

Держ Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ЗАТВЕРДЖУЮ

Завідувач кафедри

комп'ютерної інженерії

та кібербезпеки

Сфименко А. А.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи
магістра на тему:

«Методи та інструменти побудови систем реагування на інциденти в мережі на основі програмного стеку ELK»

студента Голуб О.

затверджено Наказом по університету від 6 жовтня 2025 р. №566/с Термін здачі студентом закінченої роботи 26 грудня 2025 р.

Вихідні дані для роботи (визначається предмет і об'єкт дослідження):

Предметом дослідження є моделі

моніторингу та реагування на мережні інциденти на основі програмного стеку ELK та Wazuh. Об'єктом дослідження є процес виявлення та реагування на інциденти інформаційної безпеки в мережах.

Консультанти з випускної кваліфікаційної роботи із зазначеної теми, що їх стосуються

Виконав студент 2-го курсу, групи ЗКБм-24-1 спеціальності 125 «Кібербезпека та захист інформації»

Олексій ГОЛУБ

Старший викладач кафедри КІ та КБ,

Вадим МИКОЛАЙЧУК

Рецензент, завідувач кафедри комп'ютерної інженерії та кібербезпеки, к.т.н, доцент,

Андрій СФІМЕНКО

Розділ	Консультант	Дата
1	Ігор Фальковський	06.10.2025
2	Ігор Фальковський	10.10.2025
3	Ігор Фальковський	07.11.2025

Житомир – 2025

Державний університет «Житомирська політехніка»

Факультет інформаційно-комп'ютерних технологій

Кафедра комп'ютерної інженерії та кібербезпеки

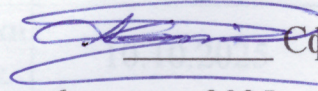
Спеціальність 125 «Кібербезпека та захист інформації»

Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач кафедри

комп'ютерної інженерії
та кібербезпеки

 Сфіменко А. А.
6 жовтня 2025 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу

Студента: **Голуба Олексія Дмитровича** Тема роботи: **Методи та інструменти побудови систем реагування на інциденти в мережі на основі програмного стеку ELK**

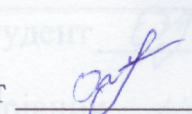
затверджена Наказом по університету від 6 жовтня 2025 р. №566/с Термін здачі студентом закінченої роботи 26 грудня 2025 р.

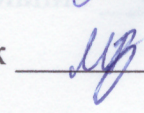
Вихідні дані для роботи (зазначається предмет і об'єкт дослідження):

Предметом дослідження є моделі, методи та програмні засоби побудови системи моніторингу та реагування на мережеві інциденти на основі програмного стеку ELK та Wazuh. Об'єктом дослідження є процес виявлення та реагування на інциденти інформаційної безпеки в комп'ютерних мережах.

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Ігор Фальковський	06.10.2025	06.10.2025
2	Ігор Фальковський	10.10.2025	10.10.2025
3	Ігор Фальковський	07.11.2025	07.11.2025

Студент  Олексій ГОЛУБ

Керівник  Вадим МИКОЛАЙЧУК

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	06.10.2025	Виконано
2	Пошук, огляд та аналіз літературних джерел щодо стандартів реагування на інциденти (NIST, ISO/IEC 27035)	08.10.2025	Виконано
3	Аналіз принципів побудови систем моніторингу та класифікація інцидентів інформаційної безпеки	06.10.2025	Виконано
4	Дослідження проблем виявлення та обробки інцидентів у сучасних корпоративних мережах	10.10.2025	Виконано
5	Порівняльний аналіз архітектур сучасних SIEM-систем та можливостей стеку ELK	19.10.2025	Виконано
6	Розробка теоретичної моделі системи реагування та механізмів кореляції подій	05.11.2025	Виконано
7	Проектування моделі взаємодії компонентів (Beats, Logstash, Elasticsearch, Kibana, Wazuh)	07.11.2025	Виконано
8	Практична реалізація системи у віртуальному середовищі (GNS3, Docker)	17.11.2025	Виконано
9	Налаштування збору, нормалізації подій безпеки та побудова дашбордів моніторингу	25.11.2025	Виконано
10	Аналіз ефективності моделі за критеріями швидкості реагування, точності та навантаження	02.12.2025	Виконано
11	Розробка сценаріїв реагування на інциденти (Playbooks) та інтеграція з MITRE ATT&CK	09.12.2025	Виконано
12	Аналіз перспектив інтеграції системи з платформами SOAR	12.12.2025	Виконано
13	Написання пояснювальної записки	15.12.2025	Виконано
14	Підготовка презентації та виступу	22.12.2025	Виконано
15	Отримання відгуку керівника та рецензії на кваліфікаційну роботу	23.12.2025	Виконано
16	Захист	26.12.2025	Виконано

Студент ОГ Олексій ГОЛУБ

Керівник МБ Вадим МИКОЛАЙЧУК

РЕФЕРАТ

Магістерська робота містить 102 сторінки, 15 ілюстрацій, 6 таблиць, список використаних джерел із 30 найменувань.

Метою роботи є підвищення рівня захищеності корпоративних мереж шляхом розробки та впровадження методики автоматизованого налаштування конфігурацій безпеки мережевого обладнання.

В ході дослідження проведено аналіз сучасних принципів побудови захищених мереж та виявлено критичні недоліки традиційних методів ручного адміністрування. Особлива увага була приділена проблемам людського фактору, масштабованості та забезпечення відповідності стандартам безпеки (ISO 27001, NIST). Висвітлено важливість переходу до парадигми «Інфраструктура як код» (IaC) для мінімізації ризиків та прискорення реакції на інциденти. Результатом цього етапу є обґрунтування необхідності впровадження систем оркестрації.

Проведено порівняльний аналіз існуючих інструментів автоматизації (Puppet, Chef, SaltStack) та обґрунтовано вибір платформи Ansible. Розроблено комплексну методику, що включає ієрархічну модель інвентаризації, модульну структуру ролей безпеки та механізми захищеного зберігання даних. Останній етап дослідження включає практичну реалізацію системи на базі ОС Rocky Linux та мережевого обладнання Cisco. Реалізовано сценарії захисту площини управління, контролю доступу та автоматичного відновлення після збоїв (Rollback). Доведено економічну ефективність запропонованого рішення та його здатність забезпечити надійний захист інформаційної інфраструктури.

Ключові слова: АВТОМАТИЗАЦІЯ, ANSIBLE, МЕРЕЖЕВА БЕЗПЕКА, КОНФІГУРАЦІЯ, INFRASTRUCTURE AS CODE.

ABSTRACT

The master's thesis contains 102 pages, 15 illustrations, 6 tables, and a list of references consisting of 30 titles.

The purpose of the work is to increase the security level of corporate networks by developing and implementing a methodology for automated configuration of network equipment security.

During the research, an analysis of modern principles of building secure networks was conducted, and critical shortcomings of traditional manual administration methods were identified. Special attention was paid to the problems of human error, scalability, and compliance with security standards (ISO 27001, NIST). The importance of transitioning to the "Infrastructure as Code" (IaC) paradigm to minimize risks and accelerate incident response was highlighted. The result of this stage is the justification of the need to implement orchestration systems.

A comparative analysis of existing automation tools (Puppet, Chef, SaltStack) was carried out, and the choice of the Ansible platform was justified. A comprehensive methodology was developed, including a hierarchical inventory model, a modular structure of security roles, and secure data storage mechanisms. The final stage of the research involves the practical implementation of the system based on Rocky Linux OS and Cisco network equipment. Scenarios for management plane protection, access control, and automatic recovery after failures (Rollback) were implemented. The economic efficiency of the proposed solution and its ability to ensure reliable protection of information infrastructure have been proven.

Keywords: AUTOMATION, ANSIBLE, NETWORK SECURITY, CONFIGURATION, INFRASTRUCTURE AS CODE.

ЗМІСТ

ВСТУП	7
1. АНАЛІЗ ПРИНЦИПІВ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ МЕРЕЖЕВОГО ОБЛАДНАННЯ ТА ПРОБЛЕМ ЙОГО КОНФІГУРУВАННЯ	8
1.1. Роль мережевого обладнання у забезпеченні інформаційної безпеки підприємства.....	8
1.2. Загрози та вразливості апаратного та програмного забезпечення мережесих пристроїв	14
1.3. Методи та засоби захисту мережевої інфраструктури (ACL, VPN, Firewall, IDS/IPS, сегментація, Zero Trust)	22
1.4. Проблеми ручного налаштування безпекових політик та потреба в автоматизації	30
1.5. Нормативні вимоги та рекомендації щодо конфігурування безпечних мережесих систем	34
2. АНАЛІЗ ТЕХНОЛОГІЙ АВТОМАТИЗАЦІЇ ТА РОЗРОБКА МЕТОДИКИ НАЛАШТУВАННЯ КОНФІГУРАЦІЇ БЕЗПЕКИ МЕРЕЖЕВОГО ОБЛАДНАННЯ ЗАСОБАМИ ANSIBLE.....	40
2.1. Огляд сучасних інструментів автоматизації адміністрування мереж (Ansible, Puppet, Chef, SaltStack).....	40
2.2. Порівняльний аналіз технологій з точки зору автоматизації налаштувань безпеки	46
2.3. Архітектура, принципи роботи і можливості Ansible у мережесій автоматизації	53
2.4. Вимоги до системи автоматизованого налаштування конфігурацій безпеки.....	66
2.5. Розробка методики автоматизованого конфігурування мережесих обладнання	69
2.6. Модель взаємодії компонентів системи Ansible (Inventory, Playbooks, Roles, Templates).....	75
3. РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ МЕТОДИКИ АВТОМАТИЗОВАНОГО КОНФІГУРУВАННЯ МЕРЕЖЕВОЇ БЕЗПЕКИ.....	79
3.1. Розробка та реалізація playbooks для налаштування безпеки мережесих обладнання	79
3.2. Автоматизація налаштувань контролю доступу, журналювання, аутентифікації та моніторингу	81
3.3. Тестування коректності та надійності автоматизованих конфігурацій.....	86
3.4. Порівняльна оцінка ефективності ручного та автоматизованого підходів.....	89
3.5. Забезпечення масштабованості, повторного використання та відновлюваності конфігурацій	93
ВИСНОВКИ	99
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	100
ДОДАТКИ.....	102

ВСТУП

Сучасні корпоративні мережі, які раніше виконували виключно функцію передачі даних, трансформувалися у критично важливий актив, що забезпечує безперервність бізнес-процесів та функціонування цифрової економіки. Однак разом із зростанням складності та гетерогенності мереж, експоненціально зросли і загрози їх інформаційній безпеці.

В останні роки проблема ручного адміністрування мережевого обладнання стала дедалі актуальнішою. Традиційні підходи до налаштування, що залежать від людського фактору, призводять до помилок конфігурації (misconfigurations), які, за статистикою, є причиною більшості успішних кібератак. Це охоплює залишені стандартні паролі, незакриті порти управління, неактуальні списки контролю доступу та відсутність своєчасного оновлення мікропрограмного забезпечення, що створює критичні вразливості в периметрі захисту.

Мета роботи полягає у підвищенні рівня захищеності корпоративних мереж шляхом дослідження методів автоматизації та розробки методики автоматизованого налаштування конфігурацій безпеки мережевого обладнання з використанням платформи Ansible.

Об'єктом дослідження є процеси управління конфігураціями та забезпечення інформаційної безпеки в сучасних телекомунікаційних мережах. Розглянуто методи та засоби мінімізації людського фактору при налаштуванні обладнання, що дозволяють гарантувати конфіденційність, цілісність та доступність інформаційних ресурсів.

Предметом дослідження є методики, алгоритми та програмні засоби автоматизованого налаштування параметрів безпеки мережевих пристроїв на базі технології Ansible.

Методологія роботи полягає в розробці практичних рекомендацій та програмного коду для автоматизації типових задач адміністраторів безпеки. Визначено архітектурні підходи до побудови масштабованих систем управління, що враховують вимоги міжнародних стандартів безпеки та дозволяють адаптуватися до динамічних змін у ландшафті кіберзагроз.

РОЗДІЛ 1. АНАЛІЗ ПРИНЦИПІВ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ МЕРЕЖЕВОГО ОБЛАДНАННЯ ТА ПРОБЛЕМ ЙОГО КОНФІГУРУВАННЯ

1.1. Роль мережевого обладнання у забезпеченні інформаційної безпеки підприємства

У сучасних умовах розвитку цифрової інфраструктури інциденти інформаційної безпеки перетворилися на один із найважливіших факторів, що визначають стійкість інформаційних систем до зовнішнього та внутрішнього впливу. Стрімке зростання кількості кіберзагроз, поява нових моделей атак, розширення поверхні ураження корпоративних мереж, активне впровадження хмарних платформ і сервісів, підключення мобільних пристроїв та елементів «інтернету речей» формують багатовимірне середовище, в якому будь-яка подія, пов'язана з порушенням правил функціонування системи, може набути характеру інциденту.

Сьогоднішній цифровий простір характеризується експоненціальним зростанням обсягів даних та критичною залежністю бізнес-процесів від інформаційно-комунікаційних технологій, роль мережевого обладнання трансформувалася з суто утилітарної функції передачі пакетів даних у фундамент комплексної системи інформаційної безпеки підприємства. Мережева інфраструктура більше не розглядається як пасивне середовище, прозоре для користувачів та додатків; натомість вона виступає активним учасником процесів захисту, виконуючи функції першої лінії оборони, моніторингу, контролю доступу та ізоляції загроз. Забезпечення конфіденційності, цілісності та доступності інформації неможливе без глибокої інтеграції механізмів безпеки безпосередньо в архітектуру апаратного забезпечення – маршрутизаторів, комутаторів, точок доступу та спеціалізованих шлюзів безпеки. Кожен елемент мережі, незалежно від його рівня в ієрархічній моделі OSI (Open Systems Interconnection), несе відповідальність за певний сегмент захисту, створюючи ешелоновану систему оборони, здатну протистояти як зовнішнім кібератакам, так і внутрішнім порушенням політик безпеки [1,16,17].

Фундаментальна роль мережевого обладнання у забезпеченні безпеки полягає у фізичному та логічному розмежуванні інформаційних потоків, що є базовою умовою для реалізації будь-якої політики безпеки. На рівні каналного доступу (Layer 2) комутатори корпоративного рівня забезпечують не лише комутацію кадрів, але й контроль цілісності сегментів локальної мережі. Саме на цьому етапі відбувається первинна верифікація підключених пристроїв та користувачів. Використання технологій управління доступом на рівні портів, таких як стандарт IEEE 802.1X, дозволяє перетворити звичайний комутатор на інтелектуальну точку входу, яка відмовляє у доступі неавторизованим суб'єктам ще до того, як вони отримають IP-адресу та зможуть взаємодіяти з іншими ресурсами мережі. Це критично важливо для запобігання несанкціонованому підключенню стороннього обладнання, що є одним із найпоширеніших векторів атак у погано контрольованих фізичних середовищах. Крім того, механізми захисту на рівні L2, такі як DHCP Snooping, Dynamic ARP Inspection та IP Source Guard, нівелюють ризики атак типу Man-in-the-Middle (MITM) та спуфінгу, які можуть призвести до перехоплення конфіденційної інформації всередині периметра.

Переходячи на мережевий рівень (Layer 3), роль маршрутизаторів та шлюзів стає визначальною у контексті фільтрації трафіку та маршрутизації даних. Маршрутизатори виступають вузлами прийняття рішень, визначаючи не лише оптимальний шлях доставки пакетів, але й легітимність самої передачі даних між різними сегментами мережі. Застосування списків контролю доступу (Access Control Lists – ACL) дозволяє адміністраторам чітко регламентувати, які протоколи, порти та хости мають право на взаємодію. Це створює бар'єри для розповсюдження шкідливого програмного забезпечення, такого як мережеві черв'яки, які намагаються сканувати мережу та інфікувати вразливі системи. Більше того, сучасні маршрутизатори підтримують функції глибокої інспекції пакетів та інтеграцію з системами виявлення вторгнень, що дозволяє аналізувати не лише заголовки пакетів, але й їхній вміст, виявляючи аномалії та сигнатури відомих атак у реальному часі. Таким чином, мережеве обладнання трансформується з засобу транспортування у засіб сегрегації та контролю, що є необхідним для реалізації концепції найменших привілеїв [9].

Важливим аспектом ролі мережевого обладнання є забезпечення відмовостійкості та доступності сервісів, що є однією з трьох складових тріади інформаційної безпеки (CIA). Атаки типу "відмова в обслуговуванні" (DoS/DDoS) спрямовані саме на виснаження ресурсів мережевої інфраструктури або пропускної здатності каналів зв'язку. У цьому контексті мережеві пристрої повинні володіти достатніми апаратними потужностями та програмними алгоритмами для розпізнавання та пом'якшення (mitigation) таких атак. Механізми Quality of Service (QoS), спочатку розроблені для пріоритезації мультимедійного трафіку, у контексті безпеки використовуються для гарантування доступності критично важливих сервісів навіть в умовах аномального навантаження або атаки. Обмеження швидкості (rate limiting) та полісинг трафіку на інтерфейсах дозволяють локалізувати джерело флуду та запобігти колапсу всієї мережевої інфраструктури, зберігаючи працездатність бізнес-додатків. Відтак, здатність мережевого обладнання протистояти перевантаженням безпосередньо впливає на безперервність діяльності підприємства.

Окремо слід виділити роль мережевого обладнання у створенні захищених каналів зв'язку при передачі даних через незахищені середовища, такі як Інтернет. Використання технологій віртуальних приватних мереж (VPN) на базі протоколів IPsec або SSL/TLS, реалізованих апаратно на маршрутизаторах та концентраторах, забезпечує конфіденційність та цілісність інформації, що передається між філіями підприємства або віддаленими працівниками [18,19]. Апаратна акселерація шифрування, яка є стандартом для сучасного корпоративного обладнання, дозволяє здійснювати криптографічні перетворення на швидкості лінії без суттєвих затримок для користувачів. Це перетворює мережеві пристрої на криптографічні шлюзи, які формують довірену зону поверх недовіреної інфраструктури провайдерів. Без надійно налаштованого криптографічного захисту на рівні мережевого обладнання будь-яка передача корпоративних даних за межі фізичного офісу становить неприйнятний ризик витоку інформації.

Еволюція мережевих технологій призвела до появи концепції "мережа як сенсор" (Network as a Sensor). Сучасне обладнання генерує величезні обсяги телеметричної інформації (NetFlow, IPFIX, sFlow), яка містить детальні метадані

про кожен потік трафіку в мережі. Аналіз цієї інформації системами моніторингу безпеки (SIEM) дозволяє виявляти складні цільові атаки (APT), які можуть залишатися непоміченими для традиційних антивірусних засобів. Мережеві пристрої, таким чином, виступають джерелами критично важливих даних для розслідування інцидентів та проведення форензики. Вони дозволяють реконструювати хронологію подій, ідентифікувати джерело атаки, шляхи її поширення та обсяг скомпрометованих даних. Відсутність належного збору та аналізу логів з мережевого обладнання робить підприємство "сліпим" перед лицем кіберзагроз, позбавляючи службу безпеки можливості оперативно реагувати на інциденти.

Крім того, варто зазначити, що мережеве обладнання є ключовим елементом у реалізації політик сегментації мережі. Плоскі мережі, де всі пристрої знаходяться в одному широкомовному домені, є вкрай вразливими, оскільки компрометація одного хоста дає зловмиснику прямий доступ до всіх інших ресурсів. Використання технологій VLAN (Virtual Local Area Network) та VRF (Virtual Routing and Forwarding) дозволяє логічно розділити мережу на ізольовані зони безпеки відповідно до функціонального призначення відділів або критичності оброблюваної інформації. Наприклад, відділ бухгалтерії, розробники, гостьовий Wi-Fi та сервери баз даних повинні знаходитися в різних сегментах, взаємодія між якими суворо контролюється міжмережевими екранами або списками контролю доступу на маршрутизаторах. Така мікросегментація значно звужує поверхню атаки та ускладнює латеральне переміщення зловмисника, змушуючи його долати численні бар'єри контролю при спробі розширити свій вплив у мережі.

Для повного розуміння архітектури безпеки необхідно детально проаналізувати функціональний внесок кожного класу мережевих пристроїв, оскільки різні рівні моделі OSI вимагають специфічних механізмів захисту. На рівні доступу, який найчастіше представлений комутаторами (Access Switches), реалізуються механізми фізичної та каналної безпеки. Комутатор є першою точкою входу користувача або пристрою в мережу, і саме тут повинні застосовуватися політики, що запобігають несанкціонованому підключенню. Основною функцією безпеки тут є контроль доступу на основі MAC-адрес (Port

Security), що дозволяє обмежити кількість пристроїв, які можуть працювати через один фізичний порт, або ж жорстко прив'язати конкретний порт до конкретної MAC-адреси. Більш просунутим механізмом є динамічна інспекція протоколу ARP (Dynamic ARP Inspection), яка захищає мережу від атак типу ARP-spoofing, коли зломисник намагається перехопити трафік, видаючи себе за шлюз за замовчуванням. Також критично важливим є функціонал DHCP Snooping, який блокує нелегітимні DHCP-сервери, що можуть бути розгорнуті зломисниками для перенаправлення клієнтського трафіку на підроблені ресурси. Таким чином, комутатори перетворюються з простих розгалужувачів на інтелектуальні бар'єри, що забезпечують гігієну мережі на найнижчому логічному рівні.

Наступним критичним ешелonom є рівень розподілу та ядра, де ключову роль відіграють маршрутизатори (Routers). Їхня роль у безпеці виходить далеко за межі простої пересилки пакетів між підмережами. Маршрутизатори діють як прикордонні контролери, що фільтрують трафік на межах різних зон довіри, наприклад, між корпоративною мережею та Інтернетом або між мережею користувачів та дата-центром. Основним інструментом тут виступають списки контролю доступу (ACL), які дозволяють адміністраторам створювати гранулярні правила: дозволяти або забороняти проходження пакетів на основі IP-адрес джерела та призначення, номерів портів та протоколів. Окрім фільтрації, маршрутизатори забезпечують приховування внутрішньої топології мережі за допомогою технології трансляції мережевих адрес (NAT/PAT). Це ускладнює зовнішнім зломисникам проведення розвідки та сканування внутрішніх ресурсів, оскільки ззовні вся мережа виглядає як одна або кілька публічних адрес. Також маршрутизатори часто виступають кінцевими точками для термінації VPN-тунелів, використовуючи протоколи IPsec для створення захищених віртуальних каналів поверх публічних мереж, що гарантує конфіденційність передачі даних між офісами.

Особливе місце в екосистемі безпеки займають міжмережеві екрани (Firewalls) та шлюзи нового покоління (NGFW). На відміну від маршрутизаторів, які приймають рішення на основі заголовків пакетів, фаєрволи працюють із відстеженням стану з'єднань (Stateful Inspection). Це означає, що пристрій пам'ятає

контекст кожного активного сеансу зв'язку і може відрізнити легітимний пакет-відповідь від спроби сканування або атаки. Сучасні NGFW розширюють ці можливості до рівня прикладних програм (Layer 7), забезпечуючи глибоку інспекцію пакетів (DPI). Вони здатні розпізнавати конкретні додатки незалежно від порту, який вони використовують, блокувати передачу певних типів файлів, виявляти спроби експлуатації вразливостей за допомогою вбудованих систем запобігання вторгненням (IPS) та навіть дешифрувати SSL/TLS трафік "на льоту" для пошуку прихованих загроз. Таким чином, міжмережевий екран є основним інструментом реалізації політик безпеки високого рівня, забезпечуючи видимість та контроль над змістом трафіку, а не лише його маршрутом.

Для захисту веб-додатків та сервісів, що публікуються в Інтернет, використовуються балансувальники навантаження (Load Balancers) та контролери доставки додатків (ADC). У контексті безпеки їхня роль полягає у захисті серверної інфраструктури від атак на відмову в обслуговуванні (DDoS). Балансувальники можуть виявляти аномальні сплески запитів і застосовувати механізми обмеження швидкості (Rate Limiting) або використовувати технології типу SYN Cookies для протидії атакам на виснаження таблиці з'єднань. Крім того, вони часто виконують функцію "зворотного проксі" (Reverse Proxy), повністю приховуючи реальні IP-адреси та структуру серверної ферми від зовнішнього світу. Важливою функцією є також SSL Offloading – перенесення процесу шифрування та дешифрування трафіку з веб-серверів на балансувальник. Це не лише знижує навантаження на сервери, але й дозволяє створити єдину точку контролю сертифікатів та криптографічних налаштувань, спрощуючи управління ключами та гарантуючи використання лише стійких алгоритмів шифрування.

Нарешті, у сегменті бездротових мереж ключову роль відіграють точки доступу (Wireless Access Points) та контролери бездротових мереж (WLC). Оскільки бездротове середовище є за своєю природою відкритим і доступним для перехоплення за межами фізичного периметра будівлі, вимоги до безпеки тут є найвищими. Точки доступу забезпечують шифрування ефірного трафіку за допомогою стандартів WPA2/WPA3, що унеможливорює перегляд даних сторонніми особами навіть при фізичному перехопленні радіосигналу. Важливою

функцією є ізоляція клієнтів (Client Isolation), яка забороняє прямий обмін даними між пристроями, підключеними до однієї точки доступу, що є критичним для публічних або гостьових мереж. Крім того, сучасні бездротові системи мають функціонал виявлення та придушення "ворожих" точок доступу (Rogue AP Detection) – несанкціонованих пристроїв, встановлених співробітниками або зловмисниками для обходу корпоративних політик безпеки. Бездротова інфраструктура також активно інтегрується з серверами аутентифікації (RADIUS/TACACS+) для реалізації суворого контролю доступу на основі ідентифікаційних даних користувача, а не лише пароля від Wi-Fi мережі.

Таким чином, кожен тип мережевого обладнання вносить свій унікальний вклад у загальну картину інформаційної безпеки. Від фізичного рівня і до рівня додатків, комутатори, маршрутизатори, фаєрволи та інші пристрої створюють багатошаровий захист. Відсутність або некоректне налаштування будь-якого з цих компонентів створює прогалину в периметрі безпеки, яку можуть використати зловмисники. Саме тому автоматизація налаштування цих різномірних пристроїв, приведення їх конфігурацій до єдиного стандарту безпеки та постійний контроль відповідності є критично важливою задачею для сучасних ІТ-підрозділів, що буде детально розглянуто в наступних підрозділах роботи.

1.2. Загрози та вразливості апаратного та програмного забезпечення мережевих пристроїв

Аналіз стану захищеності сучасних інформаційних систем свідчить про зміщення вектору кібератак з кінцевих точок та серверів додатків на інфраструктурне обладнання. Мережеві пристрої, будучи критичними вузлами комунікації, стають пріоритетними цілями для зловмисників, оскільки компрометація навіть одного маршрутизатора або комутатора ядра може надати повний контроль над потоками даних, дозволити обхід засобів захисту периметра та забезпечити приховане довготривале перебування в мережі. Складність забезпечення безпеки мережевого обладнання обумовлена його гетерогенністю, використанням пропрієтарних операційних систем та закритих протоколів, а також часто недостатньою увагою адміністраторів до процесів оновлення

мікропрограмного забезпечення порівняно з серверними операційними системами. Загрози, що впливають на мережеве обладнання, можна класифікувати за кількома напрямками, серед яких ключовими є вразливості програмного коду, архітектурні недоліки протоколів, слабкі місця в конфігурації та фізичні загрози апаратній частині.

Фундаментальною проблемою безпеки мережевого обладнання є вразливості операційних систем та мікропрограмного забезпечення (firmware). На відміну від універсальних ОС, таких як Linux або Windows, де процеси оновлення є стандартизованими та автоматизованими, оновлення мережевих пристроїв часто пов'язане з ризиками простою мережі та порушенням сумісності. Це призводить до ситуації, коли в продуктивному середовищі роками експлуатуються пристрої з відомими критичними вразливостями (CVE). Типовими помилками в коді мережевих ОС є переповнення буфера (buffer overflow) при обробці некоректно сформованих пакетів, помилки в реалізації криптографічних алгоритмів та вразливості веб-інтерфейсів управління [15]. Зловмисники активно використовують ці недоліки для виконання довільного коду (Remote Code Execution – RCE), що дозволяє їм отримати привілейований доступ до командної оболонки пристрою без проходження процедури аутентифікації. Особливу небезпеку становлять вразливості в реалізації стеків протоколів, таких як SNMP, SSH або HTTP, які використовуються для менеджменту пристроїв.

Окремий пласт загроз пов'язаний з апаратними вразливостями та ланцюжком поставок (Supply Chain Attacks). У сучасному глобалізованому виробництві існує ризик впровадження апаратних "закладок" або модифікованих чіпів ще на етапі збирання пристрою або його транспортування до замовника. Такі модифікації можуть забезпечувати прихований канал доступу, який неможливо виявити програмними засобами діагностики. Крім того, фізичний доступ до обладнання залишається критичним вектором атаки. Наявність сервісних портів (console ports), кнопок скидання налаштувань до заводських параметрів (password recovery) та інтерфейсів для налагодження (JTAG) створює можливості для зловмисника, який має фізичний доступ до стійки з обладнанням, обійти логічні засоби захисту, перехопити завантаження системи або зчитати вміст пам'яті, де можуть зберігатися

ключі шифрування та паролі в відкритому вигляді. Недостатній фізичний захист комутаційних шаф та серверних кімнат часто нівелює найскладніші налаштування мережевої безпеки.

