МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Система аналізу налаштувань мережевого обладнання на базі технологій Cisco в контексті інформаційної безпеки»

Завідувач

випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Великодний Д.В.**

Студента групи ІК.мз – 92с **Толбатова А.В.**

СУМИ 2021

Сумський державний університет
(назва внз)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*
Спеціальність *122 комп'ютерні науки, спеціалізація «інформаційно-комунікаційні технології»*

Затверджую:
зав.кафедрою_____

_____ Довбиш А.С.
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Толбатову Андрію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Система аналізу налаштувань мережевого обладнання на базі технологій Cisco в контексті інформаційної безпеки*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) *19.01.2021 р.*

3. Вхідні данні до проекту (роботи) *результати науково-дослідної роботи, звіт з переддипломної практики, тези на НТК, публікації, статті, монографії, технічна документація та перелік літературних джерел з матеріалами за фахом*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Огляд існуючих рішень. Постановка задачі. 2) Аналіз і формалізація правил та політик інформаційної безпеки. 3) Інформаційне та програмне забезпечення налаштування мережевого обладнання в контексті інформаційної безпеки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис і дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 10.09.2020

Керівник _____

Завдання прийняв до виконання _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1	*Огляд існуючих рішень. Постановка задач дослідження*	14.12.2020–23.12.2020	Виконано
2	*Аналіз і формалізація правил та політик інформаційної безпеки*	24.12.2020–02.01.2021	Виконано
3	*Інформаційне та програмне забезпечення налаштування мережевого обладнання в контексті інформаційної безпеки*	03.01.2021–11.01.2021	Виконано
4	*Оформлення пояснювальної записки*	12.01.2021–18.01.2021	Виконано

Студент – дипломник _____

Керівник роботи _____

РЕФЕРАТ

Записка: 51 сторінка, 10 рисунків, 22 джерела.

Мета роботи – дослідження теоретичних основ, наукове обґрунтування методів і засобів аналізу налаштувань мережевого обладнання в контексті інформаційної безпеки.

Об'єктом дослідження є мережеве обладнання на базі технологій Cisco.

Предметом дослідження є особливості налаштування мережевого обладнання.

Методи дослідження. Для вирішення поставлених задач були застосовані методи системного аналізу і методи формалізації.

Результати – за результатами дослідження сформульовано перелік правил і політик інформаційної безпеки, які мають бути активовані в комп'ютерних мережах з використанням (впровадженням) програмного забезпечення (засобу) автоматизованої перевірки відповідних налаштувань для забезпечення інформаційної безпеки. Також проведено його тестування.

МЕРЕЖЕВЕ ОБЛАДНАННЯ, ІНФОРМАЦІЙНА БЕЗПЕКА, ПРОГРАМНИЙ ЗАСІБ, МОДЕЛЬ OSI, CISCO.

ЗМІСТ

Перелік скорочень і умовних позначень

ІБ – інформаційна безпека

ІКТ – інформаційно-комунікаційні технології

ЛОМ – локальна обчислювальна мережа

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

ВСТУП

Для будь-якого системного адміністратора одним з найважливіших завдань є забезпечення безпеки комп'ютерної мережі. Ці завдання призначені для вирішення проблем брандмауера, однак комутатори, як правило, атакуються першими, тому тепер вони мають розширені можливості інформаційної безпеки (ІБ). Це передбачає не тільки захист мережі від зовнішніх атак, але й різні атаки всередині мережі, такі як заміна сервера DHCP, DoS-атаки, підробка ARP, несанкціонований доступ тощо. Література [1-22].

У деяких випадках комутатор не може повністю захистити мережу від таких атак, тому нагальним науково-технічним завданням є побудова сучасної системи аналізу налаштувань мережевого обладнання в середовищі інформаційної безпеки на основі інформаційно-комунікаційних технологій (ІКТ) Cisco . Література [10].

Давайте зробимо висновки, що вірна конфігурація мережевого обладнання безпосередньо покращує інформаційну безпеку досліджуваних систем.

Метою роботи є дослідження теоретичних основ, наукове обґрунтування методів і засобів аналізу налаштувань мережевого обладнання в контексті інформаційної безпеки.

Об'єктом дослідження є мережеве обладнання на базі технологій Cisco.

Предметом дослідження є особливості налаштування мережевого обладнання.

Апробація результатів. Основні результати дослідження доповідалися на науково-технічних конференціях AICT, CyberConf (індексується БД scopus/wos), IMA (2016-2020 р.р.) та інших.

За результатами досліджень опубліковано монографію і статті у фахових і міжнародних виданнях.

Структура роботи. Кваліфікаційна робота магістра складається зі вступу, 3-х розділів, загальних висновків, переліку використаних джерел і посилань, додатку.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ. ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Види мереж

У цьому розділі будуть визначені та проаналізовані основні поняття комп’ютерних мереж: мережа, топологія мережі, тип мережевого обладнання. Будуть описані особливості та практичні аспекти функціонування мережевого обладнання.

1.2 Мережі та їх види

Комп’ютерна мережа - це група комп’ютерів, пов’язаних лініями зв’язку, на яких працює спеціальне програмне забезпечення.

1.1.1 Тип мережі

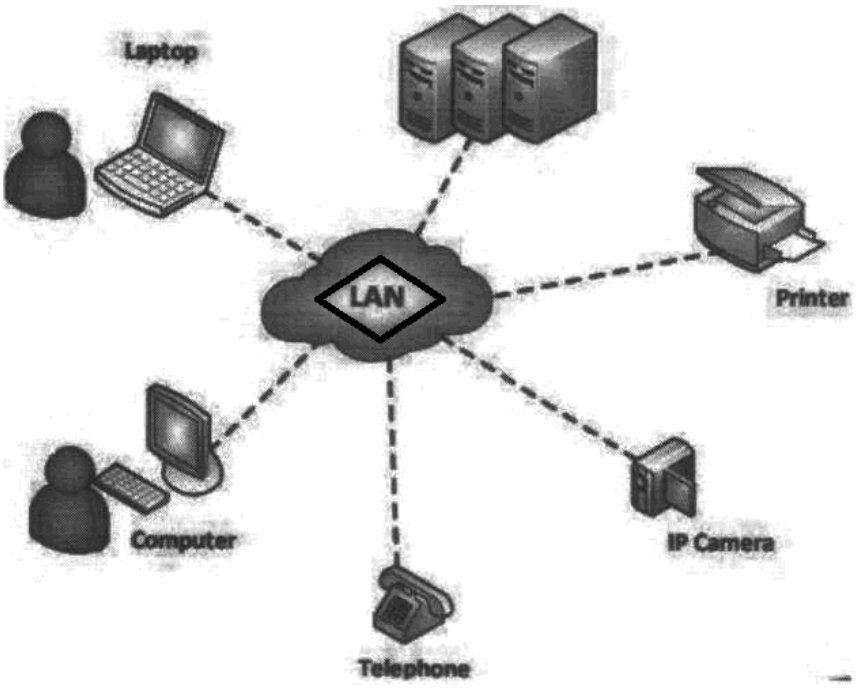
Персональна мережа (PAN) - дозволяє пристроям спілкуватися на короткі відстані. PAN поєднує в собі мишу, клавіатуру, принтер, смартфон, планшетний комп'ютер та інші пристрої. Найпоширенішою технологією підключення є Bluetooth (технологія названа на честь Харальда I, короля Bluetooth-вікінгів, який об’єднав жителів сучасної Данії та Сконе).

