вищу точність виявлення вторгнень, хоча можуть потребувати більше ресурсів для роботи.

Приклад: Suricata – відкритий IDS/IPS, який підтримує як сигнатурне, так і аномальне виявлення, що дозволяє знизити кількість хибних спрацьовувань. [21]

2.5 Механізми забезпечення цілісності даних

Забезпечення цілісності даних є важливим аспектом безпеки операційних систем з відкритим кодом. Цілісність даних гарантує, що дані не були змінені або пошкоджені без відома адміністратора або користувача, що є критично важливим для забезпечення безпеки, конфіденційності та довіри до системи. Механізми забезпечення цілісності даних включають методи контролю змін, моніторинг файлів і використання криптографічних технологій для виявлення несанкціонованих змін у даних.

Цілісність даних означає, що дані не можуть бути змінені, видалені або пошкоджені без відповідного дозволу. У контексті операційних систем з відкритим кодом основними механізмами забезпечення цілісності є:

Цифрові підписи та хешування: Використовуються для перевірки, що дані не були змінені після їх створення. Дані шифруються за допомогою криптографічних методів, що дозволяє перевірити їхню цілісність при кожному доступі.

Моніторинг змін у файлах: Для виявлення несанкціонованих змін в системі використовуються інструменти для моніторингу змін у файлах та конфігураціях.

Контроль доступу: Обмеження прав доступу до файлів та систем, щоб тільки авторизовані користувачі могли здійснювати зміни. [22]

Хешування є одним із найбільш поширених методів забезпечення цілісності даних. При хешуванні використовується алгоритм, який перетворює вхідні дані в унікальний рядок фіксованої довжини (хеш). Якщо дані змінюються, хеш також змінюється, що дає можливість виявити будь-які несанкціоновані зміни.

Цифрові підписи використовують криптографічний підхід для створення унікального підпису даних, що дозволяє перевірити їх цілісність і автентичність.

Приклад: У Linux використовується інструмент sha256sum, щоб генерувати хеші файлів. Цей інструмент дозволяє перевіряти, чи не було змінено файли за допомогою порівняння хешів оригінальних і поточних файлів.

SHA-256 – більш безпечний алгоритм для хешування порівняно з старими алгоритмами, такими як MD5 або SHA-1, які вважаються вразливими до атак, як-от колізії.

Рекомендація: Використовувати сучасні криптографічні алгоритми, такі як SHA-256, для забезпечення цілісності даних, особливо для конфіденційних даних.

Інструменти моніторингу змін у файлах Для виявлення несанкціонованих змін у системі використовуються інструменти моніторингу, які відстежують будь-які зміни в файловій системі. Це може бути корисно для виявлення змін в критичних системних файлах або конфігураціях.

AIDE (Advanced Intrusion Detection Environment) – це інструмент для моніторингу цілісності файлів у Linux. Він створює базу даних зі значеннями хешів для важливих файлів системи, після чого періодично перевіряє ці файли на наявність змін. Якщо хеш файлу змінюється, AIDE повідомляє про це.

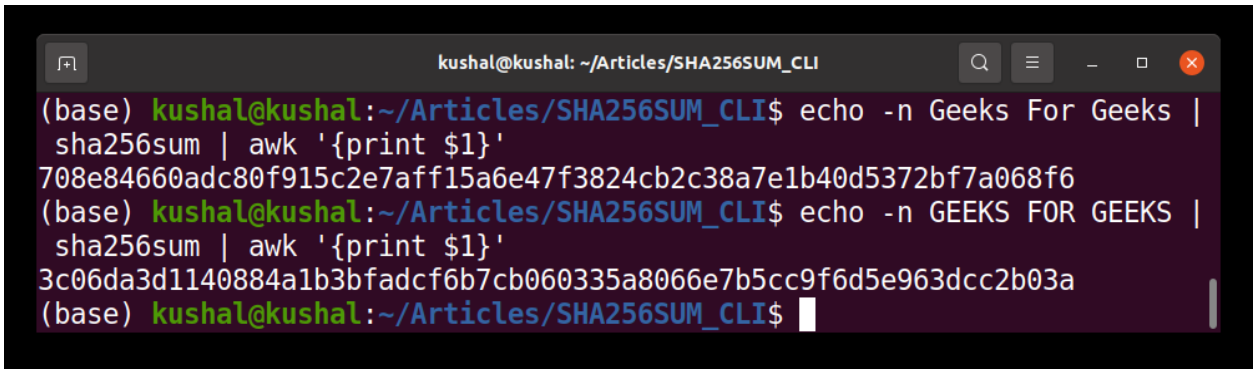

Рисунок 2.3 – Використання алгоритму SHA-256

Tripwire – ще один популярний інструмент для моніторингу цілісності файлів, який працює на подібному принципі, з можливістю створення детальних звітів про зміни в системі.

Приклад: Якщо файл /etc/passwd (файл, що містить інформацію про користувачів системи) змінюється без відома адміністратора, інструмент AIDE автоматично виявить зміну, порівнявши хеші, і повідомить про це.

AIDE підтримує численні алгоритми хешування (наприклад, SHA-256), має легку інтеграцію в більшість систем Linux.

Tripwire зазвичай використовується в більш великих інфраструктурах і надає більш розширені можливості для звітності і інтеграції в інші системи безпеки.

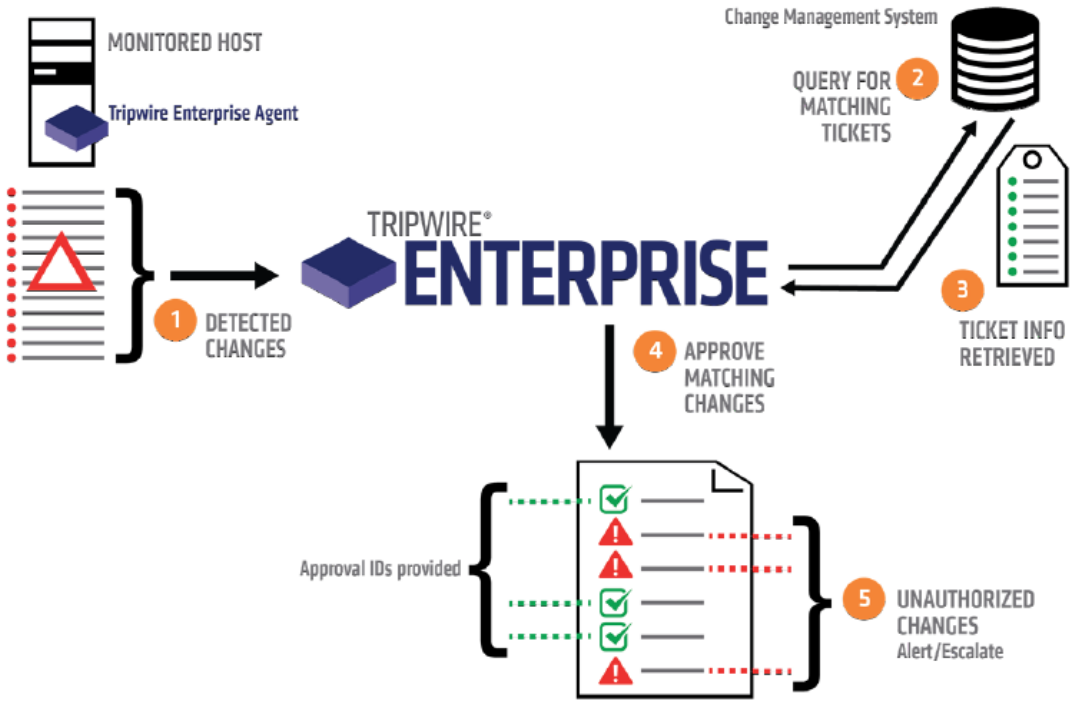

Рисунок 2.4 – Схема роботи інструменту для моніторингу цілісності файлів Tripwire

Рекомендація: Для малих та середніх організацій можна рекомендувати AIDE, оскільки він є легким у налаштуванні та ефективно працює на більшості версій Linux.

Контроль доступу та управління правами Контроль доступу є важливою частиною забезпечення цілісності даних. Обмеження прав доступу до файлів, баз даних і системи дозволяє запобігти несанкціонованим змінам і зловживанням.

SELinux (Security-Enhanced Linux) – система управління доступом, яка забезпечує детальний контроль доступу до файлів і системних ресурсів, заснований на політиках безпеки.

AppArmor – подібний механізм, що також дозволяє створювати політики для доступу додатків до системних ресурсів. AppArmor простіший у налаштуванні, ніж SELinux, але має меншу гнучкість.

Приклад: SELinux може бути налаштований для обмеження прав доступу до певних файлів або директорій для окремих користувачів або процесів. Наприклад, адміністратор може дозволити лише певним користувачам доступ до конфігураційних файлів, забезпечуючи їхню цілісність.

SELinux забезпечує більш високий рівень безпеки та гнучкості, але має складнішу конфігурацію.

AppArmor є простішим для розгортання, але має менше можливостей у порівнянні з SELinux.

Рекомендація: Для середніх і великих організацій, де потрібно забезпечити детальний контроль доступу, рекомендується використовувати SELinux. Для менших систем або для швидкого розгортання можна вибрати AppArmor. [23]

Регулярна перевірка цілісності важливих файлів: Використовувати інструменти на зразок AIDE або Tripwire для регулярної перевірки файлів і конфігурацій на наявність змін. Це дозволить швидко виявити несанкціоновані зміни та знизити ризик атаки.

Шифрування важливих даних: Для підвищення цілісності та конфіденційності даних рекомендується використовувати шифрування.

Наприклад, шифрування файлової системи за допомогою LUKS (Linux Unified Key Setup) гарантує захист даних від несанкціонованого доступу.

Використання сильних хеш-функцій: Завжди використовувати сучасні та безпечні хеш-функції, такі як SHA-256 або SHA-3, для забезпечення цілісності даних. Уникайте застарілих алгоритмів, таких як MD5 або SHA-1.

Контроль доступу на основі політик: Використовувати системи контролю доступу, такі як SELinux або AppArmor, для обмеження прав доступу до важливих системних файлів та конфігурацій. Це дозволить зменшити ймовірність несанкціонованих змін.