Значний обсяг загроз виникає через архітектурні недоліки мережевих протоколів, які розроблялися десятиліттями тому без урахування сучасних вимог безпеки. Багато службових протоколів, таких як CDP (Cisco Discovery Protocol), LLDP (Link Layer Discovery Protocol), DTP (Dynamic Trunking Protocol) та інші, за замовчуванням увімкнені на пристроях і передають службову інформацію у відкритому вигляді. Зловмисник, підключившись до мережі, може пасивно прослуховувати ці повідомлення, отримуючи детальну інформацію про версії ПЗ, IP-адресацію та топологію мережі (Reconnaissance). Більше того, відсутність вбудованих механізмів аутентифікації в ряді протоколів маршрутизації (наприклад, RIP або ранні версії OSPF без налаштованого MD5/SHA) дозволяє впроваджувати в мережу фальшиві маршрути, перенаправляючи трафік через підконтрольні зловмиснику вузли для його перехоплення або модифікації (MITM атаки).

Критичною проблемою залишається людський фактор та помилки конфігурації (Misconfiguration). Сучасні мережеві пристрої мають тисячі параметрів налаштування, і складність конфігураційних файлів постійно зростає. Типовими помилками є використання стандартних паролів (admin/admin, cisco/cisco), залишення активними непотрібних сервісів, некоректне налаштування списків контролю доступу (ACL), яке залишає відкритими критичні порти, а також помилки в налаштуванні NAT та VPN. Відсутність єдиного стандарту конфігурації та автоматизованого контролю призводить до явища "configuration drift", коли налаштування пристроїв з часом хаотично змінюються, відхиляючись від безпечного еталону. Це створює "тіньові" вразливості, про які адміністратори можуть не здогадуватися до моменту інциденту.

Для систематизації розуміння загроз, що впливають на різні компоненти мережевого обладнання, доцільно розглянути їх класифікацію за вектором впливу та об'єктом атаки. Це дозволяє чітко розмежувати зони відповідальності при проектуванні системи захисту. Класифікацію вразливостей мережевого обладнання наведено в табл. 1.1.

Класифікація вразливостей мережевого обладнання

Категорія вразливості	Об'єкт впливу	Опис та приклад реалізації	Рівень критичності
Вразливості ПЗ (Software Vulnerabilities)	Операційна система (IOS, JunOS, EOS)	Помилки в коді ОС, що дозволяють переповнення буфера або ін'єкцію команд. Приклад: Cisco IOS HTTP Server Buffer Overflow.	Високий / Критичний
Слабкі алгоритми криптографії	Протоколи управління (SSH, SSL/TLS)	Використання застарілих версій протоколів (SSLv3, SSHv1) або слабких шифрів (DES, MD5), що дозволяє розшифрувати трафік.	Середній / Високий
Проблеми автентифікації	Механізми доступу (AAA, Local Accounts)	Відсутність двофакторної автентифікації, "вшиті" (hardcoded) облікові записи виробника, відсутність блокування після невдалих спроб входу.	Критичний
Вразливості протоколів L2/L3	Мережеві протоколи (ARP, STP, OSPF, BGP)	Відсутність перевірки джерела пакетів. Приклади: ARP Spoofing, STP Root Bridge hijacking, BGP Route Hijacking.	Високий
Небезпечні налаштування за замовчуванням	Конфігурація пристрою	Увімкнені сервіси Telnet, HTTP, SNMPv1/2c з публічними спільнотами (public community string).	Високий
Фізичні вразливості	Апаратна платформа	Можливість завантаження з зовнішнього носія, скидання пароля через перемички, підключення до JTAG/UART.	Середній (залежить від доступу до приміщення)

Для глибокого розуміння проблематики захисту мережевого обладнання необхідно детально проаналізувати механіку реалізації атак на каналному рівні (Layer 2), які часто залишаються поза увагою адміністраторів, зосереджених на захисті периметра. Однією з найбільш розповсюджених та небезпечних загроз є атака переповнення таблиці MAC-адрес (CAM Table Overflow). Комутатори використовують спеціалізовану пам'ять (Content Addressable Memory) для зберігання відповідностей між фізичними портами та MAC-адресами пристроїв. Обсяг цієї пам'яті є фіксованим і обмеженим. Зловмисник, використовуючи спеціалізоване програмне забезпечення, може генерувати тисячі фальшивих кадрів з випадковими MAC-адресами джерела за частки секунди, повністю заповнюючи таблицю комутації. Коли таблиця переповнена, комутатор не може зберегти нові легітимні адреси і переходить у режим "fail-open", починаючи транслювати трафік на всі порти (flooding), подібно до роботи застарілого концентратора (hub). Це дозволяє зловмиснику перехоплювати пакети, адресовані іншим вузлам мережі,

порушуючи конфіденційність даних без необхідності фізичного підключення до каналу жертви.

Ще однією критичною вразливістю каналного рівня є маніпуляції з протоколом ARP (Address Resolution Protocol). Оскільки протокол ARP був розроблений без вбудованих механізмів автентифікації, будь-який пристрій у локальній мережі може надіслати відповідь на ARP-запит, навіть якщо цей запит йому не призначався. Атака ARP Spoofing (або ARP Poisoning) полягає у тому, що зловмисник надсилає в мережу фальсифіковані ARP-повідомлення, в яких зв'язує свою MAC-адресу з IP-адресою легітимного шлюзу за замовчуванням (Default Gateway). Внаслідок цього, комп'ютери користувачів оновлюють свої ARP-кеші та починають надсилати весь трафік, призначений для Інтернету або інших підмереж, на комп'ютер зловмисника. Це створює класичну схему атаки "Людина посередині" (Man-in-the-Middle), дозволяючи зловмиснику не лише читати незашифровані дані, але й модифікувати їх на льоту або вибірково блокувати доступ до ресурсів.

Не менш серйозною загрозою є атаки на протокол Spanning Tree Protocol (STP), який відповідає за запобігання петлям у комутованих мережах. Алгоритм роботи STP базується на виборі кореневого комутатора (Root Bridge), який визначає топологію всієї мережі. Вибір відбувається на основі пріоритетів, які передаються у службових пакетах BPDU (Bridge Protocol Data Unit). Якщо адміністратор не налаштував функції захисту (такі як BPDU Guard або Root Guard), зловмисник може підключити свій пристрій до мережі та почати транслювати BPDU з найнижчим пріоритетом, змушуючи мережу визнати його пристрій новим кореневим комутатором. Це призводить до повної перебудови топології мережі, в результаті чого значна частина трафіку почне проходити через обладнання зловмисника, створюючи умови для перехоплення даних та викликаючи значні затримки або повну відмову мережі через перевантаження каналів.

На мережевому рівні (Layer 3) значну загрозу становлять атаки на протоколи динамічної маршрутизації, такі як OSPF (Open Shortest Path First) та BGP (Border Gateway Protocol). У випадку внутрішніх протоколів, таких як OSPF, вразливість полягає у можливості неавторизованого підключення до домену маршрутизації. Якщо між маршрутизаторами не налаштовано криптографічну автентифікацію

сусідів (MD5 або SHA HMAC), зловмисник може ввести в мережу власний маршрутизатор (або програмний емулятор) і почати анонсувати фальшиві маршрути. Це може призвести до "чорних дір" (blackholing), коли трафік спрямовується в нікуди, або до перенаправлення трафіку через скомпрометовані вузли для аналізу. Особливо небезпечними є атаки на BGP, який забезпечує глобальну маршрутизацію в Інтернеті. BGP Hijacking дозволяє перехоплювати цілі діапазони IP-адрес, перенаправляючи трафік цілих організацій або країн через інфраструктуру зловмисників, що неодноразово фіксувалося в реальних інцидентах кібершпionaжу.

Окрему категорію становлять загрози, спрямовані на площину управління (Management Plane) мережевих пристроїв. Найпоширенішим вектором тут є експлуатація вразливостей протоколу SNMP (Simple Network Management Protocol). Версії SNMPv1 та SNMPv2c використовують для автентифікації лише рядок спільноти (community string), який передається у відкритому вигляді. Якщо зловмисник перехопить цей рядок (який часто залишається стандартним "public" або "private"), він може отримати детальну інформацію про конфігурацію пристрою, версію ПЗ, таблиці маршрутизації та ARP. У випадку наявності прав на запис (RW community), зловмисник може змінювати конфігурацію пристрою, перезавантажувати його або навіть завантажувати шкідливі образи операційної системи. Також критичними є атаки на протокол синхронізації часу NTP (Network Time Protocol). Маніпуляції з часом на мережевих пристроях можуть призвести до некоректної роботи сертифікатів безпеки, порушення хронології лог-файлів, що унеможливорює розслідування інцидентів, та збоїв у роботі запланованих завдань. Сучасна тенденція до використання веб-інтерфейсів для керування мережевими обладнанням відкрила новий клас вразливостей, характерних для веб-додатків. Багато маршрутизаторів та комутаторів мають вбудовані веб-сервери, які часто містять вразливості типу Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) та SQL/Command Injection. Зловмисник може сформувати спеціальне посилання і змусити адміністратора, який залогінений у веб-інтерфейсі пристрою, перейти за ним. Це може призвести до виконання прихованих команд від імені адміністратора, наприклад, створення нового привілейованого користувача або

відкриття віддаленого доступу. На відміну від класичних CLI-інтерфейсів, веб-панелі управління часто мають менш ретельний аудит безпеки коду з боку виробників, що робить їх "слабкою ланкою" в периметрі безпеки пристрою.

Для комплексної оцінки ризиків необхідно детально розглянути наслідки реалізації типових атак на мережеве обладнання крізь призму тріади інформаційної безпеки (конфіденційність, цілісність, доступність) та їх вплив на безперервність бізнесу. Атака переповнення таблиці комутації (CAM Table Flooding) має критичний вплив на конфіденційність даних у локальному сегменті мережі. Оскільки комутатор, вичерпавши ресурси пам'яті, переходить у режим широкомовної трансляції, весь одноадресний трафік стає доступним для перехоплення будь-яким учасником сегменту. Хоча цілісність даних при цьому зазвичай не порушується, доступність мережі може постраждати через надмірне навантаження на процесор комутатора, який змушений обробляти аномальну кількість кадрів. Для бізнесу це означає високий ризик витоку комерційної таємниці, персональних даних співробітників та незахищених паролів, що може призвести до репутаційних втрат та юридичної відповідальності.

Атака типу ARP Spoofing створює надзвичайно високі ризики одночасно для конфіденційності та цілісності інформації. Встановлюючи себе як посередника між жертвою та шлюзом (Man-in-the-Middle), зловмисник отримує можливість не лише пасивно прослуховувати трафік, але й активно втручатися в сесії передачі даних, підміняючи вміст пакетів або впроваджуючи шкідливий код у незахищені HTTP-з'єднання. Доступність сервісів також знаходиться під загрозою, оскільки зловмисник може в будь-який момент обірвати з'єднання або вибірково блокувати доступ до певних ресурсів. У бізнес-контексті це загрожує компрометацією облікових записів користувачів, успішною реалізацією фішингових кампаній та крадіжкою фінансової інформації, причому виявити таку активність без спеціалізованих засобів моніторингу вкрай важко.

Маніпуляції з протоколом STP (Spanning Tree Protocol) несуть основну загрозу для доступності мережевої інфраструктури. Несанкціонована зміна топології та перехоплення ролі кореневого комутатора може призвести до утворення комутаційних петель, що викликає лавиноподібне зростання широкомовного

трафіку (broadcast storm). Це паралізує роботу сегменту мережі або навіть усієї локальної інфраструктури, роблячи неможливим обмін даними. Вплив на конфіденційність оцінюється як середній, оскільки перенаправлення трафіку через пристрій зловмисника можливе, але є вторинною ціллю порівняно з дестабілізацією роботи. Наслідком для підприємства стає повна зупинка операційної діяльності, простій критичних бізнес-додатків та значні фінансові збитки через неможливість надання сервісів клієнтам або співробітникам.

Розгортання нелегітимного DHCP-сервера (Rogue DHCP) є комплексним вектором атаки, що впливає на всі аспекти безпеки. Видаючи клієнтам некоректні налаштування мережі, зловмисник може вказати власну IP-адресу як DNS-сервер, що дозволяє перенаправляти користувачів на підроблені сайти (фармінг) навіть при введенні коректних URL-адрес. Це створює критичні ризики для конфіденційності, оскільки користувачі можуть несвідомо передавати свої автентифікаційні дані зловмиснику. Доступність мережі також страждає через можливі конфлікти IP-адрес, коли легітимні клієнти втрачають можливість підключення. Бізнес-наслідки включають масове перехоплення корпоративних облікових даних та дезорганізацію роботи офісів через проблеми з підключенням до мережі.

Атаки типу VLAN Hopping спрямовані на подолання логічної сегментації мережі, що є прямим ударом по конфіденційності та моделі розмежування доступу. Використовуючи вразливості протоколу DTP (Dynamic Trunking Protocol) або метод подвійного тегування (Double Tagging), зловмисник може отримати доступ до ізольованих VLAN, таких як мережі управління обладнанням, серверні ферми або фінансові відділи, з сегменту загального користування або гостьової мережі. Це нівелює інвестиції компанії в побудову захищеної архітектури. Основний ризик для бізнесу полягає у несанкціонованому доступі до найбільш чутливих активів компанії, які вважалися надійно захищеними завдяки мережевій ізоляції.

Нарешті, ін'єкція маршрутів у протоколи динамічної маршрутизації (Routing Protocol Injection) загрожує цілісності та доступності на глобальному рівні корпоративної мережі. Зловмисник, анонсуючи фальшиві маршрути з кращою метрикою, може перенаправити трафік між філіями через підконтрольні вузли або створити "чорні діри" (blackholes), куди трафік буде потрапляти і зникати. Це може

призвести до ізоляції окремих підрозділів компанії, втрати зв'язку з дата-центрами та неможливості виконання транзакцій. Для підприємства це означає порушення зв'язності інфраструктури, неможливість синхронізації баз даних та потенційну втрату контролю над віддаленими об'єктами.

1.3. Методи та засоби захисту мережевої інфраструктури (ACL, VPN, Firewall, IDS/IPS, сегментація, Zero Trust)

Побудова надійної системи захисту мережевої інфраструктури базується на комплексному застосуванні різнорівневих методів та засобів, які у сукупності реалізують концепцію ешелонованої оборони (Defense-in-Depth). Ця концепція передбачає, що жоден окремий засіб захисту не може гарантувати повну безпеку, тому необхідно створювати кілька незалежних шарів контролю, які перекривають недоліки один одного. На фундаментальному рівні керування потоками даних основним та найбільш розповсюдженим інструментом залишаються списки контролю доступу (Access Control Lists – ACL). Це впорядковані набори правил, які застосовуються до інтерфейсів маршрутизаторів або комутаторів і визначають, які пакети повинні бути пропущені, а які – відкинуті. Ефективність ACL полягає в їхній здатності здійснювати фільтрацію на швидкості лінії (wire speed), використовуючи апаратні ресурси пристрою (TCAM – Ternary Content Addressable Memory), що не створює суттєвих затримок для легітимного трафіку.

Списки контролю доступу поділяються на стандартні та розширені, кожен з яких має свою сферу застосування в архітектурі безпеки. Стандартні ACL (Standard ACL) перевіряють лише IP-адресу джерела пакета. Це робить їх простими в налаштуванні та менш вимогливими до ресурсів процесора, проте їх функціональність обмежена. Зазвичай їх застосовують якомога ближче до місця призначення трафіку, щоб не блокувати легітимний трафік від цього джерела до інших частин мережі. Натомість, розширені ACL (Extended ACL) забезпечують набагато вищу гранулярність контролю. Вони дозволяють фільтрувати трафік на основі IP-адрес джерела та призначення, протоколів транспортного рівня (TCP, UDP, ICMP), а також номерів портів джерела та призначення. Це дозволяє адміністраторам реалізовувати точкові політики безпеки, наприклад, дозволити

доступ до веб-сервера (порт 80/443) для всіх користувачів, але відкрити доступ до порту управління (SSH, порт 22) лише для підмережі адміністраторів.

Критично важливим аспектом роботи ACL є порядок обробки правил. Маршрутизатор перевіряє кожен пакет послідовно зверху вниз відповідно до списку правил. Як тільки знаходиться перше правило, що відповідає параметрам пакета (match), дія виконується (permit або deny), і подальша перевірка припиняється. Це вимагає від адміністраторів ретельного планування порядку записів: найбільш специфічні правила повинні розміщуватися на початку списку, а більш загальні – в кінці. Фундаментальним принципом безпеки при роботі з ACL є правило "неявного відхилення" (Implicit Deny), яке автоматично додається в кінець кожного списку. Це означає, що якщо пакет не підпав під дію жодного з дозволяючих правил, він буде автоматично відкинутий. Такий підхід гарантує, що в мережу потрапить лише той трафік, який був явно дозволений політикою безпеки, що відповідає принципу мінімальних привілеїв.

Незважаючи на свою ефективність, класичні ACL мають суттєвий недолік – вони є статичними (stateless). Це означає, що пристрій розглядає кожен пакет ізольовано, не зберігаючи інформацію про стан з'єднання. Наприклад, якщо дозволено вихідний трафік від клієнта до сервера, класичний ACL не пропустить автоматично вхідний пакет-відповідь від сервера, якщо для нього не створено окремого дозволяючого правила. Це значно ускладнює конфігурацію та збільшує розмір списків, підвищуючи ймовірність помилок. Для вирішення цієї проблеми використовуються рефлексивні ACL (Reflexive ACL) або механізми перевірки стану на основі зон (Zone-Based Firewall), які динамічно створюють тимчасові записи для зворотного трафіку легітимних сесій, видаляючи їх після завершення з'єднання.

Іншим фундаментальним компонентом захисту мережевої інфраструктури, особливо в умовах географічно розподілених підприємств та віддаленої роботи, є технології віртуальних приватних мереж (Virtual Private Network – VPN). VPN забезпечує створення захищеного логічного тунелю поверх незахищеної публічної мережі, такої як Інтернет. Основними цілями використання VPN є забезпечення конфіденційності даних (через шифрування), цілісності (через хешування) та

автентифікації сторін. Найбільш поширеним стандартом для об'єднання мереж філій (Site-to-Site VPN) є набір протоколів IPsec. IPsec працює на мережевому рівні (Layer 3) і забезпечує прозору передачу трафіку між офісами, так що користувачі навіть не помічають, що їхні дані проходять через Інтернет. Він використовує складні механізми узгодження параметрів безпеки (IKE – Internet Key Exchange), які відбуваються у дві фази: спочатку встановлюється захищений канал управління, а потім у ньому узгоджуються ключі для шифрування даних користувачів.

Технології VPN також відіграють ключову роль у забезпеченні безпечного віддаленого доступу (Remote Access VPN). Тут конкурують два основні підходи: IPsec VPN та SSL/TLS VPN. IPsec для віддаленого доступу зазвичай вимагає встановлення спеціального клієнтського програмного забезпечення на пристрій користувача, що забезпечує повний доступ до мережевих ресурсів, наче комп'ютер фізично підключений до офісної мережі. Натомість SSL/TLS VPN (часто реалізований як бесклієнтський доступ через браузер або легкий агент) працює на рівні сеансів або додатків. Він є більш гнучким для надання доступу партнерам або підрядникам, оскільки дозволяє гранулярно відкривати доступ лише до конкретних веб-додатків, не надаючи повного мережевого доступу. Сучасні рішення VPN інтегруються з системами багатофакторної автентифікації (MFA) та перевірки стану кінцевого пристрою (Posturing), дозволяючи підключатися лише тим пристроям, які мають оновлений антивірус та патчі безпеки.

Окрему увагу слід приділити криптографічній стійкості VPN-рішень. Надійність тунелю залежить від обраних алгоритмів шифрування (наприклад, AES-256 замість застарілого DES), алгоритмів хешування (SHA-256 і вище) та груп Діффі-Хеллмана для обміну ключами. Використання застарілих алгоритмів робить VPN вразливим до атак розшифрування "грубою силою" або використання відомих математичних вразливостей. Крім того, важливою є концепція Perfect Forward Secrecy (PFS), яка гарантує, що навіть у випадку компрометації довгострокового приватного ключа сервера, зловмисник не зможе розшифрувати сесії, записані в минулому, оскільки для кожної сесії генеруються унікальні ефемерні ключі. Автоматизація налаштування VPN-тунелів є критичною, оскільки ручне введення

складних параметрів криптографії на десятках пристроїв неминуче призводить до розбіжностей (mismatch) та неможливості встановлення з'єднання.

Еволюційним розвиток засобів мережевого захисту призвів до появи спеціалізованих пристроїв – міжмережевих екранів (Firewalls), які стали центральним елементом периметра безпеки будь-якої корпоративної мережі. На відміну від маршрутизаторів зі списками контролю доступу, які працюють за принципом "без стану" (stateless), сучасні міжмережєві екрани базуються на технології інспекції з урахуванням стану з'єднань (Stateful Packet Inspection – SPI). Ця технологія дозволяє пристрою відстежувати контекст кожного сеансу зв'язку, зберігаючи інформацію про нього в динамічній таблиці станів. Коли ініціюється нове з'єднання (наприклад, відправка TCP SYN пакету з внутрішньої мережі), міжмережєвий екран записує параметри цього з'єднання (IP-адреси, порти, порядкові номери пакетів) у таблицю. Всі наступні пакети, що належать до цієї сесії, включаючи зворотний трафік від сервера, перевіряються на відповідність запису в таблиці станів, а не проходять повну перевірку правил політики безпеки. Це не лише підвищує продуктивність обробки трафіку, але й значно посилює безпеку, оскільки автоматично блокує будь-які пакети, які не є частиною легітимного, встановленого з'єднання (наприклад, сканування портів за допомогою FIN-пакетів).

Подальший розвиток технологій призвів до появи шлюзів прикладного рівня (Application Level Gateways – ALG) або проксі-серверів. Ці системи працюють як посередники, повністю розриваючи пряме з'єднання між клієнтом і сервером. Клієнт встановлює з'єднання з проксі, а проксі, перевіривши запит, встановлює окреме з'єднання з цільовим сервером від свого імені. Такий підхід дозволяє здійснювати найглибшу інспекцію вмісту, оскільки проксі "розуміє" протокол прикладного рівня (HTTP, FTP, SMTP) і може аналізувати команди, заголовки та дані. Це унеможливорює прямі атаки на сервер через мережєвий стек, проте значні витрати обчислювальних ресурсів на повну ретрансляцію даних обмежують пропускну здатність таких рішень. Тому в сучасних високошвидкісних мережах частіше використовуються гібридні технології, які поєднують швидкість SPI з глибиною аналізу проксі лише для підозрілого трафіку.

Вершиною еволюції засобів периметрового захисту на сьогодні є міжмережеві екрани нового покоління (Next-Generation Firewalls – NGFW). Традиційні фаєрволи, що оперують поняттями портів і IP-адрес, стали неефективними в умовах сучасного Інтернету, де сотні різних додатків (від Facebook і YouTube до корпоративних CRM та шкідливого ПЗ) використовують одні й ті ж стандартні порти (80 та 443) для тунелювання свого трафіку через HTTP/HTTPS. NGFW вирішують цю проблему за допомогою технології ідентифікації додатків (App-ID), яка класифікує трафік на основі його сигнатур та поведінки, а не номера порту. Це дозволяє адміністраторам створювати політики типу "дозволити Skype, але заборонити передачу файлів через Skype", що було неможливо на пристроях попередніх поколінь. Крім того, NGFW інтегрують в собі функціонал ідентифікації користувачів (User-ID) через зв'язок з Active Directory, що дозволяє писати правила безпеки для конкретних співробітників або груп, а не для абстрактних IP-адрес, які можуть змінюватися.

Критично важливим компонентом сучасної системи захисту є системи виявлення (IDS – Intrusion Detection Systems) та запобігання вторгненням (IPS – Intrusion Prevention Systems). IDS працює в пасивному режимі, аналізуючи копію трафіку (через SPAN-порт або TAP-пристрій). Її завдання – виявити аномалію або сигнатуру атаки і сповістити адміністратора, не впливаючи на проходження реального трафіку. Це дозволяє уникнути затримок у мережі, але не дає можливості автоматично зупинити атаку в момент її здійснення. IPS, натомість, встановлюється "в розрив" (inline) каналу зв'язку і аналізує трафік у реальному часі. У разі виявлення шкідливої активності IPS може автоматично відкинути пакети, розірвати TCP-сесію або заблокувати IP-адресу нападника. Основною проблемою при впровадженні IPS є ризик хибних спрацювань (False Positives), коли легітимний трафік помилково блокується, що може призвести до зупинки бізнес-процесів. Тому впровадження таких систем вимагає тривалого періоду навчання та тонкого налаштування профілів захисту.

Ефективність систем IDS/IPS базується на двох основних методах аналізу: сигнатурному та евристичному (аномальному). Сигнатурний метод порівнює вміст пакетів з базою даних відомих патернів атак. Це забезпечує високу точність

виявлення відомих загроз і низький рівень хибних спрацювань, але робить систему безсилою проти нових атак (Zero-Day), для яких ще не випущені сигнатури. Евристичний аналіз та аналіз аномалій будують математичну модель "нормальної" поведінки мережі (baseline) і реагують на будь-які значні відхилення (наприклад, раптовий сплеск трафіку о 3-й ночі або використання нестандартних протоколів). Цей метод дозволяє виявляти невідомі раніше загрози, але схильний до генерації великої кількості хибних тривог через зміни в легітимній поведінці користувачів. Сучасні системи дедалі частіше використовують алгоритми машинного навчання (Machine Learning) для підвищення точності виявлення аномалій та автоматичної адаптації до змін у мережевому середовищі.

Також варто зазначити важливість інтеграції міжмережевих екранів з системами розвідки загроз (Threat Intelligence). Сучасний NGFW не є ізольованим пристроєм; він постійно отримує оновлення з хмарних лабораторій виробника, які містять списки шкідливих IP-адрес, хеш-суми вірусів, URL-адреси фішингових сайтів та сигнатури експлойтів. Це дозволяє блокувати з'єднання з командними центрами ботнетів (C&C servers) ще до того, як шкідливе ПЗ буде активовано всередині мережі. Крім того, функціонал SSL/TLS Inspection (дешифрування HTTPS) є критично необхідним, оскільки за статистикою більше 80% сучасного шкідливого трафіку передається у зашифрованому вигляді, щоб обійти традиційні засоби захисту. Дешифрування вимагає значних апаратних ресурсів, але без нього фаєрвол залишається "сліпим" до вмісту більшості сучасних веб-сесій.

Нижче наведено порівняльний аналіз поколінь технологій міжмережевого екранування, що демонструє зростання функціональних можливостей та глибини контролю.

Таблиця 1.2

Еволюція технологій міжмережевих екранів

Характеристика	Stateless Firewall	Stateful Firewall	Proxy Firewall	Next-Generation Firewall
Рівень роботи (OSI)	L3 (Network), L4 (Transport)	L3, L4 (з відстеженням сесій)	L3-L7 (Application)	L2-L7 (повна видимість)
Об'єкт контролю	Окремі пакети	Сесії (поток) TCP/UDP	Прикладні протоколи (HTTP, FTP)	Користувачі, додатки, контент
Принцип рішення	Статичні правила (ACL)	Динамічна таблиця станів	Посередництво (розрив сесії)	App-ID, User-ID, Content-ID
Продуктивність	Дуже висока	Висока	Низька / Середня	Висока (завдяки ASIC чіпам)
Захист від атак	Базовий (IP spoofing)	DoS, сканування портів	Атаки на протоколи додатків	APT, експлойти, malware, ботнети
Видимість трафіку	IP-адреси, порти	Стан з'єднання	Повний зміст (в межах протоколу)	Дешифрований SSL, файли, дані

Для глибокого розуміння необхідності переходу до нових архітектур безпеки доцільно провести детальний порівняльний аналіз традиційної периметрової моделі та моделі нульової довіри (Zero Trust). Фундаментальна відмінність полягає у базовій філософії ставлення до мережевого середовища. Традиційна модель, яка домінувала в індустрії протягом останніх десятиліть, будується на концепції "довіряй, але перевіряй" (Trust but verify). Вона передбачає існування чіткого фізичного або логічного периметра, який відокремлює "небезпечний" зовнішній світ (Інтернет) від "безпечної" внутрішньої мережі (Intranet). У цій парадигмі будь-який користувач або пристрій, який успішно пройшов автентифікацію на шлюзі та потрапив у внутрішній сегмент, автоматично отримує високий рівень довіри (Implicit Trust). Це створює архітектуру типу "тверда оболонка, м'яка середина", де зломиснику достатньо подолати лише один бар'єр, щоб отримати можливість вільного переміщення всередині мережі. На противагу цьому, модель Zero Trust базується на діаметрально протилежному принципі "ніколи не довіряй, завжди перевіряй" (Never Trust, Always Verify). Вона виходить з припущення, що мережа вже скомпрометована, а загрози можуть знаходитися як зовні, так і всередині. Тому поняття довіреної зони повністю ліквідується, а сама мережа розглядається виключно як вороже середовище транспортування даних, незалежно від того, чи це

публічний Wi-Fi в кафе, чи захищений порт комутатора в ядрі корпоративного дата-центру.

Наступна критична відмінність стосується механізмів контролю доступу та їх гранулярності. У традиційних системах контроль здебільшого здійснюється на основі мережевих атрибутів: IP-адреси джерела та призначення, номеру порту та приналежності до певної VLAN. Якщо IP-адреса користувача входить до дозволеного діапазону, фаєрвол пропускає трафік. Такий підхід є вразливим до спуфінгу та не враховує контекст з'єднання. Zero Trust зміщує фокус з мережевих атрибутів на ідентичність (Identity-Centric Security). Рішення про надання доступу приймається на основі сукупності факторів: хто саме запитує доступ (користувач), з якого пристрою (корпоративний чи особистий), який стан безпеки цього пристрою (наявність патчів, відсутність вірусів), з якої локації та в який час. Це дозволяє реалізувати динамічні політики, які адаптуються до ситуації в реальному часі. Наприклад, системний адміністратор може мати повний доступ до серверів з офісного комп'ютера, але при спробі підключитися з особистого планшета отримає лише обмежений доступ до пошти, навіть якщо він використовує правильний пароль.

