НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

До захисту допущено
Завідувач кафедри

_____ **Дмитро ЛАНДЕ**
(підпис)

«_____»_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи, технології та математичні методи кібербезпеки»
спеціальності 125 «Кібербезпека»

на тему: Організаційний і технічний захист SSH серверів: підбір та автоматичне застосування налаштувань для підвищення безпеки

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи _ФБ-06__
(шифр групи)

Андреєв Данило Юрійович_____
(прізвище, ім'я, по батькові) (підпис)

Керівник к.ф-м.н., доцент кафедри ІБ, Смирнов Сергій Анатолійович___
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент к.ф-м.н., доцент кафедри ММЗІ Южакова Ганна Олексіївна___
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____
(підпис)

Київ – 2024 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 125 «Кібербезпека»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Дмитро ЛАНДЕ
(підпис)
«___»_____2024 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти

Андреєву Данилу Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Організаційний і технічний захист SSH серверів: підбір та автоматичне застосування налаштувань для підвищення безпеки_____,

керівник роботи
к.ф-м.н., доцент кафедри ІБ, Смирнов Сергій Анатолійович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від _31_ _травня_ 2024 р. №2251-с

2. Термін подання здобувачем вищої освіти роботи 13 _червня_ 2024 р.

3. Вихідні дані до роботи: алгоритм роботи протоколу SSH, особливості функціонування, наявні організаційні проблеми та технічні вразливості безпеки серверів, наявні атаки на протокол та сервери

4. Зміст роботи: дослідити роботу протоколу SSH та його сценаріїв використання; виконати аналіз поширених атак і проблем безпеки серверів SSH, а також впливу неправильних налаштувань сервера на захищеність; розробити організаційні рекомендації та програмний засіб для автоматичної конфігурації SSH серверів

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): Презентація на тему: «Організаційний і технічний захист SSH серверів: підбір та автоматичне застосування оптимальних налаштувань для підвищення безпеки»

6. Дата видачі завдання: 14 жовтня 2023 року

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Визначення тематики роботи	25.09.2023 - 10.10.2023	виконано
2	Отримання завдання	14.10.2023	виконано
3	Опрацювання літературних та інформаційних джерел	30.10.2023 - 15.11.2023	виконано
4	Дослідження роботи протоколу SSH	20.11.2023 - 10.12.2023	виконано
5	Виділення організаційних проблем захисту SSH серверів	15.01.2024 - 29.01.2024	виконано
6	Огляд проблем процесів управління SSH ключами	08.02.2024 - 20.02.2024	виконано
7	Аналіз технічних недоліків, що впливають на захист серверів	06.03.2024 - 19.03.2024	виконано
8	Огляд вразливостей з бази CVE	24.03.2024 - 10.04.2024	виконано
9	Створення організаційних рекомендацій	15.04.2024 - 30.04.2024	виконано
10	Виділення налаштувань SSH сервера, що підвищують рівень захищеності	03.05.2024 - 16.05.2024	виконано
11	Розробка скрипта для автоматичного налаштування SSH серверів	18.05.2024 - 30.05.2024	виконано
12	Оформлення дипломної роботи згідно вимог	02.06.2024 - 05.06.2024	виконано
13	Створення ілюстративного матеріалу	07.06.2024 - 12.06.2024	виконано
14	Передзахист роботи	13.06.2024	виконано

15	Захист роботи	24.06.2024	

Здобувач вищої освіти _____ __Данило Андреєв__
(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

Керівник роботи _____ __Сергій Смирнов__
(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

РЕФЕРАТ

Робота складається з трьох розділів, містить 63 сторінки (без додатків), 11 ілюстрацій, 2 таблиці, 2 додатки та 27 посилань на джерела.

Метою даної роботи є підвищення рівня захищеності SSH серверів у внутрішніх мережах організацій.

Методи дослідження включають в себе аналіз специфікацій протоколу SSH, дослідження організаційних та технічних проблем захисту SSH серверів, розгляд існуючих атак, яким піддаються сервери, створення на основі їх аналізу рекомендацій та засобів протидії.

Результатом роботи став набір організаційних рекомендацій щодо захисту SSH серверів, а також програмний засіб для автоматичної конфігурації серверів задля підвищення рівня їхньої захищеності.

Створені розробки можуть використовуватися системними адміністраторами організацій, що застосовують протокол SSH у своїх інформаційних або бізнес-процесах. Вони зможуть підвищити рівень безпеки SSH серверів та зробити зручнішим користування ними, при цьому не потребуючи значних вкладень фінансів або часу.

Ключові слова: SSH, захист SSH серверів, управління ключами, безпекові налаштування, комп’ютерні мережі

ABSTRACT

The work consists of three sections, contains 63 pages (without appendices), 11 illustrations, 2 tables, 2 appendices and 27 references to sources.

The purpose of this work is to increase the security level of SSH servers in internal networks of organizations.

Research methods include the analysis of SSH protocol specifications, the study of organizational and technical problems of protecting SSH servers, the consideration of existing attacks that servers are exposed to, the creation of recommendations and countermeasures based on their analysis.

The result of the work was a set of organizational recommendations for the protection of SSH servers, as well as a software tool for automatic servers configuration to increase their security level.

The created developments can be used by system administrators of organizations that use the SSH protocol in their information or business processes. They will be able to increase the level of security of SSH servers and make them more convenient to use, while not requiring a significant investment of money or time.

Keywords: SSH, protection of SSH servers, key management, security settings, computer networks

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ..... 8
ВСТУП.....9
1 ОГЛЯД ПРОТОКОЛУ SSH, ЙОГО МОЖЛИВОСТЕЙ ТА РЕАЛІЗАЦІЙ.....11
1.1 Основні можливості SSH.....13
1.2 Порівняння різних версій протоколу SSH.....16
1.3 SSH рукостискання.....18
1.4 Криптографічні алгоритми в SSH.....20
Висновки до розділу 1.....26
2 ОРГАНІЗАЦІЙНІ ТА ТЕХНІЧНІ ПРОБЛЕМИ ВИКОРИСТАННЯ SSH..... 29
2.1 Організаційні проблеми роботи з SSH..... 30
2.2 Технічні проблеми SSH протоколу..... 33
2.3 Аналіз реальних CVE вразливостей..... 40
Висновки до розділу 2.....46
3 РОЗРОБКА ОРГАНІЗАЦІЙНИХ РЕКОМЕНДАЦІЙ ТА ПРОГРАМНОГО ЗАСТОСУНКУ ДЛЯ ЗАХИСТУ SSH СЕРВЕРІВ..... 49
3.1 Організаційні рекомендації.....50
3.2 Технічні засоби захисту SSH серверів.....57
3.3 Програмна реалізація автоматичного конфігуратора сервера.....59
Висновки до розділу 3.....62
ВИСНОВКИ..... 64
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ..... 66
ДОДАТОК А.....69
ДОДАТОК Б..... 85

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

SSH — безпечна оболонка

FTP — протокол передавання файлів

SFTP — протокол безпечного передавання файлів

MitM — атака «людина посередині»

CVE — база даних загальновідомих вразливостей інформаційної безпеки

DES — симетричний алгоритм шифрування

3DES — блоковий шифр, що тричі застосовує алгоритм DES

RSA — криптографічний алгоритм з відкритим ключем

MAC — спеціальний набір символів, призначений для забезпечення цілісності повідомлення та автентифікації джерела

HMAC — код автентифікації повідомлень на основі гешування

DoS — атака на відмову в обслуговуванні

DDoS — розподілена атака на відмову в обслуговуванні

MFA — багатофакторна автентифікація

MD5 — алгоритм гешування

ВСТУП

Протокол SSH — це поширений інструмент, що використовується для віддаленого адміністрування та управління системами в мережі. Між клієнтом та сервером встановлюється захищене за допомогою криптографії з’єднання, за допомогою якого можна безпечно надсилати команди, файли або тунелювати трафік інших мережних протоколів. SSH прийшов на зміну протоколу Telnet, який є небезпечними через відсутність перевірок автентичності та передачу інформації у відкритому вигляді.

Проте SSH сервери часто зазнають атак різноманітного характеру через цінність даних, що на них знаходяться або можливостей, які вони можуть надати зловмисникам в разі компрометації. Тому захисту таких серверів в мережі організації необхідно приділяти особливу увагу як з організаційної, так і з технічної точки зору. Щоб розробити ефективні засоби забезпечення безпеки SSH серверів, потрібно помірковано поєднувати загальновідомі практики із специфікою роботи конкретної організації та її внутрішньої мережі.

Актуальність роботи: кібератаки на SSH сервери організації можуть мати значні наслідки, від втрати даних до зупинки критичних бізнес-процесів. Впровадження ефективних практик захисту таких серверів значно підвищить загальний рівень безпеки комп’ютерної мережі організації. Саме тому створення засобів захисту SSH серверів від наявних загроз є актуальною та корисною задачею.

Мета дослідження: забезпечення високого рівня захищеності SSH серверів в мережах організацій

Мета досягається за умови вирішення таких завдань:

1. Аналіз актуальних атак та проблем використання протоколу SSH.
2. Виділення технічних вразливостей та недоліків організаційних заходів на основі проведеного аналізу.
3. Вивчення інформаційних процесів та сценаріїв використання SSH в організації.

4. Створення списку контрзаходів з урахуванням отриманих даних.
5. Розробка організаційних рекомендацій з роботи з SSH.
6. Реалізація програмного рішення для реалізації зазначених заходів безпеки.

Об'єкт дослідження: протокол SSH та його реалізації для клієнтської та серверної частини.

Предмет дослідження: атаки на SSH сервери, вразливості протоколу, проблеми організації роботи з ним, зокрема — проблема управління ключами доступу до серверів.

Методи дослідження**:** аналіз специфікації протоколу SSH, списку знайдених вразливостей в базі даних CVE, документації для клієнтських та серверних реалізацій SSH, тестування на лабораторних машинах.

Новизна одержаних результатів: створення засобу автоматизації для конфігурування SSH серверів, що включав би найкращі налаштування безпеки, обрані на основі аналізу актуальної моделі загроз для SSH вперше надає даній роботі елемент наукової новизни.

Практичне значення одержаних результатів: програма для швидкого автоматизованого налаштування SSH серверів значно полегшить роботу системного адміністратора мережі та покращить застосування SSH в організації.

Апробація результатів роботи: результати роботи апробовано під час XXII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики» у вигляді доповіді.

Публікації: результати роботи було опубліковано у збірнику XXII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики».

1 ОГЛЯД ПРОТОКОЛУ SSH, ЙОГО МОЖЛИВОСТЕЙ ТА РЕАЛІЗАЦІЙ

Протокол SSH являє собою програмне рішення для забезпечення безпеки в комп'ютерних мережах. В процесі своєї роботи він виконує прозоре шифрування даних, тобто виконує шифрування та розшифрування непомітно для сторін. Даний протокол має архітектуру клієнт-сервер. На машині запускається програма-сервер, що може приймати SSH з'єднання, які користувачі роблять за допомогою SSH клієнтів та виконують необхідні їм дії (виконання команди, пересилання файлу тощо).

Протокол надає наступні функції [1]:

1. Конфіденційність даних, яка забезпечується за допомогою криптографії. SSH шифрує інформацію, що передається в мережі, використовуючи ключі сесії, які узгоджуються між сторонами під час процедури SSH рукостискання (SSH handshake). Протокол підтримує велику кількість алгоритмів шифрування, таких як DES, 3DES, Blowfish тощо, проте не всі підтримувані шифри є рекомендованими до використання.
2. Цілісність повідомлень, унеможливлення їх несанкціонованого змінення. Механізми забезпечення цілісності, що вбудовані у протоколи стеку TCP/IP, не можуть протистояти цілеспрямованим діям хакерів, наприклад, атаці повторного відтворення. SSH другої версії виконує перевірку цілісності за допомогою криптографії, яка не тільки забезпечує передачу даних у незмінному вигляді, але й перевіряє, що дані дійсно надходять від очікуваного відправника.
3. Автентифікація як відправника, так і одержувача. Перевірка сервера захищає SSH клієнта від перенаправлення з'єднання на іншу машину (сервер зловмисника), а також від атаки man-in-the-middle. Автентифікація клієнта відбувається за допомогою введення пароля або надання серверу згенерованого публічного ключа. На сервері такі ключі зберігаються у директорії .ssh/authorized_keys. Паролі вважаються ненадійними через ймовірність їх викрадення, перехоплення або навмисне використання

користувачами слабких паролів задля зручності. Такі паролі хакер зможе досить легко підібрати, використовуючи атаку за словником або brute-force (атаку грубої сили), якщо сервер не має засобів для протистояння таким атакам.

4. Тунелювання роботи сесій, заснованих на TCP/IP (шифрування трафіку інших протоколів). Тунелювання трафіку інших протоколів та сервісів дозволяє використовувати переваги SSH для підвищення рівня їхньої безпеки. Існує 3 види такого тунелювання: звичайне TCP тунелювання, X тунелювання та тунелювання агентів.
5. Авторизація, контроль доступу за допомогою облікових записів користувачів. SSH надає широкі можливості налаштування доступу до своїх функцій для певних користувачів. Авторизація може відбуватися як на стороні сервера (у конфігураційному файлі /etc/sshd_config), так і на стороні клієнта (у файлі авторизації для кожного користувача, наприклад, у ~/.ssh2/authorization). Це дозволяє пристосовувати сервер для потреб конкретної мережі.

Важливою відмінністю SSH від інших протоколів тунелювання трафіку (наприклад IPsec) є рівень моделі OSI, на якому він працює. IPsec знаходиться на мережному рівні і може працювати лише з IP-адресами машин і не має доступу до портів, які є частиною транспортного рівня OSI, тоді як SSH виконує свої функції на сьомому — рівні додатків (application layer). SSH може тунелювати дані лише одного застосунку одночасно, тоді як IPsec захищає весь трафік, що проходить у мережі. Також відмінність полягає у протоколі транспортного рівня, що використовується. SSH застосовує TCP, тоді як IPsec — лише UDP. Тому хоч і SSH тунеі повільніші, вони забезпечують кращу цілісність та виключають можливість втрати пакетів під час передачі мережею. До того ж, IPsec вимагає більшої кількості налаштувань ніж SSH через рівень OSI, на якому він працює, тому що кожна пара з’єднань між хостами або між мережами має бути налаштована окремо.

1.1 Основні можливості SSH

SSH протокол надає широкий список можливостей та засобів, що дозволяють використовувати його в багатьох інформаційних процесах організації. Можливість тунелювати трафік інших протоколів робить SSH гнучким та надійним інструментом захисту. Функції, що пропонує даний протокол [2]:

1. Віддалений вхід та виконання команд. SSH дозволяє входити в систему та виконувати команди на віддалених машинах. Це дуже зручно для системних адміністраторів, адже їм не потрібно мати фізичного доступу до машини, до якої вони бажають отримати доступ. Більш того, можлива навіть автоматизація виконання певних завдань на віддалених комп'ютерах з використанням bash або Python скриптів. Схожі можливості надають утиліти rlogin та rsh, проте їх використання є небезпечним. Ці команди передають дані (в тому числі логіни та паролі) у вигляді звичайного тексту, що дозволяє зловмиснику легко їх перехопити, використовуючи, наприклад, сніфери. Крім того, ці утиліти використовують файли .rhosts та /etc/hosts.equiv, які дають змогу заходити до системи без пароля, що може бути використано зловмисниками. Саме через наявність криптографічних засобів захисту та аналогічних функцій SSH повністю витіснив ці команди із загального використання.
2. Передача файлів. Аналогічно до того, як SSH надає більш безпечну альтернативу утилітам rlogin та rsh, те саме він робить для протоколу FTP та команди rcp. Знову ж таки, вони передають файли в незашифрованому вигляді, що дає змогу їх перехопити. Протокол SSH надає команду scp (secure copy), яка шифрує файл перед відправленням та розшифровує перед отриманням, що вирішує дану проблему безпеки.
3. Тунелювання. SSH може тунелювати трафік інших мережних протоколів, підвищуючи рівень захищеності. Цей механізм також часто називають port forwarding. Суть полягає у перенаправленні з'єднання TCP на порт SSH.

Далі дані пересилаються через безпечне SSH з'єднання на сервер, звідки передаються початковому отримувачу (Рисунок 1.1). Тунелювання дозволяє також обходити мережні екрани. Наприклад, якщо фаерволи налаштовані забороняти весь трафік на порт 80, але при цьому дозволяють передавати SSH трафік, можемо використати port forwarding для подолання цієї заборони та надіслати/отримати потрібні дані. Варто зазначити, що цією технікою можуть також користуватися зловмисники для обходу обмежень та отримання доступу до внутрішньої мережі організації.