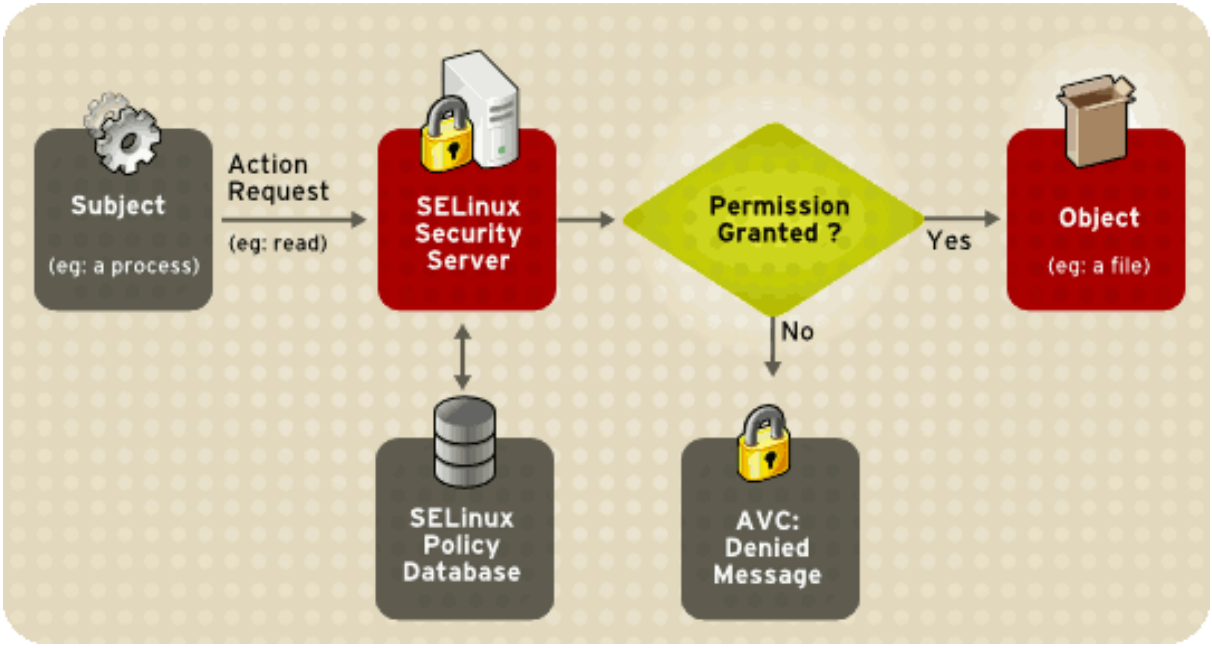

Рисунок 2.4 – Система контролю доступу SELinux

Інтеграція з системами моніторингу безпеки: Встановлення інструментів для моніторингу цілісності даних в рамках загальної стратегії безпеки (наприклад, IDS/IPS системи, SIEM) дозволить автоматично реагувати на виявлені загрози і підвищити рівень захисту системи в реальному часі.

Забезпечення цілісності даних є критичним для будь-якої операційної системи, і для операційних систем з відкритим кодом застосування

відповідних механізмів дозволяє значно підвищити рівень безпеки. Використання хешування, моніторингу змін у файлах та контролю доступу є основними методами забезпечення цілісності, які повинні бути інтегровані у щоденну практику адміністратора системи для запобігання зловживанням та несанкціонованим змінам. [24]

2.6 Методології та інструменти резервного копіювання

Резервне копіювання є критично важливим елементом стратегії забезпечення безпеки операційних систем з відкритим кодом. Це дозволяє відновити дані після втрати, пошкодження або злому системи, а також забезпечити захист від вірусів, шкідливих програм та інших загроз. Метою резервного копіювання є забезпечення безпеки даних, їх доступності в разі необхідності та цілісності.

Резервне копіювання передбачає створення копій даних та їх зберігання в окремому місці, щоб у разі втрати або пошкодження оригінальних даних можна було відновити роботу системи. Існують кілька основних типів резервного копіювання:

Повне резервне копіювання (Full Backup): Створюється повна копія всіх даних і файлів. Цей тип копіювання є найбільш надійним, але може вимагати багато часу та місця на зберігання.

Інкрементне резервне копіювання (Incremental Backup): Копіюються лише ті файли, які змінилися після останнього резервного копіювання. Це дозволяє зменшити обсяг даних для збереження, але для відновлення необхідно об'єднати всі інкрементні копії.

Диференціальне резервне копіювання (Differential Backup): Копіюються всі зміни, які відбулися після останнього повного резервного копіювання. З одного боку, це швидше, ніж повне резервне копіювання, але може вимагати більше місця для зберігання, ніж інкрементне.

Резервне копіювання в реальному часі (Continuous Data Protection, CDP): Це метод резервного копіювання, що забезпечує збереження змін в даних у реальному часі або майже в реальному часі, мінімізуючи можливі втрати даних. [25]

В операційних системах з відкритим кодом існує багато інструментів для створення резервних копій даних, що дозволяють вибрати оптимальне рішення в залежності від вимог до системи та даних.

Опис: rsync є потужним інструментом для синхронізації файлів і каталогів між локальними та віддаленими системами. Використовується як для резервного копіювання, так і для відновлення даних. Це ефективний інструмент, який дозволяє створювати інкрементні резервні копії та синхронізувати зміни з мінімальними витратами часу.

Порівняння: rsync є гнучким і потужним інструментом, що підтримує різні методи копіювання (наприклад, інкрементне або диференціальне). Однак він потребує налаштування для автоматичного створення регулярних копій (наприклад, через cron).

Опис: Bacula є повнофункціональним рішенням для резервного копіювання даних, що підходить для великих мережевих інфраструктур. Воно підтримує різні типи резервних копій, включаючи повні, інкрементні та диференціальні, а також має інтерфейс для централізованого керування.

Приклад: Bacula складається з кількох компонентів, таких як Director, Storage Daemon, і File Daemon, що дозволяє централізовано управляти копіями даних на різних машинах.

Переваги: Bacula пропонує повний набір можливостей для автоматизації, централізованого керування резервними копіями в великих середовищах.

Недоліки: Має більш складну конфігурацію і потребує значно більше ресурсів порівняно з іншими інструментами, такими як rsync або tar.

Рекомендація: Для великих підприємств або організацій з масштабними резервними потребами Bacula буде оптимальним вибором,

особливо якщо потрібно централізовано управляти копіями в різних частинах інфраструктури.

Опис: Duplicity – це інструмент для створення зашифрованих резервних копій. Підтримує інкрементне резервне копіювання та можливість автоматичного шифрування даних для забезпечення їх конфіденційності.

Переваги: Підтримка шифрування за замовчуванням і інкрементного резервного копіювання.

Недоліки: Залежить від інтерфейсу командного рядка, що може бути складно для користувачів, які не мають досвіду з командним рядком.

Рекомендація: Для користувачів, які шукають рішення для створення зашифрованих резервних копій на базі командного рядка, Duplicity є гарним вибором, особливо якщо необхідна підтримка хмарних сховищ.

Опис: Timeshift є простим у використанні інструментом для створення знімків файлової системи (snapshot), що дозволяє створювати резервні копії системних файлів і легко відновлювати їх у разі помилок або збоїв.

Приклад: Використання інтерфейсу Timeshift дозволяє користувачу автоматично створювати знімки на основі заданого розкладу (наприклад, щоденно чи щотижня).

Переваги: Простий у використанні і надає зручний графічний інтерфейс.

Недоліки: Не підтримує багато типів резервного копіювання (тільки системні знімки) і є менш гнучким порівняно з інструментами на основі командного рядка.

Рекомендація: Для користувачів, які шукають простоту і зручність у використанні для створення системних знімків, Timeshift є ідеальним рішенням. [26]

Різноманітність стратегій резервного копіювання: Використовувати комбінацію повних, інкрементних та диференціальних копій для забезпечення мінімізації витрат на зберігання, водночас зберігаючи швидкий доступ до відновлення даних.

Шифрування резервних копій: Шифрування резервних копій є необхідним для захисту конфіденційних даних. Використовувати інструменти, які підтримують шифрування, такі як Duplicity, щоб захистити резервні копії від несанкціонованого доступу.

Таблиця 2.3 – Порівняння інструментів резервного копіювання

Параметр	Bacula	rsync	Amanda
Функціональність	Клієнт-серверна архітектура Підтримка інкрементального та диференційного копіювання Планування резервних копій Автоматизація відновлення	Однонаправлене та двонаправлене синхронізування Інкрементальне копіювання Мережеве копіювання	Підтримка інкрементального копіювання Централізоване управління Підтримка шифрування резервних копій
Підтримка ОС	Linux, Windows, macOS, Unix	Linux, Windows, macOS	Linux, Unix
Типи даних, що зберігаються	Файли, бази даних, системні образи	Файли (локальні та мережеві)	Файли, бази даних
Особливості	Підтримка великих корпоративних систем Складне налаштування для початківців	Простота використання Гнучкі параметри синхронізації	Відмінний для централізованого зберігання резервних копій Проста інтеграція в середовищах Unix

Автоматизація процесу резервного копіювання: Автоматичне резервне копіювання є важливим для забезпечення регулярних резервних копій без

втручання користувача. Для цього можна використовувати такі інструменти, як rsync через cron, або налаштувати автоматичне створення знімків у Timeshift.

Тестування відновлення даних: Регулярно перевіряти працездатність резервних копій та процес відновлення даних для запобігання ситуаціям, коли резервна копія не буде коректно відновлена.

Зберігання резервних копій в різних місцях: Для додаткової безпеки важливо зберігати резервні копії не тільки на локальних машинах, але й в хмарних сховищах або на віддалених серверах для захисту від катастрофічних подій, таких як фізичні пошкодження пристроїв.

Резервне копіювання є критичним компонентом стратегії безпеки для операційних систем з відкритим кодом. Використання правильних методів та інструментів для створення резервних копій дозволяє зберегти цілісність і доступність даних у разі загрози або непередбачених ситуацій. [27]

2.7 Захист на рівні ядра операційної системи

Ядро операційної системи (ОС) є її центральною частиною, яка безпосередньо взаємодіє з апаратним забезпеченням, управлінню пам'яттю, процесами, а також контролює доступ до системних ресурсів. Оскільки ядро знаходиться на найглибшому рівні системи, захист ядра від атак є ключовим аспектом забезпечення безпеки ОС. В операційних системах з відкритим кодом, таких як Linux, особливу увагу приділяється захисту ядра для запобігання несанкціонованому доступу до системи, виконанню шкідливого коду та атак, які можуть змінити функціональність ядра.

До основних загроз на рівні ядра ОС належать:

Експлойти ядра: Атаки, які спричиняють уразливості в коді ядра, що дозволяють зловмисникам отримати доступ до привілейованих рівнів системи.

Зловмисні модулі ядра: Установка шкідливих модулів, які можуть змінювати поведінку ядра, додавати зловмисні функції або приховувати сліди зловмисної діяльності.

Зловживання привілеями: Привілейовані процеси, такі як root, можуть бути використані для доступу до чутливої інформації або для виконання небажаних операцій.

Уразливості в драйверах: Помилки в драйверах апаратного забезпечення можуть стати вектором атак, що дозволяє зловмисникам отримати доступ до пам'яті або маніпулювати апаратними ресурсами. [28]

Для захисту ядра операційних систем з відкритим кодом, особливо в Linux, розроблені різні методи та інструменти. Найважливіші з них:

SELinux – це набір механізмів безпеки, що включає політики контролю доступу для обмеження доступу до ресурсів ОС. SELinux використовує модель доступу на основі мандатних політик (MAC), що дозволяє встановлювати строгі правила для доступу до файлів, процесів і навіть ядра.

SELinux використовує політики для обмеження доступу користувачів і процесів до критичних частин системи, зокрема до модулів ядра та драйверів. Наприклад, певний процес може бути обмежений в доступі до файлової системи або певних апаратних ресурсів.

Переваги: Високий рівень контролю за доступом до ресурсів, зокрема до модулів ядра. Пропонує детальну політику безпеки, яка здатна блокувати навіть несанкціоновані спроби доступу.

Недоліки: Вимагає складної налаштування та часто може бути складним для початківців.

Рекомендація: SELinux є потужним інструментом для захисту ядра, особливо в умовах високих вимог до безпеки. Рекомендується для серверних та корпоративних середовищ, де потрібно забезпечити максимальний рівень безпеки.

Опис: AppArmor – це ще один механізм безпеки для Linux, який дозволяє створювати профілі безпеки для програм та процесів. На відміну від

SELinux, AppArmor базується на простіших профілях, які визначають, до яких ресурсів може отримати доступ конкретний процес.

Наприклад, профіль для веб-сервера може обмежити доступ до лише тих каталогів, які необхідні для роботи сервера, запобігаючи доступу до інших частин системи, зокрема до ядра.

