Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів”

Виконав здобувач вищої освіти
IV курсу, групи КБ-22-МБ
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Килимник Д.В.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук
_____ Смірнова Т.В.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *125 “*Кібербезпека*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Кібербезпека*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Килимнику Дмитру Віталійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів*

2. Керівник роботи *Смірнова Тетяна Віталіївна, канд. техн. наук*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 51-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи кібербезпеки в промислову експлуатацію.

6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи кібербезпеки	*1 аркуш*
Функціональна схема системи кібербезпеки	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Смірнова Т.В.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Килимник Д.В.
(прізвище та ініціали)

АНОТАЦІЯ

Килимник Д.В. Програмне забезпечення системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів.

Метою розробки є програмне забезпечення системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів.

Результат роботи – програмна реалізація системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Python.

Ключові слова: кібербезпека, міжмережний екран

ABSTRACT

Kilymnyk D.V. Cybersecurity system software for analyzing security risks when using firewalls. 125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed that is intended for a cybersecurity system for analyzing security risks when using firewalls.

The purpose of the development is to develop a cybersecurity system software for analyzing security risks when using firewalls.

The result of the work is a software implementation of a cybersecurity system for analyzing security risks when using firewalls.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software are provided.

The program can be used on PCs with Windows 10/11.

The program is developed in the Python environment.

Keywords: cybersecurity, firewall

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ЛОМ	–	локальна обчислювальна мережа
MME	–	міжмережні екрани
ATM	–	асинхронний режим передачі
BSD	–	адаптована для Internet реалізація операційної системи UNIX
ICMP	–	міжмережний протокол управляючих повідомлень
IP	–	Internet Protocol – міжмережний протокол
NFS	–	мережна файлова система
PPP	–	протокол передачі від точки до точки
RFC	–	опис набору протоколів Internet
RPC	–	віддалений виклик процедури
SLIP	–	міжмережний протокол для послідовного каналу
SMTP	–	Simple Mail Transfer Protocol – простий протокол передачі пошти
TCP	–	Transmission Control Protocol – протокол управління передачею
UDP	–	User Datagram Protocol – протокол користувальницьких датаграм
UNIX	–	багатозадачна операційна система
UTP	–	незахищена вита пара
URL	–	уніфікований покажчик інформаційного ресурсу

ВСТУП

Актуальність теми. Складність типової мережної інфраструктури зросла настільки, що навіть одна помилка в параметрах ключового міжмережного екрана здатна паралізувати функціонування всієї мережі. До нещастя, про ризики найчастіше замислюються лише в той момент, коли що-небудь уживати вже занадто пізно.

Історія використання міжмережних екранів (МЕ) нараховує вже більше 20 років, але ми усе ще не навчилися як варто управляти цими пристроями й використовувати їхній потенціал повною мірою. У результаті бізнес не може протистояти погрозам, з якими правильно налаштоване мережне встаткування могло б упоратися.

Серед основних недоглядів, через які виникають ризики безпеки, можна виділити наступні:

– невідповідність галузевим нормам і вимогам державних регулювальних органів;

– помилкові зміни правил роботи міжмережних екранів, що порушують функціонування бізнес-додатків;

– старіння правил, що створює погрозу неавторизованого доступу до ресурсів мережі.

Парадокс полягає в тому, що для компрометації міжмережного екрана зовсім не потрібно витончених хакерських атак – головним ворогом бізнесу найчастіше виявляються власні IT-фахівці. Тривіальні помилки й недогляди в процедурах керування екранами можуть оголити периметр мережі для зовнішніх атак і поставити під погрозу безперервність бізнес-процесів.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем для аналізу ризиків безпеки при використані міжмережних екранів.

– Дослідження системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів.

– Програмна реалізація системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі для аналізу ризиків безпеки при використані міжмережних екранів.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Якщо врахувати, що, крім керування міжмережними екранами, співробітники ІТ-відділу повинні забезпечувати роботу досить складної корпоративної мережі й вирішувати ряд інших завдань, стає очевидним: до проблеми керування екранами потрібно підійти максимально серйозно й вирішити її раз і назавжди.

Міжмережні екрани використовуються для розмежування мереж, що мають різні вимоги до безпеки. Міжмережні екрани варто використовувати щораз, коли внутрішні мережі й системи взаємодіють із зовнішніми мережами й системами й коли вимоги безпеки розрізняються в декількох внутрішніх мережах. Розглянемо, де повинні бути розташовані міжмережні екрани і як повинні бути розташовані інші мережі й системи відносно міжмережних екранів.

Тому що первісною функцією міжмережного екрана є запобігання небажаного вхідного трафіку в мережу (і в деяких випадках вихідні), міжмережні екрани повинні бути розташовані в крапках входу на логічних границях мережі. Звичайно це означає, що міжмережний екран є вузлом, у якому мережний трафік розділяється на кілька шляхів, або збирається разом у єдиний шлях. При маршрутизації міжмережний екран звичайно розташований безпосередньо перед маршрутизатором і іноді сполучається з маршрутизатором. Набагато рідше міжмережний екран розташовується після поділу трафіку на кілька маршрутів, тому що в цьому випадку міжмережний екран повинен буде стежити за кожним із цих маршрутів. Часто апаратні пристрої міжмережного екранування мають також і можливості маршрутизації, і в мережах, побудованих з використанням комутаторів, міжмережний екран часто є частиною самого комутатора, що забезпечує можливість захищати всі сегменти, що комутуються.

1.2 Область застосування

Міжмережний екран одержує трафік, перевіряє його у відповідності зі своєю політикою й виконує відповідна дія (наприклад, пропускає трафік, блокує його, виконує деяке перетворення).

Ми вже розглянули різні типи технологій міжмережного екранування. Міжмережні екрани, розташовані на границі мережного периметра, часто є апаратними пристроями з декількома мережними інтерфейсами; міжмережні екрани для хостів і персональні міжмережні екрани убудовані в ПЗ, що установлене на одному комп'ютері, і захищають тільки даний комп'ютер; пристрою персонального міжмережного екрана призначені для захисту єдиного комп'ютера або мережі невеликого офісу. У даній главі будемо розглядати тільки міжмережні екрани, розташовані на границі мережного периметра, тому що для інших типів не важлива топологія мережі.

Показано типову топологію мережі з міжмережним екраном, що функціонує як маршрутизатор. Сторона, що не захищається, міжмережного екрана з'єднана з єдиним маршрутом, позначеним як WAN, що захищається сторона з'єднана із трьома маршрутами, позначеними як LAN1, LAN2 і LAN3. Міжмережний екран діє як маршрутизатор трафіку між WAN і різними маршрутами LAN. Один з маршрутів LAN також має маршрутизатор; часто усередині мережі переважніше використовувати кілька рівнів маршрутизаторів.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

VPN-1 Gateway фірми Check Point

VPN-1 Gateway фірми **Check Point** являє собою комплект із продуктів FireWall-1 і VPN-1, крім того, існує плата VPN-1 Accelerator Card, що збільшує пропускну здатність віртуальної приватної мережі.

Чудова система керування, відмінна продуктивність один із кращих засобів протоколювання подій дозволили продукту VPN-1 Gateway посісти перше місце.

Відповідаючи тенденціям в області безпеки, фірма Check Point додала до наявних засобів захисту, заснованим на технології Stateful Inspection, прикладні модулі-посередники, розроблені для блокування аномального трафіку на прикладному рівні.

На відміну від ряду інших VPN-пристроїв, які вимагають настроювання кожного шлюзу віртуальної приватної мережі окремо шляхом уведення однієї й тої ж інформації, що управляє станція VPN-1 Gateway пропонує більше просту процедуру, що дозволяє застосувати ту або іншу стратегію захисту до зазначених firewall-системам. Графічний інтерфейс керування надає для цього все необхідне.

Була обрана наступна базова стратегія захисту: віртуальна приватна мережа на основі сімейства протоколів IPsec використовувала обмін ключами по протоколі IKE, шифрування 3DES і автентифікацію MD5.