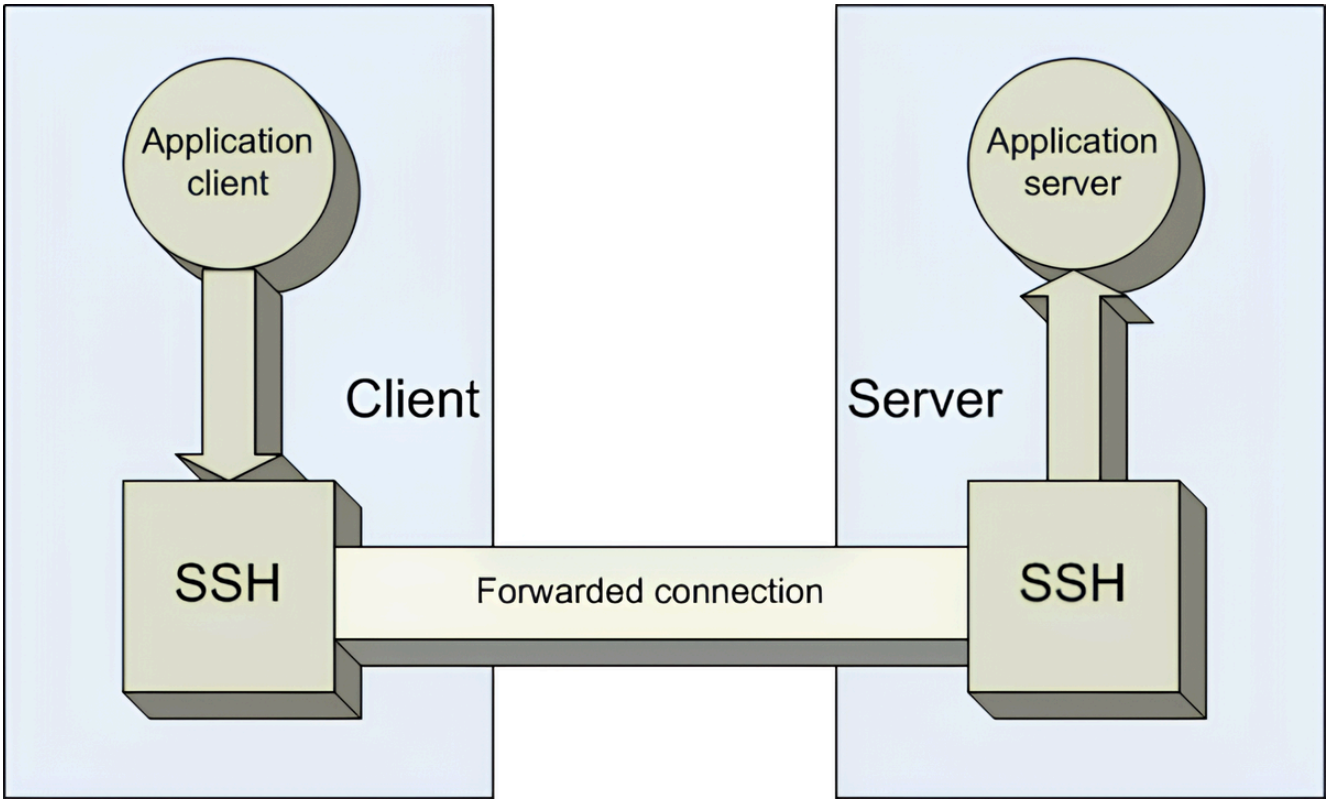

Рисунок 1.1 - Схема тунелювання трафіку за допомогою SSH

4. Тунелювання X11 трафіку. Цей пункт можна було б віднести до попереднього, адже він також стосується тунелювання трафіку, проте спосіб використання значно відрізняється, що вимагає виділити його окремо. X протокол (або X11) використовується у системах, заснованих на Unix для роботи програм з графічним інтерфейсом користувача. Його особливість полягає в розділенні системи на якій програма працює та системи, на якій вона відображається. Тому ми можемо запускати програму на віддаленій

машині, а результати її роботи відображувати локально. Але дані передаються у відкритому вигляді, що створює загрози безпеки. До того ж, такий механізм потребує наявності довіри між хостами, що беруть в ньому участь. Обидві ці проблеми може вирішити тунелювання трафіку за допомогою протоколу SSH. Для цього достатньо лише внети відповідне налаштування до конфігураційного файлу сервера та встановити з'єднання з атрибутом -X. Крім того, можливе застосування окремих налаштувань для різних хостів або навіть для окремих сесій. На рахунок Wayland, нативно він не підтримується в SSH, проте аналогічна робота з ним також можлива через додаткові утиліти (waypipe або xwayland).

Важливою особливістю протоколу SSH, якій буде присвячена частина цієї роботи, є можливість автентифікації за допомогою ключа. Використовуючи ключ, SSH клієнт може отримувати доступ до SSH серверів без необхідності вводити пароль для кожного з них. Щоб налаштувати такий доступ, достатньо згенерувати пару ключів (приватний та публічний) на клієнті за допомогою команди ssh-keygen, і додати публічний ключ на сервер використовуючи команду ssh-copy-id. При наступних підключеннях сервер буде виконувати автентифікацію за ключем і не потребуватиме введення пароля (Рисунок 1.2).

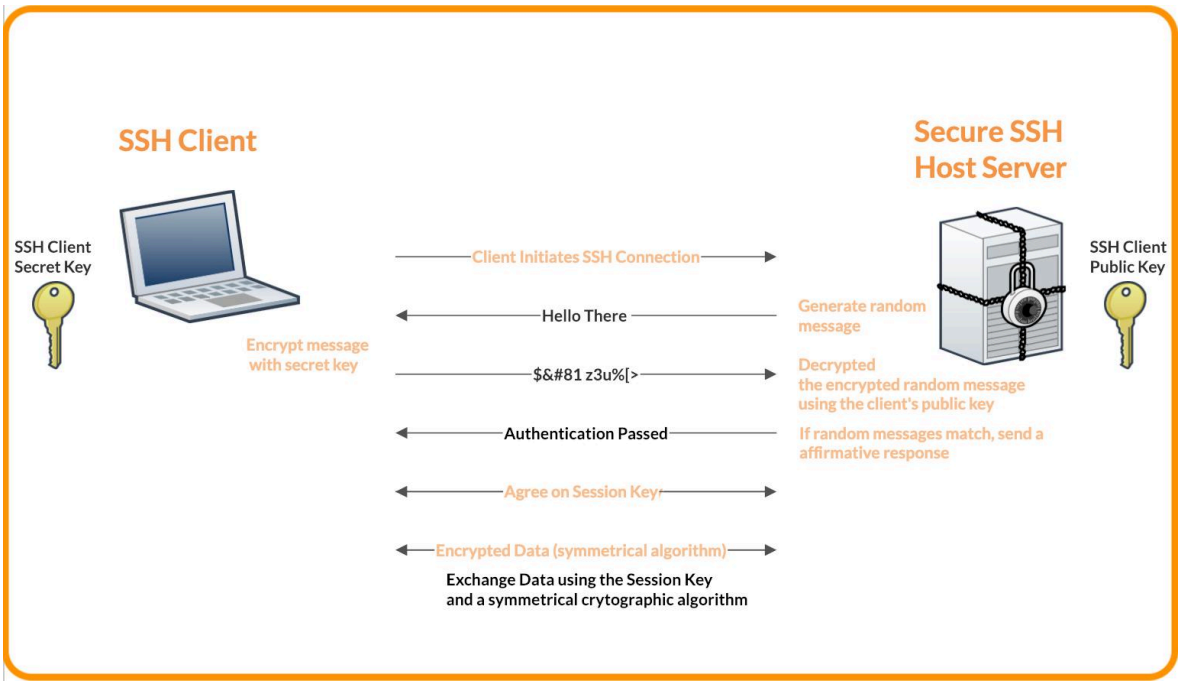

Рисунок 1.2 - Схема SSH автентифікації за ключем

Поширеною і рекомендованою практикою є повне відключення можливості входу на сервер за паролем [3]. Таким чином виключається ймовірність несанкціонованого доступу до сервера через атаки на паролі (brute force або атака за словником). Проте такий підхід має свої недоліки (хоч і не такі критичні, як у парольної автентифікації), про які мова піде у наступних розділах цієї роботи.

1.2 Порівняння різних версій протоколу SSH

Основною сучасною версією SSH протоколу, що використовується сьогодні, є SSH-2, випущений ще в 2006 році. Він містить велику кількість покращень з точок зору безпеки, зручності використання та швидкості в порівнянні з SSH-1. Через свої численні слабкості, а саме вразливість до атаки man-in-the-middle, слабкі криптографічні алгоритми та недостатньої перевірки цілісності, SSH-1 вважається застарілим і не рекомендується до використання. Проте він є цікавим для аналізу з точки зору покращень, які були внесені до наступної версії. Проаналізувавши зміни в протоколі SSH-2 в порівнянні з SSH-1, а також в порівнянні з представленим нещодавно SSH-3 можна зробити висновки про те, з якими проблемами може зіткнутися організація при роботі з поточною версією протоколу другої версії. Порівняння протоколів SSH-1 та SSH-2 [4]:

1. Вибір алгоритмів. У протоколі SSH-1 клієнт та сервер могли узгоджувати лише алгоритм шифрування, тоді як інші криптографічні аспекти було записано прямо у коді. В SSH-2 можуть узгоджуватися все необхідні алгоритми: шифрування, гешування, обмін ключами та стиснення даних.
2. Підтримувані алгоритми. SSH-1 використовує криптографічні алгоритми, що вважаються застарілими і можуть бути зламані з використанням помірної кількості обчислювальних ресурсів та часу, наприклад blowfish, RSA, 3DES. У наступній версії більшу частину цих алгоритмів було прибрано та замінено на більш сучасні та надійні, такі як aes-ctr, chacha-poly тощо.

3. Сесійний ключ. SSH-2 може змінювати сесійний ключ в межах однієї сесії, тоді як в SSH-1 всю сесію використовується той самий ключ.
4. Перевірка цілісності. В SSH-1 для забезпечення цілісності даних використовувався лише один механізм — CRC-32. Його наступник дає сторонам змогу узгоджувати алгоритм і застосовує сильніші криптографічні перевірки цілісності такі як MAC.
5. Передача файлів. Обидві версії протоколу підтримують базовий scp, проте лише SSH-2 дозволяє працювати з SFTP, який має ширші можливості та функції.
6. Архітектура. SSH-1 є монолітним, а SSH-2 розподілений на модулі, що взаємодіють між собою: SSH-TRANS, SSH-AUTH та SSH-CONN.
7. Сертифікати. У SSH-2 з'явилася можливість підписувати публічні ключі користувачів, використовуючи центр сертифікації (CA) що спрощує управління ними, дозволяє швидко їх створювати та відкликати.
8. Автентифікація. SSH-1 дозволяє лише один метод автентифікації для однієї сесії, тоді як друга версія дозволяє застосовувати декілька механізмів для отримання доступу до сервера.

SSH-3 на даний момент широко не використовується, проте вже представлена його перша реалізація і ведеться його активна розробка. Дана версія протоколу суттєво відрізняється від попередників, використовує найсучасніші мережеві технології та показує себе дуже добре [5]. Перша важлива відмінність — робота через HTTP/3. SSH-3 слухає на 443 порті замість стандартного 22, його трафік стає схожим на звичайний HTTP трафік. Це знижує ефективність мережного сканування з боку зловмисників на наявність SSH серверів і підвищує рівень захищеності мережі.

До того ж, SSH-3 може швидше встановлювати з'єднання, тунелювати протоколи, що використовують UDP (DNS, RTP), автентифікувати сервер за допомогою X.509 сертифікатів. З цього можна зробити висновок, що SSH-3 вирішує більшість проблем попередньої версії «з коробки». Проте зараз він знаходиться в стадії активної розробки і говорити про його впровадження у

мережі великих організацій поки що рано. Тому задача підвищення безпеки протоколу SSH-2 залишається актуальною.

1.3 SSH рукостискання

Процес встановлення з'єднання між клієнтом та сервером, що називається SSH рукостисканням (або SSH handshake), є одним з найважливіших з точки зору безпеки протоколу. На цьому етапі ще не встановлено захищеного криптографією каналу зв'язку, тому відкривається можливість для проведення атак на протокол. Тому необхідно розглянути процедуру рукостискання між клієнтом та сервером більш детально [6]. З'єднання між учасниками встановлюється у декілька етапів (Рисунок 1.3):

1. Клієнт за допомогою TCP надсилає серверу запит на з'єднання (зазвичай, на 22 порт — стандартний для SSH).
2. Клієнт та сервер обмінюються інформацією про те, яку версію SSH підтримує кожен з них. Якщо сторони вважають версії несумісними, процес з'єднання переривається.
3. Сторони обмінюються даними щодо криптографічних алгоритмів, які вони підтримують та порядку, в якому вони б хотіли їх використовувати.
4. На основі обраного алгоритму обміну ключами клієнт та сервер створюють сесійний ключ. Цей ключ використовуватиметься обидвома сторонами для шифрування даних у захищеному каналі SSH з'єднання.
5. Верифікація сервера. Клієнт перевіряє ключ хоста сервера. Якщо даний ключ знаходиться у файлі known_hosts, рукостискання продовжується. Якщо ні — клієнтська програма запитує у користувача, чи хоче він довіряти даному серверу. Якщо відповідь негативна — з'єднання переривається. Цей крок запобігає атакам man-in-the-middle.

6. Сторони вмикають шифрування та завершують процес автентифікації сервера. Клієнт очікує від сервера підтвердження, що зашифроване новоствореним сесійним ключем.
7. Безпечне з’єднання вважається встановленим.
8. Після цього відбувається автентифікація клієнта: за допомогою пароля або SSH ключа. Для цього клієнт надсилає серверу повідомлення SERVICE_REQUEST.

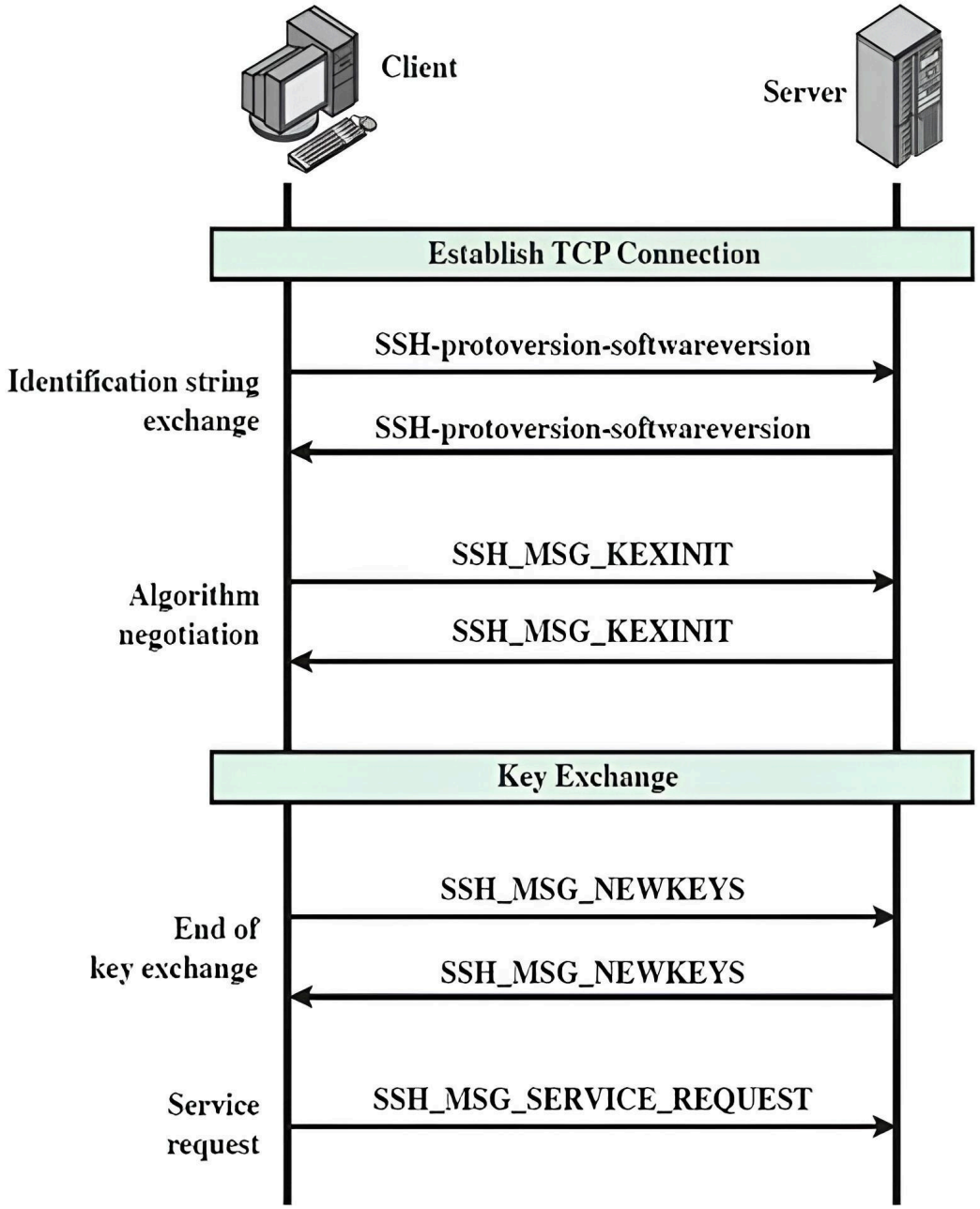

Рисунок 1.3 - Схема SSH рукостискання

1.4 Криптографічні алгоритми в SSH

Як було зазначено вище, SSH виконує свої функції за допомогою криптографії. Застосовуючи алгоритми обміну ключами, шифрування та гешування, протокол може забезпечувати цілісність та конфіденційність каналу зв'язку у мережі. Важливим фактом є те, що не всі алгоритми, які знаходяться в специфікації протоколу вважаються безпечними і застосовуються на практиці. Різні реалізації SSH серверів та клієнтів мають деякі відмінності у своїх списках алгоритмів. Часто слабкі або ненадійні функції видаляються або замінюються на сучасні, більш безпечні варіанти. Щоб зрозуміти, які алгоритми є оптимальними для забезпечення максимального рівня безпеки, необхідно проаналізувати їх з точки зору використання в SSH [7]. В якості імплементації розглядатимемо OpenSSH [8], оскільки він включає в себе як клієнт, так і сервер і є одним з найпопулярніших рішень на практиці.