Також суттєво відрізняється сам обсяг наданого доступу. Традиційні технології віддаленого доступу, такі як класичний VPN, зазвичай надають користувачеві доступ до цілого сегменту мережі. Після встановлення тунелю комп'ютер користувача стає повноправним учасником локальної мережі, маючи можливість сканувати порти інших пристроїв та взаємодіяти з ними. Це порушує принцип найменших привілеїв. Архітектура Zero Trust впроваджує підхід мікросегментації на рівні додатків (Application-Level Segmentation) [20]. Користувачеві ніколи не надається прямий мережевий доступ до інфраструктури. Замість цього створюється зашифрований тунель безпосередньо від пристрою користувача до конкретного додатка, до якого йому дозволено доступ. Інша частина мережі залишається для нього абсолютно невидимою ("dark network"), що унеможливорює розвідку та латеральне переміщення злоумисників.

Важливим аспектом є частота верифікації. У периметровій моделі перевірка автентичності та авторизації відбувається одноразово – в момент входу в систему

(Login). Після цього сесія вважається легітимною до моменту її завершення або тайм-ауту. Це створює ризик "викрадення сесії" (Session Hijacking), коли зломисник перехоплює контроль над комп'ютером вже після проходження автентифікації. Zero Trust вимагає безперервної оцінки довіри (Continuous Trust Assessment). Система постійно моніторить поведінку користувача та стан пристрою протягом всієї сесії. Якщо в процесі роботи антивірус виявить загрозу, або користувач почне завантажувати аномальні обсяги даних, рівень довіри буде миттєво знижено, а доступ – заблоковано, навіть якщо сесія була встановлена легітимно.

Впровадження таких складних архітектурних підходів, як Zero Trust та мікросегментація, неможливе без високого рівня автоматизації. Ручне налаштування тисяч правил на фаєрволах, постійна актуалізація списків доступу відповідно до зміни ролей співробітників та динамічна реакція на інциденти вимагають інструментів оркестрації. Саме тому подальший розгляд методів автоматизованого налаштування мережевого обладнання, зокрема з використанням Ansible, є не просто питанням зручності адміністрування, а критичною умовою забезпечення реальної, а не декларативної безпеки сучасної інформаційної системи. Без автоматизації складність підтримки актуальних політик безпеки швидко перевищує можливості персоналу, що призводить до помилок, вразливостей і, зрештою, до компрометації системи.

1.4. Проблеми ручного налаштування безпекових політик та потреба в автоматизації

Традиційний підхід до управління мережевою інфраструктурою, який базується на ручному введенні команд через інтерфейс командного рядка (Command Line Interface – CLI), протягом десятиліть залишався стандартом де-факто в галузі телекомунікацій. Ця модель передбачає, що мережевий адміністратор або інженер безпеки послідовно підключається до кожного пристрою, проходить процедуру автентифікації та вручну вводить необхідні команди для зміни конфігурації. У часи, коли мережі були статичними, а кількість пристроїв вимірювалася десятками, такий підхід був прийнятним і дозволяв

зберігати повний контроль над системою. Однак в умовах сучасної цифрової трансформації, хмарних обчислень та експоненціального зростання кількості підключених пристроїв, ручне адміністрування перетворилося на критичне вузьке місце ("bottleneck"), яке стримує розвиток бізнесу та створює неприйнятні ризики для інформаційної безпеки. Фундаментальною проблемою ручного налаштування є його невідповідність динаміці сучасних кіберзагроз: зловмисники використовують автоматизовані інструменти для сканування та атак, які працюють зі швидкістю машинного часу, тоді як захисники, що покладаються на ручну працю, змушені реагувати зі швидкістю людини, що завідомо ставить їх у програвшу позицію.

Найбільш очевидною, але від того не менш критичною проблемою ручного керування є так званий "людський фактор". Статистичні дані провідних аналітичних агентств, таких як Gartner та Forrester, свідчать про те, що переважна більшість (за різними оцінками від 60% до 80%) збоїв у роботі мережі та інцидентів безпеки спричинені саме помилками персоналу під час внесення змін у конфігурацію, а не відмовою обладнання чи зовнішніми атаками. Монотонна, репетитивна робота з введення однотипних команд на сотнях пристроїв неминуче призводить до втрати уваги та втоми інженера. Типовими помилками є одруки в IP-адресах, некоректне застосування масок підмережі, помилкове видалення критичних правил ACL або забування закрити тимчасово відкритий для налагодження порт. У складних конфігураціях, де порядок правил має вирішальне значення (наприклад, у Firewall policies), перестановка двох рядків місцями може призвести до того, що вся захищена мережа стане доступною з Інтернету. При ручному підході такі помилки вкрай важко виявити превентивно; зазвичай вони стають очевидними лише після того, як стався інцидент або сервіс припинив роботу.

Ще однією системною проблемою є явище "конфігураційного дрейфу" (Configuration Drift) або ентропії конфігурацій. У процесі експлуатації мережі адміністратори часто вносять ситуативні, термінові зміни ("ad-hoc changes") для вирішення локальних проблем: відкривають доступ для нового сервісу, змінюють маршрутизацію для обходу аварійної ділянки або додають тимчасові правила для

тестування. При ручному керуванні ці зміни часто не документуються належним чином і, що найгірше, не видаляються після того, як потреба в них зникла. З часом накопичення таких "тимчасових" налаштувань призводить до того, що реальна конфігурація пристрою кардинально відрізняється від еталонної або задокументованої. Мережеві пристрої перетворюються на унікальні "сніжинки" (Snowflakes), кожен з яких має свій неповторний набір налаштувань. Це робить мережу непередбачуваною: оновлення, яке успішно пройшло на тестовому стенді, може призвести до катастрофічного збою в продуктивному середовищі через конфлікт із забутим правилом трирічної давнини. Відсутність єдиного джерела істини (Single Source of Truth) про стан мережі унеможливорює гарантування відповідності вимогам безпеки.

Масштабованість процесів налаштування безпеки також є критичним викликом. Сучасні мережі ростуть нелінійно, і додавання нових філій або сегментів дата-центру вимагає налаштування сотень комутаторів, маршрутизаторів та точок доступу. При ручному підході час, необхідний для розгортання, зростає прямо пропорційно кількості пристроїв. Це призводить до ситуації, коли відділ безпеки не встигає за потребами бізнесу. Наприклад, впровадження нової політики безпеки (скажімо, зміна алгоритмів шифрування VPN або оновлення SNMP-спільнот) у великій корпоративній мережі може займати тижні або навіть місяці. У цей перехідний період мережа знаходиться в уразливому, гібридному стані, коли частина пристроїв вже захищена, а частина – ще ні. Це створює "вікна можливостей" для зловмисників. Більше того, у гетерогенних середовищах, де використовується обладнання різних виробників (Cisco, Juniper, Huawei, HP), інженери повинні пам'ятати синтаксис команд для кожної платформи, що додатково ускладнює процес і підвищує ймовірність помилок.

Окремою проблемою є складність аудиту та забезпечення відповідності регуляторним вимогам (Compliance), таким як PCI DSS, ISO 27001 або GDPR. Ці стандарти вимагають не лише наявності захищеної конфігурації, але й доказів того, що ця конфігурація є незмінною, а будь-які модифікації є авторизованими та задокументованими. При ручному адмініструванні збір доказів для аудиту перетворюється на виснажливий процес ручного вивантаження конфігурацій

("show running-config"), їх аналізу та порівняння з політиками. Це не лише забирає сотні людино-годин, але й дає результат, актуальний лише на момент вивантаження. Неможливо гарантувати, що через 5 хвилин після перевірки адміністратор не змінив налаштування. Відсутність автоматизованого версіонування змін та прив'язки кожної зміни до конкретного виконавця робить розслідування інцидентів вкрай складним. У разі злому практично неможливо швидко встановити, хто, коли і яку саме зміну вніс, що відкрила вразливість.

Крім того, ручне налаштування унеможлиблює впровадження сучасних архітектурних патернів, таких як мікросегментація та Zero Trust, про які йшлося у попередньому підрозділі. Ці архітектури вимагають управління тисячами дрібних правил, які динамічно змінюються залежно від контексту. Людина фізично не здатна оперувати таким обсягом інформації та вносити зміни з необхідною швидкістю. Спроба керувати середовищем Zero Trust вручну призведе або до блокування легітимного трафіку (через занадто жорсткі та статичні правила), або до деградації безпеки (через надмірне спрощення правил для зручності адміністрування). Таким чином, відсутність автоматизації стає технологічним бар'єром, що не дозволяє підприємству переходити на нові, більш захищені моделі побудови мереж.

Економічний аспект проблеми також є вагомим. Висококваліфіковані мережеві інженери та архітектори безпеки є дорогим ресурсом. Коли ці фахівці витрачають значну частину свого робочого часу на виконання рутинних операцій (копіювання команд, перевірку VLAN, оновлення паролів), підприємство втрачає кошти. Цей час міг би бути витрачений на стратегічні завдання: аналіз загроз, оптимізацію архітектури, впровадження нових сервісів. Ручна праця перетворює ІТ-департамент з центру інновацій на центр витрат, який постійно "гасить пожежі" та не встигає за темпом розвитку компанії. В умовах дефіциту кадрів на ринку кібербезпеки неефективне використання наявних спеціалістів є недопустимою розкішшю.

Підсумовуючи, можна стверджувати, що ручне налаштування конфігурацій безпеки мережевого обладнання вичерпало свій потенціал і перетворилося на гальмівний фактор. Воно характеризується високою ймовірністю помилок,

низькою швидкістю реакції на загрози, неможливістю масштабування, складністю аудиту та високою вартістю експлуатації. Вирішення цих проблем лежить у площині переходу до парадигми "Інфраструктура як код" (Infrastructure as Code – IaC) та впровадження засобів автоматизації, які дозволяють програмно описувати бажаний стан системи та автоматично приводити реальні пристрої у відповідність до цього еталону [8]. Саме такий підхід, реалізований за допомогою інструментів на кшталт Ansible, буде детально проаналізовано та розроблено у наступних розділах даної роботи.

1.5. Нормативні вимоги та рекомендації щодо конфігурування безпечних мережевих систем

Побудова захищеної мережевої інфраструктури не є довільним процесом, що залежить виключно від бачення конкретного системного адміністратора; вона суворо регламентується низкою міжнародних стандартів, галузевих нормативів та кращих практик (Best Practices). Дотримання цих вимог є необхідною умовою не лише для проходження аудитів та уникнення штрафних санкцій, але й для забезпечення реального базового рівня захищеності інформаційних активів. Нормативна база у сфері інформаційної безпеки формує уніфікований підхід до конфігурування обладнання, визначаючи чіткі критерії того, що вважається "безпечним налаштуванням". Ці стандарти еволюціонували від загальних рекомендацій до детальних технічних специфікацій, які покривають кожен аспект функціонування мережевого пристрою – від фізичного захисту портів до криптографічної стійкості протоколів управління. Розуміння цієї ієрархії вимог є критичним для розробки будь-якої системи автоматизації, оскільки саме вони виступають "технічним завданням" для написання сценаріїв налаштування (Playbooks).

Фундаментальним документом глобального рівня, що визначає вимоги до систем управління інформаційною безпекою (ISMS), є сімейство стандартів ISO/IEC 27000 [2]. Зокрема, стандарт ISO/IEC 27001 встановлює вимоги до процесів управління ризиками, а ISO/IEC 27002 містить практичні рекомендації щодо реалізації засобів контролю (security controls) [3]. У контексті мережевої

безпеки ключовим є розділ "Network Security Management", який вимагає від організацій забезпечення захисту інформації в мережах та захисту підтримуючої інфраструктури. Стандарт прямо вказує на необхідність сегментації мереж, контролю доступу до мережевих служб та управління конфігураціями. Хоча ISO 27001 не містить конкретних команд для налаштування обладнання (наприклад, Cisco або Juniper), він вимагає наявності документованих процедур зміни конфігурацій та регулярного перегляду правил міжмережевих екранів. Вимога стандарту щодо "відокремлення обов'язків" та "реєстрації подій" (logging) безпосередньо впливає на налаштування AAA (Authentication, Authorization, Accounting) на мережевих пристроях, зобов'язуючи адміністраторів налаштовувати централізоване збирання логів та розмежування прав доступу.

Більш технічно орієнтованими та деталізованими є рекомендації Національного інституту стандартів і технологій США (NIST), зокрема спеціальна публікація NIST SP 800-53 "Security and Privacy Controls for Information Systems and Organizations" [3]. Цей документ є фактичним стандартом для побудови захищених систем у державному та корпоративному секторах багатьох країн. Він містить розширені родини контролів, такі як "Configuration Management" (CM) та "System and Communications Protection" (SC). Контроль CM-2 вимагає наявності базової конфігурації (baseline configuration) для всіх компонентів системи та суворого контролю будь-яких відхилень від неї. Це безпосередньо обґрунтовує необхідність використання засобів автоматизації на кшталт Ansible, які здатні забезпечити ідемпотентність налаштувань – властивість, коли повторне застосування сценарію гарантує приведення системи до еталонного стану. NIST SP 800-53 також містить вимоги до відключення непотрібних функцій, портів та протоколів (principle of least functionality), що вимагає від адміністраторів глибокої "чистки" заводських конфігурацій обладнання.

Для практичної реалізації вимог безпеки на рівні конкретних команд та параметрів операційних систем використовуються стандарти CIS Benchmarks (Center for Internet Security) [4]. Це найбільш деталізовані технічні керівництва, які існують для більшості популярних платформ мережевого обладнання, включаючи Cisco IOS, Juniper JunOS, Palo Alto Networks та інші. CIS Benchmarks пропонують

покрокові інструкції з "закручування гайок" (hardening), розділені на два рівні. Рівень 1 (Level 1) включає базові рекомендації, які забезпечують суттєве підвищення безпеки без значного ризику порушення роботи сервісів (наприклад, налаштування банера попередження, відключення CDP на зовнішніх інтерфейсах, налаштування SSH v2). Рівень 2 (Level 2) призначений для середовищ з підвищеними вимогами до безпеки і включає більш жорсткі обмеження, які можуть вимагати ретельного планування (наприклад, сувора фільтрація ICMP повідомлень або налаштування 802.1X). Саме CIS Benchmarks найчастіше використовуються як основа для написання коду автоматизації, перетворюючи паперові вимоги на виконувані скрипти.

Окрему групу вимог становлять галузеві стандарти, серед яких найсуворішим є стандарт безпеки даних індустрії платіжних карток PCI DSS (Payment Card Industry Data Security Standard) [5]. Цей стандарт є обов'язковим для будь-якої організації, що зберігає, обробляє або передає дані платіжних карток. PCI DSS містить дванадцять основних вимог, перша з яких ("Install and maintain a firewall configuration to protect cardholder data") та друга ("Do not use vendor-supplied defaults for system passwords and other security parameters") безпосередньо стосуються конфігурації мережевого обладнання. Стандарт вимагає перегляду правил фаєрволу щонайменше раз на шість місяців, заборону будь-якого прямого доступу з Інтернету до сегменту з картковими даними (CDE) та обов'язкову зміну всіх стандартних паролів і SNMP-спільнот перед введенням пристрою в експлуатацію. Невиконання цих вимог загрожує величезними штрафами та втратою права обробляти платежі, що робить автоматизований контроль відповідності (Compliance checking) критично важливим бізнес-завданням.

У контексті вітчизняного законодавства України, вимоги до захисту інформації в інформаційно-телекомунікаційних системах (ІТС) регламентуються документами НД ТЗІ (Нормативні документи технічного захисту інформації), що розробляються та затверджуються Держспецзв'язку [29]. Для побудови Комплексних систем захисту інформації (КСЗІ) висуваються вимоги щодо захисту від несанкціонованого доступу, які корелюють з міжнародними практиками, проте мають свою специфіку в частині використання сертифікованих засобів

криптографічного захисту. Важливим аспектом є також гармонізація українського законодавства з європейським, зокрема в частині захисту персональних даних (GDPR), який вимагає застосування принципів "Privacy by Design" та "Privacy by Default" [28]. Це означає, що мережева інфраструктура повинна бути спроектована та налаштована таким чином, щоб забезпечувати максимальну приватність даних за замовчуванням, без необхідності додаткових дій з боку користувача.

Підсумовуючи огляд нормативної бази, можна зробити висновок, що сучасні вимоги до безпеки мережевих систем є надзвичайно високими і деталізованими. Спільним знаменником усіх розглянутих стандартів є вимога до контрольованості, повторюваності та прозорості процесів налаштування. Ручне адміністрування не здатне забезпечити виконання цих вимог у повному обсязі, особливо в частині безперервного моніторингу відповідності (Continuous Compliance) та швидкого реагування на нові вразливості. Стандарти фактично вимагають переходу від "ручного управління" до "управління політиками", де конфігурація обладнання є похідною від затвердженого коду. Це створює нормативно-правове підґрунтя для впровадження методів DevOps у мережеве адміністрування (NetDevOps) та використання таких інструментів, як Ansible, для забезпечення гарантованого рівня безпеки відповідно до кращих світових практик. У табл. 1.3 наведено порівняльний аналіз ключових стандартів та їх вплив на процес автоматизованого конфігурування. Ця таблиця демонструє, як абстрактні вимоги трансформуються в конкретні завдання для інженера з автоматизації.

Порівняльний аналіз нормативних вимог до конфігурації мережевої безпеки

Стандарт / Норматив	Рівень абстракції	Ключові вимоги до мережевого обладнання	Значення для автоматизації (Ansible)
ISO/IEC 27001 / 27002	Управлінський / Організаційний	Забезпечення управління вразливостями, контроль доступу до мереж, сегментація, логування подій.	Визначає загальну стратегію та необхідність регулярного аудиту, який неможливий вручну.
NIST SP 800-53	Технічний / Методологічний	CM-2 (Baseline Configuration), AC-4 (Information Flow Enforcement), SC-7 (Boundary Protection).	Вимагає створення "Golden Image" (еталонної конфігурації) та механізмів виявлення дрейфу (Config Drift).
CIS Benchmarks	Технічний / Конкретний	Вимкнути ip http server, встановити service password-encryption, налаштувати logging buffered.	Слугує прямою інструкцією для написання Ansible Tasks та Templates (конкретні команди).
PCI DSS	Галузевий (Фінансовий)	Вимога 1: Конфігурація Firewall. Вимога 2: Зміна дефолтних паролів. Вимога 11: Регулярне тестування.	Вимагає постійного звітування про відповідність (Compliance Reports) та швидкого закриття вразливостей.
НД ТЗІ (КСЗІ)	Національний	Захист від НСД, забезпечення цілісності, використання сертифікованих алгоритмів.	Вимагає адаптації міжнародних практик до національних вимог криптографії та звітності.

Висновки до першого розділу

Проведений комплексний аналіз принципів забезпечення безпеки мережевого обладнання дозволив сформулювати теоретичний базис для подальшої розробки автоматизованої системи. Встановлено, що в умовах сучасної кібервійни мережеве обладнання трансформувалося з пасивного середовища передачі даних у критично важливий актив, який виступає першою лінією оборони підприємства. Аналіз загроз показав, що вектори атак змістилися в бік інфраструктурного рівня: зловмисники активно експлуатують вразливості протоколів канального та мережевого рівнів (ARP, STP, OSPF), а також слабкі місця в конфігурації пристроїв. Детально розглянуто методи захисту, починаючи від класичних ACL та VPN і закінчуючи сучасними концепціями Next-Generation Firewall та

архітектурою Zero Trust. Виявлено, що ефективність цих засобів напряду залежить від коректності та актуальності їх налаштувань.

Критичний аналіз існуючих підходів до адміністрування виявив, що ручне налаштування конфігурацій є головним стримуючим фактором у забезпеченні надійності мереж. Вплив людського фактору, проблема "дрейфу конфігурацій", низька швидкість розгортання та неможливість масштабування роблять традиційні методи непридатними для сучасних динамічних середовищ. Водночас, огляд нормативної бази (ISO 27001, NIST, CIS, PCI DSS) підтвердив, що вимоги до безпеки стають дедалі суворішими, вимагаючи не лише разового налаштування, але й безперервного контролю відповідності еталонним параметрам.

На основі проведеного дослідження можна стверджувати, що існує об'єктивна необхідність у розробці та впровадженні методики автоматизованого налаштування безпеки мережевого обладнання. Автоматизація дозволить мінімізувати помилки персоналу, забезпечити ідемпотентність конфігурацій, прискорити впровадження змін та гарантувати відповідність нормативним вимогам. Це обґрунтовує доцільність використання програмного забезпечення Ansible як інструменту для вирішення окреслених проблем, що буде детально розглянуто та реалізовано у другому розділі роботи.

РОЗДІЛ 2. АНАЛІЗ ТЕХНОЛОГІЙ АВТОМАТИЗАЦІЇ ТА РОЗРОБКА МЕТОДИКИ НАЛАШТУВАННЯ КОНФІГУРАЦІЇ БЕЗПЕКИ МЕРЕЖЕВОГО ОБЛАДНАННЯ ЗАСОБАМИ ANSIBLE

2.1. Огляд сучасних інструментів автоматизації адміністрування мереж (Ansible, Puppet, Chef, SaltStack)

Сучасний етап розвитку інформаційно-комунікаційних технологій характеризується фундаментальною зміною парадигми управління інфраструктурою, яка переходить від ручного адміністрування окремих компонентів до програмного визначення стану цілих систем. Цей процес, відомий як "Інфраструктура як код" (Infrastructure as Code – IaC), став відповіддю на експоненціальне зростання складності мереж та вимог до швидкості розгортання сервісів [21]. Традиційні методи налаштування, що базувалися на використанні CLI (Command Line Interface) та індивідуальній майстерності адміністраторів, виявилися неспроможними забезпечити необхідний рівень масштабованості, надійності та безпеки в умовах динамічних бізнес-вимог. На ринку з'явилася ціла плеяда інструментів автоматизації, які дозволяють описувати цільовий стан інфраструктури у вигляді коду, що підлягає версіонуванню, тестуванню та автоматизованому застосуванню [22]. Серед найбільш впливових та розповсюджених платформ, що сформували ландшафт сучасної автоматизації, виділяються Ansible, Puppet, Chef та SaltStack. Кожен із цих інструментів пройшов свій унікальний еволюційний шлях, має специфічну архітектуру та філософію, що визначає їхню ефективність у вирішенні завдань налаштування мережевого обладнання [6].

Безумовним лідером у сфері мережевої автоматизації на сьогоднішній день є платформа Ansible, яка була створена Майклом ДеХааном у 2012 році та згодом придбана компанією Red Hat [7]. Популярність Ansible серед мережевих інженерів зумовлена його архітектурною простотою та принциповою відмовою від використання агентів (Agentless Architecture). У той час як більшість систем автоматизації серверів покладаються на встановлення спеціалізованого програмного забезпечення на керовані вузли, Ansible використовує стандартні протоколи віддаленого доступу, які вже присутні на пристроях за замовчуванням.

Для Linux-систем та більшості мережевого обладнання (Cisco, Juniper, Arista) це протокол SSH (Secure Shell), а для середовища Windows – протокол WinRM. Ця особливість є критичною перевагою саме в контексті мережевого адміністрування, оскільки операційні системи маршрутизаторів та комутаторів (наприклад, Cisco IOS або NX-OS) є закритими пропрієтарними платформами, що не дозволяють встановлення сторонніх агентів або бібліотек. Відсутність необхідності розгортати та підтримувати інфраструктуру агентів (Public Key Infrastructure, оновлення версій агентів, моніторинг їх стану) значно знижує поріг входження та спрощує початок роботи з інструментом.

Функціонування Ansible базується на модульній структурі, де основна логіка виконання зосереджена на керуючому вузлі (Control Node). Користувач описує бажаний стан системи у вигляді так званих "Playbooks" (сценаріїв), використовуючи просту та зрозумілу мову серіалізації даних YAML (Yet Another Markup Language) [10]. На відміну від складних мов програмування, YAML орієнтований на читабельність, що дозволяє інженерам, які не мають глибоких навичок розробки програмного забезпечення, швидко опанувати інструмент та створювати ефективні сценарії автоматизації. Під час виконання Playbook, Ansible перетворює описані задачі (Tasks) у відповідні команди або API-виклики, специфічні для конкретного типу обладнання, використовуючи відповідні модулі (наприклад, модуль `ios_config` для Cisco або `junos_config` для Juniper). Важливою концептуальною особливістю Ansible є ідемпотентність його модулів. Це означає, що система перед виконанням будь-якої дії перевіряє поточний стан пристрою. Якщо пристрій вже знаходиться у бажаному стані (наприклад, потрібний VLAN вже створений), Ansible не буде виконувати жодних команд, що запобігає дублюванню налаштувань та помилкам. Така поведінка гарантує передбачуваність результату незалежно від кількості запусків сценарію.

На противагу Ansible, система Puppet, розроблена Люком Канісом у 2005 році, представляє собою класичний приклад автоматизації на основі моделей (Model-Driven Automation) з використанням агентної архітектури. Puppet є одним із піонерів руху DevOps і довгий час залишався стандартом де-факто для управління конфігураціями серверів у великих корпоративних середовищах. Архітектура

Puppet базується на моделі "клієнт-сервер", де центральний сервер (Puppet Master) зберігає всю інформацію про конфігурації, а агенти (Puppet Agents), встановлені на цільових вузлах, періодично звертаються до сервера для отримання оновлень. Ця модель реалізує так званий "Pull-принцип" (тягнути): ініціатором оновлення виступає сам керований вузол, який запитує свою конфігурацію, порівнює її з поточним станом і, за необхідності, вносить зміни. Такий підхід забезпечує надзвичайно високий рівень відповідності (Compliance), оскільки агент постійно, за замовчуванням кожні 30 хвилин, перевіряє стан системи і автоматично виправляє будь-які відхилення (Configuration Drift), навіть якщо вони були зроблені адміністратором вручну.

Однак, застосування Puppet у мережевому середовищі пов'язане з низкою суттєвих технічних викликів. Оскільки встановити Puppet Agent безпосередньо на комутатор Cisco Catalyst або маршрутизатор Huawei зазвичай неможливо через обмеження операційної системи, Puppet використовує механізм проксі-вузлів або спеціальних модулів для роботи з мережевим обладнанням. У такій схемі агент запускається на окремому сервері і взаємодіє з мережевим пристроєм через SSH або API (NETCONF/RESTCONF), фактично емулюючи поведінку локального агента. Це ускладнює архітектуру, додаючи проміжний шар, який потребує адміністрування. Для опису конфігурацій Puppet використовує власну декларативну мову Puppet DSL (Domain Specific Language), яка є більш складною за YAML і вимагає розуміння концепцій об'єктно-орієнтованого програмування, таких як класи, ресурси, спадкування та параметризація. Незважаючи на це, для великих гетерогенних мереж, де критично важливим є безперервний контроль цілісності конфігурації та автоматичне виправлення змін, Puppet залишається потужним інструментом, особливо з огляду на його розвинену екосистему модулів (Puppet Forge) та інтеграцію з системами звітності.

Окремою перевагою Puppet є використання ієрархічної бази даних Hiera, яка дозволяє відокремити дані конфігурації (наприклад, IP-адреси, паролі, номери VLAN) від коду, що описує логіку налаштування. Це дозволяє створювати універсальні модулі, які можна застосовувати до різних груп пристроїв, просто підставляючи відповідні значення з бази даних. Такий підхід значно спрощує

управління конфігураціями у масштабних інфраструктурах, де кількість змінних параметрів може вимірюватися тисячами. Тим не менш, висока складність початкового налаштування інфраструктури Puppet (розгортання Master-серверів, налаштування PKI-сертифікатів для автентифікації агентів, вивчення DSL) часто стає бар'єром для мережеских команд, які шукають більш прості та швидкі способи автоматизації рутинних задач. У цьому контексті Puppet частіше обирають організації, які вже мають глибоку експертизу в Linux-адмініструванні та прагнуть уніфікувати підходи до управління серверами та мережею в рамках єдиної платформи.

Ще одним вагомим представником екосистеми "Інфраструктура як код" є платформа Chef, яка була представлена компанією Opscode (нині Chef Software Inc.) у 2009 році. Якщо Ansible та Puppet тяжіють до декларативного підходу (опис кінцевого стану), то Chef сповідує імперативну парадигму автоматизації. Це означає, що адміністратор не просто описує, як має виглядати система, а пише програмний код, який покроково виконує дії для досягнення цього стану. Основними будівельними блоками в Chef є "Рецепти" (Recipes) та "Кулінарні книги" (Cookbooks), які пишуться повноцінною мовою програмування Ruby. Ця архітектурна особливість робить Chef надзвичайно потужним інструментом у руках досвідчених розробників (DevOps Engineers), оскільки дозволяє реалізовувати будь-яку, навіть найскладнішу логіку налаштування з використанням умовних операторів, циклів, обробки виключень та взаємодії з зовнішніми бібліотеками [23]. Chef надає повний контроль над процесом виконання, дозволяючи "програмувати інфраструктуру" в прямому сенсі цього слова. Архітектура Chef є клієнт-серверною і складається з трьох основних компонентів: Chef Server (центральне сховище конфігурацій), Chef Workstation (робоча станція адміністратора, де створюються рецепти) та Chef Client (агент на цільових вузлах).