Конфігурування системи FireWall-1 дійсно є нескладною процедурою. Подібно іншим firewall-системам, заснованим на використанні правил, задати стратегію захисту в FireWall-1 означає вказати, по якому протоколі кому й до

чого дозволений доступ. Надано можливість приховання правил від перегляду і їхнє відключення. Останнє особливо до речі при редагуванні й зміні стратегії захисту firewall-системи. Правила можна встановлювати й перевіряти, і у випадку порушення роботи мережних додатків, що послужило його причиною правило може бути тимчасово відключено.

Графічний користувальницький інтерфейс керування дозволяє працювати одночасно з декількома міжмережними екранами, при цьому вся стратегія захисту задається в одному великому "плоскому" файлі, що містить всі правила. Для полегшення орієнтації, що управляє інтерфейс надає кошти для виділення кольором мережних ресурсів і груп об'єктів.

Керування firewall-системою іноді може виявитися складною справою. А керування декількома firewall-системами – це завжди складна справа, пов'язане з повсякденними проблемами, такими як додавання й модифікація правил, пошук і усунення неполадок і необхідність контролю загального стану системи захисту. Однак здатності продукту фірми Check Point до масштабування не виходять далеко за межі підтримки декількох firewall-систем і інших пристроїв, оскільки база правил стає занадто довгої й заплутаною. Може виявитися корисним наявність поля коментарю для кожного об'єкта. Продукт VPN-1 Gateway має унікальну здатність фільтрувати вивід правил на екран на основі заданих критеріїв, таких як служба, адреса призначення або джерела. Це може виявитися корисним при надмірному розростанні бази правил.

З погляду засобів протоколювання VPN-1 Gateway перевершує інші продукти. Хоча це далеко не сама помітна функція міжмережного екрана, без гарних засобів протоколювання вам довелося б працювати наосліп. Деякі аномальні події можуть не ініціювати повідомлень або аварійних сигналів, навіть при використанні локальної або зовнішньої IDS-системи. Наприклад, повільне "горизонтальне" сканування портів залишиться непоміченою більшістю firewall- і IDS-систем, але його "слід" залишиться в реєстраційному журналі. При керуванні декількома firewall-системами, зручно мати всі журнали в одному місці, але з

ростом їхнього числа ви швидко виявитеся буквально затопленими даними. Задавши набір фільтрів можна скоротити кількість виведених на екран записів, залишивши тільки ті, які становлять інтерес.

Були встановлені правила, що дозволяють вхідний HTTP-трафік для двох серверів і вихідний HTTP-трафік. Потім був запущений імітатор компонента уразливості iishack фірми eEye Digital Security, що дозволило проникнути на Web-сервер. Однак VPN-1 Gateway має унікальну функцію за назвою Security Servers, що захищає від атак сервери HTTP, FTP і SMTP. Security Servers перевіряє синтаксис протоколів прикладного рівня.

Cisco Secure PIX Firewall 520 фірми Cisco Systems

Продукт Cisco Secure PIX Firewall 520 фірми Cisco Systems, що представляє собою апаратну реалізацію міжмережного екрана, надзвичайно привабливий у точки зору продуктивності, але трохи менш привабливий з погляду керування. Його інтерфейс командного рядка важко назвати дружнім, навіть якщо ви знайомі з фірмовою мережною операційною системою фірми Cisco – IOS (Internetwork Operating System). Однак компанія забезпечила підтримку більше дружнього графічного інтерфейсу у своєму засобі керування стратегією захисту Cisco Secure Policy Manager. Менеджер CSPM призначений для мереж з більшим числом міжмережних екранів. Але будьте готові до того, що вам доведеться витратити якийсь час на вивчення CSPM і збагнення його парадигми, що відрізняється від загальноприйнятих стратегій керування firewall-системами.

CSPM – це крок уперед, але цей засіб, поза всяким сумнівом, зажадає від вас змінити підхід до захисту вашої мережі. Випливаючи парадигмі цього продукту, подібної тієї, що втілено в пакеті CiscoWorks2000, необхідно спочатку за допомогою графічного інтерфейсу побудувати модель мережі, а потім розподілити інформацію про конфігурацію системи захисту по всіх вилучених пристроях. На жаль, CSPM не може не розпізнавати пристрою, не імпортувати існуючі сценарії конфігурації, тому якщо ви помістите CSPM у діючу мережу,

вам доведеться через нього переконфігурувати інтерфейси й списки контролю доступу всіх систем PIX. На щастя, є можливість перевіряти конфігурацію перед застосуванням її до тої або іншої системи PIX.

Перегляд конфігурації системи захисту, створеної за допомогою CSPM, займає багато часу. Хоча пристрою PIX Firewall діють на основі правил, він використовує списки контролю доступу для обробки вхідного трафіку й тунелі для обробки вихідного. Синтаксис ACL незвичайний: замість правил, що встановлюють, який трафіку й куди дозволено проходити, визначається модель доступу за замовчуванням, а потім до неї додаються виключення. Інакше кажучи, у випадку звичайного синтаксису ACL досить установити правило: "Дозволити трафік з Інтернет на порт 80 вузла web.here.com". А для системи PIX потрібно задати ACL-правило із двох записів: "Заборонити весь HTTP-трафік", а потім "Крім порту 80 вузла web.here.com". Результат в обох випадках однаковий, але працювати із загальноприйнятим синтаксисом простіше, особливо, з ростом кількості правил.

Створення VPN-мереж в CSPM-менеджері – це нескладна справа, що супроводжується відносно малим числом помилок. Як і у випадку з VPN-1 Gateway, була задана стратегія захисту віртуальної приватної мережі з поділюваними секретами, завершивши загальну картину додаванням інформації про пристрої PIX і про мережі, яка захищається ними. CSPM-менеджер негайно згенерував записи схем і маршрутів, необхідні для роботи кожного пристрою PIX, що є істотним поліпшенням у порівнянні з конфігуруванням кожного пристрою окремо. Конфігурації можна зберігати локально, але в CSPM відсутні засобу контролю версій, наявні в CiscoWorks2000. Фірма Cisco планує вирішити як цю проблему, так і інші, шляхом інтеграції CSPM з CiscoWorks2000. Це повинне також забезпечити виявлення пристроїв і імпорт конфігурацій.

Удалося пробитися крізь firewall-систему PIX і одержати доступ до нашому Web-серверу. На відміну від VPN-1 Gateway, PIX Firewall 520 не пропонує можливості перевірки синтаксису HTTP, що стає нагальною потребою

для компаній, що тримають свій власний Web-вузол. Однак PIX благополучно блокував всі наші атаки на мережному рівні, а протоколювання, хоча й грубе, було цілком достатнім для швидкого відстеження подій, хоча воно навіть не наближається по якості до VPN-1 Gateway.

Raptor Firewall 6.5 фірми Axent Technologies

Продукт Raptor Firewall 6.5 фірми Axent давно користується репутацією firewall-системи, що забезпечує надійний захист як на мережному, так і на прикладному рівні, і випробування це підтвердили. Даний продукт виявився єдиним, здатним відбити атаки прикладного рівня на сервер IIS. Вилучене керування firewall-системою досить зручно. Воно здійснюється через додаток, що використовує ПЗ Microsoft Management Console (MMC). Raptor аж ніяк не найшвидший міжмережний екран

Однак у тестах по відбиттю атак Raptor Firewall виглядав блискуче. При використанні тої ж стратегії захисту, що й у випадку інших firewall-систем, не вдалося не одержати контроль над Web-сервером, не порушити його роботу. Exploit-компонент iishack працював як звичайно, але спроба підключитися до Web-сервера через telnet закінчилась відмовою в доступі. У реєстраційних журналах з'явилося повідомлення HTTP-демона системи Raptor Firewall про заблоковану спробу підключення, що містить некоректний URL-покажчик. На жаль, Raptor може бути виявлений простим скануванням портів. Якщо ви вилученим образом управляєте Raptor Firewall, то у вас будуть відкриті порти 416, 417 і 418 разом з інформацією про всі запущені служби. Цих даних, швидше за все, буде досить для ідентифікації Raptor. Але важливіше те, що при скануванні порти захищеної внутрішньої мережі, кожний IP-адреса повідомляла ті самі дані. Це вказує на те, що відповіді, очевидно, приходять від firewall-системи proxy-типу, а не від самих хостів. Проте, Raptor виявився єдиним міжмережним екраном, що визначив атаку за допомогою програми Nmap, і не дозволив сканувати що захищаються хости.