Інші технології, що дозволяють обмін даними на короткі відстані, також можуть бути використані для створення ПАН (наприклад, RFID-ідентифікація радіочастот - метод автоматичної ідентифікації об'єктів, в яких радіосигнали використовуються для зчитування даних, що зберігаються в транспондерах або RFID-мітках) . Література [10, 14, 19].

Локальна мережа (локальна мережа) - це комп’ютерна мережа, яка зазвичай охоплює невелику територію.

У цьому контексті термін "місцевий" стосується загального місцевого контролю (і не передбачає обов'язкової фізичної близькості між компонентами). Локальна мережа може бути поєднанням домашньої мережі, невеликого офісу чи великого корпоративного комп’ютера та іншого обладнання . Література [10, 14, 19].

Дротові з’єднання широко використовуються в локальних мережах, більшість з яких виготовлені з мідних проводів, а деякі з них є оптичними



волокнами.

Рисунок 1.1 – Приклад локальної мережі

Звичайно, дротові мережі працюють на швидкості від 100 Мбіт / с до 1 Гбіт / с. Більше сучасна локальна мережа може функціонувати на швидкостях до 10 Гбіт / с. Література [10, 14, 19].

- • Окремі комп'ютери можуть підключатися до локальних мереж або цілих мереж за допомогою каналів зв'язку . Література [10, 14, 19].

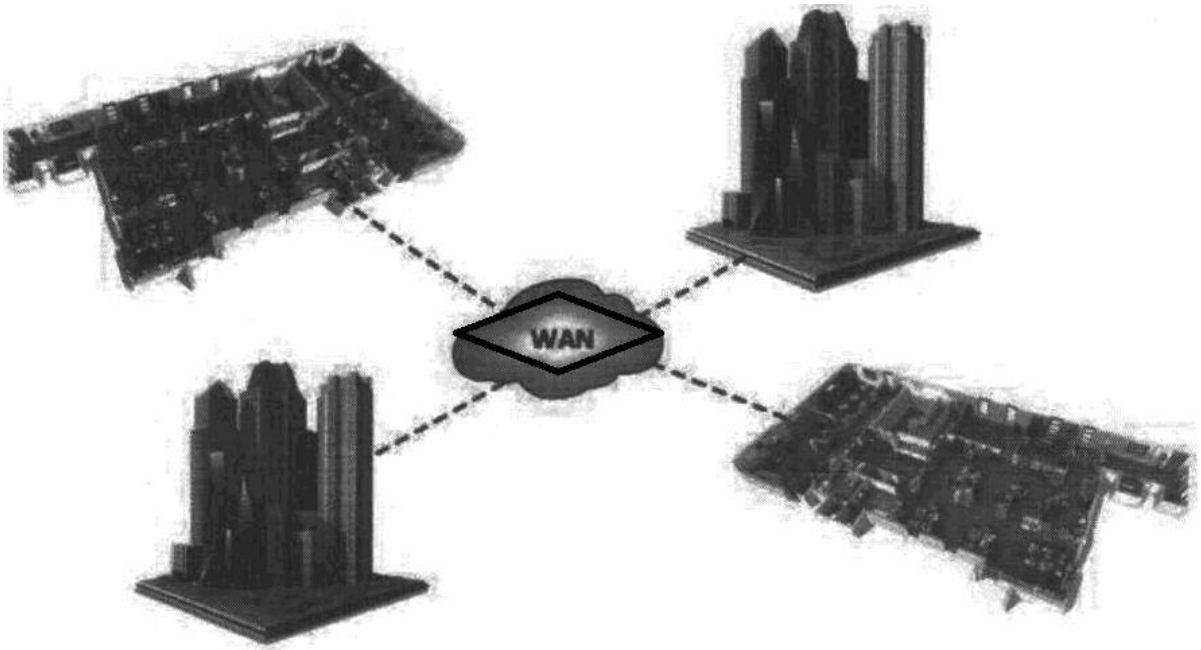


Рисунок 1.2 – Приклад глобальної мережі

1.1.2 Мережеві топології

і

• Оперативний зв'язок між комп'ютерами у локальній мережі виконується по каналах зв'язку. Вся система, залежно від фізичного з'єднання вузлів, а також найбільш геометричного розташування вузлів мережі, називається топологією мережі.

• Існують логічна та фізична топології, які не залежать одна від одної. Фізична топологія виконує геометрію побудови в мережі, а логічна встановлює в мережі для всіх потоків даних їх напрямок і спосіб передачі.

• У локальних мережах найпопулярніші фізичні топології, такі як:

• • «Автобус» (автобус);

• • «Зірка»;

• • "Каблучка";

• • а також логічний "ринг" (або Token Ring).

• Мережа з топологією шини. Тут для передачі даних використовується коаксіальний кабель (моноканальний), на його кінцях встановлені термінатори або опір терміналу. Кожен комп'ютер підключений до кабелю через Т-роз'єм (Т-роз'єм).

- Але є і недоліки:
- • обмеження кількості робочих станцій і довжини кабелю;
- • вся робота мережі може зупинитися, якщо кабель обірветься;
- • важко виявити дефекти.

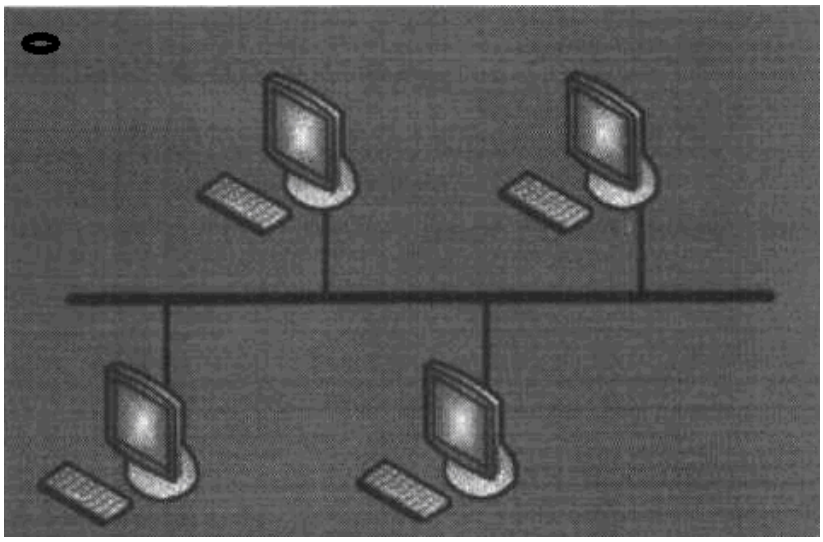


Рисунок 1.3 - Топологія “Шина”

У цій мережі кожна окрема кабельна робоча станція (кручена пара) підключена до концентратора або концентратора, який забезпечується для всіх ПК паралельним з'єднанням (усі комп'ютерні мережі можуть бути одна від одної).