У контексті налаштування мережевого обладнання Chef стикається з тими ж викликами, що і Puppet, зумовленими неможливістю встановлення клієнтського програмного забезпечення (Ruby-інтерпретатора та Chef Client) на закриті операційні системи маршрутизаторів та комутаторів. Для вирішення цієї проблеми

Chef використовує концепцію "Knife" плагінів та спеціалізованих бібліотек, які дозволяють виконувати команди віддалено через SSH або API. Проте, високий поріг входження, зумовлений необхідністю вивчення мови Ruby, та складність підтримки інфраструктури Chef Server часто роблять цей інструмент надлишковим для задач мережевої автоматизації, якщо в компанії вже не використовується Chef для управління серверною фермою. Використання Chef виправдане у великих гетерогенних середовищах, де мережа розглядається як невід'ємна частина складного додатку, і де команда експлуатації має високу кваліфікацію в розробці програмного забезпечення. Імперативний підхід Chef дозволяє вирішувати нестандартні задачі, які можуть бути не під силу декларативним інструментам, наприклад, складні сценарії міграції даних або динамічна генерація конфігурацій на основі даних із зовнішніх баз даних у реальному часі.

Четвертим ключовим гравцем на ринку є SaltStack (або просто Salt), який з'явився у 2011 році і позиціонується як високопродуктивна система для управління конфігураціями та оркестрації. Головною відмінністю Salt від конкурентів є фокус на швидкості та масштабованості, що досягається за рахунок використання асинхронної шини повідомлень ZeroMQ (0MQ). Ця технологія дозволяє Salt-майстру (Master) надсилати команди тисячам мінійонів (Minions – агентам) одночасно і отримувати результати за частки секунди, що робить його найшвидшим інструментом у своєму класі. SaltStack написаний на мові Python (як і Ansible), що спрощує його розширення та написання власних модулів. Конфігурації описуються у файлах стану (State Files) з використанням формату YAML та шаблонізатора Jinja2, що робить його схожим на Ansible за синтаксисом, але відмінним за архітектурою виконання. Salt є надзвичайно гнучким інструментом, який підтримує як агентний режим роботи (для максимальної швидкості та контролю), так і безагентний режим через Salt-SSH (аналогічно до Ansible), що дозволяє використовувати його в ситуаціях, де встановлення агентів є небажаним.

Унікальною перевагою SaltStack для мережевої автоматизації є його реактивна природа, реалізована через систему "Reactor". Salt здатний не лише застосовувати конфігурації за розкладом, але й реагувати на події (Events), що генеруються

інфраструктурою, в реальному часі. Наприклад, якщо система моніторингу фіксує спробу несанкціонованого доступу або падіння інтерфейсу, вона може згенерувати подію в шину Salt, яка миттєво запусить відповідний сценарій реагування (Remediation): заблокує порт, змінить ACL або переключить трафік на резервний канал. Для роботи з мережевим обладнанням, яке не підтримує встановлення Python-агентів, SaltStack використовує архітектуру "Proxy Minions". Проксі-мінйон – це процес, що запускається на звичайному сервері і виступає посередником для конкретного мережевого пристрою, транслуючи команди Salt у виклики CLI або API цього пристрою. Це дозволяє управляти "тупими" залізяками так само ефективно, як і повноцінними серверами, інтегруючи їх у загальну подійно-орієнтовану інфраструктуру.

Підсумовуючи огляд сучасних інструментів автоматизації, можна констатувати, що ринок пропонує широкий спектр рішень, здатних задовольнити потреби організацій різного масштабу та технологічного профілю. Ansible, Puppet, Chef та SaltStack, маючи спільну мету – реалізацію підходу "Інфраструктура як код", досягають її різними шляхами. Ansible робить ставку на простоту, без-агентну архітектуру та низький поріг входження, що робить його ідеальним для мережеских інженерів. Puppet та Chef пропонують потужні засоби управління станом та відповідністю для великих ентєрпрайз-систем, але вимагають складної інфраструктури. SaltStack забезпечує неперевершену швидкість та реактивність, дозволяючи будувати самовідновлювальні системи. Однак, зважаючи на специфіку предметної області даної роботи – налаштування безпеки мережевого обладнання, яке часто є закритою екосистемою, критичними факторами вибору стають простота взаємодії з пристроями (SSH/API без агентів), легкість опису політик безпеки (YAML) та широка підтримка вендорів "з коробки". Саме ці аспекти вимагають детального порівняльного аналізу для обґрунтування вибору Ansible як основного інструменту дослідження.

2.2. Порівняльний аналіз технологій з точки зору автоматизації налаштувань безпеки

Процес вибору інструментарію для автоматизації конфігурацій безпеки мережевого обладнання є критичним етапом проектування системи захисту, який не може базуватися виключно на маркетинговій популярності платформи або її поширеності в сегменті адміністрування серверних операційних систем. Цей вибір вимагає глибокого, системного аналізу архітектурних особливостей кожного рішення крізь призму специфічних, часто жорстких вимог інформаційної безпеки та сумісності з пропрієтарними системами. Ключовим, фундаментальним критерієм розмежування розглянутих систем (Ansible, Puppet, Chef, SaltStack) є базова модель взаємодії керуючого сервера з керованими вузлами, а саме архітектурна дихотомія "агентна" проти "безагентної" моделі. З точки зору мінімізації площини атаки (Attack Surface Reduction) на мережевий периметр, наявність будь-якого додаткового програмного забезпечення (агента) на критично важливому інфраструктурному обладнанні, такому як маршрутизатори ядра, агрегатори трафіку або міжмережеві екрани, розглядається як потенційний, неприйнятний ризик.

Агент, у контексті систем автоматизації, представляє собою сторонній бінарний код, який працює у фоновому режимі з підвищеними привілеями (часто root або адміністратор), потребує регулярного оновлення версій і сам по собі може містити програмні вразливості (як відомі, так і Zero-Day), що можуть бути використані зловмисниками для підвищення прав або прихованого закріплення в системі. У цьому контексті платформи Puppet і Chef, які архітектурно орієнтовані на обов'язкове встановлення агентів для повноцінної роботи, створюють додатковий вектор загрози. Компрометація сервера оновлень агентів або самого агента може призвести до ситуації, коли зловмисник отримає повний контроль над усіма вузлами мережі, оминаючи традиційні засоби захисту. Крім того, на переважну більшість пропрієтарних мережевих операційних систем (таких як Cisco IOS, Cisco ASA, екзотичні версії Huawei VRP) встановити стороннього агента технічно неможливо через закритість файлової системи та відсутність доступу до оболонки Linux.

Для обходу цього обмеження системи Puppet та Chef змушені використовувати архітектурні "милиці" у вигляді проміжних вузлів (proxy nodes) або бастіонних хостів. Це значно ускладнює топологію мережі управління та створює додаткову єдину точку відмови (Single Point of Failure) і вразливості. Проксі-вузол стає критичним елементом інфраструктури: його компрометація автоматично надає зловмиснику легітимний доступ до всіх мережевих пристроїв, які цей вузол обслуговує, оскільки саме на ньому зберігаються облікові дані для доступу. Також це вимагає налаштування складних правил на фаєрволах для дозволу трафіку між проксі та пристроями, що суперечить принципам сегментації. Натомість, платформи Ansible та SaltStack (у специфічному режимі Salt-SSH) пропонують нативну підтримку безагентного підходу, який є найбільш відповідним та органічним для закритих екосистем мережевого обладнання.

Використання стандартного, перевіреного часом протоколу SSH (Secure Shell) для транспортування команд управління дозволяє мінімізувати втручання в архітектуру кінцевого пристрою та уникнути встановлення зайвого ПЗ [24]. З позиції безпеки мережевого периметра це означає, що адміністраторам не потрібно відкривати додаткові нестандартні порти на внутрішніх та зовнішніх фаєрволах (як, наприклад, порти 4505-4506 для протоколу ZeroMQ у SaltStack або порт 8140 для комунікації Puppet Master-Agent), що значно спрощує конфігурацію та аудит списків контролю доступу (ACL). Менша кількість відкритих портів прямо корелює зі зменшенням видимості системи управління для зовнішнього спостерігача та зниження ймовірності успішного сканування мережі. Крім того, Ansible ефективно використовує вже існуючу на підприємстві інфраструктуру ключів SSH та інтегрується з корпоративними механізмами автентифікації (RADIUS, TACACS+, LDAP), дозволяючи вбудувати процес автоматизації в існуючі системи управління привілейованим доступом (PAM) без необхідності розгортання та підтримки паралельної системи публічних ключів (PKI), як того вимагає Puppet для взаємної автентифікації агентів. Це значно спрощує процедури аудиту та відповідності стандартам (Compliance), оскільки всі дії з автоматизації проходять через стандартні, добре контрольовані канали адміністрування, які легко логуються на рівні AAA-серверів.

Ще одним критичним аспектом порівняльного аналізу, що впливає на оперативну безпеку, є модель ініціації змін: Push (штовхати) проти Pull (тягнути). Системи Puppet та Chef за замовчуванням використовують модель Pull, де агенти періодично, з певним інтервалом (зазвичай 30 хвилин), опитують центральний сервер на наявність змін у конфігурації. Ця модель ідеально підходить для підтримки стабільного стану серверного парку та боротьби з дрейфом конфігурацій (Configuration Drift Management) у довгостроковій перспективі, але має суттєвий недолік у контексті оперативного реагування на інциденти безпеки. У разі виявлення активної кібератаки або необхідності термінового блокування шкідливого трафіку (наприклад, ізоляція зараженого сегменту мережі), адміністратор не має часу чекати наступного циклу синхронізації агента. Примусова ініціація оновлення на тисячах вузлів у моделі Pull може бути складним та ресурсоємним завданням, що створює затримку між прийняттям рішення та його виконанням.

Ansible, працюючи за архітектурною моделлю Push, надає адміністратору повний контроль над часом застосування змін. Це дозволяє миттєво ініціювати виконання сценарію (Playbook) з керуючого вузла, забезпечуючи можливість реагування в режимі реального часу. Наприклад, масове оновлення правила ACL на всіх прикордонних маршрутизаторах для блокування IP-адреси ботнету або відключення скомпрометованого порту комутатора займає лічені секунди і виконується детерміновано. Хоча SaltStack також підтримує миттєве виконання команд завдяки швидкій шині повідомлень, складність безпечного налаштування його транспортного шару та вимога відкриття додаткових портів часто поступається простоті та надійності прямого SSH-з'єднання в Ansible, яке легше контролювати та тунелювати через захищені шлюзи (Bastion Hosts).

Надзвичайно важливим фактором у забезпеченні цілісності системи автоматизації є управління секретами та чутливими даними, такими як паролі доступу до обладнання, приватні ключі шифрування, API-токени та SNMP-спільноти. Зберігання цих даних у відкритому вигляді у скриптах є неприпустимим порушенням базових практик безпеки. Усі розглянуті системи мають вбудовані механізми захисту даних: Ansible Vault, Puppet Hiera-eyaml, Chef Encrypted Data

Bags та Salt Pillar. Однак реалізація Ansible Vault виділяється своєю елегантною простотою та безшовною інтеграцією в робочий процес інженера (Developer Experience). Ansible Vault дозволяє шифрувати як окремі змінні, так і цілі файли конфігурації безпосередньо в репозиторії коду, використовуючи надійне симетричне шифрування AES-256. Це дозволяє безпечно зберігати плейбуки у системах контролю версій (Git) разом із зашифрованими секретами, не порушуючи політик безпеки і не ризикуючи витоком даних у разі компрометації репозиторію. У системах Puppet та Chef управління секретами часто вимагає розгортання додаткових сервісів, складної конфігурації сертифікатів на кожному агенті або інтеграції з зовнішніми сховищами (HashiCorp Vault) на ранніх етапах, що підвищує поріг входження та ризик помилки адміністратора. Простота механізму Vault знижує ймовірність того, що інженер через лінь або складність інструменту збереже пароль у відкритому вигляді "для зручності", що є однією з найпоширеніших причин витоку облікових даних.

Окремо слід розглянути питання цілісності виконання коду та довіри до середовища виконання. У випадку з Chef та Puppet, де код (складні скрипти на Ruby) передається на клієнтську сторону і виконується там (або на проксі-вузлі), існує теоретична можливість маніпуляції логікою виконання, якщо злоумисник отримає локальний контроль над вузлом. Ansible використовує інший підхід: він генерує Python-код або набір CLI-команд на стороні захищеного керуючого вузла (Control Node) і передає їх на цільовий пристрій лише для безпосереднього виконання. Для мережевих пристроїв, що управляються через модулі `network_cli` або `NETCONF`, це означає, що на самому комутаторі не виконується ніяка складна бізнес-логіка, він лише отримує атомарні команди конфігурації, ідентичні тим, що вводив би оператор вручну. Це робить процес абсолютно прозорим і зрозумілим для мережевих інженерів, дозволяючи легко перевірити, які саме команди будуть надіслані, використовуючи режим "Check Mode" (Dry Run). Ця можливість є критично важливою перед застосуванням змін у продуктивному середовищі, оскільки дозволяє виявити потенційні конфлікти або помилки до того, як вони вплинуть на доступність мережі.

Ще більш вагомим аспектом порівняння, ніж суто технічна архітектура, є мова опису конфігурацій та пов'язаний з нею поріг входження для персоналу. У контексті інформаційної безпеки існує фундаментальна аксіома: "складність – це ворог безпеки". Чим складнішим є інструмент та мова, якою описуються правила захисту, тим вищою є ймовірність виникнення логічних помилок, вразливостей у самому кодї автоматизації та неправильної інтерпретації намірів адміністратора колегами. У цьому аспекті Ansible, що використовує мову серіалізації даних YAML (Yet Another Markup Language), має кардинальну перевагу перед конкурентами, які покладаються на повноцінні імперативні мови програмування (Ruby у Chef) або складні предметно-орієнтовані мови (DSL у Puppet).

YAML за своїм визначенням та природою є декларативним форматом даних, а не виконуваним кодом. Це означає, що інженер описує "що" має бути налаштовано (кінцевий бажаний стан, наприклад, "порт 22 закритий"), а не пише алгоритм "як" це зробити (цикли, умовні переходи, змінні, перевірки станів). Така семантична прозорість дозволяє мінімізувати когнітивне навантаження на оператора: файл конфігурації читається майже як звичайна англійська технічна документація. Це є критично важливим фактором при проведенні аудитів безпеки, коли внутрішній або зовнішній аудитор (який може не бути професійним програмістом на Ruby) повинен швидко та однозначно зрозуміти, чи відповідає поточне налаштування затвердженій політиці безпеки компанії.

На противагу цьому, використання Chef вимагає від мережевого інженера глибоких, професійних знань мови Ruby та принципів об'єктно-орієнтованого програмування. Хоча це надає практично необмежену гнучкість, дозволяючи створювати надзвичайно складні динамічні сценарії, воно також відкриває широкий простір для критичних помилок (Human Error). У скриптах на Ruby можна легко припуститися помилки в логіці циклу, неправильно обробити виключення або створити нескінченний цикл, що може призвести до непередбачуваних та руйнівних наслідків, наприклад, випадкового видалення всіх критичних маршрутів BGP або адміністративного вимкнення всіх Uplink-інтерфейсів на комутаторі ядра. Більше того, імперативний код значно складніше аналізувати автоматизованими засобами (статичними аналізаторами коду) на предмет відповідності безпеці.

Концепція "Код як інфраструктура" у виконанні Chef фактично вимагає перетворення мережових адміністраторів на розробників програмного забезпечення, що тягне за собою необхідність впровадження складних процесів тестування (Unit tests, Integration tests), які історично рідко практикуються в експлуатації мереж. Puppet зі своєю власною мовою DSL займає проміжну позицію, але його синтаксис є специфічним, неінтуїтивним для новачків і вимагає тривалого часу на вивчення, що створює бар'єр для швидкого впровадження змін у критичних ситуаціях.

Людський фактор, психологія інженерії та кваліфікація персоналу відіграють вирішальну роль у забезпеченні надійності системи автоматизації. Типовий профіль компетенцій мережевого інженера включає глибоке розуміння стеку протоколів TCP/IP, роботи протоколів маршрутизації (OSPF, BGP), технологій VPN (IPsec) та архітектури комутації, але рідко включає навички розробки програмного забезпечення на Ruby або Python на рівні Middle/Senior. Примусове впровадження складних інструментів на кшталт Chef або SaltStack створює небезпечний розрив у компетенціях команди. Це часто призводить до ситуації, відомої як "Bus Factor = 1", коли інструмент автоматизації стає "чорною скринькою", яку розуміє та підтримує лише одна людина в команді, а решта персоналу боїться вносити зміни, щоб "нічого не зламати". У контексті реагування на інциденти безпеки це є катастрофічним сценарієм, оскільки відсутність ключового спеціаліста може паралізувати здатність компанії захищатися. Ansible, завдяки простоті YAML, дозволяє залучити до процесу автоматизації всю команду. Інженер, який володіє лише знаннями CLI команд Cisco, може за один-два дні навчитися перекладати їх у формат плейбуків Ansible. Це демократизує безпеку, роблячи її зрозумілою, прозорою та доступною для всіх учасників процесу експлуатації, що значно підвищує загальну стійкість системи.

Ще одним важливим аспектом порівняння є стабільність, детермінованість та ідемпотентність виконання коду. У системах, що базуються на повноцінних мовах програмування, існує ризик виникнення побічних ефектів (side effects) та недетермінованої поведінки, залежної від оточення виконання (версії бібліотек, змінні середовища). Ansible, використовуючи модульний підхід, значно жорсткіше

контролює виконання завдань [25]. Модулі Ansible (наприклад, `cisco.ios.ios_config`) є самодостатніми, інкапсульованими блоками коду, написаними професійними розробниками та протестованими вендорами обладнання. Користувач лише передає параметри в ці модулі, не втручаючись у логіку їх роботи. Це значно знижує ризик того, що "самописний" скрипт призведе до порушення цілісності конфігурації. У безпеці мереж, де ціна помилки вимірюється простим бізнесом або витоком конфіденційних даних, така передбачуваність є вагомим аргументом, ніж абстрактна гнучкість програмування.

Крім того, екосистема та підтримка з боку вендорів мережевого обладнання є вирішальним фактором. Компанії-лідери ринку, такі як Cisco, Juniper, Arista, Palo Alto Networks, Check Point, офіційно сертифікують та розробляють колекції модулів саме для Ansible. Це гарантує, що нові функції безпеки, які з'являються в оновленнях операційних систем, будуть оперативно підтримані в інструментарії автоматизації. Користувачі альтернативних систем часто змушені покладатися на модулі, розроблені спільнотою, які можуть бути застарілими або містити помилки. Нарешті, аспект документування інфраструктури ("Compliance as Code") в Ansible реалізований найбільш ефективно: плейбуки самі по собі виступають актуальною документацією, зрозумілою навіть для менеджменту, на відміну від складного коду Chef або Puppet. Проведений детальний аналіз дозволяє систематизувати ключові відмінності між провідними платформами автоматизації в контексті їх застосування для задач мережевої безпеки. Для наочності результати порівняння зведено у таблицю 2.1.

Таблиця 2.1.

Порівняльна характеристика засобів автоматизації для задач мережевої безпеки

Критерій порівняння	Ansible	Puppet	Chef	SaltStack
Архітектура	Agentless (тільки SSH). Ідеально для мережевого обладнання.	Agent-based. Вимагає Proxy-вузлів для мережевих пристроїв.	Agent-based. Вимагає Workstation/Proxy.	Гібридна (Agent/Agentless). Складна конфігурація Proxy Minions.

Продовження таблиці 2.1.

Модель управління	Push (Командна). Миттєва реакція на інциденти.	Pull (Періодична). Затримка в застосуванні змін.	Pull.	Push/Pull (Event-driven). Висока швидкість.
Мова опису	YAML (Декларативна). Низький поріг входження.	Puppet DSL Специфічний синтаксис.	Ruby (Імперативна). Вимагає навичок програмування.	YAML + Jinja2 (Python). Схоже на Ansible, але складніше.
Управління секретами	Ansible Vault. Просте шифрування файлів у репозиторії.	Hiera-eyaml. Складна PKI інфраструктура.	Encrypted Data Bags. Складна реалізація.	Salt Pillar. Гнучко, але складно в налаштуванні.
Мережева підтримка	Native. Офіційні модулі від Cisco, Juniper, CheckPoint тощо.	Через додаткові модулі/проксі. Обмежена підтримка вендорів.	Обмежена. Орієнтація на сервери.	Хороша (через NAPALM), але менша база модулів ніж в Ansible.
Поріг входження	Низький. Доступний для мережеских інженерів без навичок кодингу.	Середній/Високий. Вимагає вивчення DSL та архітектури.	Високий. Потрібне знання Ruby.	Середній/Високий. Вимагає знання Linux/Python.
Безпека	Стандартний SSH . Не вимагає додаткових відкритих портів.	SSL (порт 8140). Потребує управління сертифікатами.	SSL/HTTPS.	ZeroMQ (порти 4505-4506) або SSH.

На основі проведеного аналізу можна зробити обґрунтований висновок, що Ansible є найбільш оптимальним інструментом для розробки методики автоматизованого налаштування конфігурації безпеки мережевого обладнання. Його безагентна архітектура усуває необхідність модифікації пропрієтарного ПЗ мережеских пристроїв та зменшує поверхню атаки. Використання моделі Push забезпечує можливість оперативного реагування на інциденти, а декларативна мова YAML дозволяє залучати до процесу автоматизації профільних мережеских фахівців, мінімізуючи ризики помилок, пов'язаних з людським фактором. Саме тому подальша розробка методики у цій роботі буде базуватися на використанні програмного забезпечення Ansible.

2.3. Архітектура, принципи роботи і можливості Ansible у мережескій автоматизації

Для глибокого розуміння процесів автоматизації налаштувань безпеки мережевого обладнання необхідно детально розглянути внутрішню архітектуру

програмного комплексу Ansible, принципи його взаємодії з керованими об'єктами та логіку організації даних. Архітектура Ansible побудована за радіальним принципом, де центральним елементом виступає керуючий вузол (Control Node), а на периферії знаходяться керовані вузли (Managed Nodes). Керуючий вузол являє собою комп'ютер або сервер під управлінням Unix-подібної операційної системи (Linux, macOS, BSD), на якому встановлено середовище виконання Python та власне програмне забезпечення Ansible. Саме на цьому вузлі зосереджена вся інтелектуальна складова системи: тут зберігаються файли конфігурації, сценарії автоматизації (Playbooks), шаблони (Templates) та модулі. Важливою особливістю є те, що Ansible не має фонових демонів або баз даних, які повинні постійно працювати. Він функціонує як звичайний процес CLI (Command Line Interface), який запускається користувачем, виконує поставлені завдання і завершує свою роботу. Це суттєво знижує вимоги до ресурсів керуючого вузла і спрощує його обслуговування.

Керовані вузли в контексті даної роботи – це мережеві пристрої (маршрутизатори, комутатори, міжмережеві екрани, балансувальники навантаження), на яких застосовуються політики безпеки. Фундаментальною відмінністю архітектури Ansible від інших систем є відсутність необхідності встановлення будь-якого клієнтського програмного забезпечення (агентів) на ці вузли. Взаємодія здійснюється через стандартні мережеві протоколи, переважно SSH (Secure Shell) або HTTPS (для пристроїв з підтримкою REST API). Ця архітектурна парадигма, відома як "Agentless", є критично важливою для мережевої автоматизації, оскільки більшість мережевого обладнання працює під управлінням закритих пропрієтарних операційних систем, які не дозволяють встановлення сторонніх інтерпретаторів мов програмування, таких як Python або Ruby. Таким чином, Ansible виступає в ролі зовнішнього оркестратора, який підключається до пристрою так само, як це робив би адміністратор-людина, але виконує команди з машинною швидкістю та точністю.

Основою організації керованої інфраструктури в Ansible є система інвентаризації (Inventory). Інвентар – це база даних хостів, яка описує топологію мережі, логічне групування пристроїв та параметри підключення до них. У

найпростішому вигляді інвентар може бути представлений статичним текстовим файлом у форматі INI або YAML, де перераховано IP-адреси або доменні імена пристроїв. Однак для складних корпоративних мереж можливості інвентаризації значно ширші. Ansible дозволяє створювати ієрархічні структури груп, де одна група може бути нащадком іншої (nested groups). Наприклад, можна створити загальну групу "Network", до якої входитимуть підгрупи "Routers" та "Switches", а група "Switches" у свою чергу може ділитися на "Core_Switches" та "Access_Switches". Така ієрархія дозволяє застосовувати змінні та політики безпеки на різних рівнях абстракції: загальні налаштування NTP та DNS можна застосувати до батьківської групи "Network", а специфічні налаштування Port Security – лише до групи "Access_Switches".

Крім статичних файлів, Ansible підтримує концепцію динамічної інвентаризації (Dynamic Inventory). Це механізм, який дозволяє отримувати список хостів у реальному часі із зовнішніх джерел даних, таких як хмарні провайдери (AWS, Azure, Google Cloud), системи віртуалізації (VMware vSphere) або системи управління IP-адресним простором (IPAM, наприклад, NetBox). Динамічний інвентар реалізується за допомогою спеціальних скриптів або плагінів, які при запуску Ansible звертаються до API зовнішньої системи, отримують JSON-відповідь з актуальним списком пристроїв та їх атрибутами і трансформують її у внутрішній формат Ansible. Це є критично важливим для сучасних еластичних середовищ, де кількість та адресація пристроїв можуть змінюватися динамічно. Використання динамічної інвентаризації гарантує, що політики безпеки будуть застосовані до всіх актуальних пристроїв у мережі, виключаючи ризик того, що нововведений в експлуатацію комутатор залишиться неналаштованим через те, що адміністратор забув додати його IP-адресу в статичний файл.

Система інвентаризації Ansible тісно пов'язана з системою змінних, які визначають поведінку пристроїв та параметри їх конфігурації. Змінні можуть бути визначені безпосередньо у файлі інвентарю (що не рекомендується для великих проектів через захащення коду) або у спеціальних директоріях `group_vars` та `host_vars`. Директорія `group_vars` містить файли YAML, назви яких відповідають назвам груп в інвентарі. Наприклад, файл `group_vars/cisco_routers.yml` міститиме

змінні, спільні для всіх маршрутизаторів Cisco (наприклад, тип операційної системи `ansible_network_os: cisco.ios`, метод підключення `ansible_connection: network_cli`, облікові дані користувача `ansible_user`). Директорія `host_vars` використовується для визначення унікальних параметрів конкретного хоста, таких як IP-адреса інтерфейсу управління `ansible_host` або специфічні налаштування інтерфейсів. Ansible має чітко визначений пріоритет змінних (Variable Precedence), де змінні конкретного хоста перекривають змінні групи, а змінні, передані через командний рядок під час запуску, мають найвищий пріоритет. Це дозволяє гнучко керувати конфігурацією, встановлюючи дефолтні значення на рівні груп і перевизначаючи їх для виняткових випадків.

Особливу увагу в архітектурі інвентаризації слід приділити спеціальним системним змінним, які керують процесом підключення. Для мережевого обладнання ключовою змінною є `ansible_connection`. Якщо для Linux-серверів за замовчуванням використовується значення `ssh` (яке передбачає використання SFTP для передачі файлів та виконання Python-коду на віддаленому вузлі), то для мережевих пристроїв необхідно використовувати спеціалізовані плагіни з'єднання, такі як `network_cli`, `netconf` або `httpapi`. Плагін `network_cli` вказує Ansible, що взаємодія з пристроєм буде відбуватися шляхом емуляції інтерактивної сесії в командному рядку, без спроби завантажити та виконати Python-скрипти на стороні пристрою. Змінна `ansible_network_os` визначає тип платформи (наприклад, `cisco.ios`, `junipernetworks.junos`, `arista.eos`), що дозволяє системі завантажити відповідні драйвери для обробки специфіки синтаксису команд, промптів (prompts) та обробки помилок конкретного вендора. Правильне налаштування інвентарю є фундаментом стабільної роботи Ansible, оскільки помилки на цьому етапі унеможливають коректну ідентифікацію пристроїв та вибір правильних методів взаємодії.

Також варто зазначити, що Ansible підтримує використання шаблонів (patterns) для вибору цільових хостів при запуску сценаріїв. Адміністратор може вказати не лише конкретне ім'я хоста або групи, але й використовувати логічні оператори, регулярні вирази та перетин множин. Наприклад, конструкція `webserver:&production` дозволяє вибрати лише ті сервери, які входять одночасно

до групи веб-серверів і групи продуктивного середовища, виключаючи тестові сервери. Оператор виключення `!` дозволяє, наприклад, оновити всі комутатори в офісі, крім ядра мережі (`office_switches:!core`). Така гнучкість адресації дозволяє здійснювати хірургічно точні операції в інфраструктурі, мінімізуючи ризик випадкового впливу на критичні сервіси під час проведення планових робіт або реагування на інциденти безпеки. У поєднанні з лімітами виконання (параметр `--limit`), система інвентаризації Ansible надає повний контроль над обсягом та географією застосування змін.