1.4.1 Алгоритми з відкритим ключем

DSA. Цей алгоритм використовується в SSH для автентифікації. Клієнт може згенерувати ключ за допомогою DSA та додати його на SSH сервер. Далі, коли клієнт надсилає запит на автентифікацію, сервер надсилає повідомлення, яке клієнту потрібно підписати своїм приватним ключем. Далі сервер перевіряє підпис за допомогою публічного ключа, який він має та на основі цієї перевірки приймає рішення про надання доступу. Проте через надто малу довжину ключа цей алгоритм було замінено на RSA разом з SHA-2 гешуванням.

RSA. На відміну від DSA, цей алгоритм також може застосовуватись для шифрування. Крім того, довжина ключа в нього набагато більша, проте процес генерації займає більше часу, ніж для DSA. Цей алгоритм є одним з

найпопулярніших через свою надійність та можливість його використання на більшості SSH серверів та клієнтів.

ECDSA. Алгоритм цифрового підпису, що застосовує еліптичні криві. Він є набагато швидшим та ефективнішим за RSA, що важливо у випадках, коли потужності в середовищі є обмеженими. До того ж, алгоритм забезпечує високий рівень безпеки навіть при використанні відносно малої довжини ключа. Наприклад, 256-бітний ключ по рівню захищеності співставний з 3072-бітним ключем RSA.

1.4.2 Алгоритми забезпечення цілісності.

HMAC. Використовується для автентифікації повідомлень, що передаються захищеним каналом. По суті являє собою покращену версію MAC, включає в себе механізм з секретним ключем та геш-функцію (може уточнюватись у назві: HMAC-MD5, HMAC-SHA512 тощо). Коли SSH шифрує дані, він також обраховує HMAC повідомлення та додає його до шифрованого тексту. Далі отримувач також обраховує значення HMAC та порівнює їх, на основі чого приймає рішення про цілісність отриманих даних. Даний алгоритм забезпечує незмінність даних в разі використання з сильним алгоритмами гешування, такими як SHA-256 та SHA-512.

GMAC. В SSH протоколі GMAC застосовується в комбінації із симетричними алгоритмами, наприклад AES-GCM. Основною перевагою даного алгоритму є висока ефективність та швидкість, в зв’язку з чим часто саме йому надається перевага. Процес перевірки цілісності повідомлення аналогічний до HMAC.

1.4.3 Симетричні алгоритми шифрування

AES. Спільний сесійний ключ для роботи цього алгоритму створюється на основі спільного секрету, який обраховується під час механізму обміну ключами (наприклад, алгоритм Діффі-Хеллмана або ECDH). Дані можуть шифруватися в одному з декількох режимів: CBC (Cipher Block Chaining) та GCM (Galois/Counter Mode). Режим CBC забезпечує конфіденційність, проте потребує додатковий механізм перевірки цілісності (наприклад, HMAC), що ускладнює його. GCM одночасно забезпечує конфіденційність та цілісність (використовується GMAC), що спрощує процес шифрування. Незалежно від обраного режиму дані розбиваються на блоки, які шифруються за допомогою AES.

ChaCha20. Потоковий шифр, що застосовується для шифрування повідомлень у захищеному каналі SSH. Зазвичай використовується в парі механізмом забезпечення цілісності повідомлень Poly1305. Тут також використовується ключ сесії, створений на основі спільного секрету після завершення алгоритму обміну ключами. Далі на базі ключа створюється псевдовипадкова ключова послідовність. Після цього над даними, що передаються та ключовою послідовністю виконується операція XOR. Перевірка цілісності за допомогою Poly1305 виконується аналогічно до HMAC та GMAC. Серед переваг ChaCha20 можна виділити простоту, швидкість та ефективність. До того ж, шифр стійкий до багатьох криптографічних атак, такий як brute force, атак дня народження, timing атак.

1.4.4 Алгоритми обміну ключами

Протокол Діффі-Хеллмана. Застосовується для створення спільного секрету між сторонами у незахищеному середовищі під час SSH рукостискання.

Далі цей секрет використовується для утворення сесійного ключа за допомогою функції формування ключа (key derivation function), яка може генерувати секретні ключі на основі певного секретного значення, застосовуючи псевдовипадкову функцію (наприклад, геш-функцію). Схему роботи даного алгоритму наведено на Рисунку 1.5.

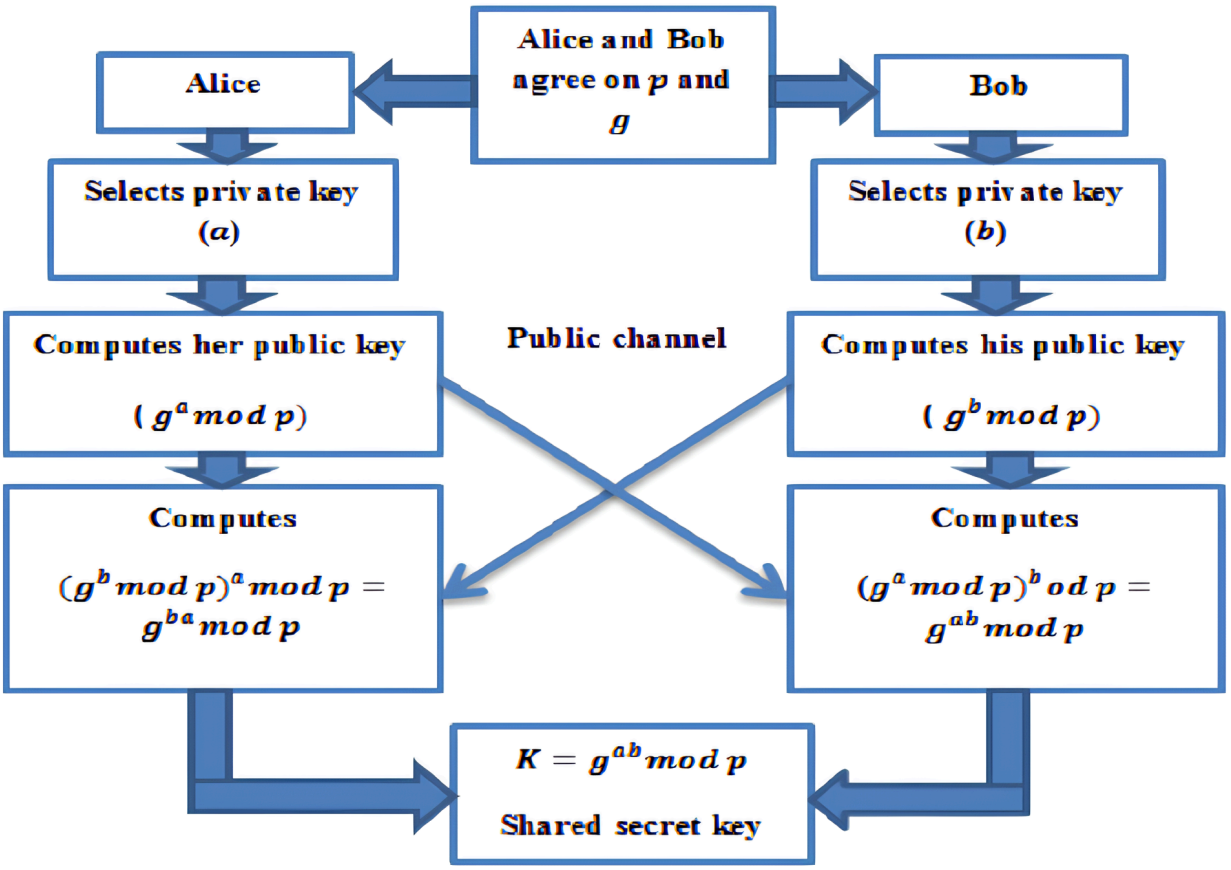

Рисунок 1.4 - Блокова діаграма обміну ключами за протоколом Діффі-Хеллмана

ECDH. Варіація протоколу Діффі-Хеллмана з використанням еліптичної криптографії. В зв’язку з цим змінюються принципи виконання алгоритму, але результат його роботи залишається незмінним. Через застосування еліптичних кривих даний протокол має кілька переваг: швидкість роботи, високий рівень захищеності при відносно малих довжинах ключа. Тому на сьогоднішній день під час SSH комунікації перевага надається саме ECDH. З мінусів можна виділити складність реалізації такого алгоритму.

1.4.5 Вплив криптографії на безпеку SSH

В зв’язку з тим, що вся робота SSH протоколу базується на використанні криптографічних механізмів, вкрай важливим є рівні конфіденційності та цілісності, які ці алгоритми надають. Необхідно застосовувати лише сучасні алгоритми для шифрування даних, що передаються мережею.

В SSH є можливість налаштовувати протоколи, що застосовуються для роботи. Така конфігурація можлива як для сервера (у файлі sshd_config), так і для клієнта (у файлі ssh_config) [9]. В окремих реалізаціях SSH часто вже встановлено шифри для забезпечення максимального рівня безпеки. Проте варто постійно здійснювати моніторинг актуальних атак на SSH сервери та змінювати список алгоритмів в разі необхідності. Також необхідно брати до уваги швидкодію криптографічних алгоритмів в залежності від потреб мережі та інформаційних процесів організації, що розглядається.

Крім того, навіть при використанні найкращих криптографічних протоколів існує можливість зламу такого шифру зловмисниками. Причинами цього можуть стати наступні проблеми:

1. Компрометація приватного ключа. Компрометація приватного ключа є важливою проблемою для будь-якого криптографічного алгоритму. Приватні ключі використовуються та становлять собою основу безпечного SSH з'єднання. Якщо такий ключ буде викрадено зловмисником, це призведе до повної компрометації з'єднання. Така ситуація може відбутися через незахищене зберігання ключа, некоректні налаштування доступу.
2. Мала довжина ключа. Мала довжина криптографічних ключів може суттєво знизити рівень безпеки. Короткі ключі більш вразливі до brute force атак і можуть бути знайдені сучасними методами та обчислювальними потужностями. Рекомендовано використовувати ключі з довжиною не менше 3072 біт для RSA або аналогічні за рівнем безпеки ключі для інших алгоритмів (наприклад, 256-бітні ключі для ECDSA) [10]. Довші ключі (в залежності від алгоритму) забезпечують вищий рівень безпеки.

3. Вразливості в реалізації клієнта або сервера. Проблеми можуть міститися також в програмних імплементаціях клієнтської або серверної частини (OpenSSH, PuTTY тощо). До цього пункту можна віднести недоліки в коді, некоректне управління пам'яттю, логічні помилки в алгоритмах автентифікації або шифрування.

Висновки до розділу 1

В даному розділі було наведено основні теоретичні відомості, що стосуються протоколу SSH та його використання. Розглянуто основні механізми, що використовуються в SSH для забезпечення конфіденційності та цілісності даних, які передаються захищеним каналом.

Також досліджено процес SSH рукостискання, який є одним з найважливіших при захисті SSH серверів, адже під час нього ще не створено захищений канал для обміну інформацією і зловмисник може перехоплювати або змінювати дані при виконанні атаки man-in-the-middle.

Розглянуто ключові відмінності версій SSH-1 та SSH-2. Дане порівняння дозволить зрозуміти, які аспекти роботи протоколу є пріоритетними в процесі забезпечення його безпеки. Цей аналіз допоможе виділити основні аспекти забезпечення безпеки SSH серверів і реалізувати їх у власному програмному рішенні.

Крім того, проаналізовано механізми шифрування, гешування, та обміну ключами. Розуміння сильних та слабких сторін криптографічних алгоритмів дозволить враховувати їх при побудові системи для захисту SSH серверів та адаптувати її до потреб організації.

На ринку існує низка рішень, що спрямовані на підвищення захищеності SSH серверів та інших систем:

1. SSHGuard. Даний інструмент захищає SSH та інші служби від brute force атак, блокуючи IP-адреси, з яких вони відбуваються.

2. OpenSCAP. Утиліта, що використовується для сканування системи на предмет її відповідності певним наборам правил та політик, визначеним завчасно. Після цього OpenSCAP надає звіт, на основі якого до системи можна внести зміни налаштувань, що підвищити загальний рівень її захищеності.
3. Bastille Linux. Цей програмний знижує ймовірність компрометації за допомогою проактивної конфігурації системи, тобто впровадження більш жорстких та безпечних налаштувань.
4. Lynis. Аналог OpenSCAP, має схожі функції, проте дозволяє також використовувати кастомні плагіни та перевірки.

Можна побачити, що ті інструменти, які існують зараз і які можна використати для підбору безпечних налаштувань, мають недоліки, що можуть змусити системного адміністратора відмовитися від їхнього використання. Вони або спрямовані саме на мережну безпеку та роботу з мережними екранами і не враховують специфіки протоколу SSH, або виконують загальний аудит системи, що змушує адміністратора вручну аналізувати знайдені проблеми, підбирати методи їх вирішення та застосовувати їх. Крім того, такі утиліти мають занадто широкий спектр призначення і ускладнюють роботу, якщо користувача або адміністратора цікавить лише безпека SSH серверів. До того ж, наведені рішення призначені для застосування досвідченими фахівцями, що мають відповідні знання і не підходять для звичайних користувачів, які хочуть налаштувати власний SSH сервер для персонального використання. Також, подібні рішення враховують лише технічну сторону питання, не беручи до уваги організаційні проблеми, такі як політики паролів, оновлення програмного забезпечення та управління SSH ключами.

В зв’язку з цим, створення програмного рішення для захисту саме SSH серверів, яке б повною мірою враховувало його сильні та слабкі сторони, а також рекомендацій організаційного характеру, є актуальною задачею та потребує подальшої роботи.

Загалом, глибокий аналіз засад роботи SSH протоколу дозволить комплексно підійти та ґрунтовно підійти до створення організаційних та технічних рекомендацій щодо підвищення рівня захищеності SSH серверів у внутрішній мережі компанії. Такий підхід є ефективним, бо дозволяє зосередити увагу лише на критичних механізмах та процесах та не витрачати час на аспекти, які слабко впливають на безпеку протоколу.

В наступних розділах буде виділено актуальні організаційні та технічні проблеми роботи з SSH серверами, проаналізовано сучасні атаки на SSH та запропоновано відповідні рішення, що дозволять їх уникнути.

2 ОРГАНІЗАЦІЙНІ ТА ТЕХНІЧНІ ПРОБЛЕМИ ВИКОРИСТАННЯ SSH

Через свій широкий спектр можливостей, SSH протокол використовується системними та мережними адміністраторами для організації централізованого та ефективного керування своїми ресурсами. Завдяки криптографії SSH дозволяє створювати з'єднання, захищені від прослуховування, а застосування тунелювання може убезпечити мало не весь трафік, що проходить у мережі. Проте використання SSH протоколу та серверів також становити небезпеку для мережі з організаційних або технічних причин. В зв'язку з цим компаніям необхідно правильно ідентифікувати та оцінити ризики, пов'язані з SSH та на основі цих даних вжити відповідних заходів для їх уникнення або пом'якшення.

До організаційних проблем використання SSH можна віднести хибне або недостатнє управління SSH ключами для доступу до серверів, а також відсутність політики оновлення програмного забезпечення або її неправильність. Часто цим питанням не приділяється достатньо уваги, проте вони можуть становити таку саму небезпеку як, наприклад, brute force атаки і призвести до компрометації SSH сервера.

Технічний захист має вирішальне значення у забезпеченні захищеності серверів. Так як даний протокол використовується для організації віддаленого доступу, SSH сервери часто піддаються атакам мережного сканування, man-in-the-middle та DDoS атакам. Але одна з найважливіших проблем — це неправильні налаштування. Слабка конфігурація сервера (дозвіл root користувача, парольної автентифікації, увімкнення слабких криптографічних алгоритмів) дозволить зловмиснику отримати доступ до SSH сервера, а далі за допомогою тунелювання обійти фаерволи і скомпрометувати інші частини мережі.

В даному розділі буде детально розглянуто організаційні та технічні небезпеки, пов'язані з SSH, а також основні види атак, що здійснюються на такі сервери. Крім того, будуть проаналізовані реальні знайдені вразливості SSH протоколу та його імплементацій, що були занесені до бази даних CVE [11], будуть виокремлені фактори, що зробили ці атаки можливими, а також наслідки в

разі їхньої успішної реалізації. Наприклад, буде досліджено відому атаку, яку було знайдено у 2023 році — Terrapin Attack, що використовує вразливість у механізмі SSH рукостискання для зниження рівня захищеності SSH з'єднання. На основі даних, отриманих за допомогою такого аналізу, будуть створені відповідні рекомендації та програмне рішення (скрипт) для забезпечення безпеки, швидкого та правильного налаштування SSH серверів у мережі організації.