Raptor відрізняється від всіх інших розглянутих firewall-систем тим, що його правила не діють по ієрархічному принципі "зверху долілиць". Замість цього він стежить за вхідної трафіком і підшукує найбільш відповідне для нього правило. Хоча це може здатися незвичайним, модель "найкращої відповідності" змушує ретельніше думати про те, як конкретно реалізується стратегію захисту. Ієрархії правил, використовувані в інших системах, набагато "поблажливіше" до помилок. Тут же конкретне правило має пріоритет над загальним. Наприклад, якщо встановити правило дозволити HTTP-трафік у внутрішній мережі й крім того встановити правило, відповідно до якого HTTP-трафік до одному з Web-вузлів – Tweety, – не пропускається, то тому що друге правило має пріоритет над першим, то доступ до Tweety буде закритий.

Пропускна здатність при використанні Raptor виявилася вкрай низкою. Був досягнутий рівень усього лише 16 Мбіт/с, коли Raptor почав катастрофічно скидати пакети. При 14 Мбіт/с приблизно на 26 тис. пакетів доводився максимум 75 скинутих – цілком прийнятна цифра. Але при 16 Мбіт/с частка загублених пакетів склала понад 90%. Після досягнення цього порога ВЧС припинила передавати трафік, що привело до необхідності перезапуску firewall-системи.

Керування firewall-системою, як локальне, так і вилучене, здійснюється через додаток консолі MMC. З урахуванням широкого набору функціональних можливостей, конфігурація являє собою не потребуючих особливих зусиль процес. "Спускаючись" долілиць по галузях "керуючого дерева", можливо звертатися майже до будь-якого елемента firewall-системи. Додаткові можливості надаються за допомогою натискання правої кнопки миші. Особливо корисними були знайдені звіти про зроблене конфігурування. Raptor забезпечує широкі можливості для створення звітів, що містять детальну інформацію про конфігурацію як системи в цілому, так і різних сполучень її елементів. Цей інструмент дає можливість точно побачити, як визначені правила, не заглядаючи в кожне з них окремо.

Хоча вдалося підключитися й управляти декількома системами Raptor через MMC-консоль, не було можливості управляти всіма пристроями як єдиним цілим, як це можна робити, працюючи із продуктами фірм Check Point і Cisco. Тут кожна firewall-система являє собою окрему сутність і повинна конфігуруватися окремо, що багато в чому нагадує інтерфейс вилученого управління продукту NetScreen-100. Це стає критично важливим при настроюванні ВЧС. Установка параметрів ВЧС не представляє особливої складності, але дуже стомлююча. Працюючи з VPN-1 Gateway і PIX Firewall 520, можливо визначати стратегію захисту ВЧС, установлювати взаємини між шлюзами й розподіляти інформацію з firewall-систем за один крок. Використовуючи продукти Raptor Firewall, NetScreen-100 і Sidewinder ви змушені працювати з кожною системою окремо, уводячи ті самі параметри по кілька разів. Це незмінно веде до незначного на вид, але порою фатальним помилкам.

Відверто говорячи, можливості засобів генерації звітів системи Raptor залишають бажати багато кращого. Даний продукт особливо слабшав в області деталізованого протоколювання сеансів зв'язку й подання цієї інформації у звітах. Хоча тести на продуктивність і захист згенерували досить великий обсяг записів у журналах, розібратися в них було досить важко. Відкинуті спроби доступу породжували по трьох запису в журналі, але ці три записи рідко випливають один за одним. Деякі із записів про події не містили відповідних IP-адрес джерела або призначення, а деякі були зовсім незбагненні для розуміння. Ще гірше те, що відсутній спосіб негайно визначити, чи були відхилені ті або інші спроби встановлення з'єднань. При скануванні портів просто повідомлялося про спроби з'єднань.

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки та мови програмування

Python – це об’єктноорієнтована мова програмування високого рівня загального призначення з відкритим кодом. Це визначення може бути важким для новачків, тому розглянемо кожну характеристику окремо, щоб зрозуміти, що вона означає:

– Відкритий вихідний код: це безкоштовно та доступно для подальших покращень, таких як додавання корисних функцій або виправлення помилок.

– Об’єктноорієнтована: заснована не на функціях, але в об’єктах з певними атрибутами й методами.

– Високий рівень: зручний для людини, а не для комп’ютера.

– Загальне призначення: можна використовувати для створення будь-яких програм.

Ця мова використовується в будь-якому програмному забезпеченні, про яке ви тільки можете подумати. Ви можете використовувати його для створення вебсайтів, штучного інтелекту, серверів, програмного забезпечення для бізнесу та багато іншого. Також застосовується в науці про дані, аналізі даних, машинному навчанні, інженерії даних, веброзробці, розробці програмного забезпечення та інших галузях.

Переваги та недоліки Python

Переваги:

– Її легко читати, вчити та писати. Це мова програмування високого рівня з англійським синтаксисом. Це полегшує читання та розуміння коду. Її дійсно легко зрозуміти і вивчити, тому багато людей рекомендують Python новачкам. Вам потрібно менше рядків коду для виконання того ж завдання в порівнянні з іншими основними мовами, такими як C/C++ та Java.

– Підвищує продуктивність. Це дуже продуктивна мова. Завдяки її простоті розробники можуть зосередитися на розв’язанні проблеми. Їм не

потрібно витрачати багато часу на розуміння синтаксису або поведінку мови програмування. Ви пишете менше коду та виконуєте більше завдань.

– Інтерпретована мова. Python мова, що інтерпретується, а це означає, що вона безпосередньо виконує код по рядку. Якщо сталася помилка, вона зупиняє подальше виконання та повідомляє про її виникнення. Вона показує лише одну помилку, навіть якщо у програмі їх кілька. Це спрощує налагодження.

– Динамічно типізована. Python не визначає тип змінної, доки ми не запустимо код. Вона автоматично надає тип даних, коли відбувається процес виконання. Фахівець може не турбуватися про оголошення змінних та типи даних.

– Безкоштовна та з відкритим вихідним кодом. Ця мова постачається під схваленою OSI ліцензією з відкритим вихідним кодом. Це робить його безкоштовним для використання та розповсюдження. Ви можете завантажити вихідний код, змінити його та навіть розповсюджувати свою версію. Це корисно для організацій, які хочуть використати свою версію для розробки.

– Підтримка великих бібліотек. Стандартна бібліотека Python є величезною, ви можете знайти майже всі функції, необхідні для вашого завдання. Таким чином ви не залежите від зовнішніх бібліотек.

– Портативність .У багатьох мовах, таких як C/C++, потрібно змінити свій код, щоб запустити програму на різних платформах. З Python все інакше. Ви тільки пишете один раз і запускаєте її будь-де.

Недоліки:

– Низька швидкість. Вище ми обговорювали, що це інтерпретована мова з динамічною типізацією. Порядкове виконання коду часто призводить до повільного виконання. Динамічна природа Python також є причиною її низької швидкості, оскільки їй доводиться виконувати додаткову роботу при виконанні коду. Тому вона не підходить для цілей, де швидкість важливий аспект проєкту.

– Неефективна для пам’яті. Ця мова програмування використовує великий обсяг пам’яті, це може бути недоліком при створенні програм, коли віддають перевагу оптимізації пам’яті.

– Слабка у мобільних обчисленнях. Python зазвичай використовується у серверному програмуванні. Ми не бачимо – її на стороні клієнта або в мобільних програмах з таких причин: вона не заощаджує пам’ять і має повільну обчислювальну потужність у порівнянні з іншими мовами.

– Доступ до бази даних. Програмувати на цій мові легко, але коли ми взаємодіємо з базою даних, її не вистачає. Рівень доступу до бази даних у Python примітивний та недостатньо розвинений у порівнянні з іншими популярними технологіями.

– Помилки виконання. Це мова з динамічною типізацією, тому тип даних змінної може змінюватись у будь-який час. Змінна, що містить ціле число, у майбутньому може містити рядок, що може призвести до помилок виконання.