Переваги: Легкість у налаштуванні та управлінні порівняно з SELinux. AppArmor забезпечує ефективний контроль доступу без складних налаштувань.

Недоліки: Має менш гнучку політику безпеки порівняно з SELinux, що може бути недоліком для великих і складних інфраструктур.

Рекомендація: Для більшості користувачів і середовищ AppArmor є хорошим компромісом між простотою налаштування і ефективністю контролю доступу. Рекомендується для менш критичних застосунків та в невеликих організаціях.

Опис: Живе патчування ядра (Kernel Live Patching) дозволяє застосовувати оновлення без перезавантаження системи. Це критично важливо для забезпечення безпеки в реальному часі, оскільки дозволяє виправляти вразливості ядра без необхідності перезапуску ОС.

Інструменти, як kpatch або ksplice, дозволяють віддалено застосовувати патчі до ядра Linux без перезавантаження, знижуючи час простою та мінімізуючи ризик відсутності критичних оновлень.

Переваги: Мінімальний час простою системи, негайне усунення уразливостей, що робить систему менш уразливою до атак.

Недоліки: Може бути складно інтегрувати в середовища з власними або кастомними ядрами. Не всі патчі підтримують живе патчування.

Рекомендація: Для організацій, які потребують високої доступності та мінімального часу простою, варто розглянути застосування технологій живого патчування ядра. Вони повинні бути інтегровані в систему моніторингу та управління безпекою.

Grsecurity – це набір патчів до ядра Linux, що забезпечує додаткові механізми безпеки, включаючи посилення безпеки процесів, захист від експлойтів і контроль доступу.

Grsecurity дозволяє застосовувати стратегії для захисту пам'яті, обмеження привілеїв процесів, а також запобігання виконанню шкідливого коду.

Переваги: Додає додаткові заходи безпеки для системи, посилюючи її захист від відомих і нових уразливостей.

Недоліки: Потрібно більше ресурсів для налаштування та тестування, а також може призвести до проблем із сумісністю з деякими програмами та модулями ядра.

Рекомендація: grsecurity – це потужне доповнення для операційних систем, які потребують додаткового рівня захисту, особливо для критичних інфраструктур. Рекомендується для організацій, де потрібна висока безпека на рівні ядра. [29]

Використовувати SELinux або AppArmor для обмеження доступу до ключових компонентів системи та ядра, обираючи в залежності від складності і вимог до безпеки.

Регулярно оновлювати ядро та застосовувати патчі для усунення вразливостей, використовуючи інструменти, такі як kpatch або ksplice, для живого патчування.

Активувати grsecurity для посилення безпеки ядра шляхом додавання додаткових політик безпеки, таких як захист пам'яті, контроль доступу та запобігання виконанню шкідливого коду.

Перевіряти сумісність нових патчів та модулів ядра з наявними системами, щоб не порушити функціональність системи.

Захист ядра операційної системи з відкритим кодом є критично важливим для забезпечення безпеки всієї системи.

Таблиця 2.4 – Порівняння основних механізмів захисту ядра

Характеристика	SELinux	AppArmor
Гнучкість	Надзвичайно гнучкий: дозволяє створювати складні політики безпеки Підтримка контекстів доступу	Менш гнучкий у порівнянні з SELinux Використовує профілі, які легше створювати та змінювати
Продуктивність	Вплив на продуктивність може бути значним при неправильному налаштуванні політик	Мінімальний вплив на продуктивність Менш ресурсомісткий у порівнянні з SELinux
Сумісність з іншими системами	Підтримка в основному на Red Hat-based системах (RHEL, CentOS, Fedora) Інтеграція з Docker і Kubernetes	Підтримка на Debian-based системах (Ubuntu, Debian) Також сумісний з Docker і Kubernetes
Особливості	Використовує контекстну модель для захисту файлів, процесів та сокетів Складний у налаштуванні для початківців	Спрощене налаштування через профілі Зручний для менш складних середовищ

Використання таких інструментів, як SELinux, AppArmor, grsecurity та технології живого патчування, допомагає зменшити ризик атак на ядро. Регулярне оновлення ядра та моніторинг системи забезпечують стійкість до нових загроз. [30]

3 ОПТИМІЗАЦІЯ МЕТОДІВ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ В ОПЕРАЦІЙНИХ СИСТЕМАХ З ВІДКРИТИМ КОДОМ

У цьому розділі буде детально розглянута оптимізація одного з методів забезпечення безпеки в операційних системах з відкритим кодом. Для прикладу оптимізуємо метод контролю доступу, який є одним з основних механізмів захисту системи. Основною метою цього розділу є виявлення можливих проблем у реалізації методу і покращення його ефективності та продуктивності.

3.1 Вибір методу для оптимізації безпеки в операційних системах з відкритим кодом

Вибір методу для оптимізації є важливим кроком у процесі підвищення безпеки операційної системи з відкритим кодом. Оптимізація повинна бути орієнтована на конкретні проблеми або слабкі місця, виявлені у поточній реалізації безпеки. Оскільки операційні системи з відкритим кодом мають різноманітні механізми для забезпечення безпеки, вибір методу для оптимізації потребує комплексного підходу, що базується на наступних факторах:

Тип загроз: необхідно оцінити, який саме тип загрози найімовірніше вплине на систему. Це можуть бути атаки через мережу, несанкціонований доступ до файлів, порушення цілісності даних або інші проблеми безпеки.

Сучасний стан системи: для вибору правильного методу оптимізації необхідно розуміти, які існуючі механізми вже використовуються і які з них потребують вдосконалення.

Продуктивність системи: оптимізація повинна враховувати не лише безпеку, а й ефективність роботи системи, щоб уникнути її уповільнення чи зайвого споживання ресурсів.

Простота адміністрування та інтеграція: вибір методу повинен враховувати легкість впровадження і сумісність із вже наявними системами та процесами адміністрування.

Враховуючи ці фактори, розглянемо основні методи, які можуть бути вибрані для оптимізації безпеки операційної системи з відкритим кодом. Одним із найбільш критичних аспектів, який потребує оптимізації в більшості операційних систем на базі Linux, є методи контролю доступу.

3.1.1 Визначення проблемних зон для оптимізації

Перед тим як зупинитись на оптимізації конкретного методу, важливо детально проаналізувати поточну реалізацію та визначити проблемні зони:

У більшості випадків користувачі мають права на доступ до системи, які можуть бути надмірно широкими. Наприклад, користувачі можуть мати права суперкористувача (root), що дозволяє їм виконувати будь-які дії, навіть якщо це не відповідає їхній ролі в організації.

Проблеми можуть виникнути і через неправильне налаштування прав доступу до важливих файлів або системних ресурсів.

Багато адміністраторів систем віддають перевагу простим, але менш безпечним підходам до конфігурації прав доступу, що робить систему уразливою до атак.

Часом механізми контролю доступу в операційних системах з відкритим кодом не інтегруються належним чином з іншими системами безпеки, такими як системи виявлення вторгнень (IDS) або антивірусне програмне забезпечення.

В організаціях, де чисельність користувачів велика, важливо забезпечити гнучке та ефективне управління правами доступу, щоб мінімізувати можливості для зловмисників. Часто існуючі інструменти надають обмежену можливість автоматизації цього процесу.

3.1.2 Оцінка можливих методів для оптимізації

Вибір конкретного методу для оптимізації повинен ґрунтуватися на аналізі його переваг і недоліків, а також на врахуванні специфічних потреб організації. Ось кілька методів, які можна вибрати для оптимізації контролю доступу:

Вибіркове керування доступом (DAC – Discretionary Access Control). У DAC адміністратор або власник ресурсу може самостійно визначати, хто має доступ до його ресурсів. Це одна з найпростіших реалізацій контролю доступу.

Переваги: Простота в налаштуванні та управлінні правами.

Недоліки: Уразливість до помилок користувачів і надмірних прав доступу. Погано масштабуються в великих системах.

Оптимізація: Вдосконалення політик доступу через чітке розмежування прав користувачів, автоматизація контролю через скрипти.

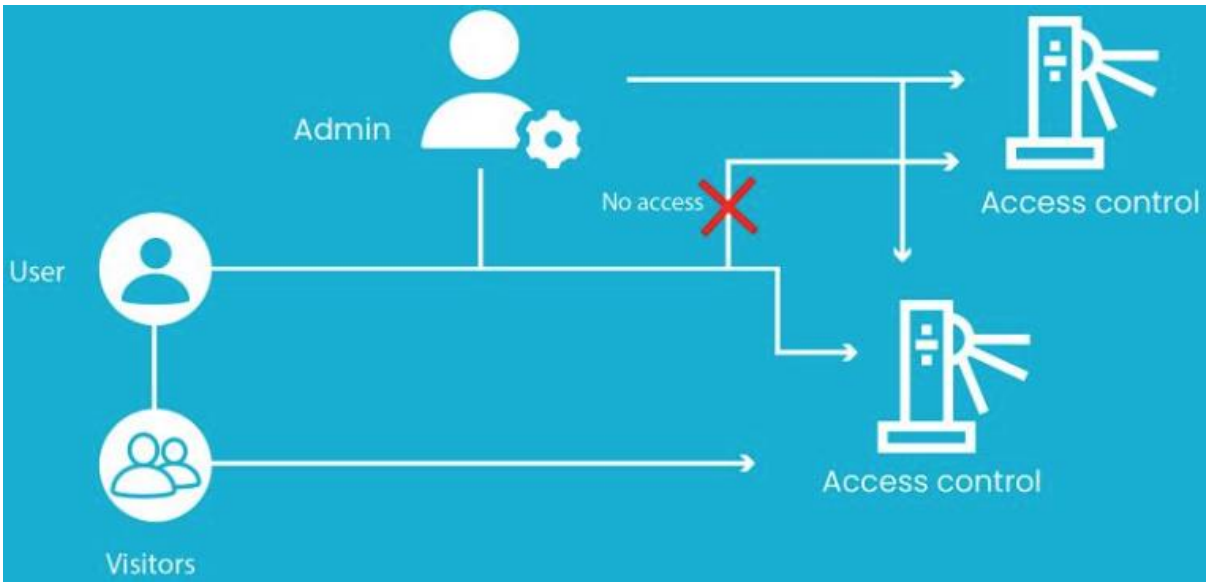

Рисунок 3.1 – Вибіркове керування доступом

Мандатне керування доступом (MAC - Mandatory Access Control). MAC забезпечує більш строгий контроль доступу, де системи визначають правила

доступу для користувачів і об'єктів незалежно від їх бажання чи власників.

Переваги: Вища безпека завдяки суворим політикам доступу. Недоліки: Складність у налаштуванні та адмініструванні. Оптимізація: Впровадження динамічних політик, які дозволяють змінювати рівень доступу в залежності від умов, автоматизація застосування політик, інтеграція з іншими безпековими механізмами (наприклад, IDS).