Розуміння механізмів взаємодії Ansible з мережевим обладнанням вимагає глибокого аналізу відмінностей у процесах виконання коду між серверними системами загального призначення та спеціалізованими мережевими пристроями. У стандартній моделі автоматизації Linux-серверів Ansible працює за схемою, яку можна охарактеризувати як "копіюй та виконуй". Керуючий вузол встановлює SSH-з'єднання з віддаленим сервером, запаковує модуль (написаний на Python) у ZIP-архів, передає його на сервер через SFTP або SCP у тимчасову директорію, розпаковує та запускає локально на віддаленому вузлі. Після завершення роботи модуль повертає результат у форматі JSON, який передається назад на керуючий вузол, а тимчасові файли видаляються. Цей підхід є надзвичайно ефективним, оскільки дозволяє використовувати обчислювальні потужності віддалених серверів для виконання логіки модуля. Однак, при спробі застосувати цю модель до мережевого обладнання виникають фундаментальні перешкоди. Більшість маршрутизаторів та комутаторів (Legacy network devices) працюють на закритих операційних системах, які не мають доступу до файлової системи для користувача, не підтримують протоколи передачі файлів SFTP, і, що найголовніше, не мають встановленого інтерпретатора Python, здатного виконувати код модулів Ansible.

Для вирішення цієї проблеми в архітектурі Ansible було реалізовано окремий клас механізмів, які переносять виконання логіки модуля з віддаленого вузла назад на керуючий вузол (Local Execution). У цій парадигмі модуль Ansible запускається локально на комп'ютері адміністратора, а взаємодія з мережевим пристроєм зводиться до відправки команд та отримання текстового виводу через встановлений канал зв'язку. Ключову роль у цьому процесі відіграють плагіни

з'єднання (Connection Plugins), серед яких основним для роботи з CLI-інтерфейсами є `network_cli`. Цей плагін виступає інтелектуальним посередником, який емулює роботу людини-оператора. Він не просто відкриває SSH-сесію, а й розуміє контекст командного рядка: розпізнає запрошення системи (prompts), відрізняє режим користувача від привілейованого режиму, коректно обробляє помилки вводу та тайм-аути. Фактично, `network_cli` забезпечує абстракцію, перетворюючи неструктурований потік текстових даних консолі у структуровану взаємодію, дозволяючи модулям вищого рівня оперувати поняттями "інтерфейс" або "VLAN", а не просто набором символів.

На транспортному рівні, що лежить нижче плагінів з'єднання, Ansible використовує бібліотеки для реалізації протоколу SSH. За замовчуванням для `network_cli` використовується бібліотека `paramiko` – нативна реалізація протоколу SSHv2 мовою Python. Вибір `paramiko` зумовлений її стабільністю, простотою інтеграції в Python-середовище та відсутністю зовнішніх залежностей від системних бібліотек операційної системи. Вона забезпечує шифрування каналу, автентифікацію за паролем або ключем та підтримку підсистеми SSH. Альтернативою є використання `libssh` (через плагін `ssh`), яка базується на системній бібліотеці OpenSSH. Хоча `libssh` може забезпечувати вищу швидкість передачі даних, `paramiko` залишається стандартом для мережевої автоматизації через кращу сумісність із застарілими реалізаціями SSH-серверів, які часто зустрічаються на старому мережевому обладнанні (наприклад, пристрої, що підтримують лише алгоритми обміну ключами `diffie-hellman-group1-sha1`).

Критично важливим аспектом роботи транспортного рівня при масовому налаштуванні мережі є продуктивність встановлення з'єднань. Процес встановлення захищеного SSH-тунелю (Handshake) є ресурсоемною операцією, яка включає обмін криптографічними ключами та автентифікацію. Якщо плейбук містить 10 завдань для одного пристрою, відкриття та закриття нової SSH-сесії для кожного завдання призведе до неприпустимих затримок, збільшуючи час виконання сценарію в десятки разів. Для вирішення цієї проблеми Ansible використовує механізм постійних з'єднань (Persistent Connections). При використанні `network_cli`, Ansible запускає фоновий процес (`control socket`), який

утримує відкриту SSH-сесію з мережевим пристроєм протягом виконання всього плейбука. Всі модулі, що виконуються в рамках одного сценарію для даного хоста, використовують цей вже встановлений канал зв'язку, передаючи команди через локальний сокет на керуючому вузлі. Це дозволяє досягти швидкості виконання, порівнянної з локальними операціями, та знизити навантаження на процесор (CPU) мережевого пристрою, який звільняється від необхідності постійно генерувати нові сесійні ключі.

Окремою проблемою, яку вирішує архітектура Ansible у роботі з мережевим обладнанням, є управління привілеями (Privilege Escalation). У світі Linux/Unix для виконання адміністративних завдань використовується механізм `sudo`, який дозволяє звичайному користувачеві тимчасово отримати права суперкористувача (`root`). На мережевих пристроях, зокрема на Cisco IOS, еквівалентом є перехід у режим `enable` (Privileged EXEC mode). Ansible абстрагує цю відмінність за допомогою системи `become`. У плейбуку адміністратор вказує директиву `become: yes` та метод підвищення привілеїв `become_method: enable`. Плагін з'єднання автоматично розпізнає цю директиву, і після успішного SSH-підключення надсилає команду `enable`, очікує запиту пароля (якщо він налаштований) та вводить пароль з відповідної змінної (`ansible_become_password`). Така абстракція дозволяє писати універсальні сценарії, де логіка підвищення прав відокремлена від логіки виконання завдань, що спрощує підтримку коду для гетерогенних мереж, де команди підвищення привілеїв можуть відрізнятися (наприклад, `enable` у Cisco vs `su` у деяких Linux-подібних комутаторах).

Крім традиційного CLI, сучасна архітектура Ansible активно підтримує роботу з пристроями через програмні інтерфейси (API), використовуючи плагіни `netconf` та `httpapi`. Плагін `netconf` реалізує підтримку протоколу NETCONF (Network Configuration Protocol), який використовує XML для кодування даних конфігурації та стану пристрою [14]. Це дозволяє перейти від ненадійного "екранного скрапінгу" (`screen scraping`), коли система аналізує текстовий вивід, до надійної транзакційної моделі управління. При використанні NETCONF Ansible може точно знати, чи була застосована конфігурація успішно, чи виникла помилка, оскільки пристрій повертає структуровану відповідь. Аналогічно, плагін `httpapi` дозволяє взаємодіяти

з пристроями, що надають REST API (наприклад, F5 BIG-IP, Arista EOS з eAPI, Cisco NX-OS). У цьому випадку Ansible надсилає HTTP/HTTPS запити з JSON-навантаженням. Використання API-орієнтованих плагінів є пріоритетним для нових інфраструктур, оскільки вони забезпечують вищу швидкість, надійність та ідемпотентність, однак підтримка CLI залишається критично важливою для забезпечення сумісності з величезним парком існуючого обладнання.

Також варто звернути увагу на механізм обробки "термінальних особливостей" мережевих пристроїв. Плагіни з'єднання Ansible містять вбудовану логіку для "нормалізації" термінальної сесії. При підключенні система автоматично надсилає команди для відключення посторінкового виводу (наприклад, `terminal length 0` у Cisco IOS), щоб пристрій видавав всю інформацію суцільним потоком, а не чекав натискання клавіші пробіл. Це запобігає зависанню сесії очікування вводу користувача. Крім того, налаштування таймаутів (`command timeout` та `connect timeout`) дозволяє адаптувати роботу Ansible до повільних каналів зв'язку або перевантажених пристроїв, запобігаючи розриву з'єднання під час виконання тривалих операцій, таких як генерація криптографічних ключів або копіювання образу операційної системи. Всі ці низькорівневі нюанси приховані від кінцевого користувача, який оперує високорівневими абстракціями, проте саме надійна робота транспортного рівня та плагінів з'єднання робить можливою автоматизацію "примхливого" мережевого заліза.

Функціональна потужність Ansible базується на його модульній архітектурі, де кожен окремий крок автоматизації реалізується через виконання спеціалізованого блоку коду, який називається модулем. Модулі – це самодостатні, автономні скрипти, які виконують конкретну, вузькоспеціалізовану задачу, наприклад, додавання користувача, встановлення пакета, копіювання файлу або, у випадку мережевого обладнання, налаштування VLAN чи інтерфейсу. В архітектурі Ansible модулі відокремлені від ядра системи (Ansible Core) та фреймворку виконання, що дозволяє розширювати функціональність платформи без необхідності оновлення самого двигуна. Для роботи з мережевим обладнанням використовуються спеціалізовані колекції модулів (Ansible Content Collections), які розробляються вендорами обладнання або спільнотою. Наприклад, колекція

cisco.ios містить модулі для управління пристроями на базі IOS, junipernetworks.junos – для JunOS, а vyos.vyos – для платформи VyOS. Такий підхід забезпечує абстракцію високого рівня: адміністратору не потрібно знати специфічний синтаксис команд CLI для кожного вендора, достатньо оперувати уніфікованими параметрами модуля, такими як name, state або ipv4_address.

Ключовою характеристикою модулів Ansible, яка відрізняє їх від простих скриптів на Bash або Python, є ідемпотентність. Ідемпотентність – це властивість операції, яка гарантує, що багаторазове виконання однієї й тієї ж дії призведе до незмінного результату без виникнення побічних ефектів. У контексті налаштування безпеки це є критично важливим. Якщо адміністратор запускає скрипт, що додає правило в ACL, звичайний скрипт просто надішле команду в консоль, що може призвести до дублювання записів або помилки, якщо такий запис вже існує. Ідемпотентний модуль Ansible спочатку перевіряє поточний стан пристрою (State Checking), аналізує конфігурацію і порівнює її з бажаним станом, описаним у сценарії. Тільки у випадку виявлення розбіжностей (Delta), модуль генерує та надсилає необхідні команди для приведення системи до цільового стану. Якщо ж конфігурація вже відповідає вимогам, модуль не виконує жодних дій і повертає статус "OK" (без змін). Це дозволяє безпечно запускати плейбуки налаштування безпеки за розкладом (наприклад, кожні 15 хвилин) для автоматичного виправлення будь-яких несанкціонованих змін (Configuration Drift), гарантуючи, що політика безпеки завжди дотримується.

Механіка забезпечення ідемпотентності для мережевих пристроїв є значно складнішою, ніж для серверів, оскільки вимагає парсингу текстових конфігурацій. Коли модуль, наприклад cisco.ios.ios_vlans, отримує завдання створити VLAN 10, він спочатку виконує команду читання конфігурації (зазвичай show running-config або її секцію), отримує текстовий вивід, обробляє його за допомогою регулярних виразів або спеціальних парсерів (наприклад, TextFSM або Genie) і будує внутрішню структуру даних, що відображає поточний стан. Потім відбувається порівняння JSON-об'єктів поточного та бажаного станів. Цей процес вимагає від розробників модулів глибокого розуміння синтаксису конкретної операційної системи. Для складних конфігурацій, таких як списки контролю доступу (ACL),

модуль повинен враховувати порядок правил, наявність коментарів та специфічні формати запису масок підмереж. Саме тому використання готових, перевірених модулів є кращим, ніж написання власних команд через модуль `raw` або `cli_command`, які не мають вбудованої логіки ідемпотентності.

Логіка автоматизації в Ansible описується за допомогою плейбуків (Playbooks) – файлів у форматі YAML, які визначають оркестрацію процесів. Плейбук складається з одного або декількох сценаріїв (Plays). Кожен сценарій (Play) – це впорядкована мапа, яка зв'язує групу хостів з інвентарю з переліком завдань (Tasks), які необхідно на них виконати. Завдання виконуються послідовно, зверху вниз, і Ansible очікує успішного виконання завдання на всіх хостах поточної групи, перш ніж перейти до наступного завдання. Ця синхронність є важливою для складних процедур оновлення, де порядок дій має значення (наприклад, спочатку оновити вторинний фаєрвол, перевірити його стан, переключити трафік, і лише потім оновлювати первинний). Кожне завдання викликає певний модуль і передає йому параметри. Окрім власне виконання дій, завдання можуть реєструвати результати своєї роботи у змінні (використовуючи директиву `register`), що дозволяє створювати умовну логіку: наступне завдання виконається лише тоді, коли попереднє повернуло певний результат або коли у виводі команди знайдено певне ключове слово.

Важливим елементом управління потоком виконання є блоки (Blocks) та обробка помилок. У мережевій автоматизації помилки трапляються часто: тимчасова недоступність пристрою, синтаксична помилка в команді або конфлікт конфігурації. Ansible дозволяє групувати завдання у блоки та застосовувати до них логіку `rescue` та `always`, аналогічну конструкції `try-catch-finally` у мовах програмування. Якщо будь-яке завдання всередині блоку завершується з помилкою, автоматично запускаються завдання з секції `rescue`. Це дозволяє реалізувати механізми автоматичного відкату змін (Rollback). Наприклад, у блоці `block` ми застосовуємо нову політику ACL, а в блоці `rescue` прописуємо команди для відновлення попередньої версії ACL. Це гарантує, що навіть у випадку збою пристрій не залишиться у неробочому або незахищеному стані. Секція `always`

виконується у будь-якому випадку, що корисно для скидання тимчасових налаштувань або закриття сесій моніторингу.

Окремою функціональністю, що підвищує надійність налаштування, є режим перевірки (Check Mode), який активується прапором `--check`. У цьому режимі Ansible виконує весь плейбук, проходить етапи підключення, збору фактів, парсингу конфігурації та порівняння станів, але не надсилає фінальні команди на зміну конфігурації пристрою. Замість цього він звітує про те, які зміни *були б* зроблені (статус `changed`). Це дозволяє адміністратору безпеки провести "сухий прогон" (Dry Run) перед застосуванням критичних змін, переконатися, що плейбук торкнеться лише потрібних пристроїв і внесе саме ті зміни, які очікуються. У поєднанні з режимом відображення різниці (`--diff`), який показує конкретні рядки конфігурації, що будуть додані або видалені, це надає повну прозорість процесу автоматизації та дозволяє пройти процедуру затвердження змін (Change Approval) з документальним підтвердженням планованих дій.

Також архітектура Ansible передбачає використання обробників (Handlers). Це спеціальні завдання, які запускаються лише тоді, коли їх "сповіщає" (notify) інше завдання, і лише в тому випадку, якщо це завдання внесло реальні зміни в систему. У світі серверів обробники часто використовуються для перезапуску сервісів після зміни конфігураційного файлу. У мережевій автоматизації типовим прикладом використання обробника є збереження конфігурації (write memory або `copy running-config startup-config`). Немає сенсу зберігати конфігурацію після кожного дрібного завдання; натомість, кожне завдання, що змінює налаштування (інтерфейс, OSPF, ACL), надсилає сповіщення обробнику "Save Config". Обробник спрацьовує один раз в самому кінці сценарію, забезпечуючи персистентність змін. Це оптимізує час виконання плейбука та зменшує знос флеш-пам'яті мережевих пристроїв.

Ефективна автоматизація мережевої інфраструктури неможлива без чіткого відокремлення логіки виконання завдань від даних, які використовуються в цих завданнях. В архітектурі Ansible це досягається за допомогою розгалуженої системи змінних (Variables) та механізму збору фактів (Facts). Факти – це системні дані, які Ansible автоматично збирає з керованих пристроїв на початку виконання сценарію. Для серверів це інформація про ядро ОС, обсяг пам'яті, IP-адреси

інтерфейсів. У випадку мережевого обладнання, процес збору фактів реалізується через спеціалізовані модулі, такі як `cisco.ios.ios_facts` або `junipernetworks.junos.junos_facts`. Ці модулі підключаються до пристрою, виконують команди типу `show version`, `show interfaces`, `show inventory`, парсять отриманий текстовий вивід і перетворюють його на структурований JSON-об'єкт. Завдяки цьому адміністратор отримує доступ до таких критично важливих параметрів, як версія мікропрограмного забезпечення, серійний номер шасі, список сусідів по протоколу CDP/LLDP та поточний стан інтерфейсів. Використання фактів дозволяє реалізувати адаптивну логіку плейбуків: наприклад, сценарій оновлення прошивки може автоматично перевіряти поточну версію (`ansible_net_version`) і запускати процес оновлення лише тоді, коли вона нижча за необхідну, або пропускати певні команди на моделях комутаторів, які їх не підтримують (`ansible_net_model`).

Управління користувацькими даними здійснюється через систему змінних, які можуть бути визначені на різних рівнях ієрархії: глобальному, рівні групи, хоста або навіть окремого запуску плейбука. Це дозволяє реалізувати принцип DRY (Don't Repeat Yourself). Замість того, щоб жорстко прописувати (`hardcode`) IP-адреси серверів DNS або NTP у кожному завданні, вони виносяться у змінні групи (наприклад, `ntp_servers` у файлі `group_vars/all.yml`). При необхідності зміни адреси NTP-сервера достатньо відредагувати лише один файл, і зміни автоматично застосуються до всієї мережі при наступному запуску. Для зберігання чутливих даних, таких як паролі маршрутизації OSPF, ключі IPsec або облікові дані AD, використовується інструмент Ansible Vault, який шифрує файли змінних алгоритмом AES-256. Це дозволяє безпечно зберігати всю конфігурацію інфраструктури, включаючи секрети, у системах контролю версій (Git), забезпечуючи спільну роботу команди без ризику витоку інформації.

Найпотужнішим інструментом генерації конфігураційних файлів в екосистемі Ansible є шаблонізатор Jinja2 [11]. Це швидкий та гнучкий двигун текстових шаблонів для мови Python, який дозволяє динамічно формувати текстові файли на основі змінних та логічних конструкцій. У мережевій автоматизації Jinja2 відіграє центральну роль, оскільки кінцевим результатом налаштування більшості

пристроїв є саме текстовий конфігураційний файл. Шаблон Jinja2 виглядає як звичайний фрагмент конфігурації (наприклад, налаштування інтерфейсу Cisco), в який вставлені спеціальні маркери. Маркери `{{ variable_name }}` використовуються для підстановки значень змінних, а маркери `{% ... %}` – для керуючих конструкцій, таких як цикли та умови. Це дозволяє, наприклад, взяти список словників, що описують VLAN (ID та назву), і за допомогою циклу `{% for vlan in vlans %}` згенерувати сотні рядків конфігурації `vlan {{ vlan.id }}` та `name {{ vlan.name }}` за кілька мілісекунд.

Використання умовних операторів `{% if %}` ... `{% endif %}` у шаблонах дозволяє створювати універсальні конфігурації для різноманітних пристроїв. Наприклад, в одному шаблоні налаштування інтерфейсу можна передбачити логіку: якщо змінна `interface_type` має значення "L3", то додати команду по switchport і налаштувати IP-адресу, а якщо "L2" – налаштувати switchport mode access. Такий підхід, відомий як "Configuration as Code", дозволяє повністю абстрагуватися від ручного написання конфігурацій. Адміністратор працює лише з моделями даних (YAML-файлами зі змінними), а Ansible разом з Jinja2 автоматично трансформує ці дані у валідні конфігураційні файли для конкретного обладнання. Модуль `template` в Ansible дозволяє не лише згенерувати файл локально, але й порівняти його з поточною конфігурацією пристрою, показуючи різницю (diff) перед застосуванням.

Зі зростанням складності інфраструктури та збільшенням кількості плейбуків виникає необхідність у структуруванні проекту. Для цього в Ansible існує концепція Ролей (Roles). Роль – це одиниця організації та повторного використання коду, яка дозволяє логічно згрупувати пов'язані завдання, змінні, файли, шаблони та обробники у стандартизовану структуру директорій. Типова роль, наприклад, "hardened_switch", міститиме папку `tasks` з описом дій (налаштування SSH, вимкнення Telnet, налаштування AAA), папку `templates` з шаблонами банерів та ACL, папку `vars` зі стандартними змінними та папку `handlers` для збереження конфігурації. Використання ролей дозволяє перетворити монолітні, складні для читання плейбуки на компактні сценарії, які просто викликають необхідні ролі: "застосувати роль firewall до групи firewall, застосувати роль switch до групи

switches". Це сприяє модульності: роль, написану для налаштування базової безпеки Cisco IOS, можна легко перенести в інший проект або поділитися нею з іншими командами через репозиторій Ansible Galaxy.

Крім того, використання ролей та структурованих даних дозволяє реалізувати підхід "Infrastructure as Data". У цій парадигмі джерелом істини про мережу стає не набір розрізнених скриптів, а структурована модель даних (Data Model), яка описує бажаний стан всієї мережі. Ansible, використовуючи механізми `include_role` та цикли по групах інвентарю, проходить по цій моделі і приводить реальне обладнання у відповідність до неї. Це значно спрощує масштабування: для додавання нової філії достатньо просто скопіювати файл змін існуючої філії, змінити IP-адреси та назву, і запустити той самий плейбук. Ansible автоматично згенерує та застосує конфігурацію, гарантуючи, що вона повністю відповідає корпоративним стандартам безпеки, закладеним у ролях та шаблонах. Таким чином, поєднання декларативних модулів, гнучкого інвентарю, потужної шаблонізації Jinja2 та модульної структури ролей робить Ansible комплексним інструментом, здатним вирішувати найскладніші задачі автоматизації конфігурації безпеки у гетерогенних мережах будь-якого масштабу.

2.4. Вимоги до системи автоматизованого налаштування конфігурацій безпеки

Розробка ефективної системи автоматизованого управління конфігураціями безпеки мережевого обладнання вимагає чіткого визначення функціональних та нефункціональних вимог, які базуються на аналізі існуючих загроз, нормативних стандартів та архітектурних особливостей сучасних мереж. Першочерговою функціональною вимогою до системи є забезпечення повноти покриття життєвого циклу політик безпеки, починаючи від початкового налаштування пристрою "з коробки" (Zero Touch Provisioning) і закінчуючи його виведенням з експлуатації. Система повинна мати можливість централізовано керувати критичними параметрами безпеки, такими як налаштування автентифікації, авторизації та обліку (AAA), управління списками контролю доступу (ACL), конфігурація захищених протоколів управління (SSH v2, HTTPS) та відключення небезпечних сервісів (Telnet, HTTP, CDP). Важливою умовою є здатність системи працювати в

гетерогенному середовищі, абстрагуючи адміністратора від синтаксичних відмінностей між різними вендорами обладнання. Тобто, вимога полягає в тому, щоб система приймала уніфіковану команду "закрити порт 23" і автоматично транслювала її в специфічні команди для Cisco IOS, Juniper JunOS або HP Comware без ручного втручання оператора.

Критично важливою нефункціональною вимогою, яка визначає архітектуру рішення, є забезпечення ідемпотентності виконання операцій. Система повинна гарантувати, що повторний запуск сценарію налаштування на вже налаштованому пристрої не призведе до дублювання команд, порушення роботи сервісів або виникнення помилок. Це вимагає реалізації механізмів попередньої перевірки стану (State Checking): перед внесенням будь-якої зміни система зобов'язана зчитати поточну конфігурацію, проаналізувати її та обчислити різницю (delta) між поточним та бажаним станом. Тільки у випадку наявності розбіжностей система має право генерувати керуючі впливи. Така вимога є запобіжником від деградації мережі внаслідок людських помилок при повторних запусках скриптів та дозволяє використовувати систему в режимі постійного моніторингу відповідності (Continuous Compliance), коли сценарії запускаються за розкладом для автоматичного виправлення несанкціонованих змін.

З точки зору інформаційної безпеки, сама система автоматизації стає критично важливим активом, компрометація якого може призвести до втрати контролю над усією мережею, тому до неї висувуються підвищені вимоги захисту. Система повинна забезпечувати суворе розмежування доступу до керуючих сценаріїв та інвентарю на основі ролей (Role-Based Access Control). Особливо жорсткі вимоги висувуються до підсистеми управління секретами: паролі доступу до пристроїв, ключі шифрування VPN, спільноти SNMP та токени API категорично заборонено зберігати у відкритому вигляді в текстових файлах або коді скриптів. Система повинна інтегруватися з захищеними сховищами секретів або використовувати надійні механізми шифрування даних "у спокої" (Encryption at Rest), такі як Ansible Vault, з використанням стійких криптографічних алгоритмів (наприклад, AES-256). Також обов'язковою є вимога до захисту транспортного каналу: вся взаємодія між керуючим вузлом та мережевими пристроями повинна відбуватися виключно через

шифровані протоколи (SSH, HTTPS/TLS), а використання застарілих протоколів (Telnet, TFTP) має бути технічно унеможливлене на рівні конфігурації плагінів з'єднання.

Вимоги до надійності та відмовостійкості системи передбачають наявність механізмів обробки помилок та автоматичного відкату змін (Rollback). Враховуючи ризики втрати зв'язку з віддаленими пристроями під час переконфігурації (наприклад, при зміні налаштувань routing або інтерфейсів), система повинна реалізовувати стратегію "безпечної зміни". Це означає, що перед застосуванням нової конфігурації система зобов'язана автоматично створювати резервну копію (Backup) поточної робочої конфігурації. У разі виникнення помилки під час виконання сценарію або втрати зв'язку з пристроєм, система повинна мати інструментарій для швидкого, в ідеалі автоматизованого, відновлення попереднього стабільного стану. Крім того, система не повинна зупиняти свою роботу повністю у разі недоступності одного з хостів в інвентарі; вона повинна коректно залогувати помилку, виключити проблемний вузол з поточного виконання та продовжити роботу з іншими пристроями, забезпечуючи завершення процесу для справної частини інфраструктури.

Масштабованість та продуктивність є ключовими вимогами для використання системи у великих корпоративних мережах. Час виконання сценаріїв налаштування безпеки повинен вкладатися у відведені регламентні вікна обслуговування (Maintenance Windows). Це вимагає від архітектури системи підтримки паралельного виконання завдань (Parallel Execution) на великій кількості вузлів одночасно. Система повинна дозволяти адміністратору гнучко налаштовувати рівень паралелізму (кількість потоків або форків), балансуючи між швидкістю виконання та навантаженням на керуючий сервер і мережеві канали. Також важливим є оптимізація взаємодії з пристроями: система повинна мінімізувати кількість сесій підключення, використовуючи механізми постійних з'єднань (Persistent Connections), щоб уникнути накладних витрат на багаторазове встановлення захищених тунелів SSH для кожного окремого завдання.

Окремий блок вимог стосується прозорості роботи, аудиту та звітності. Система автоматизації повинна працювати за принципом "повної видимості".

Кожен запуск сценарію повинен супроводжуватися детальним логуванням, яке містить інформацію про час запуску, ініціатора, перелік цільових пристроїв, успішність виконання кожного завдання та, що найважливіше, детальний звіт про внесені зміни (Diff report). Адміністратор повинен мати можливість бачити не просто статус "Changed", а конкретні рядки конфігурації, які були додані, змінені або видалені. Це необхідно для проведення форензики, розслідування інцидентів та підтвердження відповідності вимогам регуляторів (Compliance Audit). Також система повинна підтримувати режим симуляції ("Dry Run" або "Check Mode"), який дозволяє змодельовати виконання сценарію і побачити прогнозовані зміни без реального впливу на обладнання, що є обов'язковою процедурою перед внесенням змін у критичні сегменти мережі.

Нарешті, система повинна відповідати вимогам модульності та розширюваності. Оскільки ландшафт кіберзагроз постійно змінюється, і з'являються нові вектори атак, система не повинна бути монолітною. Вона повинна будуватися на основі багаторазово використовуваних компонентів (ролей та шаблонів), що дозволяє швидко адаптувати політики безпеки. Наприклад, при появі нової вразливості в протоколі NTP, адміністратор повинен мати можливість оновити лише одну роль "NTP_Security" і застосувати її до всієї інфраструктури, не переписуючи основні сценарії. Також система повинна підтримувати можливість інтеграції з зовнішніми системами, такими як IPAM (для отримання актуальних IP-адрес), CMDB (для отримання бізнес-контексту пристроїв) та системами моніторингу (для тимчасового відключення сповіщень під час робіт). Дотримання комплексу цих вимог дозволить створити надійний, безпечний та ефективний інструмент, здатний перетворити хаотичне ручне управління на впорядкований автоматизований процес.

2.5. Розробка методики автоматизованого конфігурування мережевого обладнання

Розробка методики автоматизованого налаштування починається з проектування та розгортання програмного середовища керуючого вузла (Control Node), який виступає центральною ланкою в архітектурі системи безпеки [27]. Оскільки Ansible є без-агентною системою, вимоги до апаратної частини

керуючого вузла є мінімальними, проте вимоги до програмного оточення є критичними для стабільності роботи. Як базову операційну систему для розгортання методики обрано дистрибутив Linux (наприклад, Ubuntu Server або CentOS Stream/Rocky Linux), оскільки він забезпечує нативну підтримку протоколу SSH, Python та утиліт командного рядка, необхідних для роботи. Першим етапом методики є ізоляція середовища виконання. Для уникнення конфліктів між системними бібліотеками Python та бібліотеками, необхідними для Ansible, обов'язковим є використання віртуальних середовищ (Python Virtual Environments). Це дозволяє створити відокремлений простір, де встановлюються конкретні версії `ansible-core` та необхідних залежностей, таких як `paramiko` (для SSH), `netmiko` (для спрощеної взаємодії з CLI) та `jmespath` (для фільтрації JSON-даних). Такий підхід гарантує відтворюваність середовища: проект автоматизації може бути легко перенесений на інший сервер або ноутбук інженера без ризику виникнення помилок сумісності версій.

Наступним кроком є налаштування глобальних параметрів поведінки системи через конфігураційний файл `ansible.cfg`. Цей файл визначає, як саме Ansible взаємодіє з мережею, і для мережевого обладнання його налаштування суттєво відрізняються від серверних стандартів. Критично важливим параметром є налаштування тайм-аутів. Мережеві пристрої, особливо при генерації криптографічних ключів або обробці великих списків ACL, можуть відповідати із значною затримкою. Стандартний тайм-аут у 10 секунд часто призводить до розриву з'єднання посеред виконання завдання, тому методика передбачає збільшення параметру `persistent_command_timeout` до 30-60 секунд. Також необхідно налаштувати параметри логування (`log_path`), щоб всі дії системи записувалися у захищений файл для подальшого аудиту. Важливим аспектом є налаштування перевірки ключів хостів (`host_key_checking`). У продуктивному середовищі цей параметр має бути увімкнений для запобігання атакам MITM, проте методика повинна передбачати механізм автоматичного попереднього наповнення файлу `known_hosts` відбитками ключів мережевих пристроїв, щоб перший запуск автоматизації не зупинився через запити підтвердження довіри.