2.1 Організаційні проблеми роботи з SSH

Організаційні проблеми, пов'язані з SSH — це недоліки процесів роботи з протоколом та серверами (використання, управління тощо). Сюди можна віднести неправильне поводження з ключами, відсутність контролю за генерацією та розповсюдженням ключів, недотримання принципу мінімальних повноважень, відсутність або недостатність моніторингу тощо. Розглянемо кожне з питань задля виділення та аналізу основних моментів та їх урахування під час створення організаційних рекомендацій.

2.1.1 Управління ключами

Однією з основних організаційних проблем роботи з SSH протоколом є використання та управління SSH ключами, що застосовується для автентифікації користувачів при вході на сервери.

Серед проблем, що стосуються SSH ключів, можна виділити [12]:

1. Відсутність централізованого контролю. Якщо в організації відсутній централізований контроль нам генерацією і розподілом ключів до серверів, а також особа, що відповідальна за управління (наприклад, системний або мережний адміністратор), це може призвести до серйозних проблем з безпекою. Користувачі системи можуть самостійно генерувати та

розповсюджувати ключі, які, можливо, надають доступ до критичних ассетів організації. В зв'язку з цим зростає ймовірність отримання ключа доступу зловмисником та подальшої компрометації системи.

2. Дійсні ключі для неактивних користувачів. SSH ключі для доступу до серверів після їх генерації та додавання мають необмежений термін дії. Таким чином, якщо, наприклад, працівник звільняється з організації, його SSH ключі залишаються валідними у системі. Цей користувач, або зловмисник, якому він передасть свій ключ, зможе увійти на сервер.
3. Закинуті ключі. Цей пункт схожий на попередній, але, скоріше, містить його як свою підмножину. В цю категорію також входять загублені SSH ключі (наприклад, ті що залишились на старій робочій машині користувача) або ключі, що більше не використовуються в інформаційних процесах всередині організації. Такі ключі буває складно знайти та остаточно видалити з мережі.
4. Компрометація ключів. Якщо машину користувача було скомпрометовано (наприклад, через фішинг), зловмисники можуть отримати його SSH ключі і скористатися ними для входу у систему. Системному адміністратору складно відслідковувати SSH трафік через шифрування. Тому якщо, наприклад, працівник не повідомив адміністратора про компрометацію, йому буде ще складніше виявити діяльність хакерів. Така ситуація схожа на техніку Valid Accounts з фреймворку Mitre ATT&CK [13].
5. Розростання ключів (key sprawl). Цей пункт пов'язаний із стрімким зростанням кількості ключів, а також відсутністю механізму ротації. В мережах організації постійно генеруються нові ключі, але старі та вже непотрібні при цьому не видаляються, що робить управління ними дуже складною або неможливою для одного системного адміністратора задачею.
6. Стомлюючі ручні процеси. Якщо процеси управління ключами виконуються вручну, це створює велике навантаження на адміністраторів. А у великих мережах ручне керування стає неможливим через велику кількість працівників, ключів та серверів. Менеджмент обов'язково має бути

автоматизовано та централізовано для збільшення ефективності управління та зменшення навантаження на відповідального працівника.

Тепер розглянемо наслідки компрометації SSH ключів в організації. Одержавши доступ до SSH ключів, зловмисники отримують такі можливості, ресурси та дані:

1. Доступ до чутливої інформації (наприклад, до особистих даних клієнтів, що призведе до репутаційних та фінансових втрат, а також до необхідності виплачувати величезні штрафи, передбачені законодавством країни, у якій оперує організація).
2. Впровадження шкідливого програмного забезпечення у систему.
3. Компрометація сервера електронної пошти, за допомогою чого хакер знову ж таки може отримати дані.
4. При компрометації ключів адміністратора, отримання повного контролю над системами та елементами інфраструктури або мережі компанії.
5. Подальше просування по мережі та можлива компрометація інших сегментів мережі або окремих вузлів (Lateral Movement з фреймворку Mitre ATT&CK [14]).

2.1.2 Оновлення програмного забезпечення

Велика кількість вразливостей SSH стосується не самого протоколу та його внутрішніх механізмів, таких як, наприклад, процес рукостискання або шифрування, а конкретних реалізації SSH клієнта або сервера. Наприклад, вразливість CVE-2024-31497 [15], яка дозволяє зловмиснику відновити секретний ключ та підробляти підписи, перехопивши певну кількість зашифрованих повідомлень, була актуальна лише для PuTTY (реалізація SSH клієнта для Windows) версій 0.68-0.80 . Після виявлення даної вразливості її було закрито у версії 0.81.

Можна зробити висновок, що своєчасне оновлення програмного забезпечення, а також використання актуальних та безпечних версій шифрів значно підвищує рівень захищеності SSH інфраструктури в мережі організації. В компанії необхідно розробити та впровадити політику оновлення ПЗ з урахуванням потреб інформаційних та бізнес-процесів, протестувати її, створити чітку та зрозумілу документацію для працівників. Вразливості в програмах, що використовуються для роботи з протоколом SSH можуть становити серйозні ризики безпеки, тому їх оновлення слід віднести до найвищого рівня важливості і виконувати не пізніше певного проміжку часу (наприклад, доби) з моменту виходу нової версії. При цьому потрібно створювати резервні копії всіх налаштувань для забезпечення безперервності бізнес процесів, а також процедури відновлення роботи систем у випадку, якщо після оновлення виникають проблеми.

2.2 Технічні проблеми SSH протоколу

Найбільшу небезпеку для SSH серверів становлять атаки технічного характеру. Вони часто піддаються мережному скануванню, brute-force та DoS атакам тощо. Можна виділити кілька основних причин компрометації SSH серверів [16]:

1. Неправильні налаштування. Правильні налаштування SSH сервера значно підвищують рівень його захищеності. Проте часто на SSH серверах застосовуються неоптимальні з точки зору безпеки налаштування через відсутність досвідченого системного адміністратора або навмисно, задля зручності користувачів мережі. Наприклад, дозвіл парольної автентифікації на сервері може призвести до успішної атаки на паролі та компрометації системи зловмисником.
2. Brute Force атаки. Атаки перебору часто стають причиною компрометації серверів, особливо якщо на них не встановлено відповідні налаштування або програмні засоби для запобігання атакам та використовуються слабкі

паролі. Загальною рекомендованою практикою є повне відключення парольної автентифікації на сервері та застосування лише SSH ключів. Такі налаштування повністю виключають можливість brute force атаки та підвищують рівень захищеності серверів.

3. DoS та DDoS атаки. Атаки з відмовою у обслуговуванні (DoS або DDoS) можуть переповнити сервер, наприклад, запитами на автентифікацію або обмін ключами, перевантажуючи потужності сервера або мережі та роблячи його неможливим для використання легальними працівниками організації.
4. Man-in-the-middle атаки. При атаці типу “Людина посередині” зловмисник може втрутитися в процес встановлення з’єднання між сутностями та отримати доступ до каналу комунікації. Ця атака є передумовою, наприклад, для Terrapin Attack (CVE-2023-48795), яку буде розглянуто пізніше.

2.2.1 Неправильні налаштування SSH сервера (misconfiguration)

Конфігурація SSH сервера — основний метод, за допомогою якого системний або мережний адміністратор можуть вплинути на рівень захищеності SSH сервера. Проте не завжди на сервері задіяні оптимальні налаштування безпеки. Таке може відбуватись для забезпечення зручності використання системи, незнання найкращих безпекових практик конфігурації SSH або неправильну оцінку ризиків та виставлення пріоритетів захисту. Основні помилки при налаштуванні серверів:

1. Слабка автентифікація. Використання парольної автентифікації користувачів, особливо при відсутності діючої політики паролів може призвести до зламу та копмпрометації сервера. Існує велика кількість інструментів автоматизованого сканування мереж на наявність SSH серверів з подальшими спробами підбору паролів. Щоб уникнути таких атак, необхідно повністю заборонити парольну автентифікацію, замінивши її на SSH ключі, або впровадити багатофакторну автентифікацію одночасним з

використанням ключа та пароля. У випадку MFA потрібно також запровадити сильну політику паролів.

2. Недотримання принципу мінімальних повноважень. Зазвичай системні адміністратори використовують root акаунт для входу на SSH сервер, оскільки управління програмним та апаратним забезпеченням сервера потребує підвищених привілеїв. Проте застосування root користувача є небезпечною практикою, оскільки у випадку, наприклад, brute force атаки на сервер, першою ціллю для зловмисників стане саме root акаунт. Відключення цього облікового запису надає також інші переваги: дотримання принципу мінімальних повноважень (створення облікового запису для кожного працівника, який використовує сервер і надання йому привілеїв, необхідних для виконання задач), спрощений моніторинг (можливо відстежувати дії кожного користувача, що було б неможливо, якби всі використовували один і той самий акаунт root). Крім того, при необхідності, користувач зможе виконувати дії з підвищеними привілеями за допомогою команди sudo.
3. Слабкі підтримувані криптографічні алгоритми. Під час SSH рукостискання сторони обмінюються інформацією про те, які криптографічні алгоритми шифрування, гешування та обміну ключами підтримуються та на основі цього вирішують, які з них будуть використовуватись для встановлення з’єднання. Важливо налаштувати сервер таким чином, щоб заборонити застосування небезпечних алгоритмів, які піддаються зламу. До них відносяться наприклад, MD5 та DES.

2.2.2 Brute Force атаки

Brute force, або атаки грубої сили — дуже популярний клас атак на паролі, яка полягає у перебиранні комбінацій логінів та паролів поки її не буде вгадано та зловмисник не отримає доступ до сервера. Особлива небезпека полягає в тому, що

весь процес даної атаки може бути повністю автоматизовано — від пошуку до підбору паролів. Тому зловмисники можуть виконувати масоване сканування, і, у випадку успішного злому пароля — продовжувати атаку самостійно. Таким чином, якщо в мережі організації не використовуються засоби захисту від атак грубої сили, вона може стати пріоритетною ціллю хакерів.

Проте SSH сервер досить легко захистити від таких атак. Рекомендованою практикою є заміна парольної автентифікації SSH ключами або їх поєднання у багатофакторної автентифікації. Якщо на сервері встановлено MFA, зловмисник не зможе отримати до нього доступ, якщо не матиме відповідного SSH ключа. У випадку, коли використання SSH ключів є неможливим, захистити сервер можна, наприклад, за допомогою утиліти Fail2ban. Цей інструмент легко встановити та налаштувати в залежності від потреб організації. За допомогою нього можливо заборонити доступ до сервера після певної кількості спроб, встановити тривалість блокування та відправляти сповіщення про спроби brute force атак на електронну пошту.

Також дуже ефективним засобом захисту від таких атак є перенесення SSH сервера на інший порт [17]. Як правило, зловмисники атакують саме 22 порт, бо він за замовчуванням використовується для розміщення SSH сервера, проте якщо перенести його на нестандартний порт, це значно поскладнить задачу хакерів. Єдиним мінусом такого підходу є потреба в додаткових налаштуваннях сервера, утиліт, а також необхідність вказування правильного порту при підключенні легальними користувачами.

Крім того, можна створити білий список користувачів, яким дозволено виконувати вхід на сервер. Таким чином, зловмисникам доведеться вгадувати не лише пароль, але й ім’я користувача, що робить процес підбору набагато довшим та складнішим. Можливе також вказання IP-адрес хостів, з яких дозволено вхід у файлі конфігурації сервера.

2.2.3 DoS та DDoS атаки

DoS та DDoS атаки мають деякі спільни риси brute force атаками, проте мета та способи виконання значно відрізняються. Якщо головною метою атаки грубої сили є отримання несанкціонованого доступу до SSH сервера, то у випадку DoS/DDoS — це блокування доступу легальних користувачів до сервера шляхом перевантаження його ресурсів, спроможностей мережі або перевищивши встановлену кількість одночасних з’єднань для сервера. Наступна відмінність — метод атаки. Під час атаки відмови в обслуговуванні зловмисник може лише встановити зв’язок з сервером, проте не виконувати автентифікацію. Таким чином SSH сервер витрачатиме ресурси на підтримання такого з’єднання замість обробки запитів звичайних користувачів.

Для захисту від такого роду атак можна застосовувати сторонні утиліти (наприклад, Cloudflare) або сервіси або виконувати конфігурацію мережних екранів та самого сервера. Наприклад, можна встановити ліміт одночасних TCP з’єднань на порт 22 на фаерволі, проте таке рішення може також стати обмеженням для звичайних користувачів.

Ще одним методом вирішення може стати застосування списків або баз даних відомих зловмисних IP-адрес, таких як AbuseIPDB [18], і блокування всього трафіку, що надходить з них. Таким чином працівники зможуть користуватися сервером у звичайному режимі, а зловмисники не зможуть встановлювати мережне з’єднання з ним. Недоліком такого підходу є необхідність постійно оновлювати список адрес. До того ж, трафік з адрес, які не входять до списку, блокуватися ніяк не буде.

Також для захисту від DDoS атак є методи, що застосовуються проти brute force атак — перенесення сервера на інший порт, використання fail2ban, автентифікація за SSH ключами тощо.

2.2.4 Man-in-the-middle атаки

Атака типу man-in-the-middle, або «людина посередині» полягає в тому, що зловмисник втручається в комунікацію сторін на стадії SSH рукостискання та узгоджує криптографічні параметри одночасно з сервером та з клієнтом [19]. Нижче наведено, як протікає атака man-in-the-middle, коли сторони намагаються встановити з'єднання, а також описано усі основні кроки під час такої атаки (Рисунок 2.2):

1. Клієнт запитує SSH сесію у сервера, зловмисник перехоплює запит через dnsspoof та перенаправляє його до своєї системи.
2. Далі зловмисник ініціює SSH з'єднання зі справжнім SSH сервером.
3. Хакер надає власний публічний ключ хоста та тимчасовий ключ сервера. Це ключовий момент в атаці, від якого залежить успіх її проведення. PuTTY (приклад SSH клієнта) показує клієнту попередження про новий невідомий ключ хоста сервера. (Рисунок 2.1) Єдиним способом запобігти атаці буде вибір опції Відмінити, що одразу ж розірве з'єднання.

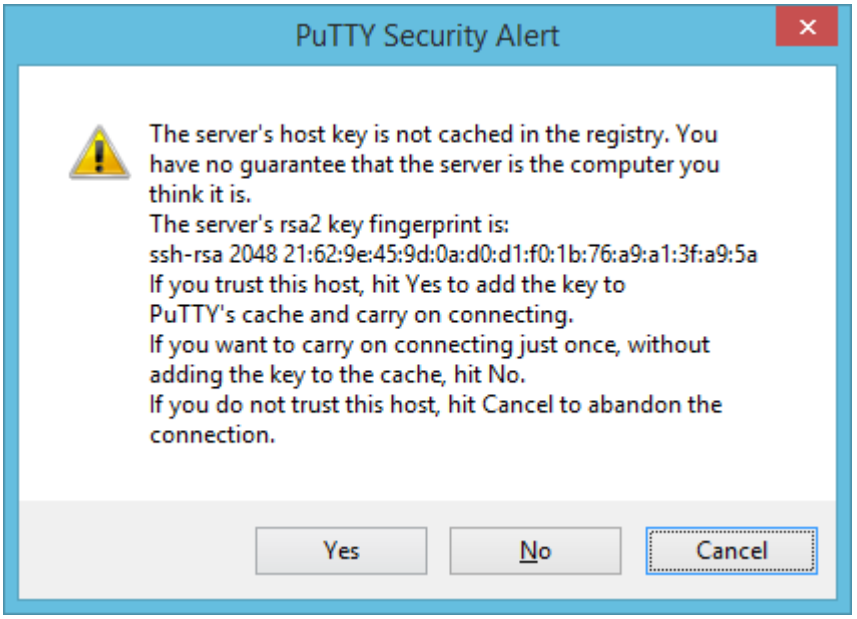

Рисунок 2.1 - Попереджувальне вікно PuTTY про зміну публічного ключа хоста

4. Якщо клієнт натисне Так або Ні — клієнт згенерує 256-бітний ключ сесії, використовуючи ключі, надіслані зловмисником і проведення атаки можна вважати успішним.
5. Справжній сервер надсилає свої ключі зловмиснику.
6. Хакер генерує 256-бітний ключ сеансу, використовуючи відкритий ключ хоста і тимчасовий ключ справжнього сервера.
7. Зловмисник вмикає шифрування і показує список доступних для використання шифрів, серед яких можуть бути blowfish, 3DES, тощо.
8. Клієнт вибирає метод шифрування і запитує автентифікацію на рівні користувача.
9. Сервер вмикає шифрування і показує список доступних для використання шифрів.
10. Зловмисник вибирає метод шифрування і запитує автентифікацію на рівні користувача.