Застосування Python:

– Для аналізу даних. Дані стали цінним активом у будь-якій сучасній галузі, і більшість компаній зацікавлені у збиранні, обробці та аналізі релевантних даних, щоб витягти з них цінну інформацію для бізнесу. І тут Python виходить за межі будь-якої конкуренції. Python особливо цінна тим, що крім великої стандартної бібліотеки надає величезний набір додаткових модулів, розроблених спеціально для аналітичних цілей. Найвідоміші бібліотеки Python для аналізу даних – це pandas і NumPy . Ці інструменти дозволяють робити з вашими даними майже все, наприклад, очищати і аналізувати їх, вивчати статистику або візуалізувати приховані тенденції у ваших даних.

– Для візуалізації даних. Візуалізація даних – це окрема частина аналізу даних, яка допомагає нам подавати інформацію, необроблену чи очищену, у більш змістовній формі. Тут Python знову входить у гру, пропонуючи широкий спектр інструментів візуалізації даних. Найпопулярніші з них – matplotlib і

заснований на ній seaborn. Використовуючи їх, ми можемо створювати буквально всі види візуалізації: від найпростіших до складніших.

– Для машинного навчання. Машинне навчання (ML) є основою більшості завдань науки даних. Він є областю штучного інтелекту, пов'язаною з використанням алгоритмів, що дозволяють машинам вивчати закономірності та тенденції на основі історичних даних, щоб робити прогнози на основі невідомих даних. – Використовуючи методи ML, ми можемо створювати моделі, які можуть точно передбачити швидкість відтоку клієнтів компанії, оцінити ризик виникнення у людини певного захворювання, визначити оптимальне розташування автомобілів таксі й т.д. За допомогою Python ми можемо побудувати модель ML, використовуючи лише три рядки коду.

– Для розробки програмного забезпечення. Крім свого багатостороннього застосування в галузях науки про дані, Python використовується на кожному етапі розробки програмного забезпечення, включаючи контроль складання, автоматичну безперервну компіляцію, прототипування, відстеження помилок, тестування та обслуговування програмного забезпечення. За допомогою цієї мови можемо створювати аудіо- або відеопрограми на основі методів штучного інтелекту, машинного навчання, API (інтерфейсів прикладного програмування), GUI (графічних інтерфейсів) або будь-якого іншого типу програмного забезпечення.

– Для веброзробки. У той час як для створення візуальної частини вебсайту ми переважно будемо використовувати такі мови, як HTML, CSS та JavaScript, для його невидимої частини ми часто вибираємо Python. Серед масштабних вебсайтів та програм, створених за допомогою цієї мови, варто згадати Google, Facebook, Instagram, YouTube, Dropbox та Reddit.

– Для автоматизації задач/скриптингу. Це відмінний інструмент для написання програм для автоматизації різних завдань, що повторюються. Цей процес називається скриптингом. Зокрема, можна робити скрипти для роботи з файлами та папками. Наприклад, можна створювати, перейменовувати,

перетворювати, розділяти, об’єднувати або видаляти файли, перевіряти їх на наявність помилок. Ви також можете використовувати автоматизацію Python для пошуку та завантаження інформації з Інтернету, заповнення та надсилання онлайн-форм та надсилання регулярних повідомлень або електронних листів.

Яким фахівцям потрібно володіти Python:

– Фахівець з даних.

– Аналітик даних.

– Інженер даних.

– Інженер з машинного навчання.

– Журналіст даних.

– Архітектор даних.

– Повний стек веброзробника.

– Backend-розробник.

– DevOps-інженер.

– Інженер-програміст

Можемо зробити висновок, що Python ще довго буде популярною мовою, хоч і має низку недоліків. Цю мову використовують для створення вебсайтів, штучного інтелекту, серверів, програмного забезпечення для бізнесу, аналізу даних, машинного навчання, інженерії даних та для багатьох інших областей. Це перспективна і затребувана навичка, яка необхідна у всіх галузях.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи кібербезпеки контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи кібербезпеки в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Завдання бізнесу № 1: оцінка ризиків, пов'язаних із правилами МЕ

У міру ускладнення мережної інфраструктури й розростання списку правил усе сутужніше виявляти й оцінювати ризики, пов'язані з невірними або надмірно «розв'язними» правилами настроювання міжмережного екрана.

Приміром, як показало дослідження «Помилки конфігурування міжмережних екранів: основні тенденції», що провів Авишай Вул, професор Тель-авівського університету, 42% всіх міжмережних екранів різних виробників дозволяють зовнішній доступ до мережі для служб Microsoft, що служать каналом поширення безлічі інтернет-хробаків. У ході дослідження було виявлено, що 75% найбільш складних моделей екранів мають понад 20 помилок у конфігурації. Крім того, 80% МЕ, перевірених на предмет наявності проломів у системі безпеки, пропускали трафік, що підлягає блокуванню, через те, що були неправильно налаштовані.

Таким чином, основна причина ризиків, пов'язаних із застосуванням політики безпеки екранів, складається в недостатнім розумінні завдань, які даний екран повинен вирішувати в певний момент часу. Нормальне функціонування додатків і безперешкодна циркуляція трафіку ще не є надійними показниками захищеності мережі.

Рішення. Кожний професіонал в області IT і мережної безпеки безупинно аналізує свої дії й виникаючі в результаті потенційні погрози. Зміни, внесені в політики міжмережних екранів, здатні як зміцнити безпека мережі, так і збільшити ризики, оскільки навіть самі досвідчені адміністратори часом роблять ненавмисні помилки. Не маючи точної картини стану справ, неможливо оцінити, наскільки захищений бізнес.

Найкращий спосіб звести ризики до мінімуму – оцінити критичність наявних активів, використовувати переваги інтелектуальних міжмережних екранів і мережних пристроїв і застосовувати автоматизовані інструменти для своєчасного виявлення й усунення слабких місць мережної інфраструктури, поки вони не перетворилися в проблему. Розвинені автоматизовані інструменти опираються на визнану найкращу практику керування міжмережними екранами й здатні аналізувати поточне середовище для виявлення пробілів у безпеці. За рахунок точної ідентифікації трафіку, що дане конкретне правило пропускає через МЕ, ряд інструментів допомагає підсилити занадто нестрогі правила (наприклад, сервіс «ANY»).

Застосування ефективних інструментів у сполученні з ручним аналізом дозволяє перейти від усунення ризиків «за фактом» до проактивним дій, коли проблема виявляється задовго до виникнення критичної ситуації.

Завдання бізнесу № 2: керування змінами

В IT-галузі зміни випливають одне за іншим безперервною низкою. Керування ними є однієї з головних завдань бізнесу. Нездатність грамотно управляти змінами обіцяє значні ризики, починаючи від таких необразливих проблем, як блокування «легального» трафіку, і закінчуючи повною неприступністю мережної інфраструктури. На даний момент не існує простого стандартного рішення для керування змінами в міжмережних екранах. Труднощі керування змінами на МЕ викликані наступними причинами.

Відсутність формальних правил. При розгляді особливостей міжмережних екранів набори правил (політику) нерідко плутають із формальною політикою інформаційної безпеки. Необхідно чітко розділяти ці поняття й сформувати офіційну політику керування змінами, яка б охоплювала й питання, пов'язані з міжмережними екранами.

Нечіткі процедури, що залишаються без уваги. Одна справа мати політику у вигляді приписань «що потрібно робити», і зовсім інше – мати формальний опис послідовності кроків для реалізації політики. Щоб успішно

управляти змінами в міжмережних екранах, необхідно чітко й докладно визначити дії, які повинен почати кожний співробітник, якщо йому доведеться вносити зміни. Будь-які виключення повинні бути враховані, чітко викладені й затверджені.

Відсутність взаємодії між співробітниками відділу ІТ. Ще одна серйозна перешкода для успішного забезпечення безпеки мережі – низька якість комунікації між особами, відповідальними за забезпечення ефективної роботи екранів. У поганій комунікації криється одна з головних причин непідготовлених змін, які, у свою чергу, викликають безліч збоїв у системі безпеки.