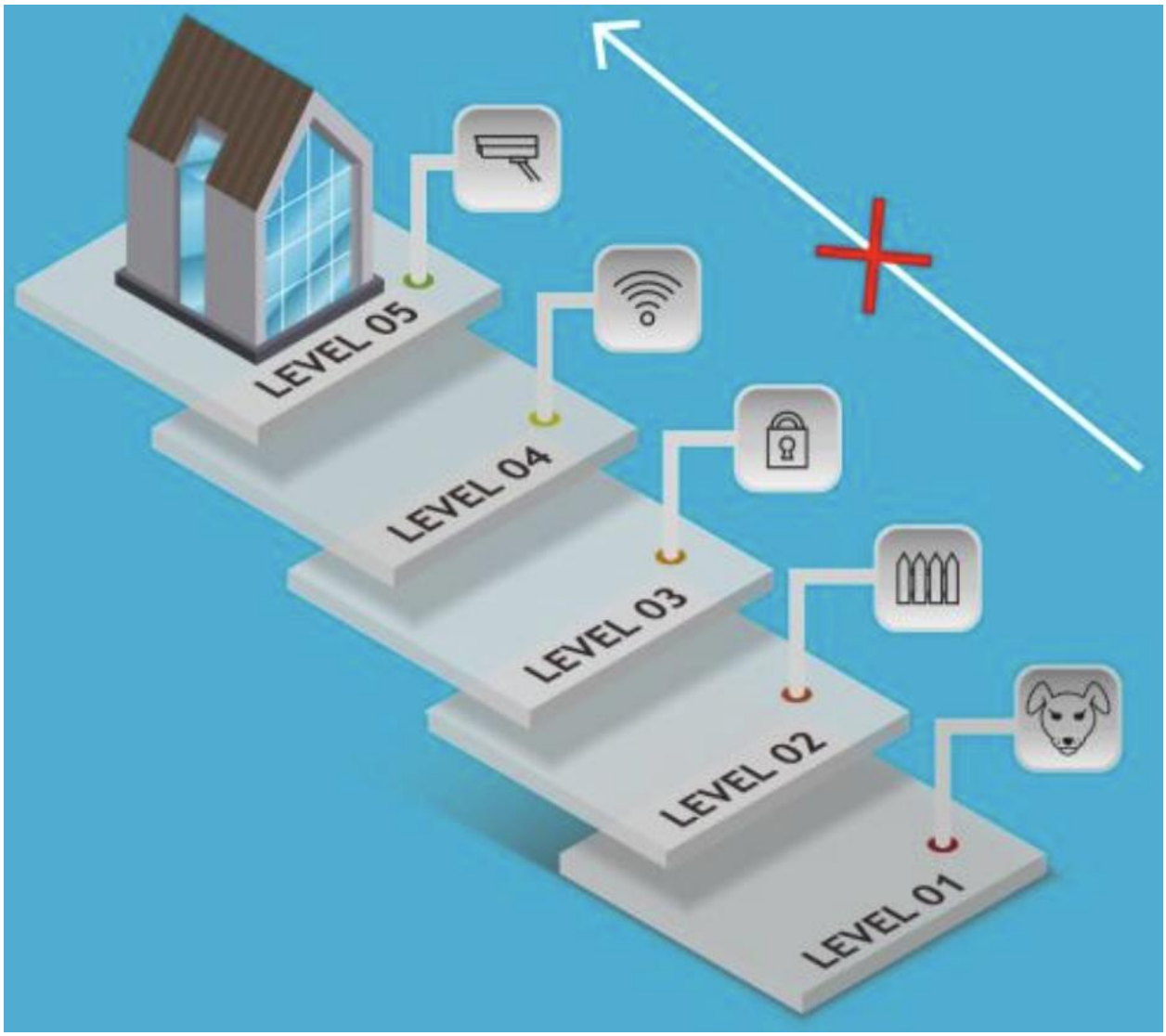

Рисунок 3.2 – Мандатне керування доступом

Керування доступом на основі ролей (RBAC – Role-Based Access Control). У RBAC доступ до ресурсів визначається на основі ролі

користувача, що значно спрощує управління правами доступу.

Переваги: Легкість адміністрування прав доступу, особливо у великих організаціях. Недоліки: Обмеженість у складних випадках, де потрібно враховувати більше специфічних правил. Оптимізація: Розширення ролей для більш детального контролю, автоматизація оновлень прав доступу, інтеграція з системами моніторингу та аналітики.

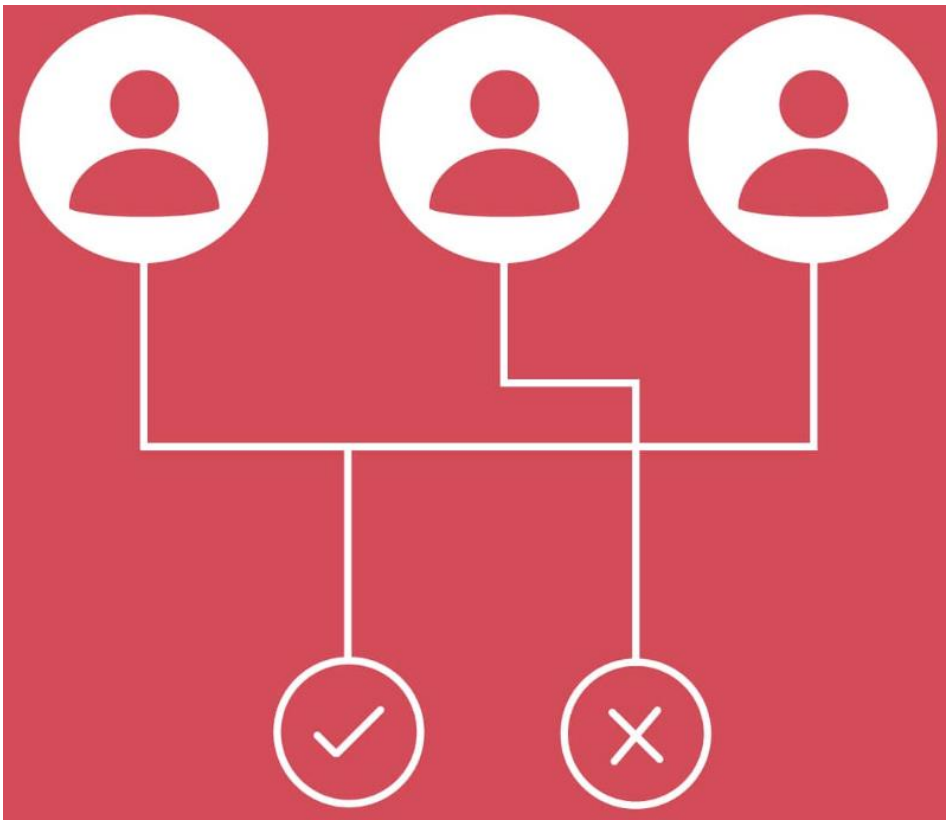

Рисунок 3.3 – Керування доступом на основі ролей

Linux з покращеним рівнем безпеки (SELinux – Security-Enhanced Linux). SELinux забезпечує найбільш суворий контроль доступу на основі політик MAC, що дозволяє адміністратору визначати, які програми можуть отримати доступ до конкретних файлів і ресурсів.

Переваги: Високий рівень безпеки, детальне управління доступом до ресурсів. Недоліки: Складність налаштування та інтеграції, висока вимогливість до ресурсів. Оптимізація: Спрощення налаштувань для

звичайних користувачів через попередньо налаштовані профілі політик, вдосконалення документації та автоматизація процесу застосування політик.

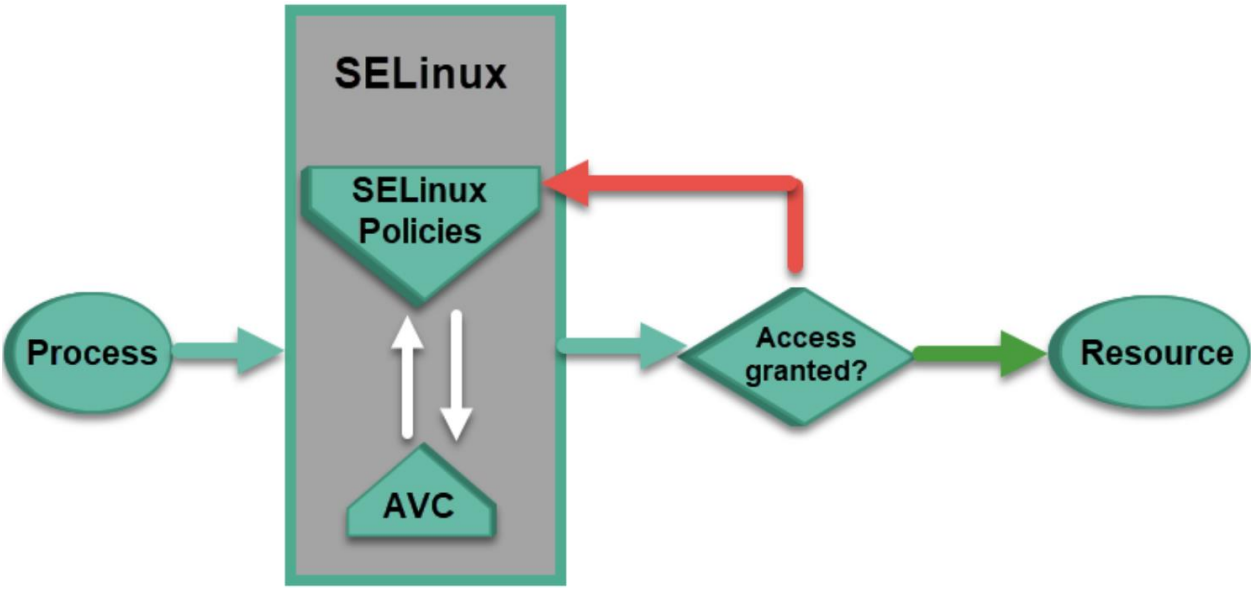

Рисунок 3.4 – Linux з покращеним рівнем безпеки

3.1.3 Вибір методу для оптимізації

Після аналізу кожного з методів можна зробити висновок, що для більшості операційних систем з відкритим кодом найбільш перспективним методом для оптимізації є RBAC (Role-Based Access Control) в поєднанні з SELinux. Це дозволить забезпечити високий рівень безпеки через жорсткий контроль доступу та водночас значно спростити адміністрування та зменшити кількість помилок, пов’язаних з неправильно налаштованими правами.

Нижче наведена стратегія для оптимізації. Використовувати RBAC для організації ролей користувачів з чітким розподілом прав доступу. Інтегрувати SELinux для забезпечення додаткового рівня захисту через політики обмеження доступу. Автоматизувати процес створення та оновлення ролей через інструменти адміністрування та налаштування, щоб забезпечити безперервність та актуальність прав доступу.

Таким чином, оптимізація методу контролю доступу дозволить підвищити рівень безпеки операційної системи та спростити її адміністрування.

3.2 Оцінка поточної реалізації методу контролю доступу

Оцінка поточної реалізації методу контролю доступу є ключовим етапом у вдосконаленні безпеки операційних систем з відкритим кодом. Визначення сильних і слабких сторін поточної реалізації дозволяє не тільки виявити уразливості, але й з'ясувати, які аспекти потребують оптимізації для підвищення загальної безпеки та ефективності.

У більшості операційних систем з відкритим кодом, таких як Linux, за замовчуванням використовуються декілька різних механізмів контролю доступу, серед яких DAC (Discretionary Access Control), MAC (Mandatory Access Control) та RBAC (Role-Based Access Control). Кожен з цих методів має свої сильні та слабкі сторони, які можна оцінити за допомогою наступних критеріїв:

DAC (Discretionary Access Control). Визначається правами доступу, які даються власником файлів або директорій. Права доступу можуть бути змінені власником ресурсу.

Переваги: Простота налаштування та адекватність для малих і середніх за розмірами організацій. Недоліки: Легко здійснити помилки при налаштуванні прав доступу, що може призвести до порушення політики безпеки. Не забезпечує суворого контролю за доступом до важливих системних файлів.

MAC (Mandatory Access Control). Системи на базі MAC визначають, хто може доступитися до ресурсів незалежно від бажання власника.