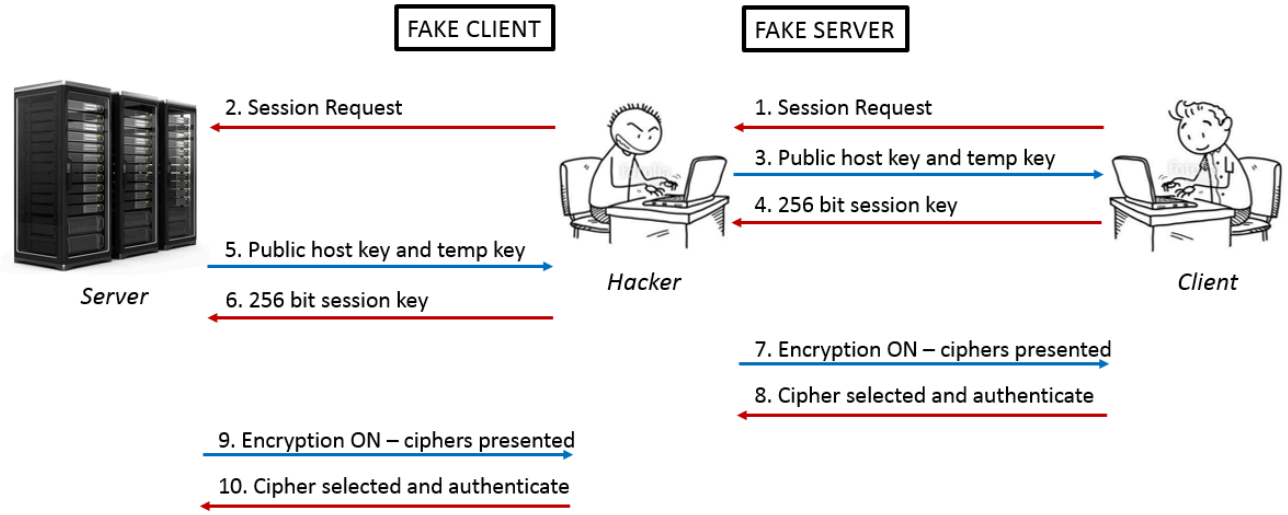

Рисунок 2.2 - Покрокова схема атаки man-in-the-middle

Після цього хакер може читати та модифікувати повідомлення, якими обмінюються клієнт і сервер (в тому числі паролі) або вставляти свої команди для виконання непомітно для сторін. Існує багато готових інструментів для проведення таких атак: sshmitm, intercepter-ng, ssh-mitm тощо.

Для захисту від main-in-the-middle, допоможе автентифікація за ключами або багатофакторна (MFA) замість звичайної парольної. Крім того, варто провести інструктаж працівників організації з приводу прийняття нових відбитків сервера (попередження PuTTY, про яке йшла мова вище). Потрібно інформувати користувачів про зміни на SSH серверах і якщо вони не проводились, то до зміни відбитка потрібно поставитись з особливою пильністю. Але цей захід скоріше відноситься до організаційних, ніж до технічних.

2.3 Аналіз реальних CVE вразливостей

Для розуміння того, які засоби безпеки необхідно застосовувати у сучасних реаліях використання SSH, необхідно проаналізувати нещодавно знайдені вразливості протоколу, їх серйозність та складність експлуатації. Завдяки цьому вдасться вивести оптимальні налаштування та практики для забезпечення захищеності SSH серверів у мережах організацій.

2.3.1 Terrapin Attack

Атака усічення префікса (prefix truncation), яка порушує цілісність захищеного каналу. Дана атака працює лише для шифру ChaCha20-Poly1305 або іншого з режимом ланцюга блоків (CBC) в поєднанні з Encrypt-then-MAC, проте згідно з [20], 57% серверів вказують цей режим шифрування як той, якому надається перевага (Таблиця 2.1).

Таблиця 2.1 - Статистика підтримуваних режимів шифрування SSH серверами

Режим шифрування	Бажаний		Підтримується	
ChaCha20-Poly1305	8,379 тис.	57.64%	10,247 тис.	67.58%
CTR-EaM	3,964 тис.	26.14%	4,200 тис.	27.70%
GCM	1,219 тис.	8.04%	10,450 тис.	68.92%
CTR-EtM	828 тис.	5.46%	10,685 тис.	70.46%
CBC-EaM	359 тис.	2.37%	1,585 тис.	10.46%
CBC-EtM	14 тис.	0.09%	2,614 тис.	17.24%
Інші	2 тис.	0.01%	-	-
Невідомо / Немає KexInit	36 тис.	0.24%	-	-
Всього	15,164 тис.	100%		

Атака базується на двох фактах:

1. SSH не захищає повну розшифровку рукостискання. Хоча автентифікація сервера використовує підпис для перевірки цілісності рукостискання, підпис формується над фіксованим списком повідомлень рукостискання, а не повної стенограми. Ця прогалина в автентифікації дозволяє зловмиснику зловмиснику вставляти повідомлення в рукостискання і таким чином маніпулювати порядковими номерами.
2. Протокол не скидає номери послідовності на початку захищеного каналу. Замість цього SSH збільшує порядкові номери (sequence numbers) монотонно, незалежно від стану шифрування. Будь-які маніпуляції з порядковими номерами перед захищеним каналом переносяться у сам канал.

Щоб реалізувати атаку, зловмисник повинен мати можливість перехоплювати дані і підміняти пакети (тобто виконати атаку man-in-the-middle). Кірм того, він має визначати довжину пакетів, що передаються, навіть коли поле Length зашифровано. Сама атака відбувається у два етапи (Рисунок 2.3):

1. Зловмисник використовує техніку RcvIncrease (Рисунок 2.2) для збільшення C.Rcv (receive sequence number) на одиницю, наприклад, відправивши клієнту IGNORE-повідомлення клієнту перед повідомленням NEWKEYS.

Використання цієї техніки можливе через вимогу стандарту SSH про те, що всі його реалізації мають мусять ігнорувати повідомлення IGNORE у будь-який час, яка від початку була введена для захисту від аналізу трафіку.

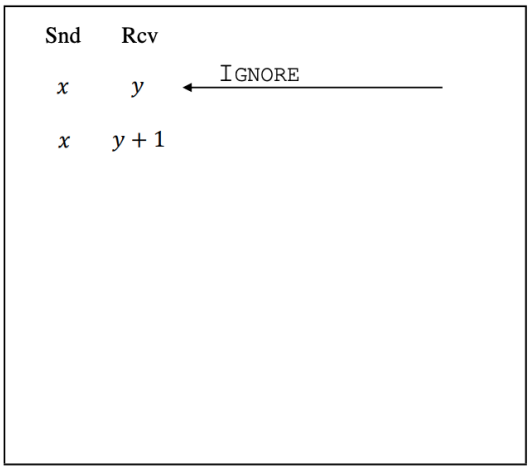

Рисунок 2.3 - Схема роботи техніки RcvIncrease

2. Зловмисник видаляє перше повідомлення SC1.

Використовуючи цю атаку, зловмисник може маніпулювати лічильниками збільшення та отримання (increase and receive counters) на сервері та на клієнті, вставляючи повідомлення у процес рукостискання. Це, в свою чергу, може призвести, наприклад, до атаки на узгодження розширень (extension negotiation attack). Це процес, коли сервер та клієнт узгоджують алгоритми та засоби захисту, що використовуватимуться під час з'єднання. Втручання в цей процес може призвести до зниження рівня безпеки тунелю.

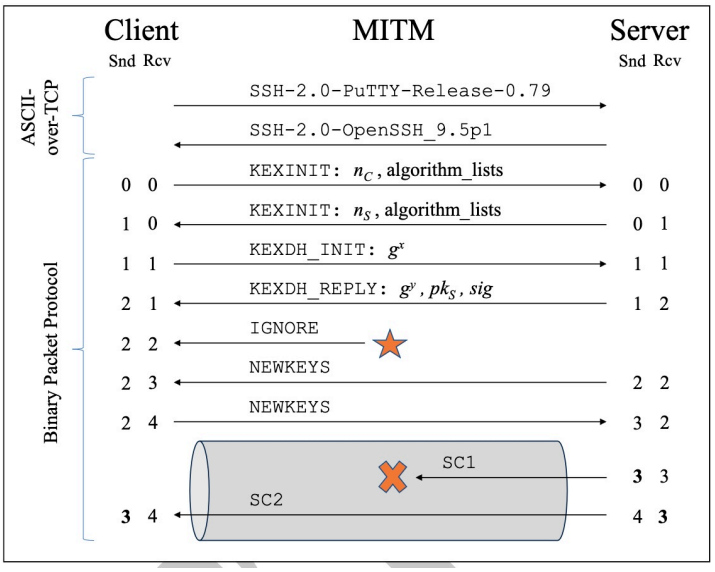

Рисунок 2.4 - SSH рукостискання під час атаки усікання префіксу

Видалення одного повідомлення — це окремий випадок атаки усікання префіксу. Загалом, можлива маніпуляція з більшою кількістю повідомлень. Така атака називається Усікання префіксу (Ns, Nc). Від вже розглянутої вона відрізняється лише кількістю повідомлень IGNORE (Ns повідомлень для сервера та Nc — для клієнта), які зловмисник вставляє в канал комунікації між клієнтом та сервером. Варіант, що розглядався вище, буде атакою усікання префікса (1, 0).

Отже, якщо відомо, що зловмисник може видалити декілька повідомлень на початку захищеного каналу (після NEWKEYS), необхідно визначити, що саме являють собою ці повідомлення так як їх видалення може вплинути на рівень захищеності SSH каналу. Якщо сторони підтримують механізм узгодження розширень (про що вони повідомляють на початку комунікації), то першим повідомленням в захищеному каналі буде ExtInfo, яке включає в себе інформацію про кількість розширень, їхні імена та дані, пов’язані з кожним з них.

Видалення повідомлення ExtInfo призводить до зниження рівня захищеності каналу комунікації. Наприклад, якщо сервер не надасть інформацію про підтримку розширення ping@openssh.com, то захист від аналізу часових проміжків натискання клавіш (keystroke timing countermeasures) буде відключено, а відсутність розширення server-sig-algs призведе до послаблення алгоритмів підпису. Розширення, що надсилаються сервером до автентифікації клієнта, наведено на рисунку 2.4. ping@openssh.com не входить сюди, бо сервер надсилає його у другому повідомленні ExtInfo вже після автентифікації клієнта.

Таблия 2.2 - SSH розширення, що надсилаються сервером у повідомлення ExtInfo до автентифікації клієнта

Назва розширення	Разів запропоновано	
server-sig-algs	637,466	76.81%
publickey-hostbound@	73,040	8.80%
delay-compression	283	0.03%
no-flow-control	283	0.03%
global-requests-ok	283	0.03%

Атака на узгодження розширень відрізняється в залежності від шифру, що використовується під час комунікації. Проте для ChaCha20-Poly1305 вона ніяк не відрізняється від тієї, яку було розглянуто вище:

1. Зловмисник вставляє повідомлення IGNORE, що збиває лічильник отриманих повідомлень.
2. На початку захищеного каналу видаляється повідомлення ExtInfo, яке відповідає за узгодження розширень. За рахунок цього важливі розширення безпеки не відправляються сервером і додаткові функції, за які вони відповідають, відключаються, що знижує загальний рівень захищеності з'єднання.

Пробелма, що робить атаку Terrapin Attack можливою, знаходиться в самій специфікації SSH. Автори атаки пропонують наступні зміни до специфікації для закриття даної вразливості [20]:

1. Скидання номерів послідовності на початку захищеного з'єднання. Таким чином, будь-які маніпуляції під час SSH рукостискання не будуть впливати на захищений канал.
2. Перевірка всього процесу SSH рукостискання. За допомогою автентифікації всієї процедури з обох сторін дозволить виявити втручання під час атаки man-in-the-middle та зміну номерів послідовності.

Проте системний або мережний адміністратор не може вносити зміни в специфікацію SSH, тому потрібно застосовувати інші контрзаходи для захисту від Terrapin Attack. Ефективним та доступним способом буде відключення криптографічних алгоритмів, вразливих до даної атаки. Використання аналогічних за характеристиками шифрів не вплине на безпеку, проте дозволить швидко закрити цю вразливість.

2.3.2 CVE-2024-31497 (PuTTY ECDSA keys)

Дана вразливість була знайдена в Putty (реалізації SSH клієнта для Windows) версій 0.68-0.80 [21]. Вразливість актуальна лише для ключів типу

521-bit ECDSA. Усі схеми підпису DSA вимагають випадкового значення під час підписання, яке позначають буквою k.

Якщо зловмисник зможе вгадати значення k, яке було використано, або знайти будь-які два підписи, які згенеровано з тим самим k, то він зможе негайно відновити закритий ключ. Крім того, хакер зможе підробляти підписи, отримуючи доступ до серверів, на яких використовувався даний закритий ключ.

Це означає, що небезпечно створювати підписи DSA в системах без високоякісного джерела випадковості. На момент створення Putty у Windows ще не існувало механізму генерації випадкових чисел, тому використовувався детермінований метод на основі гешування. Спочатку створювався SHA-512 геш, а потім відбувалося скорочення за модулем q, де q - це порядок групи, що використовується в системі DSA. Але якщо q має 512 біт, то це не дає ефекту — на виході старші 9 біт числа k будуть дорівнювати нулю, все інше не змінюється. Такого зсуву достатньо, щоб дозволити атаку з підбором ключа, якщо у зловмисника є приблизно 60 підписаних повідомлень. Проте щоб отримати підписані повідомлення, хакеру необхідно отримати доступ до SSH сервера, на який користувач входить з даним ключем. Але є й інший спосіб — отримати підписи з комітів, які система контролю версій git підписала за допомогою SSH ключа. Дану вразливість було закрито у наступних версіях PuTTY.

Щоб заподіяти даній вразливості, необхідно регулярно виконувати оновлення як серверного, так і клієнтського програмного забезпечення (згідно розробленої політики), а також використовувати сильні криптографічні алгоритми і оновлювати список підтримуваних шифрів на сервері за необхідності.

Висновки до розділу 2

В даному розділі було виділено та проаналізовано основні проблеми роботи з SSH серверами організаційного та технічного характеру. Метою цього аналізу є пошук та виділення способів вирішення, що можуть бути впроваджені в організації. Таким чином можливо розробити ряд правил, процедур та алгоритмів,

що значно підвищать рівень захищеності SSH серверів в організації та допоможуть уникнути компрометації всієї мережі або окремих її елементів.

Серед організаційних ризиків було виділено проблеми з генерацією та управлінням SSH ключами для доступу до серверів. Це організаційне питання є комплексним і включає в себе багато аспектів від необхідності централізованого контролю за створенням та розповсюдженням ключів до стрімкого зростання їхньої кількості та потреби в видалені тих, що не використовуються. Крім того, при розв'язанні задачі управління ключами слід враховувати особливості інформаційних та бізнес-процесів організації для знаходження найкращого рішення. Наприклад, у випадку малих організацій або стартапів керування ключами може здійснювати один системний адміністратор вручну, бо їхня кількість відносно невелика. Проте для більших компаній з великою кількістю серверів та користувачів слід впроваджувати рішення (певне програмне забезпечення) для централізованого управління ключами, а також, за потреби, наймати додатковий персонал.

Ще одним організаційним питанням, що потребує уваги, є політика оновлення програмного забезпечення. Воно актуальне для всіх інформаційних організацій, навіть якщо вони не користуються SSH. Проте у разі застосування даного протоколу питання оновлення серверного та клієнтського програмного забезпечення стає ще гострішим, оскільки нові версії часто містять в собі зміни, що закривають знайдені актуальні вразливості, що становлять ризики компрометації серверів або ключів клієнтів.

Технічний аспект захисту SSH серверів є критичною складовою забезпечення захищеності SSH інфраструктури в мережі, що потребує від системного адміністратора застосування поміркованих кроків та процедур. В даному розділі було розглянуто brute force, MitM, а також DoS/DDoS атаки. Проте найважливішим чинником є саме налаштування SSH серверів. Правильна конфігурація запобігає більшості атак та сприяє зручному та ефективному використанню SSH протоколу. Наприклад, впровадження автентифікації за

ключами або MFA дозволяє значно ускладнити або навіть унеможливити атаки на паролі (brute force або атаки за словником).

Крім того, було проаналізовано реальні вразливості, що стосуються SSH — CVE-2023-48795 (Terrapin Attack), яка дозволяє за допомогою маніпуляцій з повідомленнями та номерами послідовності під час SSH рукостискання знизити рівень захищеності каналу, що встановлюється, а також CVE-2024-31497, яка дає можливість зловмиснику відновити секретний ключ користувача в разі отримання певної кількості підписаних повідомлень. Детальний аналіз цих вразливостей дозволяє краще зрозуміти, яким аспектам захисту SSH серверів потрібно приділити найбільшу увагу.

В наступному розділі на основі виконаного аналізу буде створено перелік організаційних рекомендацій, спрямованих на поліпшення процесів генерації, розповсюдження, моніторингу та видалення SSH ключів в мережах організацій. Впровадження цих заходів значно полегшить роботу системного адміністратора та запобігне компрометації ключів.

Запропоновані технічні заходи будуть в основному спрямовані на конфігурацію SSH серверів, а саме на застосування налаштувань, що підвищують рівень безпеки та знижують ймовірність компрометації внаслідок тих чи інших атак. Крім того, буде розглянуто застосування інших утиліт та засобів, що можуть позитивно вплинути на захищеність сервера або полегшити роботу з ним. Вибір налаштувань та значень для параметрів буде виконуватись на основі недоліків, проблем та вразливостей, які було згадано у цьому розділі.