Нерозуміння супутніх ризиків для бізнесу. Необхідно мати повне подання про те, які погрози й уразливості можуть відбитися на роботі підприємства. Проте нерідко недооцінюється руйнівний ефект, що неписьменне керування змінами в міжмережних екранах може зробити на бізнес. Найчастіше величезні ресурси витрачаються на те, щоб не допустити зловмисника в мережу, при цьому забувається, що гірший ворог – це ми самі.

Складність мереж. Складна мережа сама по собі вже може бути джерелом безлічі помилок, особливо якщо експлуатуються численні МЕ різних виробників, для кожного з яких сформований окремий набір правил. Складність – ворог безпеки, тому потрібно почати всі можливі кроки для зниження складності конфігурації екранів і спрощення процедур керування цими пристроями.

Нерозуміння наслідків змін у міжмережних екранах. Відмова від аналізу й з'ясування того, який ефект зробить на мережну інфраструктуру кожне, навіть саме незначна зміна, чреватий серйозними неприємностями. Не провівши належної оцінки, фахівець ризикує не довідатися:

– які додатки й з'єднання можуть постраждати в результаті змін;

– які нові уразливості принесе із собою та або інша зміна;

– як зміни відіб'ються на працездатності й прозорості мережної інфраструктури.

Переконатися, що всього кілька невірних дій можуть обернутися справжньою катастрофою, можна на прикладі компанії зі сфери електронної торгівлі, що була одним із провідних постачальників цих послуг на американському ринку. Один раз все підприємство виявилося відключеним від мережі протягом декількох годин: всі робітники транзакції – і вхідні, і вихідні – не виконувалися. Згодом з'ясувалося, що ряд співробітників, відповідальних за керування міжмережними екранами, зробили непідготовлені (і непротестовані) зміни на основному міжмережному екрані, через що порушився зв'язок між торговельним додатком і Інтернетом.

Слідами інциденту було проведене розслідування за участю вищого керівництва компанії, і винні одержали стягнення. Сотні тисяч доларів були витрачені на те, щоб з'ясувати причину того, що відбулося. Виявилося, що системні адміністратори не тестували внесені зміни, прагнучи піти від «занадто довгих і складних» процедур на основі ITIL, і не замислювалися про наслідки.

Рішення. Якщо IT-фахівці здатні управляти змінами на систематичній основі протягом тривалого часу, половина справи вже зроблена. Компанія одержить не тільки більше захищене мережне середовище, але й поверне IT-відділу його первісне призначення: допомагати підприємству, а не ускладнювати ведення бізнесу. Варто пам'ятати, що ефективні автоматизовані інструменти повинні:

– розробляти й впроваджувати процедури для різних процесів керування змінами в політику безпеки;

– маючи знання мережної топології, ідентифікувати міжмережні екрани, на які поширюються передбачувані зміни;

– моделювати ефект зміни для завчасного виявлення ризиків і порушень вимог регуляторів;

– зіставляти вироблені зміни із запитуваними, щоб виявити всі непідзвітні зміни.

При хмарних обчисленнях ризики через неправильне керування змінами в міжмережних екранах, багаторазово підвищуються, тому що одна невірна дія може паралізувати роботу не однієї, а відразу багатьох компаній.

Без достатнього розуміння й передбачуваності процесів бізнес приречений на провал, а успіх можливий тільки при наявності ретельно продуманих і документованих політик і процедур, доповнених мірами контролю. Іншими словами, коли зазначені інструменти й процеси інтегровані із загалькорпоративною системою керування змінами, компанія впевнено стає на шлях процвітання, не сподіваючись на споконвічний либонь.

Завдання бізнесу № 3: підтримка правил в оптимальному стані

Безладдя в наборах правил міжмережних екранів – не естетична проблема, а джерело прямих ризиків для бізнесу, таких як відкриття непотрібних портів і тунелів VPN, поява крапок входу для мережних атак через конфлікт правил. Крім того, невпорядкованість істотно підвищує складність мережі в цілому.

Крім того, надмірне розростання наборів правил значно ускладнює проведення аудита, коли потрібно проаналізувати кожне правило й знайти відповідне йому бізнес-підстава. У підсумку виникають додаткові витрати й витрачається коштовний час IT-фахівців. От тільки деякі приклади потенційно проблемних правил:

– невикористовувані правила;

– правила, що перекриваються;

– застарілі правила;

– правила без коментарів;

– правила, розташовані в неоптимальному порядку (наприклад, коли найбільше часто застосовуване правило перебуває наприкінці списку, що приводить до зайвого навантаження на міжмережний екран).

Все це змушує підприємства час від часу проводити «чищення» або «повторну перевірку» міжмережних екранів. Ще більший масштаб зазначені проблеми здобувають на великих підприємствах, де розгорнута безліч МЕ від

різних виробників. Ріст складності приводить до погіршення прозорості, утрудняє облік і чреватий несподіване виникнення проломів у системі безпеки.

Проведення аудита правил роботи міжмережних екранів може стати тяжким завданням, особливо якщо відповідальний за це фахівець слабко розбирається в технічних питаннях або погано вивчив інспектуєму мережу. Ситуація збільшується ще й тим, що перевірка конфігурації екранів найчастіше взагалі виключається із процесу аналізу системи безпеки, тому що вважається, що «якщо мережа працює, то все в порядку».

Для перевірки можна провести простий експеримент: попросити мережних адміністраторів або керівників служби безпеки надати схему мережі. Найчастіше виявляється, що ніякої схеми немає або вона сильно застаріла. З наборами правил все точно так само. Але подібне положення справ не завжди результат помилки однієї конкретної людини, скоріше це побічний ефект від неправильної організації роботи IT-відділу в цілому, коли співробітники зайняті затиканням всі нових «пробоїн», замість того щоб систематично, день у день працювати над зміцненням захищеності мережі. Ця проблема, разом зі складністю мережі й правил, приводить до зниження ефективності керування екранами, що приводить до виникнення відповідних ризиків для бізнесу.

Рішення. Необхідний більше формалізований підхід до підтримки правил в оптимальному стані, аналогічний застосовуваному при керуванні змінами, де кожна процедура описана формально й має відповідального за її виконання. Періодична перевірка бази правил допоможе завчасно усунути проблеми й забезпечити надійний захист мережі.

Для ефективного керування наборами правил потрібні підходящі якісні інструменти, які допоможуть не тільки визначити, які правила можна видалити або оптимізувати, але й скласти уявлення про наслідки зміни конкретного правила. Найбільш зроблені інструменти автоматизують цей процес, рятуючи співробітників від рутинних дій і помилок при ручних перевірках.

Крім того, необхідно, щоб керування наборами правил охоплювало відразу всі міжмережні екрани. Концентрація уваги тільки на одному пристрої означає довільне звуження області оцінки, результати якої будуть створювати помилкове враження захищеності. Звичайно, починати треба з найбільш критичних міжмережних екранів, але в підсумку необхідно обстежити набори правил на всіх пристроях.

Завдання бізнесу № 4: відповідність внутрикорпоративним політикам і вимогам регуляторів

Забезпечити відповідність різним нормативно-правовим вимогам (поряд з виконанням умов клієнтів і бізнес-партнерів і дотриманням власних внутрикорпоративних стандартів), як правило, дуже непросто. Для цього необхідно дати чіткі відповіді на наступні питання:

– Що дійсно потрібно для дотримання даної вимоги/політики?

– Які технічні і юридичні умови закріплені в контрактах і угодах про рівень обслуговування?

– Яким образом прийняті в компанії методи керування міжмережними екранами співвідносяться з її основною діяльністю?

Всі стандарти й норми в сфері захисту інформації, такі як PCI DSS, GLBA і HIPAA, в основі своєї спрямовані на збереження конфіденційності й цілісності інформації, а також забезпечення доступності мережної інфраструктури й додатків. Це не що інше, як найкращі методи роботи, які відомі й практикуються протягом багатьох років.