Переваги цієї зіркової топології:

• легке підключення нового ПК;

• централізоване управління;

• стійкість мереж до відмов ПК;

• стійкість до розрізання окремих з'єднань

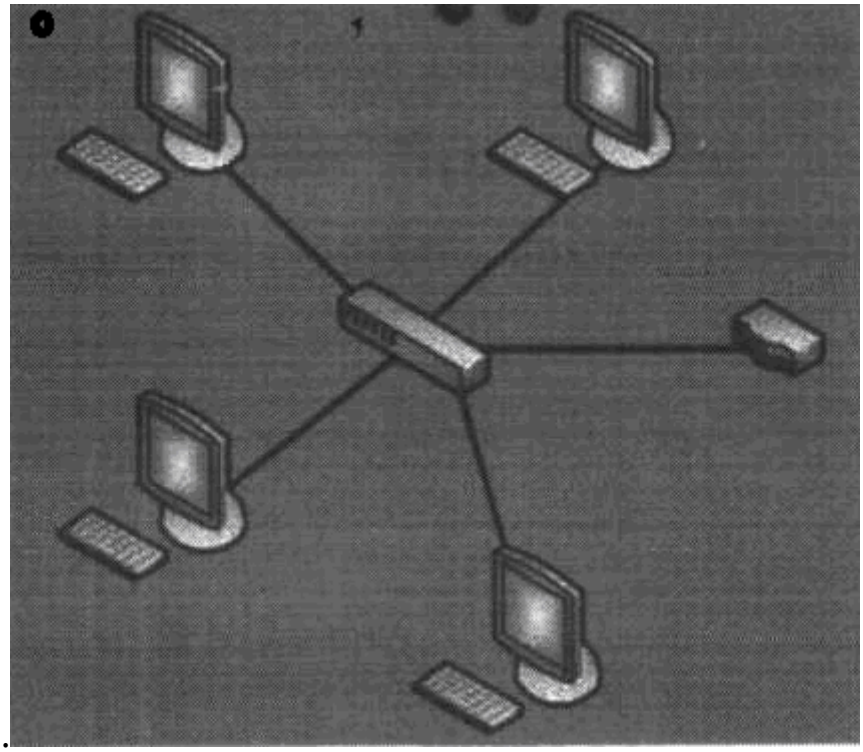


Рисунок 1.4 - Топологія “Зірка”

Єдиним недоліком кільцевої топологічної мережі є те, що якщо хоча б одна лінія зв'язку пошкоджена або виходить з ладу, ефективність всієї мережі порушується . Література [10, 14, 19].

Через деяку ненадійність у чистому вигляді цей тип топології використовується рідко. На практиці в основному використовуються модифікації різних топологій кільця.

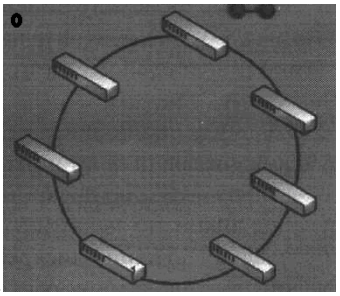


Рисунок 1.5 - Топологія “Кільце”

1.3. Постановка задач дослідження

Проаналізувавши проблеми дослідження та джерела літератури, ми

формулюємо мету та завдання подальших досліджень.

Метою роботи є вивчення теоретичних основ, дослідження збору методів та інструментів для аналізу розташування мережевого обладнання в контексті інформаційної безпеки.

Постановка цілей:

1. Аналіз та розробка правил та політик безпеки, які слід активувати на рівні регіональної мережі.

2. Визначте особливості роботи мережевого обладнання та технології для використання з ним ..

3. Розробка програм для аналізу налаштувань мережевого обладнання на відповідність зазначеним правилам.

Висновки до розділу 1

Перший розділ детально описує предмет дослідження, а саме особливості побудови мережевого обладнання та цілі дослідження. Можливі загрози ІБ можна використовувати в наступному розділі для побудови політики інформаційної безпеки.

2 АНАЛІЗ І ФОРМАЛІЗАЦІЯ ПРАВИЛ І ПОЛІТИК ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

У цьому розділі аналізуються і описуються всі правила та політики, які повинні бути встановлені на рівні регіональної мережі для досягнення рівня захисту.

2.1 Перелік правил інформаційної безпеки

Dynamic ARP verification

• Port security

• SSH settings

• EXEC wait time

• TACACS + authentication

• Disable unused services

• Logging of best practices

• Improved collection of Crashinfo files

• Network time protocol

• Re-lock the login password

• Exclusive access to configuration changes

• Flexible configuration of Cisco IOS software . Література [2, 3, 6, 7]

2.2 Аналіз поданих правил

• Списки доступу

• Основний інструмент контроля. Правильний доступ. “Правило - кожен активний список доступу повинен бути "білим" і не повинен містити більше двох "будь-яких" в одному рядку. ”

• Структура: (дозвіл / заборона) - (протокол) - (джерело ip) - (порт джерела) - (адреса призначення IP) - (порт призначення).

• “Рядок доступу дозволяє або забороняє походження трафіку

даного протоколу з певного порту хосту (або всієї мережі) до вказаного хоста (або всієї мережі)" на даному порту. Правило обережності - писати рядки там, де явно використовуються два рази. На малюнках нижче наведено приклад списку доступу, який відповідає цьому правилу чи ні.

- **"Dynamic ARP inspection"**

Динамічна інспекція "ARP (DAI) може бути використана для пом'якшення атак ARP-отруєнь на місцеві сегменти. DAI перехоплює та перевіряє зв'язок між IP та MAC-адресою всіх пакетів ARP на ненадійних портах. У середовищах DHCP DAI використовує дані, створені за допомогою функції відстеження DHCP. Пакети ARP, отримані на надійних інтерфейсах, не перевіряються, а недійсні пакети на надійних інтерфейсах відкидаються. В інших середовищах, окрім DHCP, використовується ACL." Література [2, 3, 6, 7]. "**Правило** - мають бути активовані дві команди: "

```
!
ip dhcp snooping
ip dhcp snooping vlan <vlan-range>
!
```

та

```
!
ip arp inspection vlan <vlan-range>
!
```

- **Port security**

Безпека порту використовується для управління MAC-адресою в інтерфейсі доступу. Порт безпеки може використовувати динамічно вивчені MAC-адреси для полегшення початкової конфігурації. Як тільки безпека порту виявить порушення MAC, вона може використовувати один із чотирьох режимів порушення. Ці режими призначені для захисту, обмеження, відключення та відключення VLAN. У випадках, коли порт забезпечує доступ лише до однієї робочої станції із використанням

стандартних протоколів, максимальної кількості може бути достатньо. Протоколи, які використовують віртуальні MAC-адреси, такі як HSRP, не працюють, якщо максимальна кількість встановлена в один . Література [2, 3, 6, 7].