3 РОЗРОБКА ОРГАНІЗАЦІЙНИХ РЕКОМЕНДАЦІЙ ТА ПРОГРАМНОГО ЗАСТОСУНКУ ДЛЯ ЗАХИСТУ SSH СЕРВЕРІВ

SSH сервери є доволі популярною мішенню для атак з боку зловмисників. Це пов'язано з цілою низкою факторів, що приваблюють хакерів та спонукають їх здійснювати атаки саме на SSH інфраструктуру. Серед них можна виділити наступні:

- Доступність ззовні. За необхідності SSH сервер можуть зробити доступними із зовнішньої мережі, що робить його видимим та досяжним для зловмисників, які застосовують автоматизовані засоби сканування мереж для знаходження відкритих SSH портів і виконання подальших спроб атак.
- Можливість атак на паролі. Автентифікація клієнта під час входу на SSH сервер передбачає використання паролів. За відсутності належних засобів протидії, хакери можуть здійснювати атаки на паролі, такі як атаки грубої сили. Крім того, такі атаки також можуть бути автоматизовані і вимагатимуть мінімальної участі з боку хакера, якого буде сповіщено, наприклад, лише в разі успішного знаходження правильного пароля.
- Віддалений доступ. В разі компрометації SSH сервера зловмисник може отримати повний або частковий (в залежності від привілеїв облікового запису, з якого здійснено вхід) доступ до ресурсів та даних. Отримавши контроль над сервером, хакер може красти інформацію, порушувати інформаційні або бізнес-процеси організації або застосовувати сервер для виконання більш масштабних атак із серйознішими наслідками.
- Уникнення виявлення. Зловмисник може використовувати шифрування, яке надає SSH протокол для приховування своїх дій у системі та уникнення виявлення засобами моніторингу. Скомпрометувавши сервер, хакер має можливість використовувати його для подальших атак на інші елементи мережі (Lateral movement).

Отже, SSH сервери потребують надійного, комплексного та продуманого захисту, який зможе запобігти атакам та підвищити рівень їхньої безпеки. В даному розділі буде створено організаційні рекомендації, а також запропоновано технічні заходи та налаштування серверів, що допоможуть протистояти атакам на SSH та унеможливити компрометацію серверів. Крім того, надані заходи зможуть полегшити роботу системного адміністратора та спростити використання протоколу SSH для працівників організації.

3.1 Організаційні рекомендації

Процеси організації роботи з SSH серверами грають не меншу роль, аніж технічні засоби захисту. Обидві ці компоненти захисту мають ефективно взаємодіяти одна з одною, щоб забезпечувати належний рівень безпеки серверів у мережі організації. Їх не може бути запроваджено окремо, або обрано лише одну з них, оскільки такий підхід не принесе бажаного результату. Наприклад, встановлення автентифікації за SSH ключами або MFA не зможе забезпечити достатній рівень захищеності, якщо ці ключі не будуть відстежуватися та контролюватися системним адміністратором і їх може бути скомпрометовано внаслідок фішингу або інших атак. Отже, необхідно розробити та впровадити в компанії організаційні заходи так, щоб вони логічно співвідносилися з технічними, а також відповідали потребам компанії.

3.1.1 Порівняння підтримуваних методів автентифікації SSH

Окрім вже названих парольної автентифікації, а також автентифікації за ключами, протокол SSH підтримує декілька інших методів та процедур для перевірки користувачів. Кожен з них має особливості імплементації, ряд переваг та недоліків, а також сценарії застосування, в яких він проявляє себе краще за

інших. Вибір оптимального способу автентифікації слід робити, беручи до уваги потреби організації, а також особливості її інформаційних та бізнес-процесів. Щоб розуміти, який метод краще підійде для тих чи інших умов, слід проаналізувати кожен з них, виділити сильні та слабкі сторони та порівняти їх між собою.

Під час порівняння важливо також враховувати, для вирішення яких задач в організації використовується протокол SSH. В основному є два напрямки: інтерактивний доступ для живих користувачів та доступ для програм або скриптів з метою виконання автоматизованих завдань в мережі. Отже, потрібно брати до уваги пристосованість того чи іншого метода автентифікації до цих способів роботи з протоколом та SSH серверми.

SSH підтримує 4 типи автентифікації [22]:

- Парольна. Для отримання доступу до сервера використовується пароль. Цей спосіб є дуже розповсюдженим через свою простоту та легкість налаштування. Користувач може увійти на сервер з будь-якого пристрою, якщо він знає пароль. Проте коли виникає потреба працювати з великою кількістю серверів з унікальним паролем для кожного, працівник може почати записувати їх або використовувати той самий пароль всюди (якщо він має змогу самостійно його обирати), що знижує рівень захищеності. Крім того, така автентифікація погано підходить для автоматизованих процесів (наприклад, програм або скриптів) оскільки не надає широких можливостей контролю доступу, доступних для інших методів. У випадку використання даного методу, необхідно впроваджувати ротацію та політику паролів, що знизить зручність користування для працівників організації.
- На основі хоста (host-based). В даному методі для автентифікації користувача використовується ключ хоста, з якого було виконано запит на з’єднання. Тобто хост клієнта має свій відкритий та закритий ключ і процесі автентифікації сервер перевіряє його. Цей процес є аналогічним до того, як клієнт перевіряє автентичність сервера при встановлені з’єднання. Таким чином, між хостами в мережі встановлюються довірчі відносини. Проте

даний метод погано підходить як для звичайних користувачів, оскільки не має інтерактивного логіну, так і для автоматизованих процесів, оскільки не надає можливостей для обмеження виконуваних команд. Крім того, такі довірчі відносини між хостами полегшують зловмисникам подальше пересування в мережі (lateral movement) в разі успішного проникнення до неї.

- За допомогою Kerberos. Даний метод використовується в поєднанні з LDAP або Active Directory для створення єдиної точки входу і централізованого зберігання облікових записів користувачів. Тобто, клієнту необхідно виконати автентифікацію в центрі розподілення ключів, після чого він зможе отримати доступ до SSH серверів, що знаходяться в тому ж домені (Рисунок 3.1). З одного боку, таке централізоване керування дозволяє адміністратору швидко надавати або забирати доступ і мати повний контроль над системою. З іншого боку, така система може створювати певні неявні довірчі зв'язки і доступи, якими можуть користуватися зловмисники. Для звичайних працівників Kerberos є зручним способом автентифікації, особливо якщо необхідно працювати з великою кількістю серверів. Для автоматизації цей метод не підходить, оскільки має недостатньо можливостей для обмеження виконання команд.

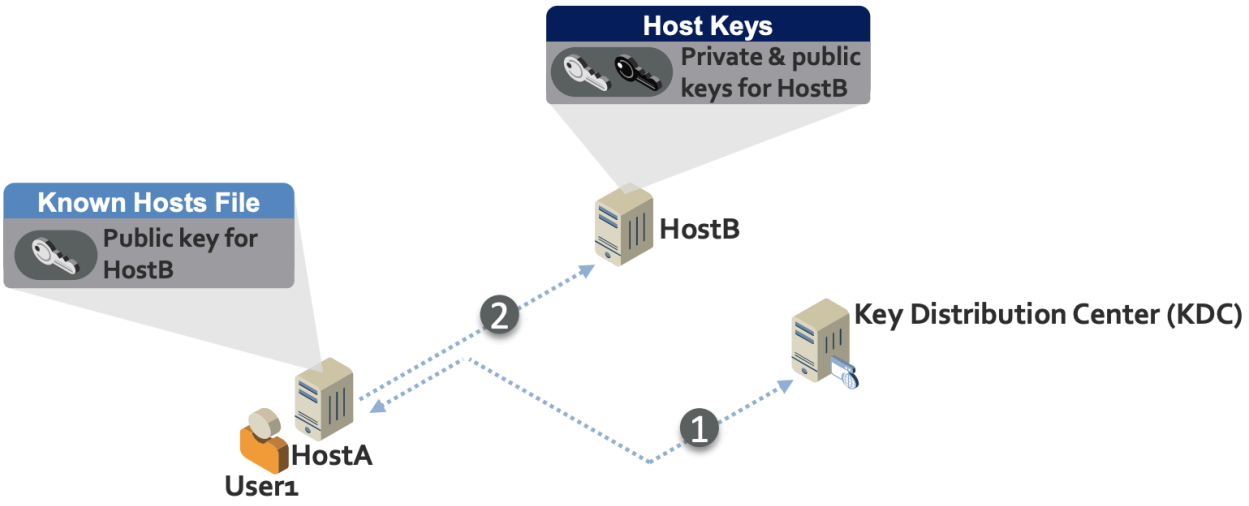

Рисунок 3.1 - Автентифікація за допомогою Kerberos

- З використанням публічних ключів. Даний метод передбачає пару ключів (відкритий і закритий) для кожного користувача або процесу, яка використовується для автентифікації. Відкритий ключ завантажується на SSH сервер (у файл authorized_keys) і при встановленні з'єднання перевіряється секретний ключ клієнта. Можливе також введення обмежень для авторизованих ключів, в тому числі на використання тих чи інших команд. Важливою перевагою цього методу є явне встановлення довірчих зв'язків між серверами та користувачами. Тобто, доступ не може бути отримано без явного дозволу адміністратора системи (за наявності централізованого контролю та заборони створювати і встановлювати ключі на сервери звичайним користувачам).

Отже, обираючи метод автентифікації, що буде використовуватись для доступу до SSH серверів в мережі, в першу чергу потрібно брати до уваги потреби організації, особливості процесів, а також способи застосування протоколу. За можливості обраний метод має працювати як для інтерактивного входу користувачів, так і для автентифікації автоматичних процесів.

Загалом, оптимальним та найбільш часто вживаними методом є автентифікація через публічні ключі. Вона є зручною для звичайних користувачів, оскільки дозволяє виконувати вхід за допомогою одного ключа на велику кількість серверів і не запам'ятовувати пароль, а також для організації роботи автоматизованих рішень, бо дає можливість вводити обмеження на виконувані команди. Тому автентифікація за відкритим ключем надалі буде розглядатися як основна і будуть запропоновані рекомендації з її використання, а також з управління парами ключів всередині мережі.

3.1.2 Управління SSH ключами

Однією з найважливіших організаційних задач захисту SSH серверів в компанії є управління SSH ключами доступу. Вона включає в себе створення,

передавання, моніторинг та видалення ключів в межах мережі організації. Проте перед вирішенням даної задачі, необхідно з'ясувати, що являють собою ці ключі та як вони використовуються для автентифікації.

Загалом для автентифікації за ключами використовується асиметрична криптосистема. Тобто існує два ключа — відкритий та закритий. Якщо використовується OpenSSH, клієнт може згенерувати нову пару ключів за допомогою команди ssh-keygen, і передати публічний ключ на сервер командою ssh-copy-id. Таким чином, приватний ключ зберігатиметься на стороні клієнта у директорії .ssh, а відкритий — на сервері у файлі ~/.ssh/authorized_keys, проте це можна змінювати у налаштуваннях. Коли клієнт надсилає запит на з'єднання, сервер надсилає йому виклик (challenge). Далі користувач, застосовуючи свій приватний ключ та виклик, створює автентифікатор, який надсилає назад. Сервер перевіряє автентифікатор та надає або не надає доступ.

З цього можемо зробити висновок, що головна задача управління SSH ключами в організації — не допустити компрометації закритих ключів, що знаходяться на комп'ютерах користувачів, та їх використання зловмисниками для отримання несанкціонованого доступу до серверів.

Отже, на основі аналізу проблем, проведеного у Розділі 2, можна виділити наступні рекомендації стосовно управління SSH ключами [22]:

1. Впровадження системи контролю ключів. Ретельне централізоване управління ключами, відстежування руху ключів в організації запобігає витоку ключів та дозволяє ефективно керувати доступом до серверів. Для цього можна використовувати спеціальне програмне забезпечення, яке можна або придбати (наприклад, Universal SSH Key Manager) або розробити самостійно.
2. Моніторинг активності. Моніторинг логів SSH серверів та мережних екранів значно підвищує рівень безпеки та швидкість реагування на інциденти, пов'язані з протоколом. Це може включати аналіз журналів входу, відстеження змін файлів та інших дій користувачів.

3. Ротація ключів та заборона доступу для неактивних користувачів. Потрібно періодично змінювати ключі SSH для зменшення ризику компрометації. Це може бути загальною політикою або стосуватись лише деяких користувачів або серверів. Також важливо забороняти доступ для неактивних користувачів.
4. Використання парольних фраз для захисту ключа. При генерації пари ключів, наприклад, за допомогою команди ssh-keygen, можливо задати парольну фразу для шифрування закритого ключа. Таким чином, якщо приватний ключ буде скомпрометовано зловмисником, він не зможе виконати автентифікацію на сервері, не знаючи парольну фразу. Щоб даний засіб був ефективним, необхідно також вводити політику, що визначає мінімальну довжину та інші вимоги до парольної фрази.
5. Копіювання ключів. Для зниження ризику компрометації приватних ключів в мережі організації, необхідно ввести заборону на копіювання приватних ключів як для звичайного інтерактивного входу, так і для автоматизованих процесів, на інші машини. Якщо внутрішні процеси компанії потребують такого копіювання, потрібно регламентувати, куди ключі можуть копіюватися та як можуть використовуватися.
6. Контроль доступу до ключів. Для забезпечення централізованого управління доступом до SSH серверів, лише системний адміністратор системи повинен мати право встановлювати нові авторизовані ключі. Якщо дозволити це робити звичайним користувачам системи, це призведе до утворення нових бекдорів, зниження ефективності систем моніторингу та аудиту, втрати адміністратором контролю над довірчими зв’язками та каналами доступу в мережі. Рекомендованою практикою є перенесення файлу authorized_keys в інше місце в системі, доступ до якого має, наприклад, лише root користувач.
7. Обмеження для автоматизованих процесів. Якщо в організації використовуються автоматизовані рішення, що вимагають доступу до SSH серверів для виконання завдань, цей доступ має бути коректно надано та налаштовано. Необхідно вводити обмеження на виконувані команди та

IP-адресу клієнта. Таким чином вдасться запобігти застосуванню програм не за призначенням і несанкціонованому доступу до сервера в разі компрометації закритого ключа.

8. Заміна ключів при компрометації. Якщо системний адміністратор або адміністратор безпеки в організації має підозру щодо компрометації закритих SSH ключів або про неї вже точно відомо, слід негайно перегенерувати та замінити ці ключі. Ефективність буде вищою, якщо працівники самі будуть повідомляти про підозрілі або аномальні дії що стосуються SSH доступу.

Ці рекомендації дозволять швидко та ефективно керувати доступом до SSH серверів в організації, знизити навантаження на системного адміністратора та пом’якшити наслідки в разі компрометації закритих ключів.

3.1.3 Оновлення програмного забезпечення

Вразливості в реалізації клієнтської або серверної частини можуть використовуватися зловмисниками для отримання несанкціонованого доступу. Ці недоліки можуть мати різну природу: помилка в програмному коді, слабкі налаштування,наприклад, дозвіл на використання слабких криптографічних алгоритмів, а також слабкості протоколів, наприклад, якщо застосовується SSH-1.

Необхідно регулярно оновлювати програмне забезпечення SSH серверів, включаючи бібліотеки криптографії за необхідності. Оновлення можуть включати в себе виправлення вразливостей та покращення безпеки. Потрібно вводити політики оновлення програмного забезпечення, які регламентують категорії оновлень, наприклад, критичні та звичайні, часові проміжки, під час яких їх має бути застосовано тощо.

Оновлення можуть виконуватись централізовано системним адміністратором або кожним користувачем окремо. При застосуванні останнього підходу, необхідно також проводити навчання персоналу, де будуть пояснюватися

політики оновлень, необхідність їх своєчасного застосування, а також відповідальність за недотримання політики.

3.2 Технічні засоби захисту SSH серверів

Маючи дані про поширені атаки на SSH сервери, а також проаналізувавши реальні вразливості SSH з бази даних CVE, можна виділити ряд заходів та налаштувань, що забезпечать високий рівень захищеності сервера від поширених атак. Перелік заходів, налаштувань та процедур, що підвищують безпеку SSH серверів в мережі організації [23]:

1. Відключення протоколу SSH-1. На сьогоднішній день протокол SSH 1 першої версії вважається застарілим та небезпечним і не рекомендується до використання. Ця версія протоколу має загальновідомі вразливості, застосовує слабкі криптографічні протоколи та не забезпечує цілісність на достатньому рівні, що відкриває можливість для man-in-the-middle атак.
2. Конфігурація логування. Для ефективного моніторингу доступу та використання SSH серверів в мережі організації, необхідно коректно налаштувати створення логів. Для параметра LogLevel у конфігураційному файлі сервера оптимальним значенням буде «VERBOSE». Також можливе підключення логів SSH сервера до SIEM систем для централізованого контролю та швидкого реагування на аномалії або спроби атак.
3. Вимкнення слабких методів автентифікації. Необхідно підібрати оптимальний механізм авторизацій користувачів та процесів на SSH серверах в залежності від потреб організації, а інші заборонити. При застосуванні оптимального для компанії методу автентифікації системний адміністратор отримає більше контролю над зв’язками між сутностями в мережі, працівникам буде зручніше користуватися SSH інфраструктурою, а рівень захищеності буде підвищено.