Використовує політики безпеки, які обмежують доступ користувачів та програм до ресурсів, засновані на категоріях і рівнях безпеки.

Переваги: Вища безпека, бо системи контролю доступу більш жорстко обмежують доступ. Недоліки: Складність налаштування та керування, що може бути проблемою для адміністраторів, які не мають достатнього досвіду.

RBAC (Role-Based Access Control). Доступ до системних ресурсів надається на основі ролей користувачів у організації. Легше масштабувати, адже додавання нових користувачів просто передбачає призначення ролей.

Переваги: Простота адміністрування і хороша сумісність з багатьма великими організаціями. Недоліки: Може бути недостатньо гнучким для складних середовищ, де потрібно створити політики доступу з високим рівнем деталізації.

Надмірний доступ до системних файлів: В одному з поширених варіантів налаштування прав доступу за допомогою DAC, користувачі часто отримують доступ до файлів та директорій, які не відповідають їхній ролі. Це призводить до того, що навіть звичайний користувач може змінювати або читати чутливі файли.

Приклад: У деяких конфігураціях Linux, файли, які містять налаштування ядра чи ключі шифрування, можуть бути доступні користувачам без належних прав, що дозволяє потенційним зловмисникам використовувати ці файли для атаки.

Недосконалість політик контролю доступу: В разі використання DAC, адміністратори часто не враховують усіх можливих сценаріїв доступу, а це може привести до помилок у налаштуванні. Користувачі з "базовими" правами можуть неосвідомлено змінювати системні файли або навіть порушувати принципи принципу найменших прав (Least Privilege Principle).

Приклад: Звичайний користувач може мати права на видалення чи зміну конфігураційних файлів, що може стати вразливістю в разі внутрішньої загрози або атаки.

Складність управління правами доступу: При масштабуванні системи та збільшенні кількості користувачів, зростає складність управління правами доступу. В операційних системах з відкритим кодом адміністраторам часто

доводиться вручну налаштовувати доступ для кожного користувача. Це ускладнює адміністрування, підвищує ймовірність помилок і порушень політики безпеки.

Приклад: У великих організаціях з тисячами співробітників налаштування доступу до кожного окремого файлу або програми вручну може призвести до високих витрат часу та ресурсів, а також створити потенційні діри в безпеці.

Проблеми із сумісністю з іншими системами безпеки: Інтеграція механізмів контролю доступу з іншими системами безпеки, такими як IDS (Intrusion Detection Systems), SIEM (Security Information and Event Management), та антивірусними програмами, може бути проблематичною. Багато сучасних систем контролю доступу не мають вбудованої підтримки для тісної інтеграції з цими інструментами, що знижує загальний рівень безпеки.

Для того, щоб оцінити ефективність поточної реалізації механізмів контролю доступу в операційних системах з відкритим кодом, необхідно провести такі аналізи:

Аудит прав доступу: Оцінка поточної реалізації прав доступу за допомогою аудиту дозволяє виявити надлишкові права у користувачів та програмах. Для цього можна використовувати такі інструменти, як Auditd в Linux, який дозволяє відслідковувати всі спроби доступу до файлів та ресурсів.

Приклад: Якщо звичайний користувач має доступ до каталогу /etc, де зберігаються конфігураційні файли, це буде серйозною вразливістю.

Аналіз політик доступу: Важливо оцінити наявні політики доступу та їх здатність відображати потреби організації. Інструменти, як AppArmor та SELinux, дозволяють визначити, чи відповідають політики встановленим вимогам безпеки.

Приклад: В організаціях з високими вимогами до безпеки, політики SELinux можуть допомогти обмежити доступ до критичних системних файлів.

Тестування продуктивності системи: Оцінка впливу поточних політик контролю доступу на продуктивність операційної системи. Надмірна кількість перевірок доступу та складні політики можуть негативно впливати на продуктивність, особливо в середовищах з високими навантаженнями.

Приклад: Якщо в системі встановлені дуже складні політики через SELinux, це може привести до уповільнення роботи програм, особливо на старих або слабких серверах.

Використання комбінованих методів контролю доступу: Замість того, щоб обирати один метод контролю доступу, рекомендується поєднувати RBAC для управління ролями та MAC для забезпечення додаткового захисту системи. Така комбінація дозволяє обмежити доступ до ресурсів, зберігаючи при цьому гнучкість у масштабуванні системи.

Інтеграція з іншими системами безпеки: Для забезпечення більш високого рівня безпеки важливо інтегрувати механізми контролю доступу з іншими інструментами безпеки, такими як IDS/IPS та SIEM. Це дозволить вчасно виявляти несанкціоновані спроби доступу і швидко реагувати на потенційні загрози. Покращення автоматизації та аудиту: Для зниження ймовірності помилок рекомендується автоматизувати процеси перевірки прав доступу та аудитів. Регулярні перевірки політик доступу допоможуть вчасно виявляти потенційні вразливості і порушення.

Навчання адміністраторів: Важливим аспектом є навчання адміністраторів системи на всіх рівнях управління доступом. Це допоможе забезпечити належну конфігурацію прав доступу та впровадження найкращих практик у управлінні безпекою.

Оцінка та тестування нових інструментів: Регулярне оновлення інструментів контролю доступу, таких як SELinux, AppArmor та Auditd, дозволить покращити загальний рівень безпеки.

Таблиця 3.1 – Результати оцінки поточного методу контролю доступу

Показник	Опис	Оцінка (за шкалою 1–5)	Коментар
Рівень безпеки	Наскільки метод забезпечує захист від несанкціонованого доступу	4	Забезпечує базову безпеку, але є ризик обхідних шляхів
Гнучкість	Легкість адаптації під різні сценарії використання	3	Складність у налаштуванні для складних систем
Простота використання	Зручність у впровадженні та адмініструванні	5	Простий для користувачів, не вимагає значних технічних знань
Масштабова-ність	Здатність працювати ефективно у великих системах	3	Потребує додаткових ресурсів для великих організацій
Продуктив-ність	Вплив на продуктивність системи	4	Негативний вплив мінімальний
Захист від внутрішніх загроз	Рівень захисту від дій привілейованих користувачів	3	Немає достатнього контролю за діями адміністратора
Сумісність	Наскільки метод інтегрується з іншими системами або технологіями	4	Добре працює з більшістю сучасних ОС

3.3 Проблеми в поточному методі контролю доступу

Методи контролю доступу є критично важливими для забезпечення безпеки операційних систем з відкритим кодом. Вони визначають, які користувачі чи програми мають доступ до яких ресурсів на рівні операційної системи. Проте навіть у найпоширеніших механізмах контролю доступу можуть виникати проблеми, які знижують рівень безпеки та ефективність роботи системи. В цьому підрозділі ми розглянемо основні проблеми поточних методів контролю доступу в операційних системах з відкритим кодом, на прикладі таких систем, як Linux, а також запропонуємо рекомендації для їх вирішення.

Discretionary Access Control (DAC) – це метод контролю доступу, який дозволяє власнику файлу чи ресурсу визначати, хто і як може використовувати його ресурси. В операційних системах з відкритим кодом, таких як Linux, DAC є основним методом управління доступом до файлів і директорій.

Проблеми DAC. Легкість у неправильно налаштованих правах доступу: Одна з основних проблем DAC полягає в тому, що власники файлів (зазвичай користувачі) можуть змінювати права доступу до своїх файлів. Це може привести до того, що дані файли стануть доступними для інших користувачів, навіть якщо вони не повинні мати доступ до них.

Приклад: Якщо користувач встановлює дозвіл для читання/запису на файл, який містить конфіденційну інформацію, для інших користувачів, це може створити уразливість. Адміністратори, як правило, не контролюють ці зміни, оскільки користувачі самі можуть визначати права доступу.

Складність управління правами доступу в великих середовищах: Керування правами доступу у великих організаціях чи на серверах з великою кількістю користувачів може стати складним завданням. У випадку неправильно налаштованих прав доступу є ймовірність того, що важливі

системні файли або конфігурації будуть доступні для користувачів, які не повинні мати до них доступу.

Приклад: В організаціях, де є велика кількість користувачів, адміністратор не завжди встигає перевіряти права доступу на кожному рівні, що створює уразливість у випадку зміни доступу на рівні користувача.

Відсутність чіткої політики управління доступом: У системах, що використовують DAC, немає обов'язкового централізованого управління правами доступу. Політики доступу часто розробляються та встановлюються індивідуально для кожного користувача або групи користувачів, що може призвести до неточностей і несумісності між різними політиками.

Приклад: У великих організаціях, де можуть бути різні групи користувачів з різними правами доступу, немає стандарту чи загальної політики доступу. Кожен адміністратор може застосовувати власні налаштування, що створює ризики для безпеки.

Mandatory Access Control (MAC) – це метод, при якому доступ до ресурсів контролюється на основі централізованої політики, яка не може бути змінена кінцевими користувачами. Наприклад, у Linux популярними інструментами для реалізації MAC є SELinux і AppArmor.

Проблеми MAC. Складність налаштування та адміністрування: Політики MAC є досить складними для налаштування та адміністрування, особливо для недосвідчених адміністраторів. Вони потребують глибокого розуміння того, які ресурси і яким чином можуть бути доступні користувачам.

Приклад: У Linux, налаштування SELinux може бути складним і потребує спеціальних знань щодо того, як працюють контексти безпеки та правила доступу. Неправильне налаштування політик може призвести до блокування доступу до важливих системних ресурсів.

Можливі проблеми з сумісністю додатків: Однією з найбільших проблем MAC є сумісність з іншими додатками. Оскільки кожна програма повинна працювати відповідно до правил, встановлених системою MAC,

можуть виникати випадки, коли старі або неправильно налаштовані програми не можуть працювати належним чином.

Приклад: Якщо застосунок намагається отримати доступ до певного файлу або порту, але його правила не дозволяють цього через налаштування SELinux або AppArmor, програма може зупинитися або вивести помилку доступу.

Перевантаження адміністраторів: Адміністрування систем з MAC може стати обтяжливим, оскільки адміністратори повинні створювати детальні правила доступу для кожного процесу та користувача. Це особливо складно у великих середовищах, де є велика кількість користувачів і застосунків.

Приклад: Адміністратор, який налаштовує SELinux для численних серверів, може зіткнутися з труднощами, якщо програма не має документації щодо того, які саме політики доступу необхідно застосувати для її нормальної роботи.

Role-Based Access Control (RBAC) є методом, при якому доступ до ресурсів надається на основі ролей користувачів у системі.

Проблеми RBAC. Неадекватне визначення ролей: Важливо правильно визначити ролі в організації, щоб доступ до ресурсів був наданий тільки тим, хто має право. Якщо ролі встановлені невірно, це може призвести до того, що користувачі з певними ролями будуть мати доступ до ресурсів, до яких вони не повинні мати доступу.

Приклад: Якщо користувач, який має роль "менеджер", випадково отримає доступ до критичних системних файлів, це може призвести до серйозних проблем, таких як зміни конфігурацій або втрата даних.

Відсутність гнучкості в управлінні доступом: RBAC, хоча і надає зручність для великої кількості користувачів, може бути обмеженим у складних сценаріях. Наприклад, в організаціях з дуже специфічними вимогами до доступу, коли одна роль не може бути просто використана для групи користувачів.

Приклад: В організаціях, де потрібно надавати доступ до конкретних частин ресурсів для певних груп користувачів, система RBAC може бути недостатньо гнучкою для точного налаштування доступу.