“Правило - на кожному активному інтерфейсі має бути увімкнена фільтрація або за однією MAC-адресою, або максимум за першими 3-ма: ”

```
!
interface <interface>
  switchport
  switchport mode access
  switchport port-security
  switchport port-security mac-address sticky
  switchport port-security maximum <number>
  switchport port-security violation <violation-mode>
```

- **Налаштування SSH**

SSH працює на захищеному транспортному рівні.

“Правило - таймаут має бути меншим або рівним 9 хвилинам. ”

-

```
!
ip ssh time-out 60
ip ssh authentication-retries 3
!
ip ssh version 2
!
line vty 0 4
transport input ssh
!
```

“Правило - на пристрої має бути налаштована автентифікація TACACS+:”

```
!
aaa new-model
aaa authentication login default group tacacs+
!
tacacs-server host <ip-address-of-tacacs-server>
tacacs-server key <key>
!
```

- **“Disable Unused Services”**

  “Правило - наступні команди мають бути увімкнені: ”

```
!
no service tcp-small-servers
no service udp-small-servers
no ip finger
no ip bootp server ????
ip dhcp bootp ignore
no mop enabled ????
no ip domain-lookup
no service pad
no ip http server
no ip http secure-server
no service config
!
```

- **“Logging best practices”**

  “Правило - має бути увімкнена команда з кількістю файлів для зберігання 3. ”

```
Client:
ntp authenticate
ntp authentication-key 5 md5 ciscotime
ntp trusted-key 5
ntp server 172.16.1.5 key 5
Server:
ntp authenticate
ntp authentication-key 5 md5 ciscotime
ntp trusted-key 5
!
```

- **“Login Password Retry Lockout”**

“Правило - кількість невдалих спроб має дорівнювати”

```
!
aaa new-model
aaa local authentication attempts max-fail <max-attempts>
aaa authentication login default local
!
username <name> secret <password>
!
```

- **“Exclusive Configuration Change Access”**

Література [2, 3, 6, 7] “**Правило -** включена за замовчуванням ця функція. ”

```
!
aaa new-model
aaa local authentication attempts max-fail <max-attempts>
aaa authentication login default local
!
username <name> secret <password>
!
```

- **“Cisco IOS Software Resilient Configuration”**

Література [2, 3, 6, 7]

“Правило - наступні команди мають бути ввімкнуто. ”

```
!
secure boot-image
secure boot-config
```

Висновки до розділу 2

В другому розділі аналізується перелік правил, яких варто дотримуватися для забезпечення найкращого рівня інформаційної безпеки і захисту в цілому. В кожному підрозділі докладно аналізуються проблеми і засоби їх вирішення.

Розроблені правила щодо забезпечення інформаційної безпеки будуть використані при написанні програмного засобу.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ НАЛАШТУВАННЯ МЕРЕЖЕВОГО ОБЛАДНАННЯ В КОНТЕКСТІ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

У цьому розділі виконується і аналізується процес розробки а також тестування програмного забезпечення.

У розділі 2 запропоновано перелік правил, активація яких повинна суттєво покращити безпеку мережі від можливих загроз з боку зловмисників.

Слід зазначити, що основа запропонованого нами способу захисту мереж від несанкціонованих перешкод, шляхом правильної конфігурації обладнання CISCO, базується на найкращих тенденціях та рекомендаціях, які були сформовані та перевірені самою CISCO.

Проаналізовано декілька джерел та вибрано найбільш значущі рекомендації. Звичайно, цей список можна розширювати нескінченно, але деякі функції не можуть бути перевірені програмно. Література [2, 3, 6, 7].

Розроблено підхід, який дозволяє використовувати лише бібліотеку Paramiko.

У наступних підрозділах будуть описані найважливіші моменти та продемонстровано деякі висновки програми . Література [2, 3, 6, 7].