4. Відключення root користувача. Варто відключити можливість автентифікації суперкористувача на сервері та створити нових користувачів з необхідними привілеями, застосовуючи принцип мінімальних повноважень.
5. Перенсення сервера на інший порт. Зміна порту SSH сервера з 22 на інший менш популярний дозволяє уникнути мережних сканувань. Але цей метод може спричинити проблеми з деякими реалізаціями SSH клієнтів і затримки відповідей сервера. Крім того, це ускладнить роботу з серверами для працівників організації.
6. Відключення слабких або вразливих криптографічних алгоритмів [24]. На прикладі Terrapin Attack було продемонстровано, як вразливі алгоритми можуть впливати на безпеку і що їх використання необхідно уникати.
7. Створення білого списку користувачів [3]. Налаштування SSH сервера дозволяють використання білого списку облікових записів, яким дозволено вхід на сервер. Задати список можна в параметрі AllowUsers. Крім імені користувача, можна також вказати IP-адресу, з якої він може заходити на сервер. Використання такого підходу забезпечує контрольоване розмежування доступу та дозволяє запобігти отриманню доступу хакерами в разі компрометації приватного ключа користувача.
8. Налаштування Banner Message [25]. SSH дозволяє налаштувати повідомлення, яке буде виведено для клієнта при спробі входу на сервер. Хоча такий текст прямо не впливає на захищеність, він також може мати позитивний вплив. Банер можна використовувати, наприклад, для відображення дисклеймера про обмеженість доступу до сервера. Таким чином, при порушеннях зловмисників буде легше притягнути до відповідальності.
9. Обробка неактивних з’єднань. Щоб уникнути перевантаження сервера внаслідок обслуговування неактивних з’єднань, слід задати значення для параметрів ClientAliveInterval та ClientAliveCountMax. Якщо їх було налаштовано, SSH сервер буде надсилати клієнту пакети і очікувати відповідь. Якщо відповіді не буде, сервер розірве з’єднання.

ClientAliveInterval регулює інтервали між відправкою пакетів, ClientAliveCountMax — кількість повторень перед закриттям каналу.

10. Встановлення ханіпотів. Даний пункт не є обов'язковим і не підвищує рівень захищеності сервера напряму. Проте ханіпот дозволяє застосовувати проактивний підхід для захисту SSH інфраструктури, виявляти та аналізувати атаки до того, як вони досягнуть справжніх SSH серверів і запроваджувати додаткові захисні заходи в залежності від типу атаки.

3.3 Програмна реалізація автоматичного конфігуратора сервера

Маючи перелік заходів, що мають бути застосовані для SSH серверів, можемо реалізувати скрипт для автоматичної конфігурації. Програма має автоматично застосовувати оптимальні налаштування, зазначені вище, а також встановлювати і конфігурувати (за згоди користувача) додаткові інструменти та утиліти, що підвищують рівень захищеності сервера.

Для створення скрипта було обрану мову програмування Python 3, оскільки через свою простоту та гнучкість вона дозволяє швидко реалізувати необхідні функції.

Програмний код скрипта наведено у Додатку А.

3.3.1 Реалізація автоматичного та ручного режимів

Програма може працювати в двох основних режимах роботи (Рисунок 3.2):

1. Автоматичний. Скрипт самостійно застосовує найкращі налаштування безпеки, встановлює та конфігурує ханіпот та інші допоміжні утиліти. Від користувача вимагається лише мінімальне втручання для підтвердження встановлення програм, створення білого списку.

2. Ручний. Цей режим призначений для більш досвідчених користувачів та дозволяє більш гнучко керувати роботою програми в залежності від потреб системного адміністратора або організації. Користувач програми може самостійно приймати рішення щодо увімкнення або вимкнення автентифікації за паролем, входу для root користувача. Також можна налаштувати власне повідомлення для банера, підтвердити або відхилити створення ханіпота.

```
Welcome to S3H (Super Secure Shell)!
   _________ __  __
  / ___/__  // / / /
  \__ \ /_ </ /_/ /
 ___/ /__/ / __  /
/____/____/_/ /_/


Please choose execution mode:


[1] SSH server autoconfig (recommended)
[2] Manual SSH server configuring
[3] Restore from backup (only after first successfull s3h run)
[4] Exit


Execution mode:
```

Рисунок 3.2 — Вибір режиму роботи в програмі налаштування сервера

Таким чином, програма є одночасно простою в застосуванні для користувачів, що не мають навичок системного адміністрування та гнучкою для досвідчених адміністраторів.

3.3.2 Реалізація внесення налаштувань

Програма під час своєї роботи вносить зміни до конфігураційного файла SSH сервера (/etc/ssh/sshd_config), таким чином виконуючи налаштування. Список налаштувань, що вносяться скриптом (Рисунок 3.3):

1. Перенесення SSH сервера на інший порт, відмінний від 22 (за вибором користувача).
2. Вимикає парольну автентифікацію і доступ root користувача до сервера.

3. Налаштовує ClientAliveInterval та ClientAliveCountMax для переривання неактивних з’єднань.
4. Налаштовує оптимальні алгоритми шифрування та гешування, в тому числі прибирає алгоритм ChaCha20, який використовувався для Terrapin Attack.
5. Встановлює рівень логування (LogLevel) на значення VERBOSE, для монітрингу роботи SSH сервера.
6. Налаштовує Banner Message — повідомлення, що виводиться на екран клієнта під час підключення до сервера. У ручному режимі користувач може сам вибрати текст для повідомлення.
7. Налаштування білого списку — переліку користувачів, яким дозволено заходити на SSH сервер. Його може бути введено в інтерактивному режимі.
8. В будь-якому режимі забороняється використання пустих паролів.
9. Забороняється використання SSH протоколу першої версії.
10. Налаштовує значення для MaxAuthTries, яке регулює максимальну кількість спроб автентифікації.

```
Execution mode: 1
Config backup created at /var/lib/s3h/sshd_config_backup
Port in SSH config file set to 2222
PermitRootLogin in SSH config file set to no
PasswordAuthentication in SSH config file set to no
Set up public key authentication to gain access to server!
Setting client alive intervals...
Setting optimal cryptographic algorithms...
Optimal cryptographic algorithms are already set.
Setting log level to verbose...
Setting banner text to default...
Whitelist is detected in the configuration file. Do you want to change it? (y/n)
 [y]: n
Skipping whitelist creation...
Endlessh is already installed and set up. Skipping tarpit setup. Run 'sudo syste
mctl stop endlessh && sudo rm /usr/local/bin/endlessh && sudo rm /etc/systemd/sy
stem/endlessh.service' to reinstall
Restarting Fail2ban...
Fail2ban restarted successfully.
```

Рисунок 3.3 — Застосування налаштувань до SSH сервера

3.3.3 Додаткові можливості

Також програма має декілька додаткових можливостей, направлених на підвищення захищеності сервера.

По-перше, це резервне копіювання попередньої версії конфігураційного файлу. Під час запуску програми, вона робить копію поточного стану файлу /etc/ssh/sshd_config, та зберігає у власній директорії — /var/lib/s3h. Таким чином, у разі внесення некоректних або небажаних налаштувань, користувач може застосувати додатковий режим роботи — відновлення з резервного файлу і повернути попередні налаштування сервера.

По-друге, програма також встановлює та налаштовує дві додаткові утиліти: endlessh та fail2ban.

Endlessh [26] — це SSH ханіпот, що слухає з’єднання на порту, вибраному користувачем (рекомендується 22) та у разі встановлення з’єднання починає посилати нескінченно довге повідомлення баннера, що не несе жодної корисної інформації, таким чином «блокуючи» зловмисника, поки він сам не вирішить розірвати з’єднання.

Fail2ban [27] — відома утиліта для протидії атакам грубої сили. Відслідковує підозрілу активність в мережі і самостійно конфігурує правила мережного екрана для блокування зловмисника. При використанні автентифікації за ключами необхідність в даній утиліті відпадає. Проте в залежності від потреб організації парольна автентифікація може бути актуальною або використовуватися як частина MFA. Адміністратор може вирішувати, чи потрібно йому встановлювати дану утиліту на конкретний SSH сервер.

Висновки до розділу 3

В даному розділі було розроблено організаційні та технічні рекомендації для забезпечення безпеки SSH серверів в мережі організації. Заходи було створено

на основі аналізу проблем, вразливостей та завдань, наведеного в 2 розділі даної роботи.

Організаційні рекомендації в основному концентруються на процесі автентифікаї під час встановлення SSH з‘єднання, а саме на виборі методу автентифікації в залежності від потреб та завдань компанії. Було зроблено висновок, що метод з використанням відкритих ключів є універсальним та підходить для більшості сценаріїв застосування протоколу в мережі. Особливу увагу було приділено процесам управління ключами клієнтів. Правильні генерація, розповсюдження, моніторинг та видалення ключів значно знижують ризики компрометації закритих ключів і, як наслідок, SSH сервера. Крім того, було також розглянуто варіанти використання методу з відкритими ключами для автентифікації автоматизованих процесів і запропоновано обмеження та правила, які необхідно вводити для запобігання зловживанням.

Технічні засоби захисту SSH серверів спрямовані на протидію поширеним атакам з одночасним забезпеченням достатнього рівня зручності користування для звичайних користувачів. Головним чином запропоновані методи полягають у встановленні налаштувань, що найкращим чином впливають на безпеку сервера. Конфігурація — це швидкий та простий спосіб підвищити захищеність для системного адміністратор, що не потребує закупівлі дорогого додаткового обладнання або налаштування комплексних програмних рішень.

Для автоматизації процесу налаштування було розроблено скрипт мовою Python, який завдяки наявності двох режимів роботи може використовуватися як системними адміністраторами, що бажають спростити та прискорити свою роботу, так і звичайними користувачами, які хочуть захистити свій SSH сервер, призначений для персонального використання, від зловмисників.

ВИСНОВКИ

В ході виконання дипломної роботи було детально розглянуто особливості функціонування протоколу SSH: механізмів встановлення з'єднання, захисту даних, що передаються захищеним каналом, основних та додаткових функцій. Також проаналізовано та порівняно між собою криптографічні алгоритми, що застосовуються для забезпечення безпеки каналу зв’язку для виявлення тих з них, яким варто надавати первагу. Розглянуто процес SSH рукостискання, який є одним з найважливіших у роботі протоколу, оскільки на цьому етапі між сторонами ще не встановлено захищеного з’єднання. Виявлено, яким чином зловмисники можуть втручатися в цей процес та як вони можуть атакувати сервер, використовуючи його.

Виділено поширені організаційні помилки та недоліки роботи з SSH протоколом в мережах організацій. Зокрема, виділено проблему управління ключами, що використовуються в процесі автентифікації за відкритим ключем. До неї можна віднести процес розростання ключів та утворення неактивних та закинутих ключів, що створює небезпеку компрометації та значно збільшує навантаження на системного адміністратора мережі.

Також було проаналізовано поширені атаки, яким піддаються SSH сервери, такі як brute-force, man-in-the-middle, DDoS тощо. Розглянуто природу цих атак, а також методи, які застосовуються для їх проведення для розробки оптимальних методів протидії. Крім того, розглянуто реальні актуальні вразливості, знайдені в протоколі SSH з бази вразливостей CVE з метою виділення причин появи даних вразливостей та їх усунення в подальшому.

В результаті аналізу організаційних та технічних проблем безпеки SSH серверів та SSH інфраструктури в мережі в цілому, було створено рекомендаційні заходи та програмне рішення (скрипт) для підвищення рівня захищеності серверів в організації та автоматизації, полегшення роботи системного або мережного адміністратора.

Організаційні заходи спрямовані на подолання проблем, пов’язаних з автентифікацією за відкритим ключем. Серед них можна виділити впровадження централізованого контролю за ключами в організації, ротацію ключів, створення політик копіювання ключів на інші машини та введення режиму доступу до файлу з авторизованими ключами на SSH серверах.

Скрипт, написаний мовою Python, створений для автоматичного застосування налаштувань безпеки SSH сервера, які було отримано шляхом аналізу поширених атак та підбору заходів для протидії ним. Він дозволяє швидко конфігурувати SSH сервери, що може значно полегшити адміністрування та підвищити рівень захищеності.

Поєднання організаційних та технічних засобів, запропонованих у даній роботі, дозволить забезпечити надійний захист SSH серверів від багатьох типів атак, спростити отримання доступу до серверів для звичайних працівників компанії та отримати повний контроль над обігом SSH ключів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Barrett D. SSH, The Secure Shell: The Definitive Guide Second Edition [Текст] / D. Barrett, R. Silverman, R. Byrnes. — O'Reilly Media, Inc, 2005. — 564 с.
2. Lucas M. SSH Mastery Second edition. [Текст] / M. Lucas — Tilted Windmill Press, 2018. — 242 с.
3. Rimuhosting. Preventing Brute Force SSH Attacks [Електронний ресурс] / Rimuhosting — Режим доступу: https://rimuhosting.com/knowledgebase/linux/misc/preventing-brute-force-ssh-attacks
4. Emtec. Comparison of SSH v2 vs SSH v1 [Електронний ресурс] / Emtec — Режим доступу: https://www.emtec.com/ssh/ssh-v2.html
5. ElNiak. SSH3: Revolutionising Secure Shell Protocol [Електронний ресурс] / Medium — Режим доступу: https://medium.com/@elniak/ssh3-revolutionising-secure-shell-protocol-067627567cec
6. Beling J. Conducting SSH Man in the Middle attacks with sshmitm [Електронний ресурс] / GIAC Certifications — Режим доступу: https://www.giac.org/paper/gsec/2034/conducting-ssh-man-middle-attacks-sshmitm/103515
7. McNab C. Network Security Assessment [Текст] / McNab C. — O'Reilly Media, Inc, 2017. — 493 с.
8. OpenSSH. Official website [Електронний ресурс] / OpenSSH — Режим доступу: https://www.openssh.com
9. Linux.die.net. sshd_config man page. [Електронний ресурс] / linux.die.net — Режим доступу: https://linux.die.net/man/5/sshd_config
10. StrongDM Team. Comparing SSH Keys: A Comprehensive Guide (RSA, DSA, ECDSA) [Електронний ресурс] / Strongdm — Режим доступу: https://www.strongdm.com/blog/comparing-ssh-keys
11. CVE. SSH : Security Vulnerabilities [Електронний ресурс] / CVE Details — Режим доступу: https://www.cvedetails.com/vulnerability-list/vendor_id-120/SSH.html
12. Jogi P. What is an SSH Key? Its Features, Types, Uses and Importance [Електронний ресурс] / Ssl2buy — Режим доступу: https://www.ssl2buy.com/cybersecurity/ssh-secure-shell-key
13. MITRE ATT&CK. Valid Accounts [Електронний ресурс] / MITRE — Режим доступу: https://attack.mitre.org/techniques/T1078/

14. MITRE ATT&CK. Lateral Movement [Електронний ресурс] / MITRE — Режим доступу: https://attack.mitre.org/tactics/TA0008/
15. Bäumer F. CVE-2024-31497: Secret Key Recovery of NIST P-521 Private Keys Through Biased ECDSA Nonces in PuTTY Client [Електронний ресурс] / Openwall — Режим доступу: https://www.openwall.com/lists/oss-security/2024/04/15/6
16. Soltani C.B. Why Poorly Secured Linux SSH Servers Are a Vulnerability Crisis? [Електронний ресурс] / Medium — Режим доступу: https://medium.com/@cbochras/why-poorly-secured-linux-ssh-servers-are-a-vulnerability-crisis-96e05ca34a79
17. Piwowarek G. How to Secure an SSH Server in Linux [Електронний ресурс] / Baeldung — Режим доступу: https://www.baeldung.com/linux/secure-ssh-server
18. AbuseIPDB. Abusive IPs [Електронний ресурс] / AbuseIPDB — Режим доступу: https://www.abuseipdb.com/
19. Nazrul S.S. Common SSH Protocol Vulnerabilities [Електронний ресурс] / Medium — Режим доступу: https://medium.com/@sadatnazrul/common-ssh-protocol-vulnerabilities-43d7d7f3e0a9
20. Bäumer F. Terrapin Attack: Breaking SSH Channel Integrity By Sequence Number Manipulation [Електронний ресурс] / Terrapin Attack — Режим доступу: https://terrapin-attack.com/TerrapinAttack.pdf
21. Richard M. Stairwell threat report: Vulnerable PuTTY SSH libraries (CVE-2024-31497) [Електронний ресурс] / Stairwell — Режим доступу: https://stairwell.com/resources/stairwell-threat-report-vulnerable-putty-ssh-libraries-cve-2024-31497/
22. Ylonen T. Security of Interactive and Automated Access Management Using Secure Shell (SSH) [Електронний ресурс] / NISTIR 7966 — Режим доступу: https://nvlpubs.nist.gov/nistpubs/ir/2015/nist.ir.7966.pdf
23. Garn D. Eight ways to protect SSH access on your system [Електронний ресурс] / Redhat — Режим доступу: https://www.redhat.com/sysadmin/eight-ways-secure-ssh
24. SSH Academy. sshd_config - How to Configure the OpenSSH Server? [Електронний ресурс] / SSH Academy — Режим доступу: https://www.ssh.com/academy/ssh/sshd_config
25. Saive R. How to Show Security Warning Message to SSH Unauthorized Users [Електронний ресурс] / Tecmint — Режим доступу: https://www.tecmint.com/ssh-warning-message-before-login/
26. Skeeto. Endless: an SSH tarpit [Електронний ресурс] / Github — Режим доступу: https://github.com/skeeto/endlessh