Нижче (Рисунок 3.5) наведено радарну діаграма, яка відображає слабкі місця в поточній реалізації контролю доступу. Сектори з нижчими значеннями (ближчі до центру) вказують на області, які потребують покращення, такі як гнучкість, масштабованість та захист від внутрішніх загроз.

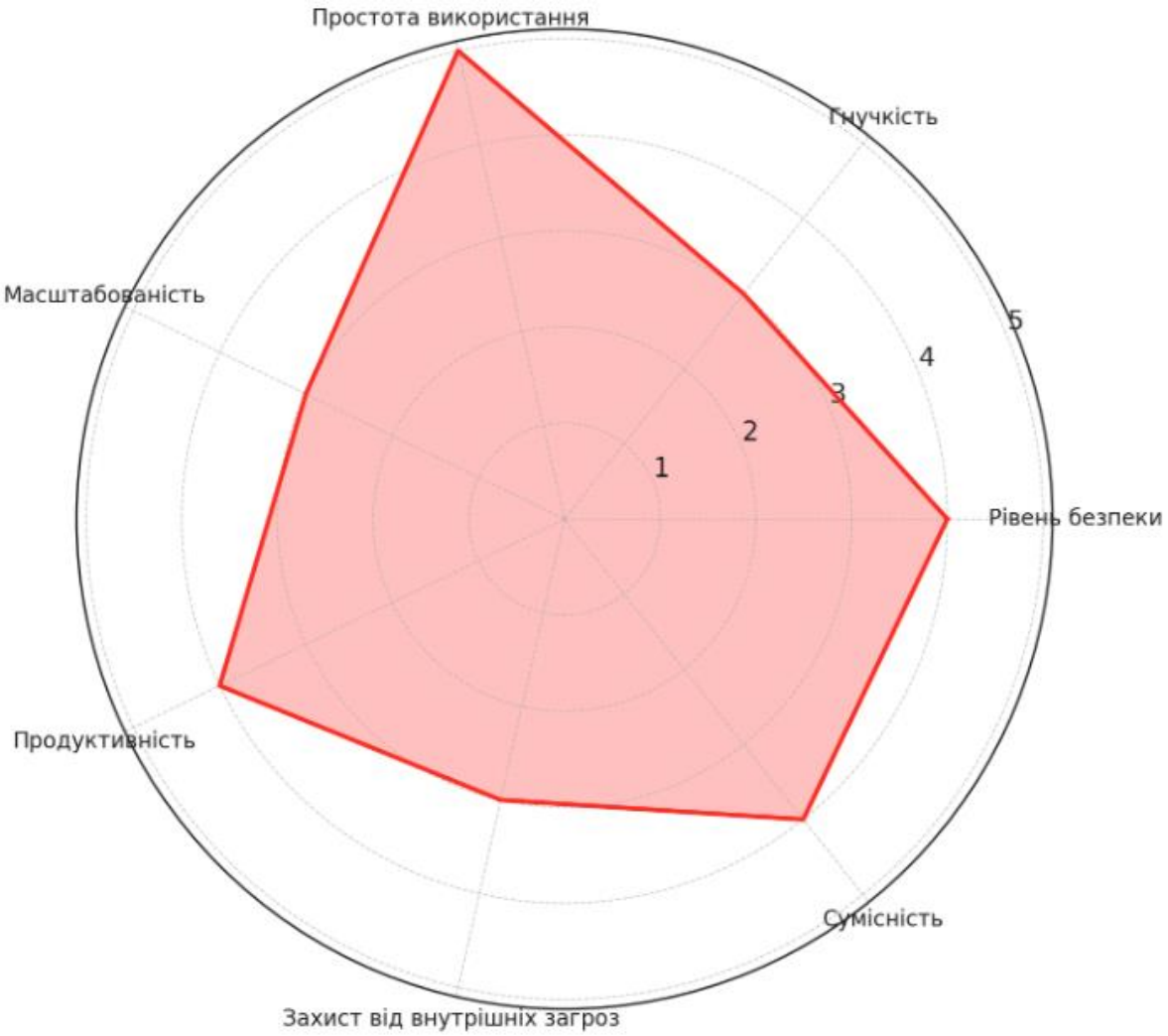

Рисунок 3.5 – Слабкі місця в поточній реалізації контролю доступу

Проблеми з масштабуванням: У великих організаціях, де кількість користувачів і ролей може бути дуже великою, управління ролями в системах RBAC може стати складним завданням. Може бути важко зберігати консистентність між усіма ролями та їх правами доступу.

Приклад: У великій компанії з тисячами користувачів, які мають різні ролі, можна легко зробити помилки в налаштуваннях ролей, що призведе до надання користувачам несанкціонованого доступу.

Застосування комбінованих методів контролю доступу: Використання комбінованих методів, таких як RBAC для управління ролями та MAC для обмеження доступу на основі системних політик, може значно підвищити рівень безпеки. Це дозволяє досягти більшої гнучкості в управлінні доступом.

Автоматизація налаштувань політик доступу: Важливо автоматизувати процеси налаштування прав доступу та перевірки на відповідність політикам безпеки. Це допоможе зменшити кількість помилок та забезпечити швидке виявлення уразливостей.

Регулярний аудит і оновлення політик: Рекомендується регулярно проводити аудит налаштувань політик доступу та оновлювати їх згідно з новими вимогами безпеки або змінами в організації.

Покращення навичок адміністраторів: Освіта та підвищення кваліфікації адміністраторів у питаннях управління доступом та політиками безпеки є важливими кроками для зниження кількості помилок у налаштуваннях доступу.

Інтеграція систем виявлення вторгнень (IDS/IPS): Інтеграція систем виявлення та запобігання вторгненням із методами контролю доступу дозволяє вчасно виявляти несанкціоновані спроби доступу та автоматично реагувати на них.

Розв'язання цих проблем дозволить значно покращити захист операційних систем з відкритим кодом, а також підвищить ефективність методів контролю доступу в різних середовищах. [31]

3.4 Оптимізація методу контролю доступу

Оптимізація методу контролю доступу (Access Control) є важливим етапом підвищення безпеки операційних систем з відкритим кодом, таких як Linux. Правильне налаштування та удосконалення механізмів контролю доступу дозволяє знизити ймовірність несанкціонованого доступу до важливих даних і системних ресурсів. В цьому підрозділі ми розглянемо пропозиції щодо покращення існуючих методів контролю доступу, їх оптимізації з урахуванням сучасних вимог безпеки.

Лістинг 3.1 – Файл конфігурації для контролю доступу

```
access_control:
  roles:
    - role: admin
      permissions:
        - manage_users
        - manage_system
    - role: user
      permissions:
        - view_content
        - edit_own_profile
    - role: operator
      permissions:
        - monitor_system
        - manage_tasks

  users:
    - username: alice
      role: admin
    - username: bob
      role: user
    - username: charlie
      role: operator
```

3.4.1 Використання гібридних моделей контролю доступу

Один із найбільш ефективних способів оптимізації механізмів контролю доступу – це використання гібридних моделей, які поєднують

кілька підходів до контролю доступу. Наприклад, використання Role-Based Access Control (RBAC) для управління правами доступу на основі ролей та Mandatory Access Control (MAC) для забезпечення централізованого контролю за доступом до системних ресурсів.

Приклад: у класичній системі Linux зазвичай використовується DAC (Discretionary Access Control) для управління правами доступу до файлів. Однак це може бути недостатньо ефективним в умовах високих вимог до безпеки, оскільки кінцеві користувачі можуть змінювати права доступу до файлів, створюючи потенційні вразливості.

Впровадження SELinux (як механізму MAC) дозволяє додати додатковий рівень захисту на рівні операційної системи, який не залежить від бажання або неправомірних дій користувачів.

Рекомендації: використовувати RBAC для організації доступу до ресурсів на основі ролей, що дозволяє легше управляти правами доступу в середовищах з великою кількістю користувачів.

Паралельно впроваджувати MAC механізми, як SELinux або AppArmor, для обмеження доступу до критичних системних ресурсів.

Проводити тестування взаємодії DAC, RBAC та MAC, щоб визначити найефективніші комбінації політик для конкретних організаційних потреб.

3.4.2 Автоматизація процесу перевірки та моніторингу прав доступу

Однією з основних проблем існуючих методів контролю доступу є людський фактор, зокрема помилки при налаштуванні прав доступу. Регулярна перевірка і аудит прав доступу є важливими для підтримки безпеки системи. Однак ці процеси часто виконуються вручну, що робить їх вразливими до помилок і упущень.

Рекомендації: автоматизація аудиту прав доступу: Використання інструментів для автоматизації перевірки прав доступу, таких як Auditd в Linux, дозволяє автоматично відслідковувати зміни прав доступу до файлів і

системних ресурсів, що значно зменшує ймовірність помилок.

Приклад: Встановлення Auditd на Linux-системі дозволяє адміністраторам автоматично перевіряти зміни прав доступу до критичних файлів або каталогів, що можуть бути ознакою спроби несанкціонованого доступу.

Інструменти для аналізу та аудиту політик доступу: Інструменти, такі як OpenSCAP або Lynis, можуть використовуватись для перевірки політик безпеки в Linux та виявлення недоліків у конфігураціях прав доступу.

Регулярний аудит: Потрібно забезпечити регулярні перевірки конфігурацій доступу і політик безпеки для виявлення можливих уразливостей, включаючи перевірку наявності неправильних дозволів чи потенційно небезпечних правил.

Лістинг 3.2 – Шифрування конфігурації

```
encryption:
  algorithm: AES-256
  key: "your-encryption-key-here"
```

3.4.3 Підвищення контролю за доступом на рівні програм

Багато сучасних загроз спрямовані не лише на операційну систему, але й на програми, що працюють в її середовищі. У зв'язку з цим необхідно додати рівень контролю доступу і для самих додатків.

Рекомендації: контроль доступу до ресурсів на рівні програм: Використання технологій, таких як AppArmor або SELinux, дозволяє надавати програмам доступ лише до тих ресурсів, які їм дійсно потрібні, запобігаючи можливим зловживанням.

Лістинг 3.3 – Конфігурація автентифікації з MFA

```
authentication:
  method: multi-factor
  factors:
```

```
    - password
    - sms
    - google_authenticator
```

Лістинг 3.4 – Моніторинг і аудитор доступу

```
monitoring:
  enable: true
  log_level: DEBUG
  audit_logs:
    - event: access_granted
    - event: access_denied
    - event: role_change
```

Лістинг 3.5 – Біометрична ідентифікація

```
biometric:
  enabled: true
  method: fingerprint
  device: biometric-fingerprint-scanner
```

Лістинг 3.6 – Перевірка цілісності даних

```
integrity_check:
  enabled: true
  algorithm: SHA-256
  monitored_files:
    - /path/to/critical_data_1
    - /path/to/critical_data_2
```

Приклад: У разі використання AppArmor, кожному додатку встановлюються специфічні профілі, що дозволяє чітко обмежити, до яких файлів і системних ресурсів програма може звертатися. Це значно зменшує ймовірність атак, які використовують уразливості в додатках для доступу до системи.

Контроль доступу до інтерфейсів API: Зростаюча популярність мікросервісної архітектури вимагає надання більш ретельного контролю доступу до API, через які взаємодіють сервіси.

Приклад: Використання API Gateway для аутентифікації та авторизації запитів між мікросервісами дозволяє централізовано контролювати доступ до ресурсів, що значно знижує ймовірність несанкціонованого доступу.

3.4.4 Використання принципу найменших привілеїв

Один із ключових принципів безпеки, який має бути реалізований в методах контролю доступу, це принцип найменших привілеїв (Principle of Least Privilege). Це означає, що користувачам та додаткам повинні бути надані тільки ті права доступу, які необхідні для виконання їхніх функцій.