Труднощі в тому, що співробітники IT-відділів виявляються «між багатьох вогнів» і відсторонені від питань нормативно-правової відповідності більш, ніж коли-або колись. Найчастіше вони завантажені настільки, що не встигають займатися стратегічними завданнями, у тому числі забезпеченням відповідності нормативним вимогам. Час – самий дефіцитний ресурс в IT, внаслідок чого, на жаль, виникає згадана відстороненість IT-керівників і мережних адміністраторів, які виявляються у вкрай скрутному стані, коли справа стосується нормативно-

правової відповідності і юридичного забезпечення ІТ. Як це ні парадоксально, все перераховане лише шкодить бізнесу, стримуючи його розвиток або навіть погіршуючи стан справ. Від цього страждають і міжмережні екрани як один з компонентів системи мережної безпеки.

Рішення. Навіть із урахуванням всіх складностей виконання вимог стандартів по захисту інформації жодне підприємство не може дозволити собі ігнорувати погрози, які обіцяє невиконання цих вимог. Необхідно призначити фахівця, що забезпечує відповідність систем загальноприйнятим нормам. Ідеальним рішенням буде створення формального комітету, куди ввійшли б керівники кадрового, виробничого, юридичного й ІТ-відділів. Залучення осіб, що приймають рішення, допоможе прищепити культуру відповідальності за безпеку даних у масштабі всієї організації. Тим самим буде гарантоване поширення інформації про всі вимоги, норми й правила, які визначають її діяльність.

Варто мати на увазі, що регулюючі норми, з якими зіштовхується компанія, являють собою найкращі практичні застосування в області безпеки, які найчастіше вже тією чи іншою мірою впроваджені в організації. Крім того, багато вимог не занадто розрізняються. Розглядаючи завдання нормативно-правової відповідності в перспективі стратегічного керування інформаційними ризиками, керівництво зможе знизити погрози в сфері безпеки й забезпечити дотримання всього спектра вимог.

Для спрощення завдання забезпечення відповідності нормативним вимогам варто застосовувати автоматизовані інструменти, що використовують інформацію про мережу (про те, які підмережі є захищеними зонами PCI) і враховуючі завдання безпеки (конкретні вимоги PCI і внутрикорпоративних політик). Подібні інструменти допоможуть оцінити правила міжмережного екранування на предмет відповідності вимогам і виявити виключення, що бідують в особливій увазі.

Дотримання всіх вимог може здаватися непід'ємним завданням, однак, пам'ятаючи про цілях, які переслідують регулятори, керівництво підприємства

здатне перетворити їх у заходи щодо керування міжмережними екранами – особливо при наявності необхідних фахівців і інструментів, коли кожний співробітник розуміє, що саме він повинен робити для забезпечення безпеки.

Завдання бізнесу № 5: правильна оцінка ситуації

Ключовий аспект мережної безпеки – володіння оперативною інформацією, що, у свою чергу, залежить від прозорості мережі й здатності в будь-який момент одержати оцінку ступеня захищеності й відповідності вимогам.

Оцінка ефективності роботи міжмережних екранів може знадобитися в багатьох ситуаціях, наприклад:

– IT-аудитор проводить обстеження системи керування;

– керівництву компанії потрібно одержати більше детальне подання про захищеність мережі;

– є причини думати, що мережа атакована;

– експерт ІБ робить аналіз пролому в системі безпеки після атаки.

Найчастіше не ми управляємо міжмережними екранами, а екрани «управляють» нами. У такому випадку, зв'язані по руках і ногам високою складністю мереж і обмеженістю ресурсів, мережні адміністратори навіть приблизно не представляють, що насправді відбувається в мережі.

Правильна оцінка ситуації – це ключ до дотримання вимог і стратегічному керуванню ризиками. Якщо не можна точно визначити, наскільки добре захищена мережа в конкретний момент, то в чому зміст використання екранів? Чи можна затверджувати, що безпека мережі дійсно перебуває під контролем? Швидше за все, відповідь буде негативним.

Рішення. Можливість з'ясувати стан захищеності мережі й ступінь дотримання вимог залежить від того, наскільки якісно реалізовані механізми контролю. До технічних засобів контролю ставляться такі функції, як ведення журналів аудита й тривожних оповіщень – вони убудовані буквально в кожний міжмережний екран. Однак найчастіше мережні адміністратори й профільні керівники зневажають навіть цими засобами.

Проблема типових рішень полягає в тому, що вони легко стають некерованими, коли в мережі є більше десятка екранів. Для великих підприємств, особливо тих, де використовується встаткування різних виробників, додатка для керування екранами від незалежних постачальників можуть стати реальною панацеєю й дадуть краще подання про роботу мережі. Подібні інструменти здатні навіть створювати звіти для миттєвого одержання оцінки відповідності галузевим нормам і внутрішнім політикам, що дозволяє заощаджувати коштовний час для підготовки до аудита.

Дійте на випередження

Захист інформації може бути досить дорогою справою, але й одночасно неефективним: її надійність залежить не тільки від обсягу інвестицій, але й від того, як вона організована. Завдання, пов'язані з керуванням міжмережними екранами, часто упускаються з виду. Однак ті речі, які на перший погляд здаються банальн і другорядними, можуть вплинути на бізнес, особливо якщо щось піде не за планом. IT-інфраструктура повинна служити досягненню поставлених перед компанією цілей, тому керівництву варто зробити все можливе, щоб застрахувати себе, мережних адміністраторів і весь свій бізнес від невдач. Необхідно постаратися за бітами й байтами побачити тісний взаємозв'язок, що існує між захищеністю мережі й грамотним керуванням міжмережними екранами.

Основне завдання керівництва – надання ефективних інструментів і організація продуманих процедур, за допомогою яких стане можливим приймати інформовані рішення в сфері мережної безпеки. Сьогодні необхідно володіти точною інформацією про роботу мережі й управляти мережею в систематичному, робітнику режимі. Без впровадження життєво важливих інструментів і процедур IT-фахівці навряд чи зможуть вірогідно затверджувати, що інфраструктура міжмережних екранів функціонує надійно й повністю готова до виконання всіх вартих перед нею завдань.

3.2 Розробка структурної схеми

Багато апаратних пристроїв міжмережного екранування мають функціональність, називану **DMZ** – це скорочення від демілітаризованої зони, що встановлюють між воюючими країнами. Хоча й не існує одного визначення для DMZ, звичайно вони є інтерфейсами в міжмережному екрані, для яких можливо задавати правила маршрутизації, і аналогічні інтерфейсам, розташованим на стороні, яка захищається, міжмережного екрана. Основне розходження полягає в тому, що трафік, що проходить між DMZ і іншими інтерфейсами на стороні, яка захищається, міжмережного екрана, проходить через міжмережний екран, і до нього може застосовуватися певна політика. DMZ часто використовується в тому випадку, якщо існують хости, яким необхідно, щоб весь трафік оброблявся політиками міжмережного екрана (наприклад, для того, щоб хости в DMZ були б спеціально посилені), але трафік від хостів до інших систем у мережі також повинен проходити через міжмережний екран. В DMZ розташовують привселюдно доступні сервера, такі як веб-сервер або поштовий сервер. Приклад такої топології показаний на рисунку 3.1. Трафік з інтернету проходить через міжмережний екран і маршрутизируется до систем, розташованим із захет сторони, що, міжмережного екрана, або до систем в DMZ. Трафік між системами, розташованими на захищеній стороні, і системами, розташованими в DMZ, проходить через міжмережний екран, і до них можуть застосовуватися політики міжмережного екрана.

Більшість мережних архітектур є ієрархічними. Це означає, що єдиний маршрут ззовні розділяється на кілька маршрутів усередині мережі. Уважається, що ефективніше всього розміщати міжмережний екран у тім місці, де відбувається розгалуження маршруту. Перевага цього складається також і в тому, що в цьому випадку не виникає питання, що вважати "поза", а що вважати "усередині". Проте можуть існувати також причини, по яких міжмережні екрани варто встановлювати усередині мережі, наприклад, для захисту однієї безлічі

комп'ютерів від іншого. Якщо мережна архітектура не є ієрархічної, то ті самі політики міжмережного екрана повинні використовуватися для всіх крапок доступу в мережу. Часто забувають, що в мережу існує кілька входів і ставлять міжмережний екран тільки на одному з них, залишаючи інші відкритими. У цьому випадку шкідливий трафік, що блокується на основному вході, може проникнути в мережу іншими способами.