27. Fail2ban. Fail2Ban: ban hosts that cause multiple authentication errors [Електронний ресурс] / Github — Режим доступу: https://github.com/fail2ban/fail2ban

ДОДАТОК А

Реалізація функцій програми

```
import subprocess
import re
import os
import shutil
import configparser
import pyfiglet
import colorama

colorama.init(autoreset=True)

DEFAULT_PORT = 2222
CONFIG_FILE_PATH = '/etc/ssh/sshd_config'
DEFAULT_BACKUP_DIR = '/var/lib/s3h'


def read_file(file_path):
    with open(file_path, 'r') as file:
        return file.read()

def write_file(file_path, file_content):
    with open(file_path, 'w') as w:
        w.write(file_content)


def ask_permission(message, default_input):
    valid_inputs = {'yes', 'no', 'y', 'n'}
```

```
    while True:
        user_input = input(message).lower().strip()
        if user_input in valid_inputs:
            return user_input == 'yes' or user_input == 'y'
        if user_input == '':
            return default_input
        else:
            print("Invalid input.")

def check_sudo():
    return 'SUDO_UID' in os.environ


def hello_message():
    print('Welcome to S3H (Super Secure Shell)!')
    styled_text = pyfiglet.figlet_format('S3H', font='slant')
    print(colorama.Fore.GREEN + styled_text)


reserved_ports = [20, 21, 23, 25, 53, 80, 110, 143, 443, 3306]
def get_new_port():
    while True:
        port = input("Choose port to run SSH server on: [2222]: ")
        if port == "":
            port = 2222
            return port
        else:
            try:
                port = int(port)
```

```
            if port < 1 or port > 65535:
                raise ValueError("bounds_error")

            if port in reserved_ports:
                raise ValueError('another_service_error')

            if port == 22:
                confirm = ask_permission("Port 22 is the default SSH port and might not
be safe. Are you sure you want to use it? (y/n) [n]: ", False)
                if confirm:
                    return port
                continue

            return port
        except ValueError as e:
            if "invalid literal" in str(e):
                print("Invalid port number. Please enter a valid integer port number.")
            if "bounds_error" in str(e):
                print('Port number must be between 1 and 65535')
            if "another_service_error" in str(e):
                print("This port might be used by another service. Using it may cause
problems.")



def update_port(port):
    config_content = read_file(CONFIG_FILE_PATH)
    config_content = re.sub(r'^\s*#?\s*Port\s+\d+', f'Port {port}', config_content,
flags=re.MULTILINE)
    write_file(CONFIG_FILE_PATH, config_content)
```

```
    print(f"Port in SSH config file set to {port}")


def set_permit_root_login(ask):
    if ask:
        permit_root_login = ask_permission("Do you want to permit root login? (y/n)
[n]:", False)
    else:
        permit_root_login = False

    config_content = read_file(CONFIG_FILE_PATH)

    if permit_root_login:
        config_content = re.sub(r'^\s*#?\s*PermitRootLogin\s+\w+', 'PermitRootLogin
yes', config_content, flags=re.MULTILINE)
    else:
        config_content = re.sub(r'^\s*#?\s*PermitRootLogin\s+\w+', 'PermitRootLogin no',
config_content, flags=re.MULTILINE)

    write_file(CONFIG_FILE_PATH, config_content)
    print(f"PermitRootLogin in SSH config file set to {'yes' if permit_root_login else
'no'}")


def set_permit_password_login(ask):
    if ask:
        permit_password_login = ask_permission("Do you want to permit password login?
(y/n) [n]:", False)
    else:
        permit_password_login = False
```

```
    config_content = read_file(CONFIG_FILE_PATH)

    if permit_password_login:
        config_content = re.sub(r'^\s*#?\s*PasswordAuthentication\s+\w+',
'PasswordAuthentication yes', config_content, flags=re.MULTILINE)
    else:
        config_content = re.sub(r'^\s*#?\s*PasswordAuthentication\s+\w+',
'PasswordAuthentication no', config_content, flags=re.MULTILINE)

   # empty passwords always no
    config_content = re.sub(r'^\s*#?\s*PermitEmptyPasswords\s+\w+',
'PermitEmptyPasswords no', config_content, flags=re.MULTILINE)

    write_file(CONFIG_FILE_PATH, config_content)
    print(f"PasswordAuthentication in SSH config file set to {'yes' if
permit_password_login else 'no'}")
    if not permit_password_login:
        print('Set up public key authentication to gain access to server!')


def create_backup(source_file=CONFIG_FILE_PATH,
backup_dir=DEFAULT_BACKUP_DIR):
    if not os.path.exists(backup_dir):
        os.makedirs(backup_dir)

    file_name = 'sshd_config_backup'
    backup_file = os.path.join(backup_dir, file_name)
    shutil.copy(source_file, backup_file)
    print(f"Config backup created at {backup_file}")
```

```
    return backup_file


def restore_from_backup(backup_file='/var/lib/s3h/sshd_config_backup',
destination_file='/etc/ssh/sshd_config'):
    if not os.path.exists('/var/lib/s3h/'):
        print('No backup directory found')
        return False

    if not os.path.isdir('/var/lib/s3h'):
        print('Backup path is not a directory')
        return False

    files = os.listdir('/var/lib/s3h')

    if files:
        try:
            shutil.copy(backup_file, destination_file)
            print(f'Successfully restored from {backup_file}')
            return True
        except Exception as e:
            print(f"Error restoring from backup: {e}")
            return False
    else:
        print('No files found in backup directory')
        return False


def set_banner(ask):
    if ask:
```

```
        allow_banner = ask_permission("Do you want to set banner text? (y/n) [y]: ", True)
    else:
        allow_banner = False

    config_content = read_file(CONFIG_FILE_PATH)
    config_content = re.sub(r'^\s*#?\s*Banner\s+\S+', 'Banner /etc/issue.net',
config_content, flags=re.MULTILINE)
    write_file(CONFIG_FILE_PATH, config_content)

    if not allow_banner:
        write_file('/etc/issue.net', 'This server is protected by S3H.' + '\n')
        print('Setting banner text to default...')
    else:
        try:
            subprocess.run(['nano', '/etc/issue.net'])
            print('Banner text set successfully')
        except Exception as e:
            print(f'Error setting banner text: {e}')


def set_timeouts():
    config_content = read_file(CONFIG_FILE_PATH)
    print('Setting client alive intervals...')
    config_content = re.sub(r'^\s*#?\s*ClientAliveInterval\s+\d+', 'ClientAliveInterval
300', config_content, flags=re.MULTILINE)
    config_content = re.sub(r'^\s*#?\s*ClientAliveCountMax\s+\d+',
'ClientAliveCountMax 3', config_content, flags=re.MULTILINE)
    config_content = re.sub(r'^\s*#?\s*TCPKeepAlive\s+\w+', 'TCPKeepAlive yes',
config_content, flags=re.MULTILINE)
    write_file(CONFIG_FILE_PATH, config_content)
```

```
def set_cryptography():
    print('Setting optimal cryptographic algorithms...')
    optimal_ciphers = [
        "Ciphers aes128-ctr,aes192-ctr,aes256-ctr",
        "HostKeyAlgorithms
ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,ssh-rsa,ssh-dss",
        "KexAlgorithms
ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-excha
nge-sha256",
        "MACs hmac-sha2-256,hmac-sha2-512"
    ]

    config_content = read_file(CONFIG_FILE_PATH)
    present_ciphers = [cipher for cipher in optimal_ciphers if cipher in config_content]

    if len(present_ciphers) == len(optimal_ciphers):
        print("Optimal cryptographic algorithms are already set.")
        return

    with open(CONFIG_FILE_PATH, 'a') as file:
        file.write('\n')
        for cipher in optimal_ciphers:
            if cipher not in present_ciphers:
                file.write(cipher + '\n')

    print("Optimal cryptographic algorithms added successfully.")
```

```
def set_logging():
    config_content = read_file(CONFIG_FILE_PATH)
    print('Setting log level to verbose...')
    config_content = re.sub(r'^\s*#?\s*LogLevel\s+\w+', 'LogLevel VERBOSE',
config_content, flags=re.MULTILINE)
    write_file(CONFIG_FILE_PATH, config_content)

def set_protocol():
    config_content = read_file(CONFIG_FILE_PATH)
    print('Disabling SSH-1...')
    if 'Protocol 2' in config_content:
        return


    with open(CONFIG_FILE_PATH, 'a') as w:
        w.write('Protocol 2\n')




def set_maxauthtries():
    config_content = read_file(CONFIG_FILE_PATH)

    while True:
        max_tries = input("Enter max auth tries value: [3]: ")
        if max_tries == "":
            max_tries = 3
            break
        else:
            try:
```

```
            max_tries = int(max_tries)
            if max_tries < 1:
                raise ValueError("bounds_error")
            break
        except ValueError as e:
            if "invalid literal" in str(e):
                print("Invalid max tries number. Please enter a valid integer max tries
number.")
            if "bounds_error" in str(e):
                print('Max tries number must be 1 or more')


    config_content = re.sub(r'^\s*#?\s*MaxAuthTries\s+\d+', f'MaxAuthTries
{max_tries}', config_content, flags=re.MULTILINE)
    write_file(CONFIG_FILE_PATH, config_content)
    print(f'MaxAuthTries set to {max_tries}')


def set_whitelist():
    valid_inputs = {'yes', 'no', 'y', 'n'}
    default_input = True
    allow_whitelist = True

    while True:
        config_content = read_file(CONFIG_FILE_PATH)
        allow_users_line = [line.strip() for line in config_content.split('\n') if
line.strip().startswith('AllowUsers') and not line.strip().startswith('#')]

        if allow_users_line:
```

```
        change_whitelist_input = input("Whitelist is detected in the configuration file.
Do you want to change it? (y/n) [y]: ").lower().strip()
        if change_whitelist_input in valid_inputs:
            allow_whitelist = change_whitelist_input == 'yes' or change_whitelist_input
== 'y'
            if not allow_whitelist:
                print('Skipping whitelist creation...')
                return
            break
        elif not change_whitelist_input:
            allow_whitelist = default_input
            break
        else:
            print("Invalid input. Please enter 'yes' or 'no'.")
            continue
    else:
        allow_whitelist = True
        break

  if allow_whitelist:
    while True:
        users_input = input("Enter a space-separated list of usernames: ").strip()
        users = [user.strip() for user in users_input.split(' ')]
        invalid_users = [user for user in users if not user.isalnum()]

        if invalid_users:
            print(f"Error: Invalid usernames detected: {', '.join(invalid_users)}")
            retry_input = input("Do you want to retry entering usernames? (y/n) [y]:
").lower().strip()
            if retry_input not in valid_inputs:
```

```
                print("Invalid input.")
                continue
            elif retry_input == 'yes' or retry_input == 'y':
                continue
            else:
                print("Skipping whitelist creation...")
                return
        else:
            break

    config_content = re.sub(r'^\s*#?\s*AllowUsers\s+.*', f'AllowUsers {"
".join(users)}', config_content, flags=re.MULTILINE)
    write_file(CONFIG_FILE_PATH, config_content)
    print("Whitelisted users added successfully.")
else:
    print('Skipping whitelist creation...')



def set_up_tarpit(ask):
    if os.path.exists("/usr/local/bin/endlessh") and
os.path.exists("/etc/systemd/system/endlessh.service"):
        print("Endlessh is already installed and set up. Skipping tarpit setup. Run 'sudo
systemctl stop endlessh && sudo rm /usr/local/bin/endlessh && sudo rm
/etc/systemd/system/endlessh.service' to reinstall")
        return

    if ask:
        allow_tarpit = ask_permission('Do you want to set up Endlessh? (y/n) [y]: ', True)
    else:
        allow_tarpit = True
```

```
    if allow_tarpit:
        while True:
            tarpit_port = input("Enter the port to run Endlessh on (22 is recommended) [22]: ").strip() or '22'
            try:
                tarpit_port = int(tarpit_port)
                if tarpit_port < 1 or tarpit_port > 65535:
                    raise ValueError
                break
            except ValueError:
                print("Invalid port number. Please enter a valid integer port number.")

        subprocess.run(['sudo', 'systemctl', 'stop', 'ssh'])
        if not os.path.exists("endlessh"):
            os.system("git clone https://github.com/skeeto/endlessh.git")
        os.chdir("endlessh")

        if not os.path.exists("/usr/local/bin/endlessh"):
            subprocess.run(["make"])
            os.system('sudo mv endlessh /usr/local/bin/')

        if not os.path.exists("/etc/systemd/system/endlessh.service"):
            os.system('sudo cp util/endlessh.service /etc/systemd/system/')
            os.system("sudo setcap 'cap_net_bind_service=+ep' /usr/local/bin/endlessh")

            service_config = read_file('/etc/systemd/system/endlessh.service')
            service_config = re.sub(r'^\s*#?\s*AmbientCapabilities=CAP_NET_BIND_SERVICE\s+\S+',
```

```
'AmbientCapabilities=CAP_NET_BIND_SERVICE', service_config,
flags=re.MULTILINE)
        service_config = re.sub(r'^\s*#?\s*PrivateUsers=true\s+\S+',
'#PrivateUsers=true', service_config, flags=re.MULTILINE)

        write_file('/etc/systemd/system/endlessh.service', service_config)

    if not
os.path.exists("/etc/systemd/system/multi-user.target.wants/endlessh.service"):
        os.system('sudo systemctl --now enable endlessh')

    config_dir = '/etc/endlessh'
    if not os.path.exists(config_dir):
        os.makedirs(config_dir)

    config_path = os.path.join(config_dir, 'config')
    if not os.path.exists(config_path):
        with open(config_path, 'w') as file:
            file.write(f'Port {tarpit_port}\n')
            file.write('LogLevel 1\n')

    if subprocess.run(['systemctl', 'is-active', '--quiet', 'endlessh']).returncode != 0:
        os.system('sudo systemctl restart endlessh')
    os.system('sudo systemctl daemon-reload')
    print(f'Endlessh setup complete. Running on port {tarpit_port}')
  else:
    print('Skipping setting up tarpit...')


def set_up_fail2ban(ask, port):
```

```
    if ask:
        allow_fail2ban = ask_permission("Do you want to set up fail2ban (brute-force
protection)? (y/n) [y]: ", True)
    else:
        allow_fail2ban = True
    if not allow_fail2ban:
        print('Skipping fail2ban set up...')
        return

    fail2ban_installed = os.path.exists('/etc/fail2ban')

    if not fail2ban_installed:
        print("Fail2ban is not installed. Installing...")
        subprocess.run(['sudo', 'apt', 'update'])
        subprocess.run(['sudo', 'apt', 'install', 'fail2ban', '-y'])
        print("Fail2ban installed successfully.")

    fail2ban_config_path = '/etc/fail2ban/jail.conf'
    config = configparser.ConfigParser()
    config.optionxform = str
    config.read(fail2ban_config_path)

    if 'sshd' in config:
        config['sshd']['port'] = str(port)
    else:
        print("Warning: 'sshd' section not found in Fail2ban configuration.")
        return

    with open(fail2ban_config_path, 'w') as file:
        config.write(file)
```

```
print("Restarting Fail2ban...")
subprocess.run(['sudo', 'systemctl', 'restart', 'fail2ban'])
print("Fail2ban restarted successfully.")
```

ДОДАТОК Б

Реалізація двох режимів роботи програми

```
def main():
    hello_message()

    if not check_sudo():
        print('Please execute this script with sudo')
        return 1

    print('Please choose execution mode:\n')
    print('[1] SSH server autoconfig (recommended)')
    print('[2] Manual SSH server configuring')
    print('[3] Restore from backup (only after first successfull s3h run)')
    print('[4] Exit\n')
    option = int(input('Execution mode: '))

    if (option not in [1, 2, 3, 4]):
        print('Invalid option. Terminating...')
        return 1

    if option == 1:
        create_backup()
        update_port(DEFAULT_PORT)
        set_permit_root_login(ask=False)
        set_permit_password_login(ask=False)
        set_timeouts()
        set_protocol()
        set_cryptography()
```

```
    set_logging()
    set_maxauthtries()
    set_banner(ask=False)
    set_whitelist()
    set_up_tarpit(ask=False)
    set_up_fail2ban(True, DEFAULT_PORT)
    subprocess.run(['sudo', 'systemctl', 'restart', 'ssh'])

elif option == 2:
    create_backup()
    new_port = get_new_port()
    update_port(new_port)
    set_permit_root_login(ask=True)
    set_permit_password_login(ask=True)
    set_timeouts()
    set_protocol()
    set_cryptography()
    set_logging()
    set_maxauthtries()
    set_banner(ask=True)
    set_whitelist()
    set_up_tarpit(ask=True)
    set_up_fail2ban(True, new_port)
    subprocess.run(['sudo', 'systemctl', 'restart', 'ssh'])

elif option == 3:
    restore_from_backup()

elif option == 4:
    return 0
```

```
if __name__ == "__main__":
    main()
```