3.1 Функціонування програмного засобу

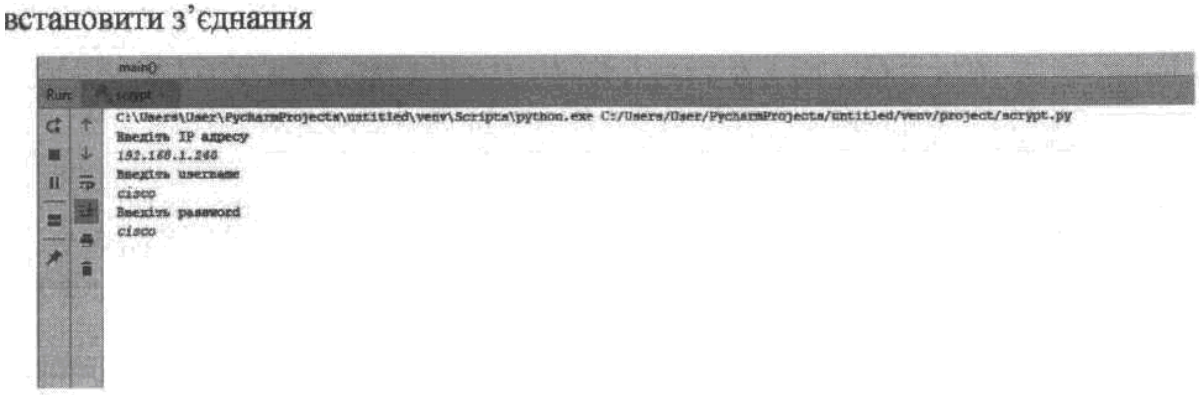


Рисунок 3.1 – Приклад форми для підключення

1. Встановлюємо SSH з’єднання

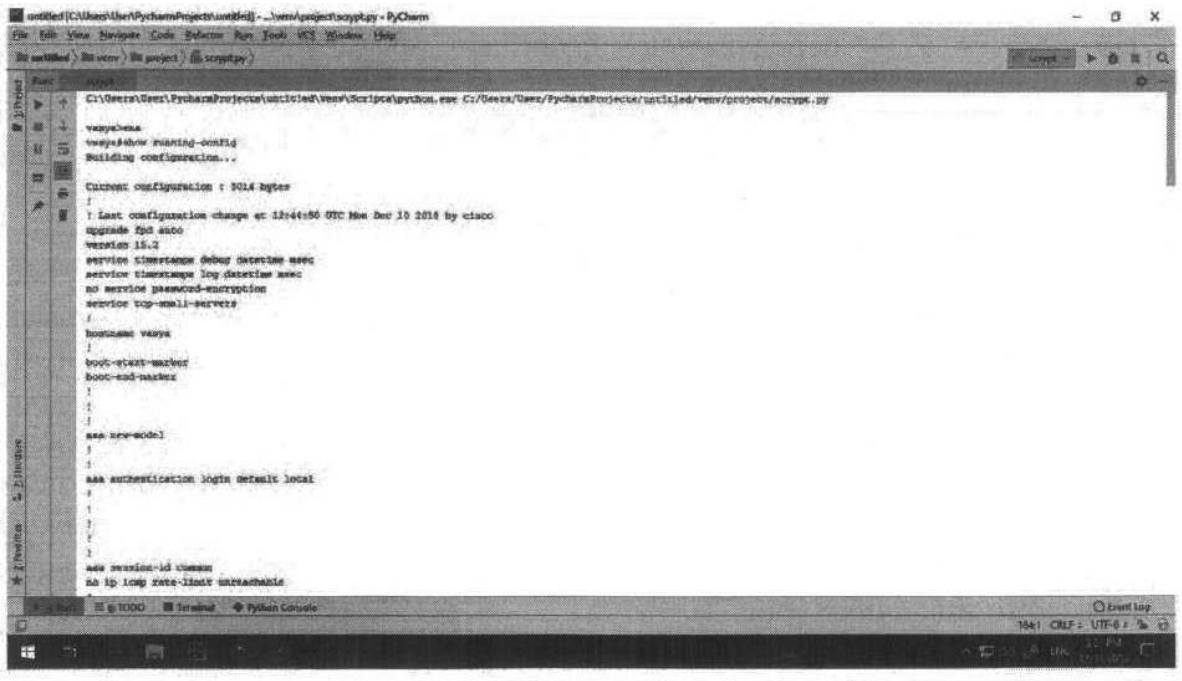


Рисунок 3.2 – Приклад частини “running-config“

2. Програма отримує конфігурації: "running-config" і розширену "running-config all"

3. Перевіряє списки активного доступу

3.2 Аналіз доступів

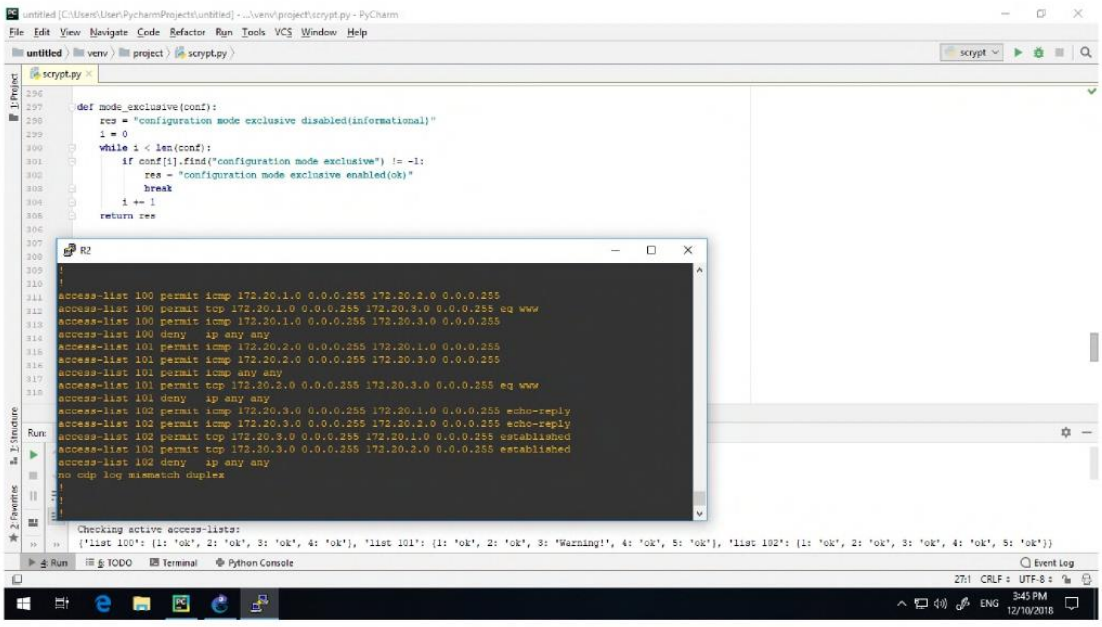


Рисунок 3.3 - Результати аналізу списків доступу

У меншому вікні ви можете побачити існуючі списки доступу в кількості 3 штуки.

У виході консолі Python результат перевірки кожного з термінів відповідно до вказаного правила. За списками доступу:

1. Все добре
2. Критична помилка рядка С
3. Все добре

3.3. Тестування розробленого програмного засобу

Для тестування було обрано симулятор GNS3, що дозволило створити віртуальні тестові маршрутизатори і комутатори . Література [3, 6, 7].

На рис. 3.5 представлено функціонування ПЗ.

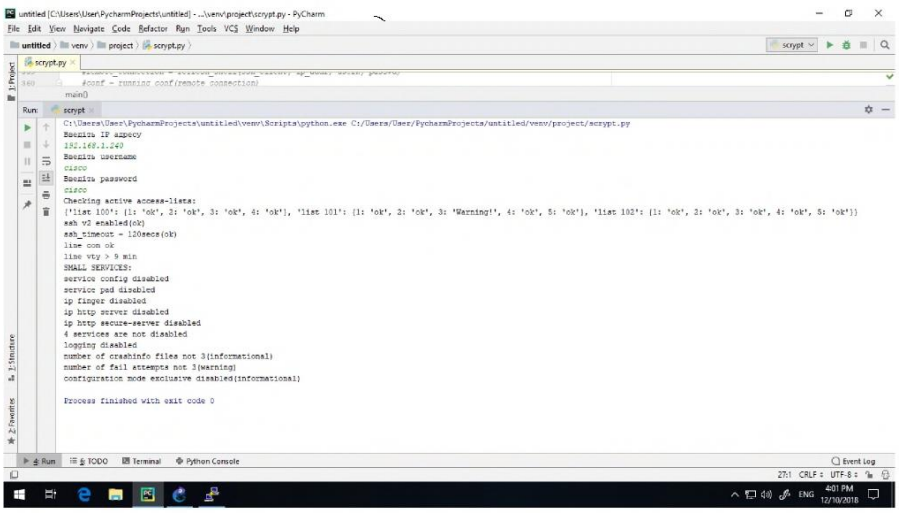


Рисунок 3.5 – Тестування програми

Висновки до розділу 3

У 3-му розділі було розроблено функції, які реалізують перевірку заданих правил і програмний засіб. Застосовано симулятор GNS3. Програма коректно реагує на усі зміни.

ЗАГАЛЬНІ ВИСНОВКИ

За результатами досліджень у кваліфікаційній роботі магістра було досягнутою мету, яка полягала у вивченні теоретичних основ, методів та засобів аналізу налаштувань мережевого обладнання в контексті інформаційної безпеки.