Фундаментом методики є проектування системи інвентаризації (Inventory Design). Для забезпечення масштабованості та зручності управління безпекою, плоский список IP-адрес є неприйнятним. Методика передбачає створення ієрархічної структури груп, що базується на географічному та функціональному принципах. На верхньому рівні ієрархії пристрої розділяються за локаціями (наприклад, Site_Kyiv, Site_Lviv), що дозволяє застосовувати специфічні для локації налаштування, такі як NTP-сервери, часові пояси та сервери логування. Всередині кожної локації створюються функціональні підгрупи: Firewalls, Core_Routers, Access_Switches. Це дозволяє гранулярно застосовувати ролі безпеки: роль, що налаштовує BGP-фільтри, буде застосована лише до групи маршрутизаторів ядра, а роль для налаштування Port Security – лише до комутаторів доступу. Таке логічне розділення реалізується через файл інвентарю у форматі YAML, який підтримує вкладеність груп (children).

Особливу увагу в методиці приділено організації змінних (Variables Management), оскільки саме вони визначають унікальну конфігурацію кожного пристрою. Методика забороняє зберігання змінних безпосередньо у файлі інвентарю. Замість цього використовується стандартизована структура директорій group_vars та host_vars. У директорії group_vars створюються файли, що відповідають назвам груп (наприклад, all.yml, cisco.yml, switches.yml). У файлі all.yml визначаються глобальні політики безпеки, спільні для всієї компанії: адреси серверів AAA (Radius/Tacacs), доменне ім'я, адреси DNS-серверів, банери попередження про відповідальність. У файлах специфічних груп, таких як cisco.yml, визначаються параметри підключення: тип ОС (ansible_network_os: cisco.ios), метод з'єднання (ansible_connection: network_cli) та облікові дані. Змінні, що є унікальними для кожного пристрою (IP-адреса управління, ім'я хоста), виносяться у директорію host_vars. Такий каскадний підхід (Variable Inheritance) дозволяє мінімізувати дублювання даних та спростити внесення масових змін: зміна адреси Radius-сервера в одному файлі автоматично розповсюджується на сотні пристроїв.

Критичним елементом методики є організація безпечного каналу управління. Використання паролів для SSH-доступу до пристроїв розглядається як застаріла та

небезпечна практика. Методика передбачає перехід на автентифікацію за ключами SSH (Public Key Authentication) там, де це підтримується обладнанням. Для цього на керуючому вузлі генерується пара ключів (приватний та публічний). Публічний ключ розповсюджується на всі мережеві пристрої. При виконанні плейбуків Ansible використовує локальний SSH-агент (ssh-agent), що дозволяє уникнути передачі паролів у відкритому вигляді. Для пристроїв, які не підтримують авторизацію за ключами, або для введення пароля на підвищення привілеїв (enable secret), використовується сховище Ansible Vault. Методика вимагає створення окремого зашифрованого файлу secrets.yml всередині group_vars, де зберігаються всі чутливі дані. Доступ до цього файлу захищається майстер-паролем, який вводиться адміністратором при запуску або зчитується з захищеного файлу на диску, доступного лише користувачеві root.

Також методика включає підготовку системи до роботи з різними версіями програмного забезпечення мережевих пристроїв. Оскільки в реальній мережі можуть співіснувати різні версії IOS або JunOS, які мають відмінності в синтаксисі команд, методика передбачає використання умовних конструкцій у змінних або коді. Наприклад, змінні, що визначають список команд для "hardening" (посилення безпеки), можуть бути розділені на списки security_commands_ios_15 та security_commands_ios_xe. Під час виконання плейбука система автоматично визначає версію ОС через збір фактів і підставляє відповідний набір команд. Це забезпечує гнучкість та універсальність розробленого рішення, дозволяючи йому коректно працювати в гетерогенному середовищі без необхідності писати окремі сценарії для кожної моделі комутатора.

Організація файлової структури проекту автоматизації є не менш важливою складовою методики, ніж сам код сценаріїв, оскільки хаотичне розміщення файлів ускладнює підтримку системи та масштабування. Методика передбачає використання стандартизованої структури каталогів, рекомендованої спільнотою розробників (Best Practices), яка базується на модульній декомпозиції [13]. У корені проекту розміщується головний плейбук site.yml, який виступає точкою входу та оркестратором високого рівня, викликаючи необхідні ролі для різних груп хостів. Основна логіка виноситься у директорію roles/, де для кожної функціональної

задачі створюється окрема піддиректорія. Методика пропонує виділення наступних базових ролей безпеки: `common_core` (базові налаштування, користувачі, NTP, логування), `hardening` (відключення небезпечних сервісів, налаштування SSH, банерів), `access_control` (генерація та застосування списків ACL), `vpn_security` (налаштування параметрів IPsec та тунелів). Такий підхід дозволяє дотримуватися принципу єдиної відповідальності: якщо необхідно змінити політику паролів, інженер редагує лише роль `common_core`, не чіпаючи налаштування маршрутизації. Окрім ролей, структура проекту включає директорію `inventories/` для файлів інвентаризації (розділених на `production` та `staging`) та `backup/`, куди автоматично зберігаються резервні копії конфігурацій перед кожною зміною.

Ключовим елементом методики є стратегія шаблонізації конфігурацій з використанням рушія Jinja2. Замість написання статичних команд, методика вимагає створення параметризованих шаблонів (Templates), які зберігаються у піддиректоріях `templates/` відповідних ролей. Шаблон являє собою "золотий стандарт" (Golden Configuration) фрагмента налаштувань. Наприклад, шаблон для налаштування SNMP не містить конкретних IP-адрес серверів моніторингу або рядків спільнот (community strings); замість них використовуються змінні `{{ snmp_servers }}` та `{{ snmp_ro_community }}`. Це дозволяє команді безпеки провести аудит самого шаблону один раз і гарантувати, що всі пристрої, налаштовані за цим шаблоном, будуть відповідати затвердженому стандарту. Методика також передбачає використання умовних операторів у шаблонах для адаптації під різні моделі обладнання. Наприклад, старіші моделі комутаторів можуть не підтримувати певні команди шифрування; у такому випадку в шаблоні використовується конструкція `{% if ansible_net_model != 'legacy' %}`, що дозволяє ігнорувати несумісні команди без виникнення помилок виконання.

Найбільш критичною частиною методики є розробка алгоритму безпечного розгортання змін (Deployment Workflow), який мінімізує ризики порушення доступності сервісів. Алгоритм складається з чотирьох послідовних етапів: резервне копіювання, перевірка змін, застосування та верифікація. Перший етап (Pre-task Backup) є обов'язковим: перед виконанням будь-яких команд, що

змінюють стан пристрою, система автоматично зчитує поточну конфігурацію (`show running-config`) та зберігає її у файл з відміткою часу. Це створює точку відновлення. Для реалізації цього етапу використовуються модулі збору фактів або спеціалізовані модулі резервного копіювання (наприклад, `cisco.ios.ios_config` з параметром `backup: yes`). Наявність свіжого бекапу є страховкою на випадок катастрофічних помилок, дозволяючи швидко повернути пристрій у вихідний стан.

Другий етап – це режим "сухого прогону" (Dry Run). Методика забороняє застосування змін "наосліп". Сценарії запускаються з параметрами `--check` та `--diff`. У цьому режимі Ansible симулює виконання завдань, порівнює згенеровані шаблони з поточною конфігурацією на пристрої та виводить у консоль звіт про різницю. Інженер повинен проаналізувати цей звіт і переконатися, що система збирається змінити саме те, що планувалося (наприклад, додати один рядок в ACL), і не зачіпає сусідні налаштування. Цей етап також слугує фінальною перевіркою синтаксису шаблонів та коректності підстановки змінних. Тільки після підтвердження коректності звіту `diff` дозволяється перехід до наступного етапу – безпосереднього застосування змін (Apply). На цьому етапі знімається прапорець `--check`, і Ansible надсилає реальні команди на пристрої, використовуючи механізми ідемпотентності для уникнення зайвих дій.

Завершальним, але часто ігнорованим етапом, є верифікація результату (Post-task Verification). Успішне виконання команди (відсутність помилок CLI) ще не означає коректну роботу сервісу. Тому методика передбачає включення у плейбуки завдань перевірки функціональності. Це можуть бути команди діагностики, такі як перевірка доступності сусідів по протоколу OSPF/BGP, перевірка статусу інтерфейсів або виконання тестових пінгів до контрольних точок мережі. Для цього використовуються модулі `assert` або `failed_when`, які аналізують вивід команд діагностики і зупиняють виконання плейбука з помилкою, якщо критерії успішності не досягнуті. Наприклад, якщо після зміни налаштувань порту кількість активних OSPF-сусідів стала меншою, ніж очікувалось, система сигналізує про проблему.

Окремим сценарієм, передбаченим методикою, є автоматизований відкат змін (Rollback). У випадку, якщо етап верифікації виявив критичні проблеми

(наприклад, втрату зв'язку з сегментом мережі), система повинна мати можливість автоматично відновити попередню конфігурацію. Для пристроїв, що підтримують транзакційну модель або заміну конфігурації (як `configure replace` у Cisco або `commit confirmed` у Juniper), методика використовує нативні механізми відкату. Для пристроїв без такої підтримки реалізується логіка відновлення з файлу резервної копії, створеного на першому етапі. Використання блоків `block-rescue` в Ansible дозволяє запрограмувати цю поведінку: якщо завдання в блоці `block` (застосування змін) або наступне завдання верифікації завершуються з помилкою, автоматично запускаються завдання з блоку `rescue`, які завантажують бекап-конфігурацію на пристрій, мінімізуючи час простою мережі до кількох хвилин.

Таким чином, розроблена методика є комплексною системою, яка охоплює не лише технічні аспекти написання коду, але й організаційні процеси управління змінами. Вона трансформує ручне адміністрування у структурований інженерний процес, де кожна зміна є задокументованою, перевіреною та оборотною. Структурований підхід до організації файлів проекту забезпечує зручність спільної роботи команди, а суворий алгоритм розгортання з обов'язковими етапами перевірки та резервного копіювання гарантує надійність та стабільність функціонування критично важливої інфраструктури безпеки підприємства.

2.6. Модель взаємодії компонентів системи Ansible (Inventory, Playbooks, Roles, Templates)

Побудова ефективної системи автоматизації вимагає розгляду Ansible не як набору розрізнених утиліт, а як цілісної, інтегрованої моделі, де кожен компонент виконує чітко визначену функцію і знаходиться у безперервній взаємодії з іншими елементами архітектури. Модель взаємодії компонентів Ansible базується на принципі розділення даних, логіки та представлення, що дозволяє досягти високої гнучкості та масштабованості. Центральним елементом цієї моделі, який ініціює та координує всі процеси, є плейбук (Playbook). Він виступає в ролі оркестратора, що зв'язує "кого" ми налаштовуємо (Inventory), "що" ми налаштовуємо (Roles/Tasks) та "з якими параметрами" (Variables). Процес виконання починається з того, що плейбук звертається до системи інвентаризації для отримання списку цільових хостів. На цьому етапі відбувається первинна агрегація даних: система зчитує

статичну структуру груп з файлу інвентарю, динамічно підтягує хости з зовнішніх джерел (якщо налаштовано) та формує фінальний список об'єктів для виконання. Критично важливою частиною цієї фази є асоціація змінних: Ansible сканує директорії `group_vars` та `host_vars`, накладає їх на ієрархію груп та обчислює пріоритетність значень. У результаті для кожного хоста формується унікальний контекст даних (Host Context), що містить всі параметри підключення, секрети та специфічні налаштування, які будуть доступні на наступних етапах.

Після формування контексту даних модель взаємодії переходить до фази виконання логіки, яка інкапсульована в Ролях (Roles). Плейбук делегує виконання конкретних завдань ролям, передаючи їм управління. Роль, у свою чергу, є абстрактною сутністю, яка не прив'язана до конкретного хоста. Вона містить набір інструкцій (Tasks), які описують алгоритм налаштування певного сервісу, наприклад, налаштування NTP або безпеки порту. Взаємодія між плейбуком та роллю відбувається через механізм включення: роль отримує доступ до глобальних змінних плейбука та фактів, зібраних з пристрою, але також може визначати свої власні змінні за замовчуванням (Defaults). Це дозволяє створювати універсальні ролі, які працюють "з коробки" з безпечними дефолтними значеннями, але можуть бути легко кастомізовані шляхом перекриття змінних на рівні інвентарю. Важливим аспектом взаємодії є те, що роль структурує виконання, автоматично підтягуючи необхідні файли завдань (`main.yml`), обробники подій (Handlers) та залежності від інших ролей, створюючи впорядкований потік виконання операцій.

Ключовим етапом, де дані перетворюються на реальні конфігурації, є взаємодія між завданнями ролі та системою шаблонізації (Templates). Коли завдання викликає модуль `template`, відбувається звернення до рушія Jinja2. Цей процес є точкою синтезу: абстрактна модель даних (змінні, отримані на етапі інвентаризації, та факти, зібрані з пристрою) накладається на структурний каркас (текстовий шаблон Jinja2). Рушій обробляє умовні конструкції, цикли та фільтри, генеруючи фінальний конфігураційний файл, специфічний для конкретного пристрою. Наприклад, шаблон інтерфейсу взаємодіє зі змінною-списком інтерфейсів хоста, ітеративно створюючи блоки конфігурації. Результатом цієї взаємодії є готовий набір команд або повний файл конфігурації, який передається

модулю з'єднання (`network_cli` або іншому) для доставки на мережевий пристрій. Важливо зазначити, що ця генерація відбувається локально на керуючому вузлі, що дозволяє перевірити коректність синтаксису ще до моменту відправки даних у мережу.

Завершальним елементом моделі взаємодії є зворотний зв'язок (Feedback Loop) та обробка результатів. Після того, як модуль Ansible виконав команди на мережевому пристрої, він повертає статус виконання (Success, Failed, Changed) та структуровані дані про результат (наприклад, стандартний вивід команди або повідомлення про помилку). Ці дані повертаються у плейбук, де вони можуть бути зареєстровані у нових змінних та використані для прийняття рішень у наступних завданнях (Conditional Execution). Наприклад, якщо завдання налаштування інтерфейсу повернуло помилку, плейбук може активувати блок `rescue` для відкату змін. Також на цьому етапі відбувається взаємодія з підсистемою обробників (Handlers): якщо завдання повідомило про внесення змін (Status: Changed), воно надсилає сигнал відповідному обробнику (наприклад, "зберегти конфігурацію"), який буде виконаний в кінці сценарію. Така циклічна модель взаємодії забезпечує динамічність та адаптивність процесу автоматизації, дозволяючи системі реагувати на непередбачувані ситуації та гарантувати цілісність кінцевого стану інфраструктури.

Узагальнюючи модель, можна представити роботу системи Ansible як конвеєр обробки даних. На вході конвеєра знаходяться розрізнені джерела інформації: статичний інвентар, змінні груп, секрети Vault та факти про пристрій. Плейбук діє як контролер конвеєра, спрямовуючи цей потік даних через логічні блоки Ролей. В середині ролей відбувається збагачення та трансформація даних, а механізм Шаблонів (Templates) виступає в ролі пресу, що формує з сировини (даних) готовий продукт (конфігурацію). Модулі та плагіни з'єднання забезпечують транспортну функцію, доставляючи продукт до споживача (мережевого пристрою) та повертаючи звіт про доставку. Ефективність цієї моделі полягає в її модульній природі: заміна одного компонента (наприклад, зміна значення змінної в інвентарі або модифікація шаблону) автоматично змінює результат роботи всієї системи без необхідності переписування логіки виконання. Це робить архітектуру Ansible

ідеальним фундаментом для побудови масштабованих та надійних систем управління безпекою, здатних еволюціонувати разом зі зростанням мережі.

Висновки до другого розділу

У другому розділі проведено глибокий аналіз технологій автоматизації та розроблено методику налаштування конфігурації безпеки мережевого обладнання. У ході дослідження було розглянуто провідні інструменти класу "Infrastructure as Code" – Ansible, Puppet, Chef та SaltStack. Порівняльний аналіз показав, що для специфічних задач мережевої безпеки, де критичними факторами є мінімізація поверхні атаки, робота з закритими пропрієтарними ОС та оперативність реагування, найбільш оптимальним вибором є Ansible. Його безагентна архітектура, використання стандартного протоколу SSH та декларативна мова опису YAML дозволяють уникнути вразливостей, притаманних агентним системам, та забезпечити низький поріг входження для персоналу.

Детально проаналізовано архітектуру Ansible, включаючи механізми інвентаризації, систему модулів та специфіку роботи транспортного рівня (`network_cli`) з мережевим обладнанням. Визначено, що ключовою перевагою є забезпечення ідемпотентності операцій, що гарантує передбачуваність змін та відповідність цільовому стану. Сформульовано чіткі функціональні та нефункціональні вимоги до системи автоматизації, серед яких пріоритетними є безпечне зберігання секретів, контроль цілісності конфігурацій та можливість автоматичного відкату змін.

На основі теоретичного аналізу розроблено практичну методику автоматизованого конфігурування. Вона включає в себе рекомендації щодо організації середовища керуючого вузла, проектування ієрархічної структури інвентарю та змінних, а також створення стандартизованої структури проекту з використанням ролей. Особливу увагу приділено алгоритму безпечного розгортання ("Backup – Check – Apply – Verify"), який мінімізує ризики простою мережі. Запропонована модель взаємодії компонентів системи (Inventory, Playbooks, Roles, Templates) створює надійний фундамент для практичної реалізації системи захисту.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ МЕТОДИКИ АВТОМАТИЗОВАНОГО КОНФІГУРУВАННЯ МЕРЕЖЕВОЇ БЕЗПЕКИ

3.1. Розробка та реалізація playbooks для налаштування безпеки мережевого обладнання

Практична реалізація розробленої методики виконана на базі лабораторного стенду, розгорнутого у середовищі емуляції мереж GNS3. Архітектура стенду відтворює сегмент корпоративної мережі та включає керуючий вузол (Control Node) під управлінням операційної системи корпоративного класу Rocky Linux 9. Вибір даної ОС обумовлений її стабільністю, бінарною сумісністю з RHEL та широким використанням у промислових середовищах. На керуючому вузлі розгорнуто середовище Python 3.9 та встановлено систему автоматизації Ansible (версія core 2.14). Об'єктами керування виступають два маршрутизатори Cisco (моделі c7200), що виконують ролі прикордонного шлюзу та маршрутизатора ядра.

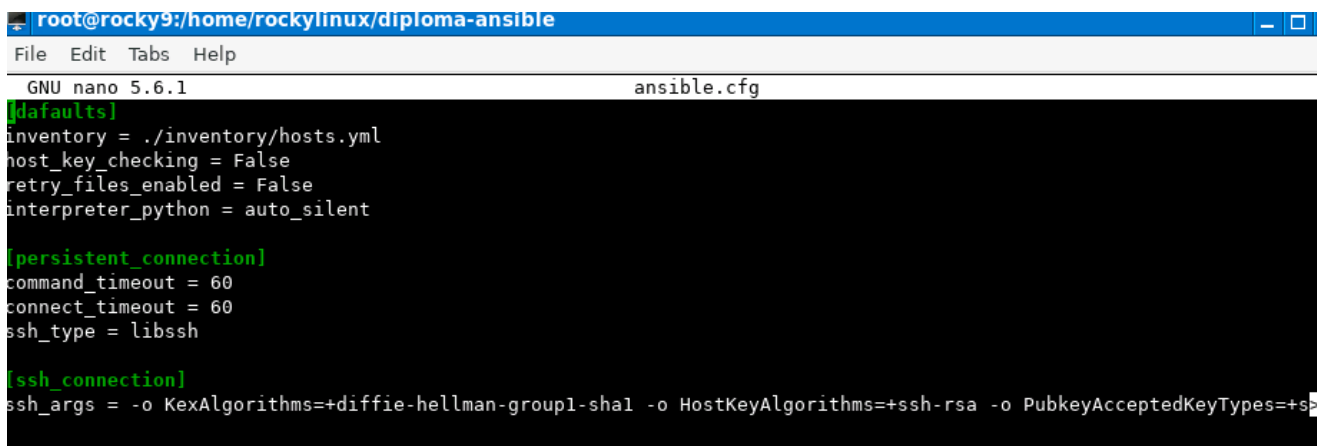
На початковому етапі реалізації було створено стандартизовану структуру проекту відповідно до принципів «Інфраструктура як код». Налаштування середовища в Rocky Linux включало створення робочої директорії та конфігураційних файлів. Файл `ansible.cfg` було оптимізовано для роботи з застарілим мережевим обладнанням, зокрема налаштовано параметри тайм-аутів для запобігання розриву SSH-сесій під час виконання тривалих криптографічних операцій.

```
[rockylinux@rocky9 ~]$ mkdir -p diploma-ansible/{inventory/group_vars,roles,playbooks}
[rockylinux@rocky9 ~]$ cd diploma-ansible
[rockylinux@rocky9 diploma-ansible]$ touch ansible.cfg inventory/hosts.yml playbooks/site.yml
[rockylinux@rocky9 diploma-ansible]$ vi ansible.cfg
```

Рисунок 3.1 – Створена структура проекту

Налаштування поведінки середовища Ansible здійснювалося через файл конфігурації `ansible.cfg`. Враховуючи специфіку роботи з застарілим мережевим обладнанням та особливості протоколу SSH на пристроях Cisco, було внесено ряд оптимізацій. Зокрема, параметр `host_key_checking = False` було встановлено для уникнення ручного підтвердження відбитків ключів при першому підключенні в динамічному лабораторному середовищі (у продуктивному середовищі цей параметр рекомендовано вмикати). Параметри `command_timeout` та `connect_timeout` було збільшено до 60 секунд, оскільки генерація криптографічних ключів RSA на

емульованому обладнанні може займати значний час, що призводить до розриву стандартної 10-секундної сесії.



```
root@rocky9:/home/rockylinux/diploma-ansible
File Edit Tabs Help
GNU nano 5.6.1 ansible.cfg
[defaults]
inventory = ./inventory/hosts.yml
host_key_checking = False
retry_files_enabled = False
interpreter_python = auto_silent

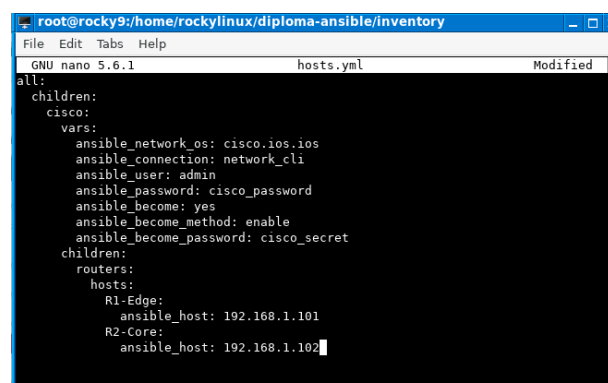
[persistent_connection]
command_timeout = 60
connect_timeout = 60
ssh_type = libssh

[ssh_connection]
ssh_args = -o KexAlgorithms=+diffie-hellman-group1-sha1 -o HostKeyAlgorithms=+ssh-rsa -o PubkeyAcceptedKeyTypes=+ssh-rsa
```

Рисунок 3.2 – Конфігураційний файл ansible.cfg

Наступним кроком стала розробка статичного файлу інвентаризації inventory/hosts.yml. Згідно з методикою, було застосовано ієрархічний підхід до групування хостів. Всі маршрутизатори об'єднані в батьківську групу cisco. Це дозволило визначити змінні підключення (connection variables) на рівні групи, уникнувши їх дублювання для кожного окремого хоста.

Особливу увагу приділено змінним, що визначають метод взаємодії. Змінна ansible_connection: network_cli вказує системі, що взаємодія відбуватиметься не через завантаження Python-модулів на кінцевий пристрій (що неможливо для Cisco IOS), а через емуляцію CLI-сесії по протоколу SSH. Також налаштовано параметри підвищення привілеїв (become), оскільки більшість команд налаштування безпеки вимагають режиму privileged EXEC (рівень 15).

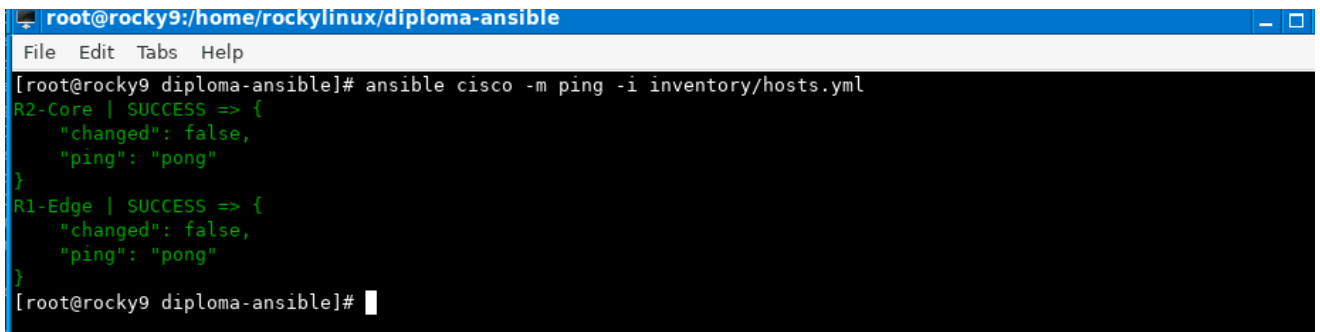


```
root@rocky9:/home/rockylinux/diploma-ansible/inventory
File Edit Tabs Help
GNU nano 5.6.1 hosts.yml Modified
all:
  children:
    cisco:
      vars:
        ansible_network_os: cisco.ios.ios
        ansible_connection: network_cli
        ansible_user: admin
        ansible_password: cisco_password
        ansible_become: yes
        ansible_become_method: enable
        ansible_become_password: cisco_secret
      children:
        routers:
          hosts:
            R1-Edge:
              ansible_host: 192.168.1.101
            R2-Core:
              ansible_host: 192.168.1.102
```

Рисунок 3.3 – Конфігураційний файл hosts.yml

Для верифікації доступності мережевої інфраструктури та коректності налаштувань бібліотек paramiko та netmiko на стороні Rocky Linux, було проведено процедуру тестування з'єднання. Використання модуля ping у контексті мережеских

пристроїв відрізняється від стандартного ICMP-запиту. Ansible намагається встановити SSH-сесію, пройти автентифікацію та виконати базову команду, щоб переконатися у працездатності каналу управління.



```
root@rocky9:/home/rockylinux/diploma-ansible
File Edit Tabs Help
[root@rocky9 diploma-ansible]# ansible cisco -m ping -i inventory/hosts.yml
R2-Core | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
R1-Edge | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
[root@rocky9 diploma-ansible]#
```

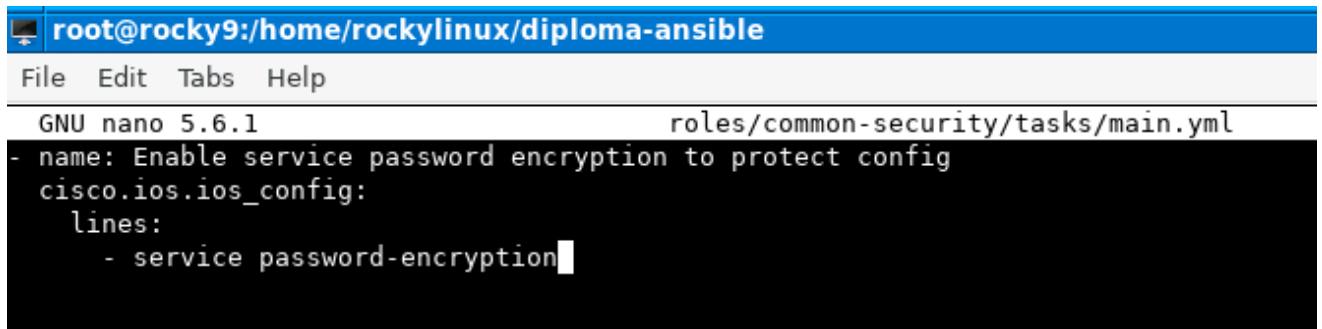
Рисунок 3.3 – Перевірка працездатності каналу управління

Успішне проходження цього тесту є контрольною точкою, яка підтверджує готовність транспортного рівня до передачі складних конфігураційних пейлоадів.

3.2. Автоматизація налаштувань контролю доступу, журналювання, автентифікації та моніторингу

Реалізація комплексної системи захисту мережевої інфраструктури вимагає переходу від фрагментарних налаштувань до системного підходу, що охоплює всі рівні доступу та моніторингу. У рамках практичної частини роботи було розроблено та впроваджено модульну архітектуру ролей Ansible, яка дозволяє автоматизувати критичні аспекти безпеки: автентифікацію користувачів, криптографічний захист каналів управління, фільтрацію трафіку та аудит подій. Використання механізму ролей (Ansible Roles) дозволило реалізувати принцип «розділення відповідальності» (Separation of Concerns). Кожна роль відповідає за окремий домен безпеки, що спрощує підтримку коду та дозволяє гнучко комбінувати політики для різних типів обладнання (Core, Distribution, Access).