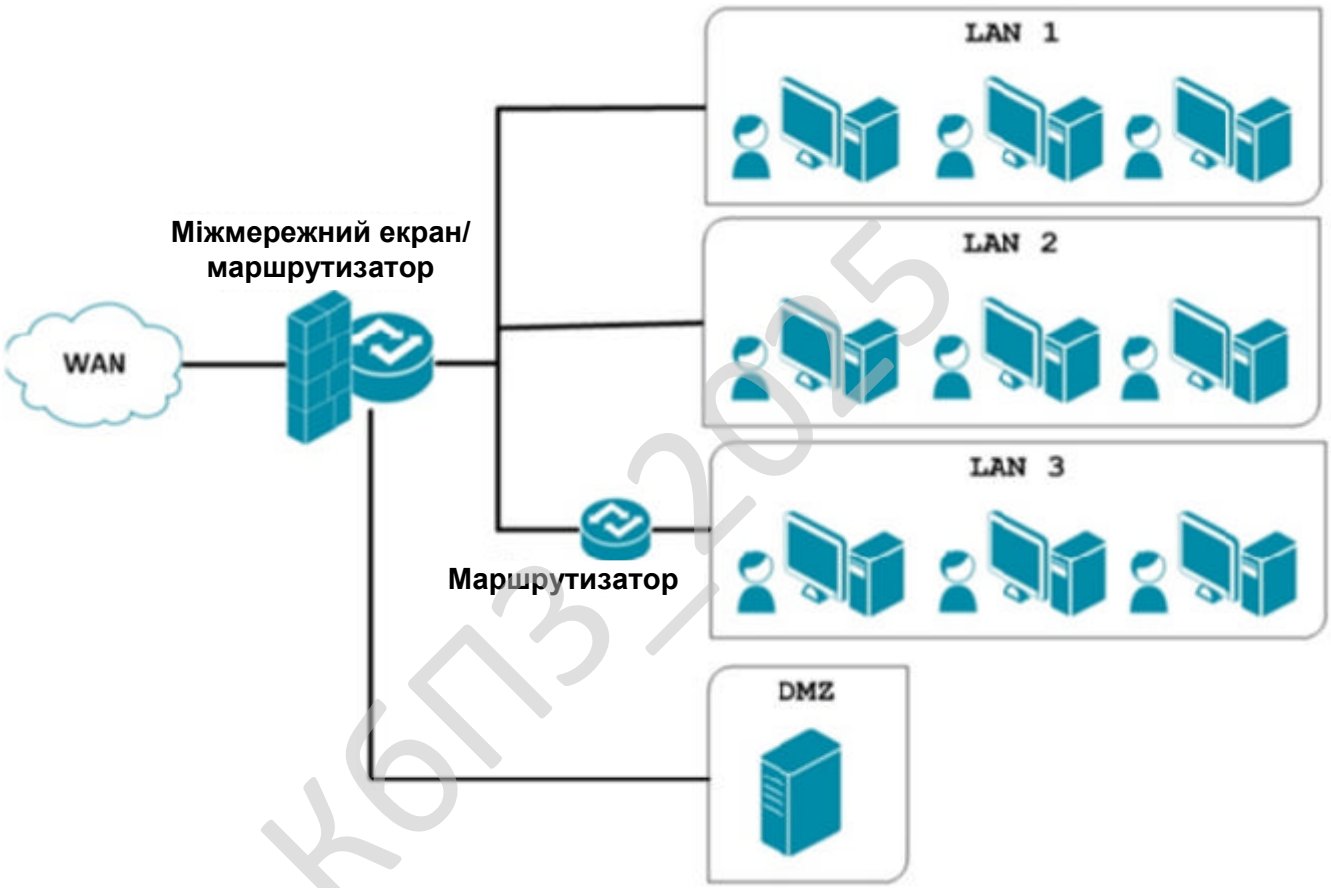

Рисунок 3.1 – Структурна схема системи

Деякі виробники пропонують відказостійкі (НА) міжмережні екрани, що означає, що один міжмережний екран заміняється іншим, якщо перший з якихось причин відмовив. НА міжмережні екрани розгортаються в парі в тому самому місці сегмента мережі, тим самим вони обоє мають ті самі зовнішні й внутрішні з'єднання. Хоча НА міжмережні екрани й збільшують надійність, вони можуть

внести й додаткові проблеми, тому що виникає необхідність комбінувати логи між цими міжмережними екранами, можлива також плутанина в адміністраторів при конфігуруванні таких міжмережних екранів (наприклад, який міжмережний екран пересилає зміни політики іншому). Функціональність НА може надаватися за допомогою технологій, розроблених виробниками.

Принципи побудови оточення міжмережного екрана

Простота (Keep It Simple). Даний принцип говорить про першому й основному, про що треба пам'ятати при розробки топології мережі, у якій функціонує міжмережний екран. Важливо приймати найбільш прості рішення – більше безпечним є те, чим легше управляти. Важкі функціональності часто приводять до помилок у конфігурації.

Використовувати пристрою по призначенню. Використання мережних пристроїв для того, для чого вони спочатку призначалися, у даному контексті означає, що не слід робити міжмережні екрани з устаткування, що не призначено для використання в якості міжмережного екрана. Наприклад, маршрутизатори призначені для виконання маршрутизації; можливості фільтрування пакетів не є їхньою вихідною метою, і це завжди треба враховувати при розробки оточення міжмережного екрана. Залежність винятково від можливості маршрутизатора забезпечувати функціональність міжмережного екрана небезпечна: він може бути легко переконфігурован. Іншим прикладом є мережні комутатори (switch): коли вони використовуються для забезпечення функціональності міжмережного екрана поза оточенням міжмережного екрана, вони чутливі до атак, які можуть порушити функціонування комутатора. У багатьох випадках гібридні міжмережні екрани й апаратні пристрої міжмережних екранів є кращим вибором, тому що вони оптимізовані в першу чергу для функціонування в якості міжмережних екранів.

Створювати оборону вглиб. Оборона вглиб означає створення декількох рівнів захисту на противагу наявності єдиного рівня. Не треба весь захист забезпечувати винятково міжмережним екраном. Де може використовуватися

трохи міжмережних екранів, вони повинні використовуватися. Де маршрутизатори можуть бути зконфігуровані для надання деякого керування доступом або фільтрації, це варто зробити. Якщо ОС сервера може надати деякі можливості міжмережного екрана, це варто застосувати.

Приділяти увагу внутрішнім погрозам. Нарешті, якщо приділяти увагу тільки зовнішнім погрозам, те це приводить у тому, що мережа стає відкритої для атак зсередини. Хоча це й малоймовірно, але варто розглядати можливість, що порушник може якось обійти міжмережний екран і одержати свободу дій для атак внутрішніх або зовнішніх систем. Отже, важливі системи, такі, як внутрішні веб- або e-mail-сервери або фінансові системи, повинні бути розміщені за внутрішнім міжмережних екранів або DMZ-Зон.

Як підсумок помітимо, що вираження "весь захист можна зламати" особливо застосовне до побудови оточень міжмережного екрана. При розгортанні міжмережних екранів варто пам'ятати про перераховані вище правилах для визначення оточень, але в кожному випадку можуть мати місце свої власні вимоги, можливо, що вимагають унікальних рішень.

Архітектура з декількома рівнями міжмережних екранів

Не існує ніяких обмежень на те, де можна розміщати в мережі міжмережний екран. Міжмережні екрани можуть розташовуватися на входах у границі логічної мережі, визначаючи поняття "усередині" і "поза" відносно міжмережного екраном, але адміністратор може захотіти мати додаткові границі усередині мережі й розгорнути додаткові міжмережні екрани для позначення цих границь. Використання декількох рівнів міжмережних екранів є типовим способом створення " оборони-в-глиб".