Під час кваліфікаційної роботи були визначені концепції та особливості роботи мережевого обладнання та супутніх ІКТ технологій, представлені і проаналізовані актуальні аспекти інформаційної безпеки. Запропоноване забезпечення, яке фактично аналізує налаштування (аудит) та видає звіт. Він також був протестований на віртуальному маршрутизаторі.

Отримане програмне забезпечення дозволяє перевірити один маршрутизатор за 10-12 секунд. Це допомагає зменшити витрачений час на кілька годин порівняно з ручною перевіркою всіх налаштувань на основі ІКТ.

Перелік використаних джерел і посилань

1. “Аналіз вимог до програмного забезпечення /І.С. Ясенова / Навчальний посібник (рекомендовано НУБіП України), Київ, Україна: НУБіП України, 2018. - 250с.”

2. “С. Гнатюк, М. Рябий, В. Лядовська, Визначення критичної інформаційної інфраструктури та її захист: аналіз підходів, *Зв'язок*, №4, С. 3-7, 2014.”

3. “О. Довгань, «Критична інфраструктура як об’єкт захисту від кібернетичних атак», *Матеріали наук.-практ. конф. «Інформаційна безпека: виклики і загрози сучасності».* К: НА СБ України, С. 17-20, 2013.”

4. “Виникнення та еволюція комп'ютерних систем бронювання. [Електронний ресурс]. Режим доступу: http://pidruchniki.com/15801117/ turizm/viniknennya_evolyutsiya_kompyuternih_sistem_bronyuvannya”

5. “S. Gnatyuk, Critical Aviation Information Systems Cybersecurity, *Meeting Security Challenges Through Data Analytics and Decision Support*, NATO Science for Peace and Security Series, D: Information and Communication Security. -IOS Press Ebooks, Vol.47, №3, pp. 308-316, 2016.”

6. “S. Gnatyuk, V. Sydorenko, M. Aleksander, Unified data model for defining state critical information infrastructure in civil aviation, *Proceedings of the 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Kyiv, Ukraine, May 24-27, 2018, pp. 37-42. ”

7. “А.В. Толбатов, Актуальні проблеми забезпечення інформаційної безпеки як функції сучасної держави / А.В. Толбатов, В.А. Толбатов, О.Б. В’юненко, Г.А. Смоляров, В.А. Ефанов / Перспективные тренды развития науки: менеджмент, юриспруденция. – Одесса: КУПРИЕНКО СВ, 2016. – С.170–180.”

8. “Толбатов А.В. Научное окружение современного человека: техника, информатика, архитектура, медицина, сельское хозяйство. Книга

2. Часть 1 : серия монографий / [авт.кол. : Линда С.Н., Львович И.Я., Преображенский А.П., Толбатов В.А., Толбатов А.В. и др.]. – Одесса: КУПРИЕНКО СВ, 2019 – 199 с.”

9. “Толбатов А.В. Інноваційна наука, освіта, виробництво і транспорт: техніка і технології, інформатика, транспорт, архітектура: монографія / [авт.кол. : В.В.Лукін, І.Я.Львович, Г.В.Пачурін, В.А.Толбатов, А.В.Толбатов та ін.]. - Одеса: КУПРІЄНКО СВ, 2019 - 180 с. : ил., табл. - (Серія «Інноваційна наука, освіта, виробництво і транспорт»; №2). ISBN 978-617-7414-78-9.”

10. “Наукове оточення сучасної людини: техніка і технології, інформатика. Книга 3. Часть 3: серія монографій / [авт.кол. : С.О. Гнатюк, А.В. Толбатов, В.А. Толбатов, С.В. Агаджанова, С.В. Толбатов та ін.]. - Одеса: КУПРІЄНКО СВ, 2020 - 83 с.”

11. “Tolbatov A. Cybersecurity of distributed information systems. The minimization of damage caused by errors of operators during group activity / Lavrov, E., Tolbatov, A., Pasko, N., Tolbatov, V. / 2017 2nd International Conference on Advanced Information and Communication Technologies, AICT 2017 – Proceedings – Lviv, 2017. – P. 83–87.”

12. “Tolbatov A. Ergonomic Support for Decision-Making Management of the Chief Information Security Officer / Sergiy Gnatyuk, Nataliia Barchenko, Olena Azarenko, Andrii Tolbatov, Victor Obodiak, Volodymyr Tolbatov / 1st International Conference on Cyber Hygiene and Conflict Management in Global Information Networks (CyberConf 2019) Lviv Ukraine, November 29, 2019. – P. 459–471. – Режим доступу: http://ceur-ws.org/Vol-2588/ (дата звернення 27.04.2020 р.).”

13. “System for monitoring the connection of USB devices for cybersecurity auditing. CEUR International Workshop Proceedings, 2020, CEUR-WS.org, online. P. 733–785.”

14. “Method of Improving the Stability of Network Synchronization in Multiservice Macro Networks. CEUR International Workshop Proceedings, 2020, CEUR-WS.org, online. P. 786–797.”

15. “Tolbatov A.Monitoring the quality of reference synchronization signals on the 4G network / Fedorova, N., Khlaponin, Y., Tolbatov, A., Odarchenko, R., Polihenko, O. // CEUR Workshop Proceedings, 2020, 2746, стр. 33–45.”

16. “Толбатов А.В. Віртуальні когнитивні центри як інтелектуальні IT системи моніторингу та оцінки роботи регіональних агропромислових комплексів / Толбатов А.В., В‘юненко О.Б., Толбатов В.А., Толбатов С.В., Агаджанова С.В. // Вимірювальна та обчислювальна техніка в технологічних процесах. – Хмельницький, 2015. –№ 2(51). – С. 112–116.”

17. “Толбатов, А.В. Дослідження процесу функціонування комплексної охоронної системи допуску / А.В. Толбатов, В.А. Толбатов, О.Ю. Калітін, О.О. Виноградова // Інформатика, математика, автоматика (IMA :: 2016) : матеріали та програма наук.-техн. конф., 18–22 квітня 2016 р. – Суми : СумДУ, 2016. – С. 156.”

18. “Tolbatov A.V. Information software providing effective activity of machine-building enterprises of ukraine in resources restrictions / Chuprina M.O, Tolbatov A.V., Viunenko O.B., Tolbatov V.A. / Modern engineering and innovative technologies Issue №7, Part 3, 2019 – P. 84-90.”

19. “Viunenko O. B. Analysis of the level of safety in e-learning systems / Viunenko O. B., Tolbatov A.V. // International periodic scientific journal Modern engineering and innovative technologies Issue №14 Part.2 November 2020 P. 35–43.”

20. “Tolbatov, A.V. Development concept modeling of business processes of modern industrial enterprises in terms of theoretical and legal approaches to the analysis information security [Текст] / A.V. Tolbatov, V.A.

Tolbatov // Вимірювальна та обчислювальна техніка в технологічних процесах. – Хмельницький, 2017. – №1 –С.196–199.”