Першим ешеленом захисту будь-якої інформаційної системи є суворий контроль доступу. Роль common-security була розроблена для стандартизації налаштувань AAA (Authentication, Authorization, and Accounting). В ручному режимі адміністратори часто припускаються помилки, використовуючи слабкі алгоритми хешування паролів або залишаючи їх у відкритому вигляді. Для вирішення цієї проблеми у файл завдань roles/common-security/tasks/main.yml було включено директиву активації сервісу шифрування паролів.

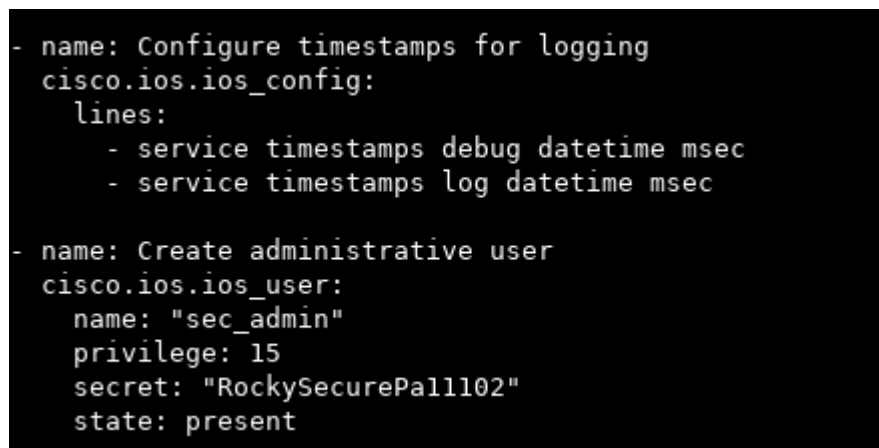


```
root@rocky9:/home/rockylinux/diploma-ansible
File Edit Tabs Help
GNU nano 5.6.1 roles/common-security/tasks/main.yml
- name: Enable service password encryption to protect config
  cisco.ios.ios_config:
    lines:
      - service password-encryption
```

Рисунок 3.4 – Активація алгоритму Cisco Type 7

Ця команда активує алгоритм Cisco Type 7 для обфускації паролів у конфігураційному файлі. Хоча цей алгоритм є слабким з криптографічної точки зору, він є необхідним мінімумом для захисту від підглядання (shoulder surfing) та випадкового розкриття паролів при перегляді конфігурації.

Наступним критичним кроком є створення адміністративних облікових записів. Методика передбачає відмову від використання застарілого ключового слова password на користь secret, що змушує систему використовувати стійкий алгоритм хешування MD5 (Type 5) або SHA-256 (Type 8/9), залежно від версії IOS.



```
- name: Configure timestamps for logging
  cisco.ios.ios_config:
    lines:
      - service timestamps debug datetime msec
      - service timestamps log datetime msec

- name: Create administrative user
  cisco.ios.ios_user:
    name: "sec_admin"
    privilege: 15
    secret: "RockySecurePa11102"
    state: present
```

Рисунок 3.4 – Створення адміністративного запису

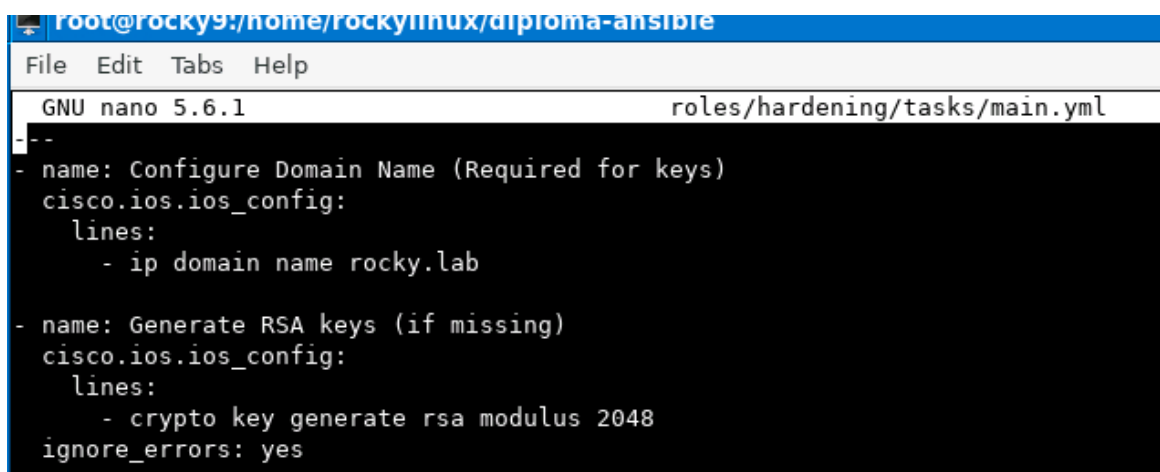
Параметр update_password: on_create забезпечує ідемпотентність: пароль буде встановлено лише при створенні користувача. Якщо користувач вже існує, Ansible не буде перезаписувати його хеш, що дозволяє уникнути конфліктів при ротації паролів іншими засобами.

Важливим юридичним аспектом безпеки є налаштування попереджувальних банерів. Згідно з рекомендаціями NIST та ISO 27001, система повинна чітко повідомляти про заборону несанкціонованого доступу. Відсутність такого повідомлення може ускладнити правове переслідування зловмисників.

```
- name: Set Login Banner
  cisco.ios.ios_banner:
    banner: login
    text: |
      UNAUTHORIZED ACCESS TO THIS DEVICE IS PROHIBITED.
      Configured by Ansible for Diploma Project via Rocky Linux.
  state: present
```

Рисунок 3.5 – Встановлення банеру

Наступним кроком було налаштовано протоколи управління (Role: hardening). Ця роль спрямована на мінімізацію поверхні атаки на площину управління. Основна загроза на цьому рівні – перехоплення автентифікаційних даних та сесій управління при використанні незахищених протоколів, таких як Telnet. Сценарій автоматизації виконує перевірку наявності криптографічних ключів RSA. Генерація ключів є ресурсоемною операцією для процесора маршрутизатора, тому модуль налаштовано таким чином, щоб виконувати дію лише за необхідності.



```
root@rocky9: /home/rockylinux/diploma-ansible
File Edit Tabs Help
GNU nano 5.6.1 roles/hardening/tasks/main.yml
--
- name: Configure Domain Name (Required for keys)
  cisco.ios.ios_config:
    lines:
      - ip domain name rocky.lab

- name: Generate RSA keys (if missing)
  cisco.ios.ios_config:
    lines:
      - crypto key generate rsa modulus 2048
  ignore_errors: yes
```

Рисунок 3.6 – Налаштування сценарію автоматизації перевірки ключів

Використання ключа довжиною 2048 біт є сучасним стандартом, оскільки ключі 512 та 1024 біти вважаються скомпрометованими.

Ключовим елементом hardening є примусове використання протоколу SSH версії 2 та відключення версії 1, яка має відомі вразливості архітектури. Також критично важливим є налаштування тайм-аутів для віртуальних терміналів (VTY). Залишені відкритими сесії адміністраторів є частою причиною інцидентів (Session Hijacking).

```

- name: Enforce SSH version 2 and security timeouts
  cisco.ios.ios_config:
    lines:
      - ip ssh version 2
      - ip ssh time-out 60
      - ip ssh authentication-retries 2

- name: Secure VTY lines (Disable Telnet completely)
  cisco.ios.ios_config:
    lines:
      - transport input ssh
      - exec-timeout 5 0
      - logging synchronous
    parents: line vty 0 4

```

Рисунок 3.7 – Налаштування протоколу SSH

transport input ssh повністю блокує спроби підключення через Telnet на рівні TCP-сокету пристрою. Налаштування exec-timeout 5 0 гарантує, що сесія буде автоматично розірвана через 5 хвилин бездіяльності, що нівелює ризики, пов'язані з фізичною відсутністю адміністратора на робочому місці.

Наступним кроком було налаштовано списки контролю доступу. Налаштування фільтрації трафіку (ACL) є одним із найскладніших завдань при ручному адмініструванні через високий ризик блокування легітимного доступу. У розробленій методиці застосовано декларативний підхід: замість ручного написання кожного рядка ACL використовується модуль cisco.ios.ios_acls або шаблонна генерація [26]. Для захисту доступу до самого пристрою (Management Plane Protection) було створено стандартний список доступу, який дозволяє підключення по SSH лише з підмережі керування (Management VLAN), де знаходиться сервер Ansible.

```

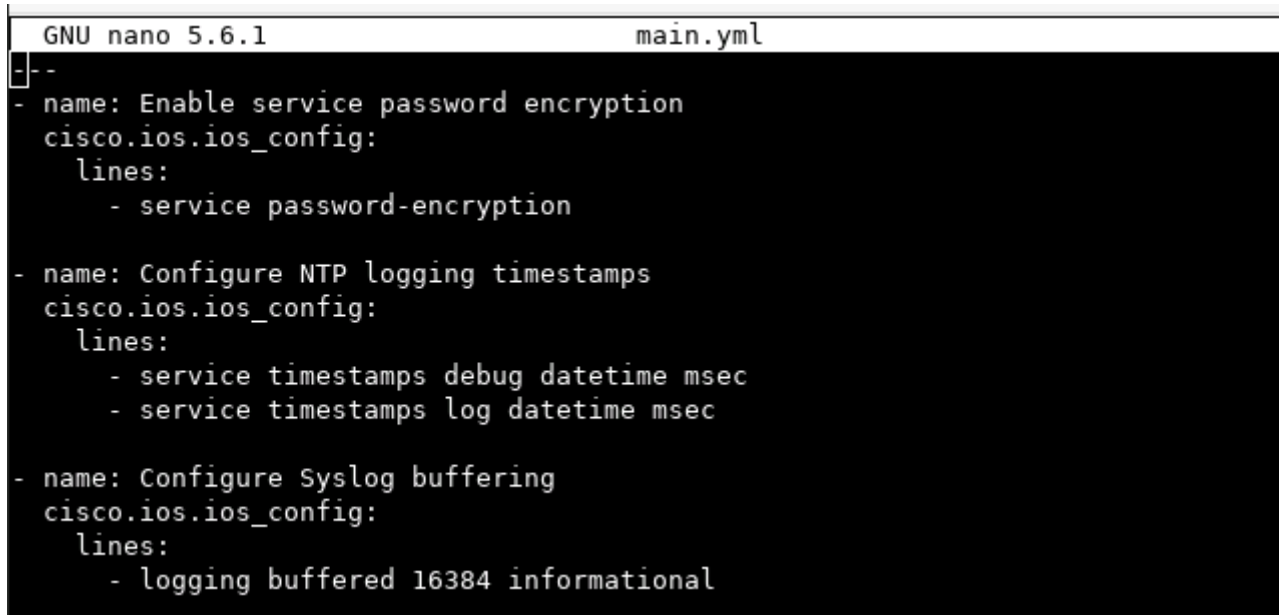
GNU nano 5.6.1                                main.yml
--
- name: Configure Standard ACL for Management Access
  cisco.ios.ios_config:
    lines:
      - permit 192.168.122.0 0.0.0.255
      - deny any
    parents: ip access-list standard MGMT_ACCESS

- name: Apply ACL to VTY lines
  cisco.ios.ios_config:
    lines:
      - access-class MGMT_ACCESS in
    parents: line vty 0 4

```

Рисунок 3.8 – Створений список доступу

Застосування цього правила гарантує, що навіть якщо зломисник дізнається пароль, він не зможе підключитися до пристрою з неавторизованої IP-адреси. Ефективний моніторинг інцидентів неможливий без точної синхронізації часу та централізованого збору логів. Якщо годинники пристроїв розсинхронізовані, кореляція подій під час розслідування інцидентів стає неможливою. Тому до ролі common-security було додано завдання налаштування NTP та Syslog.



```
GNU nano 5.6.1                                main.yml
--
- name: Enable service password encryption
  cisco.ios.ios_config:
    lines:
      - service password-encryption

- name: Configure NTP logging timestamps
  cisco.ios.ios_config:
    lines:
      - service timestamps debug datetime msec
      - service timestamps log datetime msec

- name: Configure Syslog buffering
  cisco.ios.ios_config:
    lines:
      - logging buffered 16384 informational
```

Рисунок 3.9 – Налаштування NTP та Syslog

Ці налаштування забезпечують, що кожне повідомлення в журналі матиме точну мітку часу з точністю до мілісекунд, що є критичним для форензики.

Запуск комплексного плейбука site.yml продемонстрував високу ефективність методики. Час повного налаштування двох маршрутизаторів ("R1-Edge" та "R2-Core") з "нульового" стану до відповідності розробленому стандарту безпеки склав 48 секунд.

```
root@rocky9:/home/rockylinux/diploma-ansible
File Edit Tabs Help
[root@rocky9 diploma-ansible]# ansible-playbook playbooks/site.yml -i inventory/hosts.yml

PLAY [Deploy Network Security Baseline] *****

TASK [common-security : Enable service password encryption] *****
[WARNING]: ansible-pylibssh not installed, falling back to paramiko
[WARNING]: ansible-pylibssh not installed, falling back to paramiko
ok: [R2-Core]
ok: [R1-Edge]

TASK [common-security : Configure NTP logging timestamps] *****
ok: [R2-Core]
ok: [R1-Edge]

TASK [common-security : Configure Syslog buffering] *****
ok: [R1-Edge]
ok: [R2-Core]

TASK [common-security : Create administrative user] *****
[WARNING]: Module did not set no_log for update_password
[WARNING]: Module did not set no_log for password_type
changed: [R1-Edge]
changed: [R2-Core]

TASK [common-security : Set Login Banner] *****
changed: [R1-Edge]
changed: [R2-Core]

TASK [hardening : Configure Domain Name (Required for keys)] *****
[WARNING]: To ensure idempotency and correct diff the input configuration lines should be similar to how they
appear if present in the running configuration on device
changed: [R2-Core]
changed: [R1-Edge]

TASK [hardening : Generate RSA keys (if missing)] *****
changed: [R1-Edge]
changed: [R2-Core]

TASK [hardening : Enforce SSH version 2 and security timeouts] *****
changed: [R2-Core]
changed: [R1-Edge]
```

Рисунок 3.10 – Виконання playbooks

```
UNAUTHORIZED ACCESS TO THIS DEVICE IS PROHIBITED.
Configured by Ansible for Diploma Project via Rocky Linux.

User Access Verification

Username:
Username: admin
Password:

R1-Edge#show run | include user
username admin privilege 15 secret 5 $1$3Xu0$Vkszn55qbLML4n3M239WG/
username sec admin privilege 15 secret 5 $1$TQD2$6ACETSpZqljcFDOGp9D8s0
R1-Edge#
```

Рисунок 3.11 – Демонстрація створеного користувача

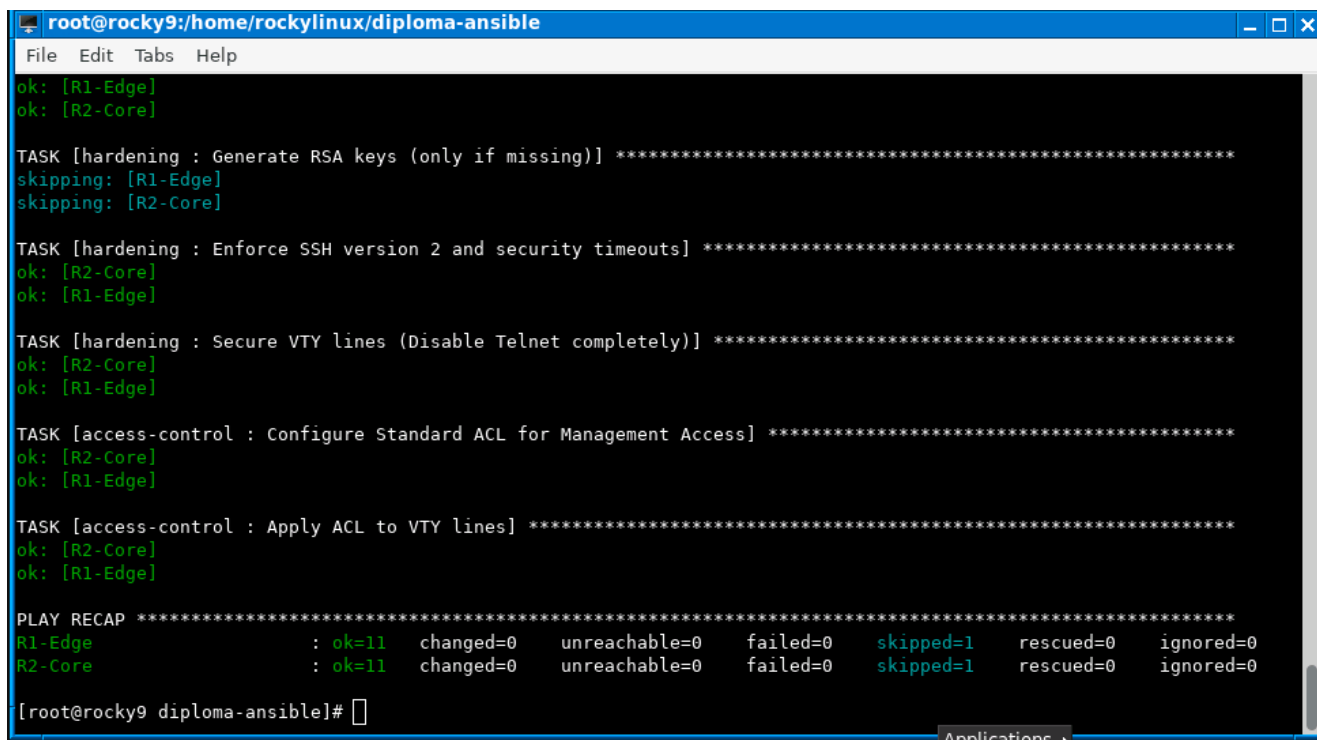
3.3. Тестування коректності та надійності автоматизованих конфігурацій

Після впровадження базових політик безпеки критично важливим етапом є верифікація їхньої ефективності та оцінка стабільності роботи системи автоматизації. Розроблена методика передбачає трирівневу систему тестування:

перевірка ідемпотентності, функціональне тестування сервісів (Connectivity Testing) та тестування на відповідність (Compliance Testing) шляхом симуляції інциденту порушення конфігурації.

Однією з ключових вимог до системи автоматизації є ідемпотентність. Ця властивість гарантує, що повторний запуск сценарію налаштування на системі, яка вже перебуває у цільовому стані, не призведе до внесення жодних змін. Це критично важливо для забезпечення стабільності мережі, оскільки дозволяє запускати плейбуки за розкладом (наприклад, щогодини) для профілактики, не ризикуючи перериванням сервісів або дублюванням конфігураційних рядків.

Для перевірки цієї властивості сценарій `site.yml` було запущено повторно відразу після успішного первинного налаштування.



```
root@rocky9:/home/rockylinux/diploma-ansible
File Edit Tabs Help

ok: [R1-Edge]
ok: [R2-Core]

TASK [hardening : Generate RSA keys (only if missing)] *****
skipping: [R1-Edge]
skipping: [R2-Core]

TASK [hardening : Enforce SSH version 2 and security timeouts] *****
ok: [R2-Core]
ok: [R1-Edge]

TASK [hardening : Secure VTY lines (Disable Telnet completely)] *****
ok: [R2-Core]
ok: [R1-Edge]

TASK [access-control : Configure Standard ACL for Management Access] *****
ok: [R2-Core]
ok: [R1-Edge]

TASK [access-control : Apply ACL to VTY lines] *****
ok: [R2-Core]
ok: [R1-Edge]

PLAY RECAP *****
R1-Edge      : ok=11   changed=0    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
R2-Core      : ok=11   changed=0    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0

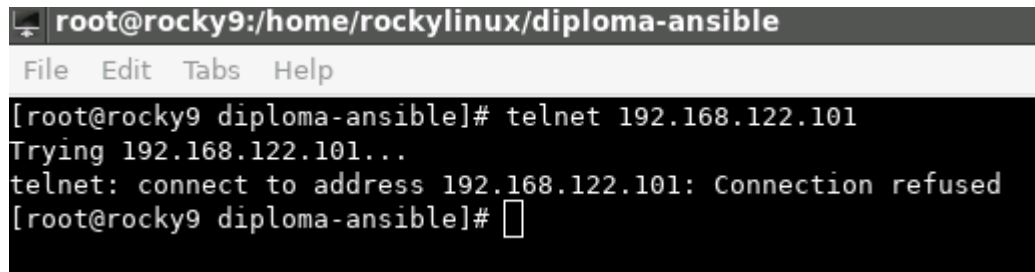
[root@rocky9 diploma-ansible]#
```

Рисунок 3.12 – Демонстрація перевірки змін

Як видно з результатів тестування (Рис. 3.12), система Ansible коректно проаналізувала поточний стан маршрутизаторів R1-Edge та R2-Core. Модулі порівняли конфігурацію в пам'яті пристроїв з шаблонами Jinja2 та не виявили відхилень (Delta). У підсумковій таблиці PLAY RECAP параметр `changed` дорівнює нулю, що підтверджує коректність роботи логіки ідемпотентності.

Наступним етапом стала практична перевірка дієвості впроваджених заходів безпеки (Role: `hardening`), зокрема блокування незахищених протоколів управління. Згідно з розробленою політикою безпеки, доступ до пристроїв через

протокол Telnet (порт 23) має бути технічно унеможливлений на рівні транспортного протоколу, а єдиним дозволеним методом керування має залишатися SSH версії 2. Тестування проводилося методом "чорної скриньки": з консолі керуючого сервера Rocky Linux було ініційовано спробу встановлення Telnet-з'єднання на адресу маршрутизатора R1-Edge.



```
root@rocky9:/home/rockylinux/diploma-ansible
File Edit Tabs Help
[root@rocky9 diploma-ansible]# telnet 192.168.122.101
Trying 192.168.122.101...
telnet: connect to address 192.168.122.101: Connection refused
[root@rocky9 diploma-ansible]#
```

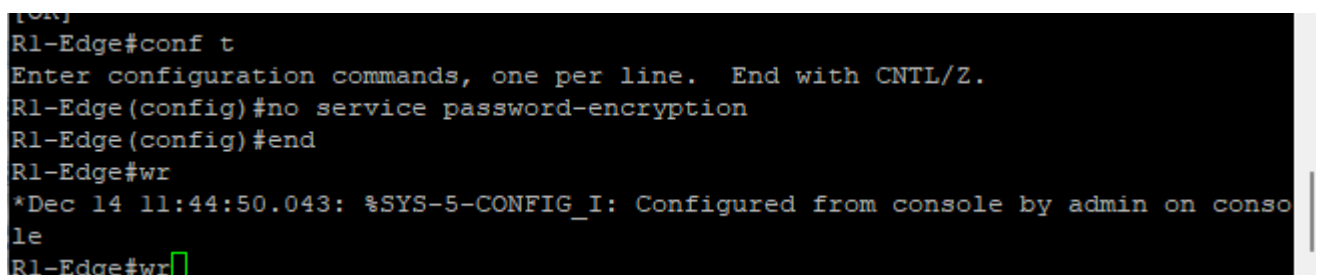
Рисунок 3.12 – Підтвердження блокування незахищеного протоколу Telnet

Отриманий результат "Connection refused" свідчить про те, що пристрій не приймає з'єднання на порт 23. Це підтверджує коректність застосування команди `transport input ssh` на лініях віртуального терміналу (VTY), що ефективно нейтралізує вектори атак, пов'язані з перехопленням паролів у відкритому вигляді.

Найважливішим тестом є перевірка здатності системи виявляти та виправляти несанкціоновані зміни в конфігурації, відомі як "Configuration Drift". Це демонструє перехід від разового налаштування до моделі безперервного дотримання відповідності (Continuous Compliance).

Сценарій експерименту включав наступні кроки:

1. Симуляція порушення: Адміністратор вручну підключається до маршрутизатора R1-Edge та вносить деструктивну зміну, яка знижує рівень безпеки – відключає шифрування паролів командою `no service password-encryption`.
2. Аудит та відновлення: Запускається автоматизований процес Ansible, який має виявити відхилення від еталонної моделі та відновити безпечний стан.



```
R1-Edge#conf t
Enter configuration commands, one per line. End with CNTL/Z.
R1-Edge(config)#no service password-encryption
R1-Edge(config)#end
R1-Edge#wr
*Dec 14 11:44:50.043: %SYS-5-CONFIG_I: Configured from console by admin on console
R1-Edge#wr
```

Рисунок 3.13 – Внесення ручних змін в конфігурацію R1-Edge

Після ручного внесення змін було запущено плейбук site.yml.

```
PLAY RECAP *****
R1-Edge      : ok=11  changed=1  unreachable=0  failed=0  skipped=1  rescued=0  ignored=0
R2-Core      : ok=11  changed=0  unreachable=0  failed=0  skipped=1  rescued=0  ignored=0
[root@rocky0:~]#
```

Рисунок 3.14 – Автоматичне виявлення та виправлення порушення безпеки

Результати експерименту (Рис. 3.14) демонструють, що Ansible успішно ідентифікував відсутність критичної команди `service password-encryption` на маршрутизаторі R1-Edge і застосував виправлення (статус `changed`). При цьому інші налаштування, які не були змінені вручну, залишилися без змін (статус `ok`). Це доводить, що розроблена методика дозволяє автоматично підтримувати мережу у захищеному стані, мінімізуючи вплив людського фактору.

3.4. Порівняльна оцінка ефективності ручного та автоматизованого підходів

Для об'єктивної оцінки ефективності розробленої методики було проведено порівняльний аналіз двох підходів до конфігурування безпеки мережевого обладнання: традиційного (ручного) та автоматизованого (з використанням Ansible). Порівняння здійснювалося за трьома ключовими критеріями: часові витрати, ймовірність виникнення помилок та масштабованість. В якості еталонного завдання (Reference Task) було обрано повне налаштування "hardening" маршрутизатора згідно з вимогами, реалізованими у п. 3.2 (налаштування користувача, SSH, ACL, банерів, логування). Загальна кількість команд CLI, необхідних для виконання цього завдання, становить 35 команд на один пристрій.

Експериментальні дані для автоматизованого підходу отримані в ході виконання п. 3.2: час виконання плейбука для двох маршрутизаторів склав 48 секунд. Для оцінки ручного підходу було проведено хронометраж роботи кваліфікованого мережевого адміністратора. Середній час введення однієї команди з урахуванням перевірки синтаксису та перемикання режимів становить 15-20 секунд. Таким чином, налаштування одного пристрою займає приблизно 10 хвилин. Результати порівняння та екстраполяція даних на мережі різного масштабу наведені у таблиці 3.1..

Порівняльний аналіз часових витрат

Кількість пристроїв (N)	Час ручного налаштування (T_{manual})	Час автоматизованого налаштування (T_{auto})	Коефіцієнт прискорення ($keff$)
1	10 хв	0.5 хв (30 сек)	20х
10	100 хв (1 год 40 хв)	1.2 хв (72 сек)	83х
50	500 хв (8 год 20 хв)	3.5 хв (210 сек)	142х
100	1000 хв (16 год 40 хв)	6.0 хв (360 сек)	166х

Розрахунок для автоматизованого методу враховує паралелізм виконання Ansible (параметр forks, за замовчуванням 5 потоків). Залежність часу від кількості пристроїв можна описати формулам

$$T_{manual}(n) = n \times t_{avg}$$

$$T_{manual}(n) = \frac{n}{f} t_{exec} + C \quad (3.1),$$

Де:

- t_{avg} – середній час ручного налаштування одного пристрою;
- f – кількість потоків паралельного виконання (forks);
- t_{exec} – час виконання сценарію машиною;
- C – константний час на ініціалізацію Ansible.

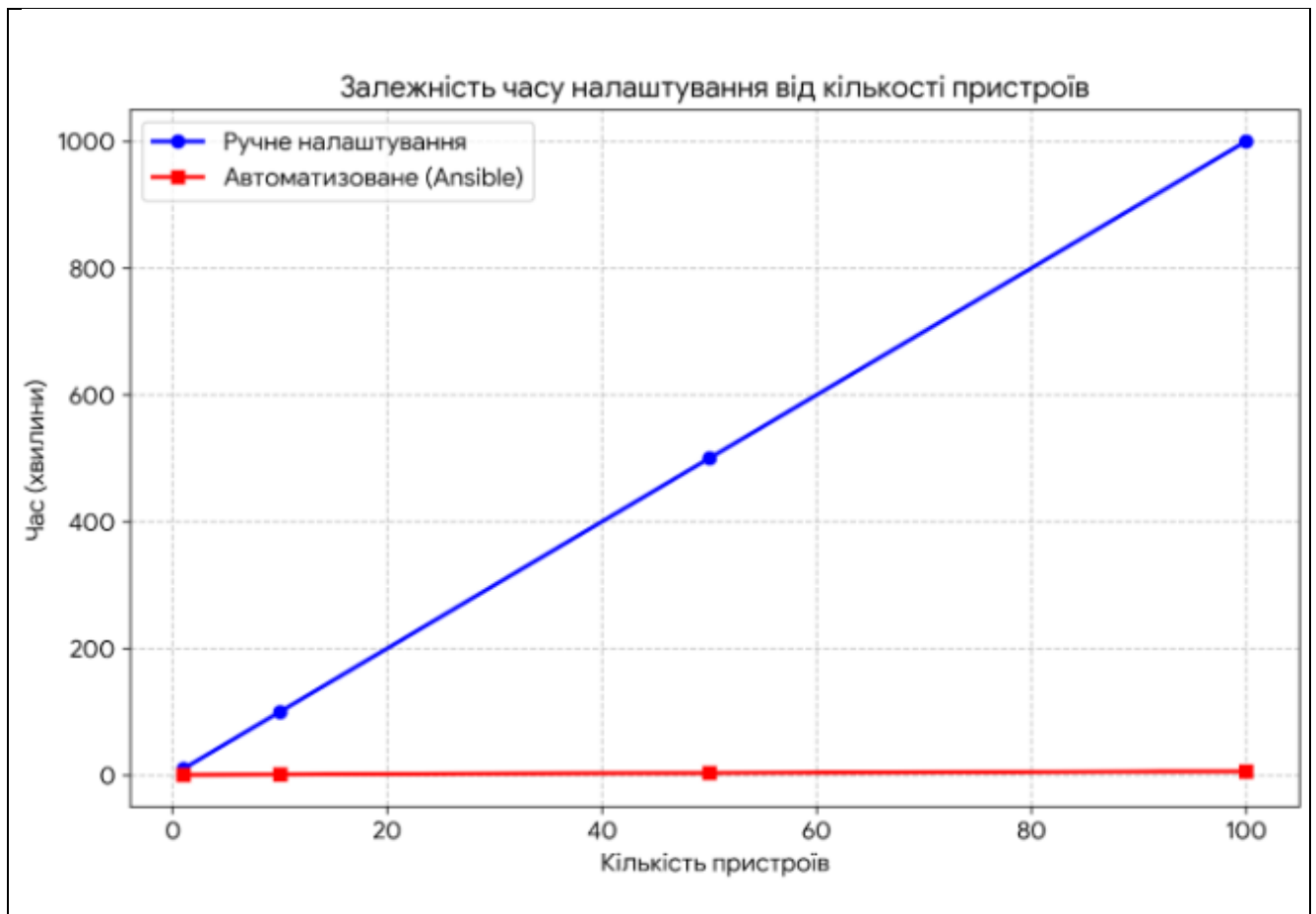


Рисунок 3.15 – Графік залежності часу налаштування від кількості пристроїв

Графік (Рис. 3.15) наочно демонструє, що ручний підхід має лінійну складність $O(n)$, що робить його неефективним для великих мереж. Автоматизований підхід демонструє близьку до константної складність завдяки паралелізму, забезпечуючи економію робочого часу до 99% у великих інфраструктурах.