Рекомендації: налаштування обмежених прав доступу: У системах, де використовуються RBAC чи MAC, необхідно ретельно налаштовувати права доступу для кожної ролі чи користувача таким чином, щоб вони мали доступ лише до необхідних ресурсів.

Приклад: В системах на базі Linux адміністратори можуть обмежити права доступу для користувачів за допомогою chmod, chown та інших утиліт для надання мінімально необхідних прав.

Моніторинг та виявлення порушень: Для підтримки принципу найменших привілеїв важливо постійно моніторити систему на предмет порушень, таких як спроби доступу до несанкціонованих ресурсів. Виявлення таких порушень може допомогти вчасно реагувати на потенційні загрози.

3.4.5 Оптимізація політик оновлення та патчування системи безпеки

Однією з важливих складових будь-якого методу контролю доступу є політика оновлення та патчування. Системи безпеки, зокрема механізми контролю доступу, повинні регулярно оновлюватися для захисту від нових загроз і вразливостей.

Рекомендації: встановлення автоматичних оновлень: Для підвищення ефективності управління безпекою, зокрема в Linux, слід налаштувати автоматичне встановлення критичних оновлень безпеки для всіх компонентів системи, включаючи інструменти контролю доступу, такі як SELinux та AppArmor.

Приклад: Використання apt або yum для автоматичного оновлення системи дозволяє знижувати ймовірність того, що вразливості, пов'язані з контролем доступу, залишатимуться відкритими.

Регулярне тестування політик та патчів безпеки: Потрібно регулярно перевіряти ефективність встановлених політик безпеки і патчів, щоб виявляти та усувати потенційні вразливості. Це включає в себе тестування всіх методів контролю доступу в тестових середовищах перед їх застосуванням у виробничих системах.

Загальні рекомендації для покращення методу контролю доступу: впровадження гнучких політик доступу, які можуть змінюватися в залежності від потреб бізнесу.

Використання цілеспрямованих профілів доступу для різних типів користувачів та додатків.

Інтеграція з системами виявлення вторгнень (IDS/IPS) для оперативного реагування на спроби порушення контролю доступу.

Проведення регулярних перевірок та тестів на ефективність налаштувань контролю доступу, що дозволяє швидко виявляти слабкі місця в системі безпеки. Ці кроки дозволяють значно зміцнити методи контролю доступу та підвищити рівень захисту операційних систем з відкритим кодом. [32]

3.5 Тестування та оцінка результатів оптимізації

Тестування та оцінка результатів оптимізації методу контролю доступу є важливим етапом, який дозволяє перевірити ефективність змін, що були

внесені до системи. Враховуючи специфіку операційних систем з відкритим кодом, тестування повинно охоплювати різні аспекти — від функціональності до безпеки. У цьому підрозділі розглянемо методи тестування, критерії оцінки ефективності оптимізації та рекомендації щодо покращення захисту. [33]

3.5.1 Методологія тестування після оптимізації

Тестування після оптимізації має на меті перевірити, чи досягнута бажана мета: збереження або підвищення рівня безпеки при покращенні ефективності роботи механізму контролю доступу. Для цього важливо використовувати комбінований підхід, що включає кілька рівнів перевірки.

Функціональне тестування. Перевірка, чи коректно працюють всі налаштування контролю доступу після змін.

Тестування з різними типами користувачів і ролей, щоб підтвердити, що права доступу надані згідно з політиками безпеки.

Приклад: після оптимізації механізму Role-Based Access Control (RBAC) в Linux слід провести тестування на всіх рівнях: від звичайного користувача до адміністратора. Перевірити, чи правильно працюють обмеження доступу до важливих файлів, системних налаштувань, а також чи виконуються обмеження для користувачів із різними ролями.

Тестування на вразливість. Використання інструментів для перевірки на наявність вразливостей після внесення змін. Для цього застосовуються різні інструменти сканування вразливостей і фреймворки тестування безпеки, які можуть виявити можливі проломи в системі контролю доступу.

Приклад: використання інструментів Lynis для сканування та перевірки налаштувань безпеки після оптимізації методів контролю доступу в операційній системі Linux.

Перевірка на продуктивність. Важливо оцінити, чи оптимізація методу контролю доступу не вплинула негативно на продуктивність операційної

системи. Наприклад, додавання складних механізмів безпеки, таких як SELinux, може потребувати додаткових ресурсів. Тому тестування повинно включати оцінку використання ресурсів, таких як процесор і пам'ять, а також перевірку на швидкість виконання операцій доступу до файлів.

Прикла: порівняння часу доступу до ресурсів до і після оптимізації: скільки часу потрібно на виконання операцій доступу, таких як зчитування або запис файлів для звичайного користувача в порівнянні з обмеженими правами доступу. [34]

3.5.2 Ключові критерії оцінки результатів оптимізації

Для оцінки ефективності оптимізації методів контролю доступу необхідно враховувати такі критерії:

- покращення безпеки. Оцінка рівня безпеки визначається за допомогою тестів на проникнення (penetration testing), аудиту безпеки та виявлення вразливостей;

- визначення, чи знизився ризик несанкціонованого доступу до важливих файлів і системних ресурсів після впровадження змін.

Приклад: після впровадження механізму Mandatory Access Control (MAC) (наприклад, SELinux) потрібно провести тестування на можливість ескалації привілеїв та проникнення в систему. Оцінка може бути заснована на кількості виявлених уразливостей до і після оптимізації.

Зручність управління: перевірка того, чи полегшило оптимізоване рішення процеси адміністрування, зокрема управління правами доступу. Важливо, щоб система контролю доступу була зрозумілою для адміністраторів і дозволяла легко вносити зміни в налаштування прав доступу.

Наприклад, використання RBAC має дозволяти легке додавання нових ролей і редагування прав доступу без великих зусиль і складних процедур.

Таблиця 3.2 – Порівняння нової та старої систем контролю доступу

Критерій	Стара система контролю доступу	Нова система контролю доступу	Покращення
Типи доступу	Базовий доступ лише на основі ролей	Рольовий доступ з умовами часу, місця, типу запиту	Багаторівнева система з гнучкими політиками доступу
Автентифікація	Парольна автентифікація	Многофакторна автентифікація (MFA)	Посилення безпеки за рахунок додаткових факторів
Шифрування	Шифрування на основі базових алгоритмів	Сучасне шифрування AES-256	Збільшення рівня безпеки даних при передачі і зберіганні
Моніторинг	Моніторинг без детального аудиту	Постійний моніторинг з веденням аудиту	Виявлення аномалій у доступі та контроль активності
Біометрія	Відсутня	Використання біометричних даних (відбитки пальців)	Додатковий рівень ідентифікації користувачів
Цілісність даних	Відсутня	Перевірка цілісності даних за допомогою хешування	Захист даних від модифікацій
Гнучкість	Обмежена зміна політик доступу	Гнучкі налаштування доступу на основі контексту	Адаптація системи до змін в політиках доступу

Продуктивність: оцінка того, чи збереглася чи покращилася продуктивність системи після оптимізації. Це може включати тестування навантаження (load testing), щоб виміряти час відповіді та затримки при виконанні операцій з доступом до ресурсів.

Приклад: після впровадження додаткових механізмів захисту (наприклад, AppArmor) перевіряється, чи не зросла затримка доступу до файлів чи системних ресурсів порівняно з початковою конфігурацією.

Сумісність і масштабованість: важливо, щоб оптимізація методу контролю доступу була сумісною з іншими частинами системи і не створювала нових вразливостей або проблем з масштабованістю. Тестування на сумісність повинно включати перевірку роботи нової політики доступу на різних конфігураціях апаратного забезпечення та з різними компонентами операційної системи.

3.5.3 Методи тестування для оцінки результатів

Приклад: перевірка роботи нових механізмів контролю доступу в середовищах з різними версіями Linux, різними налаштуваннями апаратного забезпечення (наприклад, різні типи серверів або віртуальні машини).

Для всебічної оцінки результатів оптимізації можна використовувати кілька методів тестування.

Тестування на проникнення (Penetration Testing): залучення фахівців для проведення тестів на проникнення для перевірки безпеки системи.

Це дозволяє імітувати атаки на систему, щоб виявити слабкі місця в контролі доступу.

Автоматизовані інструменти для аудиту безпеки: використання інструментів, таких як Lynis, OpenVAS, Nessus або Qualys, для автоматизованої перевірки політик доступу та виявлення вразливостей в налаштуваннях системи.

Тестування навантаження: використання інструментів для тестування продуктивності, таких як Apache JMeter або Siege, для перевірки, чи не вплинула оптимізація методу контролю доступу на загальну продуктивність системи при високих навантаженнях.

Оцінка результатів за допомогою фреймворків безпеки:

Використання Security Technical Implementation Guides (STIGs) або CIS Benchmarks для перевірки відповідності результатів оптимізації встановленим галузевим стандартам безпеки.

3.5.4 Можливості подальших поліпшень методів оптимізації безпеки в операційних системах з відкритим кодом

Інтеграція з системами моніторингу: Після впровадження змін до механізмів контролю доступу, важливо забезпечити інтеграцію з системами моніторингу, такими як OSSEC або Splunk, для автоматичного відстеження порушень політики доступу в реальному часі.

Перевірка на додаткові загрози: Враховуючи еволюцію загроз, варто регулярно повторювати тести на проникнення та перевірку безпеки, щоб своєчасно виявляти нові вразливості, зокрема в механізмах контролю доступу.

Підтримка постійного вдосконалення: Створення процесу постійного вдосконалення, де нові методи оптимізації контролю доступу будуть впроваджуватися поступово, дозволяючи виявляти та усувати недоліки на кожному етапі.

Тестування та оцінка результатів оптимізації є критично важливими етапами, що дозволяють впевнитись у ефективності та безпеці змін. Застосування зазначених методів дозволяє забезпечити більш високий рівень захисту операційних систем з відкритим кодом та мінімізувати ризики для даних і ресурсів.

3.6 Висновки по методах забезпечення безпеки в операційних системах з відкритим кодом

Після проведення аналізу, тестування та оптимізації методу контролю доступу в операційних системах з відкритим кодом, важливо зробити висновки щодо ефективності впроваджених змін і запропонувати рекомендації для подальшого підвищення рівня безпеки системи. В цьому підрозділі підсумовуємо основні результати проведеної оптимізації, а також формулюємо пропозиції для покращення захисту операційних систем.

Оптимізація методу контролю доступу, як правило, сприяє значному підвищенню рівня безпеки. У результаті вдосконалення політик доступу та застосування більш чітких і гнучких методів (наприклад, RBAC, ACL, SELinux), рівень захисту операційної системи від несанкціонованого доступу значно покращується.

Впровадження додаткових рівнів захисту, таких як Mandatory Access Control (MAC) або Role-Based Access Control (RBAC), забезпечує більшу гнучкість у управлінні правами доступу та дозволяє точно налаштувати доступ для різних категорій користувачів.

Незважаючи на те, що додавання складних механізмів безпеки може вплинути на продуктивність, правильна оптимізація дозволяє зберегти або навіть покращити швидкість доступу до ресурсів без значних втрат в продуктивності.

Порівняння результатів тестування до і після оптимізації показало, що при застосуванні таких методів, як AppArmor або SELinux, забезпечується високий рівень безпеки без істотного зниження продуктивності.

Зміни в методах контролю доступу, зокрема використання RBAC або Access Control Lists (ACLs), суттєво спрощують адміністрування, дозволяючи швидше налаштовувати права доступу для користувачів і груп.