21. “Об’єктно-орієнтоване програмування. Навчальний посібник для студентів спеціальностей 121 – «Інженерія програмного забезпечення», 122 – «Комп’ютерні науки та інформаційні технології», 123 – «Комп’ютерна інженерія» / Міловідов Ю.О. – К.НУБіП України: 2019р. – 301с.”

22. “Системне програмне забезпечення / В.Є.Бондаренко, Навчальний посібник. - К: ДУТ, 2016. - 25 с. ”

ДОДАТОК А – програмне забезпечення (засіб) автоматизованої перевірки відповідних налаштувань для забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no ip finger") != -1:
            res += "ip finger disabled\n"
            count += 1
            i += 1
            continue
        if check.find("ip dhcp bootp ignore") != -1:
            res += "dhcp bootp ignore disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip domain-lookup") != -1:
            res += "ip domain-lookup disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service pad") != -1 and res.find("service pad") == -1:
            res += "service pad disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip http server") != -1:
            res += "ip http server disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip http secure-server") != -1:
            res += "ip http secure-server disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("  ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("  ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("  ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("  ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    time.sleep(0.05)
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  mass = string.splitlines()
  return mass

def ip_route(remote_connection):
  remote_connection.send("show ip route\n")
  time.sleep(1)
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  return string

def ssh_timeout(remote_connection):
  remote_connection.send("show ip ssh | include Authentication timeout\n")
  time.sleep(2)
  i = 0
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  mass = string.splitlines()
  check = ""
  while i < len(mass):
    if mass[i].find("Authentication timeout:") != -1:
      check = mass[i]
      break
    i += 1
  last_index = check.rfind(" secs")
  res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
  time1 = check[24:last_index]
  if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("  ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("  ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("  ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("  ")
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no ip finger") != -1:
            res += "ip finger disabled\n"
            count += 1
            i += 1
            continue
        if check.find("ip dhcp bootp ignore") != -1:
            res += "dhcp bootp ignore disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip domain-lookup") != -1:
            res += "ip domain-lookup disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service pad") != -1 and res.find("service pad") == -1:
            res += "service pad disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip http server") != -1:
            res += "ip http server disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip http secure-server") != -1:
            res += "ip http secure-server disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("  ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("  ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("  ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("  ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("  ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("  ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

```
            continue
        if check.find("no ip finger") != -1:
            res += "ip finger disabled\n"
            count += 1
            i += 1
            continue
        if check.find("ip dhcp bootp ignore") != -1:
            res += "dhcp bootp ignore disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip domain-lookup") != -1:
            res += "ip domain-lookup disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service pad") != -1 and res.find("service pad") == -1:
            res += "service pad disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip http server") != -1:
            res += "ip http server disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip http secure-server") != -1:
            res += "ip http secure-server disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("  ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("  ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    time.sleep(0.05)
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  mass = string.splitlines()
  return mass

def ip_route(remote_connection):
  remote_connection.send("show ip route\n")
  time.sleep(1)
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  return string

def ssh_timeout(remote_connection):
  remote_connection.send("show ip ssh | include Authentication timeout\n")
  time.sleep(2)
  i = 0
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  mass = string.splitlines()
  check = ""
  while i < len(mass):
    if mass[i].find("Authentication timeout:") != -1:
      check = mass[i]
      break
    i += 1
  last_index = check.rfind(" secs")
  res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
  time1 = check[24:last_index]
  if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no ip finger") != -1:
            res += "ip finger disabled\n"
            count += 1
            i += 1
            continue
        if check.find("ip dhcp bootp ignore") != -1:
            res += "dhcp bootp ignore disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip domain-lookup") != -1:
            res += "ip domain-lookup disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service pad") != -1 and res.find("service pad") == -1:
            res += "service pad disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip http server") != -1:
            res += "ip http server disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip http secure-server") != -1:
            res += "ip http secure-server disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    time.sleep(0.05)
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  mass = string.splitlines()
  return mass

def ip_route(remote_connection):
  remote_connection.send("show ip route\n")
  time.sleep(1)
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  return string

def ssh_timeout(remote_connection):
  remote_connection.send("show ip ssh | include Authentication timeout\n")
  time.sleep(2)
  i = 0
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  mass = string.splitlines()
  check = ""
  while i < len(mass):
    if mass[i].find("Authentication timeout:") != -1:
      check = mass[i]
      break
    i += 1
  last_index = check.rfind(" secs")
  res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
  time1 = check[24:last_index]
  if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("  ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("  ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

```
            continue
        if check.find("no ip finger") != -1:
            res += "ip finger disabled\n"
            count += 1
            i += 1
            continue
        if check.find("ip dhcp bootp ignore") != -1:
            res += "dhcp bootp ignore disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip domain-lookup") != -1:
            res += "ip domain-lookup disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service pad") != -1 and res.find("service pad") == -1:
            res += "service pad disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip http server") != -1:
            res += "ip http server disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip http secure-server") != -1:
            res += "ip http secure-server disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("  ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("  ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    time.sleep(0.05)
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  mass = string.splitlines()
  return mass

def ip_route(remote_connection):
  remote_connection.send("show ip route\n")
  time.sleep(1)
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  return string

def ssh_timeout(remote_connection):
  remote_connection.send("show ip ssh | include Authentication timeout\n")
  time.sleep(2)
  i = 0
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  mass = string.splitlines()
  check = ""
  while i < len(mass):
    if mass[i].find("Authentication timeout:") != -1:
      check = mass[i]
      break
    i += 1
  last_index = check.rfind(" secs")
  res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
  time1 = check[24:last_index]
  if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("  ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("  ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    time.sleep(0.05)
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  mass = string.splitlines()
  return mass

def ip_route(remote_connection):
  remote_connection.send("show ip route\n")
  time.sleep(1)
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  return string

def ssh_timeout(remote_connection):
  remote_connection.send("show ip ssh | include Authentication timeout\n")
  time.sleep(2)
  i = 0
  output = remote_connection.recv(52828)
  string = output.decode("utf-8")
  mass = string.splitlines()
  check = ""
  while i < len(mass):
    if mass[i].find("Authentication timeout:") != -1:
      check = mass[i]
      break
    i += 1
  last_index = check.rfind(" secs")
  res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
  time1 = check[24:last_index]
  if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

```
            continue
        if check.find("no ip finger") != -1:
            res += "ip finger disabled\n"
            count += 1
            i += 1
            continue
        if check.find("ip dhcp bootp ignore") != -1:
            res += "dhcp bootp ignore disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip domain-lookup") != -1:
            res += "ip domain-lookup disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service pad") != -1 and res.find("service pad") == -1:
            res += "service pad disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip http server") != -1:
            res += "ip http server disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no ip http secure-server") != -1:
            res += "ip http secure-server disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("   ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("   ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    print(auth_fails(conf))
    print(mode_exclusive(conf))
    return

def main():
    ip_addr = "192.168.1.240"
    usern = "cisco"
    passwd = "cisco"
    ssh_client = paramiko.SSHClient()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    #conf_all = running_conf_all(remote_connection)
    #remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    print(conf)
    #test_router(ssh_client)
    ssh_client.close()

main()
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
            continue
        if check.find("no service config") != -1:
            res += "service config disabled\n"
            count += 1
            i += 1
            continue
        i +=1
    res += str(9 - count) + " services are not disabled"
    return res

def logging(conf):
    res = "logging disabled"
    i = 0
    while i < len(conf):
        if conf[i].find("logging host") != -1:
            pos = conf[i].rfind(" ")
            ip = conf[i][pos+1:len(conf[i])]
            res = "logging enabled to host: " + ip
            break
        i += 1
    return res

def crashinfo(conf):
    res = "number of crashinfo files not 3(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("exception crashinfo maximum files 3") != -1:
            res = "number of crashinfo files 3 (ok)"
            break
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