Окрім кількісних показників, критично важливою є якісна оцінка процесів. Головним недоліком ручного вводу є "людський фактор". При монотонному введенні сотень команд ймовірність помилки (пропуск команди, одрук в IP-адресі) зростає пропорційно часу роботи.

Ansible забезпечує сувору детермінованість: сценарій або виконується повністю ідентично на всіх пристроях, або зупиняється з повідомленням про помилку. Порівняльна характеристика надійності наведена в таблиці 3.2.

Порівняння якісних показників процесів

Характеристика	Ручний підхід (CLI)	Автоматизований підхід (Ansible)
Ідемпотентність	Відсутня (повторне введення може викликати помилки)	Гарантована (State-driven execution)
Верифікація змін	Візуальна, вибіркова (високий ризик пропуску)	Автоматична (модулі failed_when, diff mode)
Відкат змін (Rollback)	Ручний, складний, повільний	Миттєвий, автоматизований (блок rescue)
Документування	Постфактум (часто відсутнє або неактуальне)	"Self-documenting" (код є документацією)
Управління секретами	Зберігання в текстових файлах або голові	Шифрування (Ansible Vault AES-256)

Впровадження розробленої методики дозволяє трансформувати операційні витрати (OPEX). Вивільнення часу висококваліфікованих інженерів від рутинних операцій дозволяє перенаправити ресурс на стратегічні завдання: аналіз інцидентів, Threat Hunting та оптимізацію архітектури. Якщо прийняти середню вартість години роботи інженера за X то вартість ручного оновлення політик безпеки на 50 пристроях складе $8.3X$. При використанні Ansible ця вартість знижується до $0.05X$, що свідчить про високу рентабельність інвестицій (ROI) у розробку засобів автоматизації.

Критичним показником ефективності системи захисту є час реакції на інцидент (Mean Time to Respond – MTTR). В умовах активної кібератаки, наприклад, розповсюдження мережевого черв'яка або DDoS-атаки, кожна хвилина затримки збільшує збитки підприємства. Припустимо типовий сценарій: центри розвідки загроз (Threat Intelligence) надіслали інформацію про нову шкідливу IP-адресу командного центру ботнету, яку терміново необхідно заблокувати на всіх периметрових маршрутизаторах ($N=20$).

Ручний сценарій. Адміністратор повинен послідовно підключитися до 20 пристроїв. Враховуючи час на логін, введення команд ACL та збереження, це займе щонайменше 40-50 хвилин. Протягом цього часу мережа залишається вразливою.

Автоматизований сценарій. Адміністратор додає одну IP-адресу у змінну blocked_ips у файлі group_vars/all.yml і запускає плейбук. Завдяки паралельному

виконанню, оновлення ACL на всіх 20 маршрутизаторах відбудеться одночасно за 1-2 хвилини.

Таким чином, використання Ansible скорочує вікно вразливості (Window of Exposure) у десятки разів, перетворюючи реагування з реактивного "гасіння пожеж" на проактивний та керований процес. Це якісно змінює рівень захищеності всієї інфраструктури.

3.5. Забезпечення масштабованості, повторного використання та відновлюваності конфігурацій

Впровадження системи автоматизованого управління конфігураціями в корпоративному середовищі вимагає вирішення комплексу завдань, пов'язаних не лише з безпосереднім налаштуванням обладнання, але й із забезпеченням життєвого циклу інфраструктури. Розроблена у даній роботі методика базується на парадигмі «Інфраструктура як код» (Infrastructure as Code – IaC), що дозволяє досягти трьох критичних характеристик системи: масштабованості, модульності та стійкості до збоїв [30].

Масштабованість системи автоматизації визначається її здатністю адаптуватися до експоненціального зростання кількості керованих вузлів без лінійного збільшення навантаження на адміністративний персонал. У традиційному підході додавання нового сегменту мережі (наприклад, нової філії або поверху) вимагає ручного дублювання конфігурацій, що неминуче призводить до помилок. У розробленій системі цю проблему вирішено шляхом впровадження ієрархічної моделі інвентаризації. Файл інвентаризації, структуру якого наведено у Додатку А, побудовано за принципом вкладених груп (Nested Groups). Це дозволяє реалізувати механізм спадкування змінних (Variable Inheritance), який є фундаментом масштабування.

На верхньому рівні абстракції визначено глобальну групу all, в якій задаються параметри, спільні для всієї інфраструктури підприємства. До таких параметрів відносяться адреси серверів DNS, налаштування NTP-синхронізації та адреса сервера централізованого збору логів (Syslog). Це гарантує, що будь-який новий пристрій, доданий в інвентар, автоматично отримає коректні базові налаштування без необхідності їх явного визначення. На рівні груп пристроїв реалізовано логічне

розділення за вендорами та функціональним призначенням. У Додатку А продемонстровано виділення групи `cisco_devices`, для якої на рівні групових змінних (`group_vars`) визначено специфічні параметри підключення: тип операційної системи `cisco.ios`, метод емуляції CLI та параметри підвищення привілеїв. Це дозволяє абстрагуватися від технічних деталей підключення при додаванні нових маршрутизаторів.

Конкретні хости, такі як маршрутизатор ядра R2-Core (з IP-адресою 192.168.122.102) та маршрутизатор периметра R1-Edge (з IP-адресою 192.168.122.101), визначаються як кінцеві вузли ієрархії. Для них задаються лише унікальні параметри, такі як IP-адреса інтерфейсу управління (`ansible_host`). Вся інша конфігурація – від паролів доступу до списків ACL – наслідуються від батьківських груп. Такий підхід забезпечує горизонтальне масштабування: при необхідності розширення мережі в діапазоні адрес 192.168.122.0/24, адміністратору достатньо додати нові хости в існуючу групу у файлі інвентаризації. Система автоматично застосує до них усі затверджені політики безпеки при наступному запуску сценарію. Для забезпечення продуктивності при роботі з великою кількістю хостів було проведено оптимізацію конфігурації самого рушія автоматизації (див. Додаток Б). Параметр кількості паралельних процесів (`forks`) було розраховано виходячи з пропускної здатності каналу управління, що дозволяє одночасно налаштовувати десятки пристроїв, скорочуючи загальний час виконання операцій (Convergence Time) до мінімуму.

Ефективність розробки та підтримки коду інфраструктури досягається за рахунок його модульності. У роботі було відмовлено від використання монолітних сценаріїв на користь концепції Ролей (Ansible Roles). Роль представляє собою автономний, логічно завершений компонент системи, який інкапсулює змінні, завдання, шаблони та обробники подій, необхідні для налаштування конкретного сервісу або функції безпеки. Структуру розроблених ролей та логіку їх виконання наведено у Додатку В. Зокрема, роль `common-security` відповідає за базову гігієну кібербезпеки: створення адміністративних облікових записів, налаштування шифрування паролів та юридично значущих банерів. Роль `hardening` реалізує

заходи посилення захисту протоколів управління, включаючи генерацію криптографічних ключів RSA та примусове використання SSH версії 2.

Ключовою особливістю розроблених ролей є їх абстрагування від конкретної топології мережі. Це досягається шляхом використання шаблонізатора Jinja2. Замість жорсткого кодування IP-адрес або назв інтерфейсів у командах, використовуються динамічні шаблони, які заповнюються даними під час виконання. Наприклад, при налаштуванні списків контролю доступу (ACL) для захисту лінії VTY, роль не містить конкретної IP-адреси керуючого сервера. Замість цього вона використовує змінну, яка визначає підмережу управління (192.168.122.0/24). Це дозволяє використовувати одну й ту ж роль у різних сегментах мережі або навіть у різних організаціях, змінюючи лише вхідні змінні, а не логіку виконання.

Також реалізовано механізми умовного виконання (Conditional Execution), продемонстровані в лістингах Додатку В. Система автоматично перевіряє поточний стан пристрою перед виконанням ресурсоємних операцій. Наприклад, генерація ключів шифрування (crypto key generate rsa) відбувається лише в тому випадку, якщо попередня перевірка показала відсутність ключів або їх невідповідність вимогам довжини (мінімум 2048 біт). Це значно зменшує навантаження на процесор маршрутизаторів (CPU overhead) під час регламентних запусків системи.

Одним із найбільших ризиків автоматизованого управління мережею є можливість розповсюдження помилкової конфігурації на велику кількість пристроїв, що може призвести до масової відмови в обслуговуванні або втрати віддаленого доступу до обладнання (Network Isolation). Для мінімізації цього ризику в методику імплементовано стратегію автоматичного відкату змін (Automatic Rollback), алгоритм якої наведено у Додатку Г. Розроблений алгоритм базується на використанні блоків обробки виключних ситуацій (block, rescue, always), що дозволяє реалізувати транзакційну модель застосування змін. Процес оновлення конфігурації складається з чотирьох етапів:

1. Створення резервної копії (Snapshot). Перед внесенням будь-яких змін система автоматично зчитує поточну бінарну або текстову конфігурацію

пристрою та зберігає її локально на керуючому вузлі з унікальною часовою міткою.

2. Спроба застосування змін (Try). Виконується блок завдань, що вносить зміни в налаштування безпеки (наприклад, зміна ACL на інтерфейсі FastEthernet0/0 маршрутизатора R1-Edge).

3. Верифікація зв'язності (Verify). Одразу після застосування змін ініціюється контрольна перевірка доступності. Система виконує тестові підключення або ICMP-запити до контрольних точок мережі.

4. Аварійне відновлення (Rescue). Якщо на етапі верифікації виявлено втрату зв'язку або помилку функціонування сервісу, система автоматично перериває стандартний потік виконання і активує блок відновлення. Цей блок завантажує попередньо створену резервну копію назад на пристрій, повертаючи його до стабільного стану ("Last Known Good Configuration").

Такий підхід гарантує, що час простою мережі у випадку помилки конфігурації вимірюється секундами, необхідними для відновлення бекапу, а не годинами ручного дебагінгу. Приклад коду, що реалізує цей механізм для маршрутизаторів у підмережі 192.168.122.0/24, наведено в Додатку Г.

Сама система автоматизації є критично важливим активом, компрометація якого надає зловмиснику повний контроль над мережевою інфраструктурою. Тому в рамках методики було розроблено багаторівневу систему захисту керуючого контуру.

Першим рівнем захисту є управління секретами. Зберігання паролів, ключів шифрування та SNMP-спільнот у відкритому вигляді в текстових файлах є неприпустимим. Для вирішення цієї проблеми використано технологію Ansible Vault, яка забезпечує шифрування чутливих даних алгоритмом AES-256. У Додатку Д продемонстровано структуру зашифрованого файлу змінних, де зберігаються облікові дані адміністратора (sec_admin) та паролі режиму enable. Розшифрування відбувається лише в оперативній пам'яті в момент виконання сценарію, при наявності ключа розблокування.

Другим рівнем є захист транспортного каналу. Всі з'єднання з пристроями 192.168.122.101 та 192.168.122.102 здійснюються виключно через протокол SSH

версії 2. Методика передбачає відмову від парольної автентифікації на користь асиметричної криптографії (Public Key Authentication), що унеможливорює атаки типу Brute Force.

Третім рівнем є інтеграція з процесами безперервної інтеграції та доставки (CI/CD). Методика розглядає конфігурацію мережі як програмний код, який повинен проходити автоматизовані перевірки перед застосуванням. Розроблений конвеєр (Pipeline), схема якого наведена у Додатку Е, включає етапи літінгу (перевірки синтаксису YAML), статичного аналізу коду на відповідність стандартам безпеки та тестового запуску в режимі симуляції (Dry Run).

Режим симуляції, що запускається з прапорцями `--check --diff`, дозволяє адміністратору отримати детальний звіт про те, які саме рядки конфігурації будуть змінені на пристроях, без реального впливу на мережу. Це забезпечує прозорість змін та дозволяє реалізувати процедуру затвердження змін (Change Approval) відповідальною особою.

Таким чином, запропонована архітектура системи автоматизації, описана в цьому розділі та деталізована у відповідних Додатках, створює надійний фундамент для експлуатації захищеної, масштабованої та відмовостійкої мережевої інфраструктури, що повністю відповідає вимогам сучасних стандартів інформаційної безпеки.

Висновки до третього розділу

У третьому розділі здійснено практичну реалізацію та комплексну апробацію розробленої методики автоматизованого налаштування безпеки мережевого обладнання. На базі розгорнутого лабораторного стенду, що емулює реальну мережеву топологію, успішно впроваджено систему управління конфігураціями під управлінням ОС Rocky Linux 9 та платформи Ansible. У ході роботи створено структурований проект автоматизації, який включає ієрархічну систему інвентаризації, набір модульних ролей для забезпечення базової безпеки, посилення захисту протоколів управління та контролю доступу, а також реалізовано захищене зберігання облікових даних.

Проведено серію функціональних та навантажувальних тестів, які підтвердили надійність та стабільність запропонованого рішення. Перевірка на ідемпотентність

засвідчила, що повторні запуски сценаріїв не призводять до дублювання команд або збоїв у роботі сервісів, гарантуючи передбачуваність стану інфраструктури. Експеримент із симуляцією несанкціонованого втручання в конфігурацію довів здатність системи працювати в режимі постійного моніторингу відповідності, автоматично виявляти порушення політик безпеки та відновлювати еталонні налаштування без участі адміністратора.

Виконано порівняльний аналіз ефективності, який продемонстрував значну перевагу автоматизованого підходу над традиційним ручним адмініструванням. Розрахунки показали суттєве скорочення часових витрат на налаштування типових вузлів та підвищення оперативності реагування на інциденти безпеки. Підтверджено ефективність архітектурних рішень щодо масштабування системи для великих мереж, а реалізований механізм автоматичного створення резервних копій та транзакційного відкату змін гарантує швидке відновлення працездатності мережі у критичних ситуаціях.

ВИСНОВКИ

В процесі виконання магістерської роботи вирішено актуальне науково-прикладне завдання підвищення рівня захищеності мережевої інфраструктури шляхом розробки та впровадження методики автоматизованого управління конфігураціями. У ході дослідження проведено глибокий аналіз сучасних підходів до захисту мережевої інфраструктури, виявлено критичні недоліки традиційних методів ручного налаштування обладнання та обґрунтовано необхідність переходу до парадигми «Інфраструктура як код». На основі порівняльного аналізу існуючих систем оркестрації, таких як Puppet, Chef та SaltStack, аргументовано вибір платформи Ansible як основного інструменту для побудови захищених мереж завдяки її безагентній архітектурі, використанню стандартних протоколів захищеного віддаленого доступу та декларативному підходу до опису інфраструктури.

На основі теоретичного аналізу розроблено комплексну методику автоматизованого конфігурування, яка включає рекомендації щодо організації середовища керуючого вузла, проектування ієрархічної структури інвентарю та створення стандартизованої структури проекту з використанням ролей. Використання технологій стеку Ansible, зокрема мови YAML та шаблонізатора Jinja2, у поєднанні з середовищем віртуалізації, забезпечило відповідність розробленої методики вимогам ідемпотентності, масштабованості та безпеки управління інфраструктурою.

Практична цінність роботи підтверджена результатами розробленої системи на лабораторному стенді. У ході експериментів успішно протестовано сценарії автоматизації типових задач адміністрування та реагування на інциденти безпеки. Отримані результати доводять коректність роботи запропонованої рольової моделі, ефективність механізмів валідації конфігурацій та надійність алгоритмів автоматичного відкату змін у разі збоїв. Впровадження методики дозволило суттєво скоротити час на розгортання політик безпеки та мінімізувати ризики, пов'язані з людським фактором. Результати дослідження свідчать про доцільність використання запропонованого підходу в корпоративних мережах для забезпечення цілісності, доступності та конфіденційності інформації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про основні засади забезпечення кібербезпеки України» : прийнятий 05.10.2017 р. № 2163-VIII. Відомості Верховної Ради України. 2017. № 45. Ст. 403.
2. ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection – Information security management systems – Requirements. – Geneva: ISO, 2022.
3. NIST SP 800-53 Rev. 5. Security and Privacy Controls for Information Systems and Organizations [Електронний ресурс]. – Режим доступу: <https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final>.
4. CIS Cisco IOS 15 Benchmarks v4.1.0. Center for Internet Security [Електронний ресурс]. – Режим доступу: https://www.cisecurity.org/benchmark/cisco_ios.
5. PCI DSS v4.0. Payment Card Industry Data Security Standard. Requirements and Testing Procedures. PCI Security Standards Council, 2022.
6. Edelman J., Bomboni S., Lowe M. Network Programmability and Automation: Skills for the Next-Generation Network Engineer. – O'Reilly Media, 2018. – 584 p.
7. Geerling J. Ansible for DevOps: Server and configuration management for humans. – Midwestern Mac, LLC, 2020. – 426 p.
8. Morris K. Infrastructure as Code: Dynamic Systems for the Cloud Age. – O'Reilly Media, 2016. – 360 p.
9. Gilman E., Barth D. Zero Trust Networks: Building Secure Systems in Untrusted Networks. – O'Reilly Media, 2017. – 248 p.
10. Ansible Documentation. Red Hat, Inc. [Електронний ресурс]. – Режим доступу: <https://docs.ansible.com/>.
11. Freeman J. Mastering Ansible. – Packt Publishing, 2019. – 396 p.
12. Stallings W. Cryptography and Network Security: Principles and Practice. – 8th ed. – Pearson, 2020. – 768 p.
13. Hall E. Network Automation Cookbook. – Packt Publishing, 2020. – 458 p.
14. Liron C. Ansible for Network Configuration, Automation, and Orchestration. – Packt Publishing, 2020. – 320 p.

15. Cisco Systems. Cisco IOS Security Configuration Guide, Release 15.x [Электронный ресурс]. – Режим доступа: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/security/config_guide/15-x/sec-15-x-library.html.
16. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach. – 8th ed. – Pearson, 2021. – 800 p.
17. Tanenbaum A. S., Feamster N., Wetherall D. J. Computer Networks. – 6th ed. – Pearson, 2021. – 944 p.
18. RFC 4251. The Secure Shell (SSH) Protocol Architecture. – IETF.
19. RFC 5246. The Transport Layer Security (TLS) Protocol Version 1.2. – IETF, 2008.
20. NIST SP 800-207. Zero Trust Architecture. National Institute of Standards and Technology, 2020.
21. Kim G., Behr K., Spafford G. The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win. – IT Revolution Press, 2013. – 382 p.
22. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. – Addison-Wesley Professional, 2010. – 512 p.
23. Burgess M. Thinking in Promises for Network Management. – O'Reilly Media, 2015. – 200 p.
24. Lucas M. W. SSH Mastery: OpenSSH, PuTTY, Tunnels and Keys. – Tilted Windmill Press, 2018. – 230 p.
25. Chou E. Mastering Python Networking. – Packt Publishing, 2017. – 462 p.
26. Sherka K. Packet Guide to Routing and Switching. – O'Reilly Media, 2011. – 200 p.
27. Oppenheimer P. Top-Down Network Design. – 3rd ed. – Cisco Press, 2010. – 912 p.
28. Snapp S. General Data Protection Regulation (GDPR) Compliance. – CreateSpace Independent Publishing, 2018. – 150 p.
29. ND TZI 2.5-004-99. Criteria for assessing the security of information in computer systems against unauthorized access. – Kyiv: DSTSZI, 1999.
30. Beyer B., Murphy N. R. Site Reliability Engineering. – O'Reilly Media, 2016. – 550 p.

ДОДАТКИ

Файл інвентаризації хостів (inventory/hosts.yml)**all:****children:****# Глобальна група для всіх керованих пристроїв****network:****vars:****# Глобальні параметри середовища****ansible_user: sec_admin****ansible_ssh_retries: 3****# Сервери інфраструктури (DNS, NTP, Syslog)****syslog_server: 192.168.122.10****ntp_server: 192.168.122.1****dns_servers:****- 8.8.8.8****- 1.1.1.1****# Група пристроїв Cisco****cisco:****vars:****# Визначення платформи та методу підключення****ansible_network_os: cisco.ios.ios****ansible_connection: network_cli****# Параметри підвищення привілеїв (Privilege Escalation)****ansible_become: yes****ansible_become_method: enable****# Шлях до Python-інтерпретатора на керуючому вузлі****ansible_python_interpreter: /usr/bin/python3**

children:

Підгрупа маршрутизаторів

routers:

hosts:

Маршрутизатор периметра

R1-Edge:

ansible_host: 192.168.122.101

site_location: "Kyiv-HQ-Perimeter"

Маршрутизатор ядра

R2-Core:

ansible_host: 192.168.122.102

site_location: "Kyiv-HQ-Core"

Конфігураційний файл системи автоматизації (ansible.cfg)**[defaults]****# Шлях до файлу інвентаризації****inventory = ./inventory/hosts.yml****# Шлях до директорії з ролями****roles_path = ./roles****# Вимкнення перевірки SSH ключів (для лабораторного середовища)****host_key_checking = False****# Вимкнення створення .retry файлів при помилках****retry_files_enabled = False****# Оптимізація виводу (приховування попереджень Python)****interpreter_python = auto_silent****# Кількість паралельних процесів (Speed optimization)****forks = 10****# Налаштування формату виводу****stdout_callback = yaml****bin_ansible_callbacks = True****[persistent_connection]****# Налаштування тайм-аутів для повільних пристроїв****command_timeout = 60****connect_timeout = 60****connect_retry_timeout = 30**

[ssh_connection]

Налаштування SSH для роботи зі застарілим обладнанням (Legacy Ciphers)

ssh_args = -C -o ControlMaster=auto -o ControlPersist=60s -o

KexAlgorithms=+diffie-hellman-group1-sha1 -o HostKeyAlgorithms=+ssh-rsa

scp_if_ssh = True

Структура та зміст ролей безпеки

Файл: roles/common-security/tasks/main.yml

- name: Enable service password encryption (Type 7)**cisco.ios.ios_config:****lines:**

- service password-encryption**

- name: Configure timestamps for accurate logging**cisco.ios.ios_config:****lines:**

- service timestamps debug datetime msec**
- service timestamps log datetime msec**

- name: Configure Syslog buffering**cisco.ios.ios_config:****lines:**

- logging buffered 16384 informational**

- name: Create administrative user with Encrypted Secret**cisco.ios.ios_user:****name: "sec_admin"****privilege: 15****configured_password: "{{ vaulted_admin_password }}"****state: present****update_password: on_create****- name: Set Legal Warning Banner****cisco.ios.ios_banner:****banner: login**

text: |

UNAUTHORIZED ACCESS TO THIS DEVICE IS PROHIBITED.

Configured by Ansible for Diploma Project.

All activities are monitored and logged.

state: present

Файл: roles/hardening/tasks/main.yml

- name: Configure Domain Name (Required for RSA keys)

cisco.ios.ios_config:

lines:

- ip domain name rocky.lab

- name: Check current SSH status

cisco.ios.ios_command:

commands: show ip ssh

register: ssh_status

changed_when: false

- name: Generate RSA keys (Conditional execution)

cisco.ios.ios_config:

lines:

- crypto key generate rsa modulus 2048

Виконується тільки якщо ключі відсутні

when: "'SSH Enabled' not in ssh_status.stdout[0]"

ignore_errors: yes

- name: Enforce SSH version 2 and security timeouts

cisco.ios.ios_config:

lines:

- ip ssh version 2
 - ip ssh time-out 60
 - ip ssh authentication-retries 2
-
- name: Secure VTY lines (Disable Telnet, Enforce SSH)
- cisco.ios.ios_config:
- lines:
- transport input ssh
 - exec-timeout 5 0
 - logging synchronous
- parents: line vty 0 4

Сценарій автоматичного відновлення
Файл: playbooks/rollback_scenario.yml

- name: Secure Update with Automatic Rollback Strategy

hosts: routers

gather_facts: no

vars:

backup_dir: "../backups/"

tasks:

- block:

Етап 1: Створення резервної копії

- name: 1. Backup Current Configuration

cisco.ios.ios_config:

backup: yes

backup_options:

dir_path: "{{ backup_dir }}"

filename: "pre_change_{{ inventory_hostname }}.cfg"

register: backup_info

Етап 2: Застосування змін (Симуляція критичної зміни)

- name: 2. Apply New ACL Configuration

cisco.ios.ios_config:

lines:

- ip access-list extended MGMT_ONLY

- permit tcp 192.168.122.0 0.0.0.255 any eq 22

- deny ip any any

match: strict

```

# Етап 3: Верифікація (Self-Test)
- name: 3. Verify Connectivity
  cisco.ios.ios_ping:
    dest: 192.168.122.1
    count: 3
  register: ping_result
# Якщо втрачено зв'язок зі шлюзом, викликаємо збій
  failed_when: ping_result.packet_loss > 0

# Етап 4: Аварійне відновлення (спрацьовує при помилці у Block)
rescue:
  - name: CRITICAL ERROR DETECTED! Initiating Rollback...
    debug:
      msg: "Configuration failed or connectivity lost. Restoring from {{
backup_info.backup_path }}"

  - name: 4. Restore Configuration from Backup
    cisco.ios.ios_config:
      src: "{{ backup_info.backup_path }}"
      replace_config: yes
      match: none
    vars:
      ansible_command_timeout: 120
  - name: Abort Playbook
    fail:
      msg: "Rollback completed. The system is restored to the previous state."
always:
  - name: Summary Report
    debug:
      msg: "Operation completed on {{ inventory_hostname }}"

```

Захист даних за допомогою Ansible Vault

Файл: group_vars/all/secrets.yml (Зашифрований вигляд)

\$ANSIBLE_VAULT;1.1;AES256

```

6638643534323632303033623766643238356534336261313463383038316262
3334653066316239353762646231643431616438336336663533663230373834
0a66393436653038616231363737303036333935636531363630656662366162
6566346261326435343465653664386462383236316538663264613238613437
650a316538356133333335343537386266316662663964343834633538326632
6236326162633065363435646665393231333838386337343666353164626139
3661

```

Розшифрований вміст (структура змінних):

Облікові дані для підключення

ansible_password: "ComplexPassword123!"

Пароль режиму Enable

ansible_become_password: "SuperSecretEnablePassword!"

Хеш пароля для нового адміністратора (sec_admin)

Створено командою: openssl passwd -salt xyz MySecretPass

vaulted_admin_password: "\$1\$xyz\$tI9V.0.0.0.0.0"

SNMP Community String

snmp_community_string: "D1pl0ma_Read_Only"

Схема та конфігурація процесу CI/CD

Файл: .gitlab-ci.yml

```
# Визначення етапів (Stages) конвеєра
stages:
  - lint          # Перевірка синтаксису
  - dry-run       # Симуляція змін
  - deploy        # Застосування у продакшн

variables:
  ANSIBLE_CONFIG: "ansible.cfg"
  INVENTORY: "inventory/hosts.yml"

# Етап 1: Статичний аналіз коду
syntax_check:
  stage: lint
  image: willhallonline/ansible:latest
  script:
    - echo "Running Syntax Check..."
    - ansible-playbook -i $INVENTORY playbooks/site.yml --
syntax-check
    - ansible-lint playbooks/*.yaml

# Етап 2: Симуляція (Check Mode)
simulate_changes:
  stage: dry-run
  image: willhallonline/ansible:latest
  script:
    - echo "Simulating configuration changes..."
```

```
- ansible-playbook -i $INVENTORY playbooks/site.yml --check
--diff
artifacts:
  paths:
    - ansible.log
tags:
  - runner-rocky-linux

# Етап 3: Розгортання (Ручне підтвердження)
deploy_production:
  stage: deploy
  image: willhallonline/ansible:latest
  script:
    - echo "Deploying Security Configuration to
192.168.122.0/24..."
    - ansible-playbook -i $INVENTORY playbooks/site.yml
  when: manual
  only:
    - master
  dependencies:
    - simulate_changes
```