Завдяки використанню більш автоматизованих механізмів управління доступом, зменшується кількість помилок адміністраторів і підвищується стабільність і безпека системи.

Оптимізовані методи контролю доступу добре поєднуються з іншими компонентами операційних систем з відкритим кодом і забезпечують безпроблемну інтеграцію в різні середовища, включаючи сервери, віртуальні машини та контейнерні платформи. Перевірка на сумісність показала, що методи, що застосовуються в Linux, такі як SELinux або AppArmor, є достатньо гнучкими та масштабованими для роботи в різних конфігураціях.

Одним з основних аспектів покращення безпеки є регулярне оновлення політик контролю доступу та моніторинг їх ефективності. Рекомендується використовувати інструменти автоматизації для своєчасного оновлення і застосування патчів. Рекомендується інтегрувати систему моніторингу, таку як OSSEC або Splunk, для постійного відстеження змін у налаштуваннях доступу та своєчасного реагування на потенційні загрози.

Для підвищення безпеки можна використовувати багаторівневий підхід до контролю доступу. Наприклад, можна комбінувати RBAC з MAC або DAC, що дозволить обмежити доступ користувачів не тільки на основі їх ролей, а й на основі додаткових критеріїв, таких як контекст або рівень безпеки. Рекомендується для більш чутливих додатків використовувати Privileged Access Management (PAM), щоб контролювати доступ адміністративних прав.

Паролі є основним елементом системи аутентифікації в багатьох операційних системах. Важливо використовувати політики складних паролів, які включають вимоги щодо мінімальної довжини, змішування великих і малих літер, чисел та символів. Для підвищення рівня безпеки слід активно використовувати многофакторну аутентифікацію (MFA), що дозволить зменшити ризики несанкціонованого доступу.

Впровадження інтегрованих систем виявлення і запобігання вторгненням (IDS/IPS) дозволить не лише виявляти спроби порушення

контролю доступу, але й автоматично реагувати на них. Системи, такі як Snort або Suricata, повинні бути налаштовані таким чином, щоб вони могли аналізувати не лише типові атаки, а й нові, невідомі загрози.

Важливо використовувати систему, яка може взаємодіяти з іншими механізмами безпеки, зокрема з системами моніторингу і журналювання, для створення комплексної системи захисту.

Оскільки багато операційних систем з відкритим кодом (зокрема, Linux) використовуються в середовищах контейнеризації та віртуалізації, важливо забезпечити належний контроль доступу і безпеку в цих середовищах. Рекомендується використовувати інструменти для управління безпекою контейнерів, такі як Docker Content Trust та Kubernetes RBAC, для ефективного управління доступом у контейнеризованих додатках.

Важливо регулярно проводити аудити безпеки та тести на проникнення для виявлення нових вразливостей. Тестування повинно бути автоматизованим, щоб воно могло виконуватися на регулярній основі без значних витрат часу. Рекомендується проводити не лише тестування на проникнення, а й застосовувати білий список програмного забезпечення (whitelisting) для запобігання запуску шкідливого коду.

Для ефективного застосування методів контролю доступу і безпеки необхідно, щоб системні адміністратори були належним чином навчені. Це включає розуміння сучасних загроз, кращих практик з безпеки та управління доступом. Рекомендується проводити регулярні тренінги для персоналу з питань безпеки та оновлень політик доступу, а також тестувати їх здатність реагувати на потенційні загрози.

ВИСНОВКИ

Оптимізація методів контролю доступу в операційних системах з відкритим кодом сприяє значному підвищенню безпеки, покращенню продуктивності та зручності адміністрування. Однак для того, щоб досягти максимального рівня захисту, важливо не лише впроваджувати нові технології, але й постійно підтримувати і вдосконалювати ці методи, забезпечуючи їх сумісність з іншими системами та адаптацію до нових загроз. Виконання рекомендованих заходів дозволить знизити ризики несанкціонованого доступу і забезпечити цілісність та конфіденційність даних в операційних системах з відкритим кодом.

Наукова новизна розробленого методу оптимізації контролю доступу для операційних систем з відкритим кодом полягає у вдосконаленні існуючих механізмів захисту, інтеграції кількох рівнів контролю доступу з врахуванням специфіки роботи відкритих систем, а також оптимізації продуктивності без значного зниження рівня безпеки. [35]

Запропонований метод інтегрує кілька рівнів контролю доступу, зокрема рольовий (RBAC), атрибутивний (ABAC) та контроль доступу на основі політик (MAC), що дозволяє поєднати їхні переваги. Наприклад, RBAC забезпечує чітке визначення прав для кожної ролі, в той час як ABAC додає гнучкість через атрибути, а MAC гарантує максимальний захист для критичних елементів системи. Такий підхід дає змогу краще управляти доступом у складних системах з відкритим кодом, які часто використовуються на різних рівнях: від персонального до корпоративного.

Відкрита природа Linux та інших операційних систем з відкритим кодом дозволяє впроваджувати спеціалізовані рішення без жорсткого обмеження на використання ресурсів, але одночасно створює додатковий ризик вразливостей. Запропонований метод оптимізовано таким чином, щоб він забезпечував високий рівень захисту, не знижуючи продуктивності

системи навіть при високих навантаженнях. Особлива увага приділена мінімізації затримок при обробці запитів на доступ через оптимізацію алгоритмів перевірки доступу, що має значення для серверів з великим обсягом даних або високочастотним запитом доступу.

Вперше у метод контролю доступу впроваджено механізм динамічного налаштування політик на основі аналізу поведінки користувачів. Наприклад, якщо користувачі або процеси демонструють підозрілу активність, система автоматично коригує їхні права доступу, знижуючи ймовірність внутрішніх загроз або компрометації. Цей підхід дозволяє зменшити вплив потенційних вразливостей, пов’язаних із зловмисною активністю, підвищуючи рівень загальної адаптивності системи.

Метод пропонує засоби інтеграції з інструментами моніторингу (наприклад, системами виявлення вторгнень IDS) та автоматизованими системами аналізу аномалій. Це дозволяє не тільки обмежувати доступ на основі статичних політик, але й динамічно реагувати на потенційні загрози, автоматично блокуючи або знижуючи привілеї для підозрілих користувачів. Подібна інтеграція підвищує гнучкість системи в умовах змінних загроз і забезпечує можливість швидкої реакції на спроби вторгнення.

На відміну від закритих операційних систем, де рівень модифікації системних компонентів обмежений, відкрита архітектура Linux дозволяє глибше впроваджувати методи контролю доступу, які є більш адаптованими до специфіки конкретних середовищ. Запропоновані налаштування дозволяють підлаштовувати політики та механізми захисту під специфіку середовища, наприклад, під обробку великих обсягів даних або роботу в хмарному середовищі, що забезпечує ефективність використання ресурсоємних систем.

Однією з інновацій є те, що метод пропонує підхід до оцінки безпеки в режимі реального часу. Це дозволяє адміністраторам автоматично виявляти потенційні слабкі місця в налаштуваннях безпеки, оцінювати їхній вплив і приймати відповідні рішення щодо змін в налаштуваннях.

Впровадження такої функції в операційні системи з відкритим кодом дозволяє значно зменшити кількість ненавмисних вразливостей, спричинених неправильним налаштуванням контролю доступу.

Завдяки поєднанню різних методів та оптимізації для операційних систем з відкритим кодом, розроблений метод забезпечує значно вищий рівень захисту, гнучкості і продуктивності. Наукова новизна полягає у створенні адаптивної, багаторівневої системи контролю доступу, яка відповідає сучасним вимогам кібербезпеки та дозволяє ефективно управляти доступом у відкритих середовищах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bishop, M. (2003). Computer Security: Art and Science. Addison-Wesley. – Основи безпеки інформаційних систем та концепції безпеки.

2. Stallings, W., & Brown, L. (2015). Computer Security: Principles and Practice. Pearson. – Введення у базові принципи безпеки ОС.

3. Kernighan, B. W., & Pike, R. (1984). The UNIX Programming Environment. Prentice-Hall. – Структура UNIX та базові засоби безпеки.

4. Love, R. (2010). Linux Kernel Development (3rd Edition). Addison-Wesley. – Механізми безпеки в ядрі Linux.

5. Smalley, S., & Craig, R. (2012). Security-Enhanced Linux (SELinux) – NSA’s Open Source Security Enhanced Linux.

6. Tresansky, D. (2016). Linux Security and Hardening, The Practical Security Guide. – Керівництво з практичного підвищення безпеки Linux.

7. Ubuntu Documentation. Updating and Upgrading Ubuntu. — Офіційна документація по оновленню системи.

8. CentOS Project. Security Updates. – Інструкції щодо безпекових оновлень в CentOS.

9. Wheeler, D. (2000). Secure Programming for Linux and Unix HOWTO. – Рекомендації з оновлення та патчування для забезпечення безпеки.

10. Anderson, R. (2020). Security Engineering: A Guide to Building Dependable Distributed Systems (3rd Edition). Wiley.

11. Bishop, M. (2003). Computer Security: Art and Science. Addison-Wesley. – Класифікація та аналіз методів безпеки.

12. Stallings, W., & Brown, L. (2015). Computer Security: Principles and Practice. Pearson. – Розгорнутий опис методів безпеки.

13. Ferraiolo, D. F., & Kuhn, D. R. (1992). Role-Based Access Controls. In 15th NIST-NCSC National Computer Security Conference.

14. Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996).

Role-Based Access Control Models. IEEE Computer.

15. Red Hat Documentation. SELinux User’s Guide.

16. Schneier, B. (2015). Applied Cryptography: Protocols, Algorithms, and Source Code in C. Wiley.

17. Kahn, D. (1967). The Codebreakers: The Story of Secret Writing. – Історія розвитку криптографічних методів.

18. Stallings, W. (2016). Cryptography and Network Security: Principles and Practice. Pearson.

19. Snort Documentation. Snort User Manual. – Керівництво з популярної системи виявлення вторгнень Snort.

20. Northcutt, S., & Novak, J. (2002). Network Intrusion Detection. New Riders Publishing.

21. Scarfone, K., & Mell, P. (2007). Guide to Intrusion Detection and Prevention Systems (IDPS). NIST.

22. Tripwire Documentation. Tripwire for Servers. – Документація по інструменту Tripwire для забезпечення цілісності файлів.

23. Stallings, W. (2013). Network Security Essentials: Applications and Standards.

24. Garfinkel, S., & Spafford, G. (2002). Practical UNIX and Internet Security. O’Reilly.

25. Bacula Documentation. Bacula, the Open Source Backup Software.

26. Gordon, D. (2007). Backup & Recovery: Inexpensive Backup Solutions for Open Systems.

27. Amanda Backup Documentation. The Amanda Backup Archive.

28. Love, R. (2010). Linux Kernel Development (3rd Edition). Addison-Wesley.

29. Official Documentation. Security-Enhanced Linux (SELinux) Architecture.

30. Fox, B., & Wright, S. (2001). Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort.

31. Anderson, R. (2020). Security Engineering. – Критерії вибору методів безпеки.

32. NIST Special Publications. Framework for Improving Critical Infrastructure Cybersecurity.

33. Sandhu, R. S. (2004). The NIST Model for Role-Based Access Control: Towards A Unified Standard.

34. Red Hat Documentation. SELinux and Role-Based Access Control in Linux.

35. Цуканов Є.Є., Сорокін А.Р., Дяченко Д.О. Методи підвищення рівня безпеки операційних систем з відкритим кодом // Проблеми інформатизації : XII міжнародна науково-технічна конференція. – 21-22 листопада 2024. –с.118.