```
    list.pop(len(list) - 1)
    while i < len(list):
        dict[i+1] = check_access_string(list[i])
        i+= 1
    return dict

def check_access_string(string):
    res = "ok"
    a = 10
    if "permit" in string:
        if "ip" in string:
            if string.count("any") == 1:
                res = "informational"
            if string.count("any") == 2:
                res = "Warning!"
        else:
            if string.count("any") >= 2:
                res = "Warning!"
            if string.count(".") <= 6 and res != "Warning!":
                try: last_index = string.rindex(".")
                except ValueError: a = 20
                if len(string) - last_index < 7:
                    res = "Warning!"
    return res

def active_list(remote_connection):
    remote_connection.send("show running-config | section ip access-group\n")
    time.sleep(3)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
]
    res = "arp inspection disabled"
    while i < len(mass):
        if mass[i].find("ip dhcp snooping") != -1:
            res = "dhcp snooping enabled"
            if mass[i].find("ip arp inspection") != -1:
                res = "arp inspection enabled"
        i += 1
    return res

def running_conf_all(remote_connection):
    #remote_connection.send("ena\n")
    remote_connection.send("show running-config all\n")
    i = 0
    time.sleep(5)
    while i < 30:
        remote_connection.send("  ")
        i += 1
        time.sleep(0.05)
    output = remote_connection.recv(100000)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def running_conf(remote_connection):
    remote_connection.send("show running-config\n")
    i = 0
    time.sleep(5)
    while i < 5:
        remote_connection.send("  ")
        i += 1
```

```
    list = string.splitlines()
    list.pop(len(list)-1)
    list.pop(0)
    list.pop(0)
    list.pop(0)
    i = 0
    line = ""
    res = []
    while i < len(list):
        line = list[i]
        last_index = line.rindex(" ")
        res.append(line[17:last_index])
        i += 1
    return res

def ssh_version(mass):
    i = 0
    res = "ssh disabled(informational)"
    while i < len(mass):
        if mass[i].find("ip ssh version 1") != -1:
            res = "ssh v1 you must enable ssh v2(informational)"
            break
        if mass[i].find("ip ssh version 2") != -1:
            res = "ssh v2 enabled(ok)"
            break
        i += 1
    return res

def arp_inspection(mass):
    i = 0
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        time.sleep(0.05)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    return mass

def ip_route(remote_connection):
    remote_connection.send("show ip route\n")
    time.sleep(1)
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    return string

def ssh_timeout(remote_connection):
    remote_connection.send("show ip ssh | include Authentication timeout\n")
    time.sleep(2)
    i = 0
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    check = ""
    while i < len(mass):
        if mass[i].find("Authentication timeout:") != -1:
            check = mass[i]
            break
        i += 1
    last_index = check.rfind(" secs")
    res = "ssh_timeout = " + str(check[24:last_index]) + "secs"
    time1 = check[24:last_index]
    if int(time1) <= 120:
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    res = "line con > 9 min"
    output = remote_connection.recv(52828)
    string = output.decode("utf-8")
    mass = string.splitlines()
    while i < len(mass):
        if mass[i].find("exec-timeout") != -1:
            str = mass[i]
            str = str[14:len(str)]
            pos = str.find(" ")
            str = str[0:pos]
            if int(str) <= 9:
                res = "line con ok"
        i += 1
    return res

def small_services(conf_all):
    res = "SMALL SERVICES:\n"
    i = 0
    count = 0
    while i < len(conf_all):
        check = conf_all[i]
        if check.find("no service tcp-small-servers") != -1:
            res += "tcp-small-servers disabled\n"
            count += 1
            i += 1
            continue
        if check.find("no service udp-small-servers") != -1:
            res += "udp-small-servers disabled\n"
            count += 1
            i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
        continue
    if check.find("no ip finger") != -1:
        res += "ip finger disabled\n"
        count += 1
        i += 1
        continue
    if check.find("ip dhcp bootp ignore") != -1:
        res += "dhcp bootp ignore disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip domain-lookup") != -1:
        res += "ip domain-lookup disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no service pad") != -1 and res.find("service pad") == -1:
        res += "service pad disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http server") != -1:
        res += "ip http server disabled\n"
        count += 1
        i += 1
        continue
    if check.find("no ip http secure-server") != -1:
        res += "ip http secure-server disabled\n"
        count += 1
        i += 1
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки

```
    return res

def auth_fails(conf):
    res = "number of fail attempts not 3(warning)"
    i = 0
    while i < len(conf):
        if conf[i].find("aaa local authentication attempts max-fail 3") != -1:
            res = "number of fail attempts 3(ok)"
            break
        i += 1
    return res

def mode_exclusive(conf):
    res = "configuration mode exclusive disabled(informational)"
    i = 0
    while i < len(conf):
        if conf[i].find("configuration mode exclusive") != -1:
            res = "configuration mode exclusive enabled(ok)"
            break
        i += 1
    return res

def refresh_shell(ssh_client, ip_addr, usern, passwd):
    ssh_client.close()
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def first_shell(ssh_client, ip_addr, usern, passwd):
```

```
    ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh_client.connect(hostname=ip_addr, username=usern, password=passwd)
    remote_connection = ssh_client.invoke_shell()
    remote_connection.send("ena\n")
    return remote_connection

def test_router(ssh_client):
    print("Введіть IP адресу")
    ip_addr = input()
    print("Введіть username")
    usern = input()
    print("Введіть password")
    passwd = input()
    remote_connection = first_shell(ssh_client, ip_addr, usern, passwd)
    print("Checking active access-lists:")
    print(start_test_lists(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf = running_conf(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    conf_all = running_conf_all(remote_connection)
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(ssh_version(conf))
    print(ssh_timeout(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_con(remote_connection))
    remote_connection = refresh_shell(ssh_client, ip_addr, usern, passwd)
    print(line_vty(remote_connection))
    print(small_services(conf_all))
    print(logging(conf))
    print(crashinfo(conf))
```

ПЗ (засіб) для автоматизованої перевірки відповідних налаштувань забезпечення інформаційної безпеки