Типова ситуація, коли потрібно кілька рівнів міжмережних екранів, розташованих у мережі, ця наявність внутрішніх користувачів з різними рівнями довіри. Наприклад, необхідно захистити базу даних з обліковими записами користувачів від співробітників, які не повинні мати до неї доступ. Це можна зробити, розміщаючи один міжмережний екран на вході в мережу (запобігаючи

необмеженому доступу в мережу з інтернету) і іншої на вході у внутрішню мережу, тим самим визначаючи границю відділу кадрів. Внутрішній міжмережний екран буде блокувати доступ до сервера бази даних будь-якого ззовні мережі відділу кадрів, але дозволяти обмежений доступ до інших ресурсів у мережі відділу кадрів. Іншим типовим прикладом використання міжмережних екранів усередині мережі поряд з міжмережним екранів на вході в мережу є наявність недоверяемих користувачів, наприклад, випадкових відвідувачів, яким необхідний доступ в інтернет. Часто для відвідувачів створюється можливість бездротового доступу в мережу. Міжмережний екран між крапками доступу й внутрішньою мережею, що залишилася, може запобігти доступу відвідувачів у локальну мережу із привілеями співробітників.

Розташування міжмережного екрана усередині при наявності міжмережного екрана на вході повинне бути ретельно сплановано, щоб запобігти випадковим помилкам, що впливають на безпеку. При розробки політики для внутрішнього міжмережного екрана можуть бути зроблені неправильні припущення, що необхідно досить слабку політику – наприклад, якщо адміністратор внутрішнього міжмережного екрана буде вважати, що зовнішній міжмережний екран уже запобігає основним типам небажаного трафіку, а адміністратор зовнішнього міжмережного екрана згодом змінить існуючу політику, то для хостів за внутрішнім міжмережним екраном можуть виникнути додаткові погрози. Більше гарний підхід складається в дублюванні політики зовнішнього міжмережного екрана. Це може бути досить важко, якщо ці міжмережні екрани не мають можливості автоматично координувати свої політики, що буває досить часто, якщо міжмережні екрани розроблені різними виробниками.

Іншою проблемою, пов'язаної з використанням декількох рівнів міжмережних екранів, розташованих у мережі, є зростання складності трасування при виникненні яких-небудь проблем. Якщо між користувачем і сервером перебуває один міжмережний екран, і користувач не може з'єднатися із сервером,

то досить перевірити логи міжмережного екрана. Але при наявності декількох міжмережних екранів необхідно визначити послідовність їхнього проходження й перевірити всі логи. Наявність декількох рівнів шлюзів прикладного рівня особливо важко, тому що кожний шлюз може змінити повідомлення.

DMZ-мережі

Конфігурація з однієї DMZ-мережею

У більшості випадків оточення міжмережного екрана утворить так звану DMZ-мережа або мережа демілітаризованої зони.

DMZ-мережі призначені для розташування систем і ресурсів, які не повинні бути розміщені у внутрішніх захищених мережах, але до яких необхідний доступ або тільки ззовні, або тільки зсередини, або й ззовні, і зсередини. Причина в тому, що ніколи не можна гарантувати, що ці системи й ресурси не можуть бути зламані. Але злом цих систем не повинен автоматично означати доступ до всіх внутрішніх систем.

DMZ-мережі звичайно будуються з використанням мережних комутаторів і розташовуються між двома міжмережними екранами або між міжмережним екраном і прикордонним маршрутизатором. Гарною практикою є розміщення серверів вилученого доступу й кінцевих крапок VPN в DMZ-мережах. Розміщення цих систем в DMZ-мережах зменшує ймовірність того, що вилучені атакуючі будуть мати можливість використовувати ці сервери як крапка входу в локальні мережі. Крім того, розміщення цих серверів в DMZ-мережах дозволяє міжмережним екранам служити додатковими засобами для контролю прав доступу користувачів, які одержують доступ з використанням цих систем до локальної мережі.

Service Leg конфігурація

Однієї з конфігурацій DMZ-мережі є так звана "service leg" конфігурація міжмережного екрана. У цій конфігурації міжмережний екран має як мінімум три мережних інтерфейси. Один мережний інтерфейс з'єднується з інтернетом, іншої з'єднується із внутрішньою мережею, третій мережний інтерфейс формує DMZ-

мережа. Така конфігурація може привести до зростання ризику для міжмережного екрана при DoS-атаці, що буде націлена на сервіси, розташовані в DMZ-мережі. У стандартній конфігурації DMZ-мережі DoS-атака, спрямована на розташовані в DMZ-мережі ресурс, такий як веб-сервер, буде впливати тільки на цей цільовий ресурс. В service leg конфігурації DMZ-мережі міжмережний екран бере на себе основний удар від DoS-атаки, тому що він повинен перевіряти весь мережний трафік перед тим, як трафік досягне розташованого в DMZ ресурсу. У результаті це може вплинути на весь трафік організації, якщо на її веб-сервер виконаний DoS-атака.

Конфігурація із двома DMZ-мережами

При наявності великої кількості серверів з різними вимогами доступу можна розгорнути міжмережний екран з можливостями прикордонного маршрутизатора й два внутрішніх міжмережних екрани й розмістити всі зовні доступні сервери в зовнішньої DMZ між маршрутизатором і першим міжмережним екраном. Прикордонний маршрутизатор буде фільтрувати пакети й забезпечувати захист серверів, перший міжмережний екран буде забезпечувати керування доступом і захист серверів внутрішньої DMZ у випадку, якщо зовнішні сервера атаковані. Внутрішньо доступні сервери розміщаються у внутрішньої DMZ, розташованої між основним і внутрішнім міжмережними екранами; міжмережні екрани будуть забезпечувати захист і керування доступом для внутрішніх серверів, захищених як від зовнішніх, так і від внутрішніх атак.

Зовнішня DMZ-мережа з'єднана з інтернетом з використанням пакетного фільтра, що одночасно є прикордонним маршрутизатором. Раніше обговорювалося, чому використання пакетного фільтра є більше кращим.

Основний міжмережний екран є VPN-шлюзом для вилучених користувачів; такі користувачі повинні мати ПЗ VPN-клієнта для з'єднання з міжмережним екраном.

Вхідний SMTP-трафік повинен пропускатися основним міжмережним екраном.

Основний і внутрішній міжмережні екрани повинні підтримувати технологію аналізу станів і можуть також включати можливості прикладного проксі. Основний міжмережний екран повинен виконувати наступні дії:

– дозволяти зовнішнім користувачам, які були автентифіковані VPN-сервером, доступ у локальну мережу й DMZ-мережа;

– якщо основний міжмережний екран має SMTP-проксі, виконувати фільтрацію SMTP-трафіку;

– дозволяти вихідний трафік з локальної мережі.

Внутрішній міжмережний екран повинен приймати вхідний трафік тільки від основного міжмережного екрана й SMTP-сервера. Нарешті, він повинен дозволяти всі вихідні з'єднання від внутрішніх систем.

Щоб зробити даний приклад застосовним до оточень із більше високими вимогами до безпеки, можуть бути додані наступного сервера й використані наступні технології:

– внутрішні й зовнішній DNS-сервери, що забезпечить приховання внутрішніх систем;

– NAT для подальшого приховання внутрішніх систем;

– вихідний трафік від внутрішніх хостів може фільтруватися, що включає фільтрування трафіку до певних сайтів або сервісів відповідно до політики керування;

– може бути використане трохи міжмережних екранів як для збільшення продуктивності, так і для розмежування трафіку між відділами.

Кінцеві крапки VPN

Іншою функціональністю, який звичайно володіють міжмережні екрани, є можливість функціонування як кінцева крапка VPN. VPN створюється поверх існуючого мережного середовища й протоколів з використанням додаткових протоколів, що забезпечують шифрування й цілісність трафіку.

VPN застосовується для забезпечення безпечних мережних з'єднань із використанням мереж, які не є що довіряються. Наприклад, технологія VPN всі

частіше створюється для надання вилученого доступу користувача до мереж організації через інтернет. Дана технологія користується зростаючою популярністю, тому що це істотно знижує витрати, пов'язані з можливістю безпечного вилученого доступу, у порівнянні з використанням виділених каналів зв'язку. При використанні VPN з'єднання з інтернетом може також використовуватися для безпечного вилученого доступу користувачів до мереж і ресурсів організації.

З погляду використовуваного протоколу, існує кілька можливих виборів для створення VPN. Найбільше часто використовується сімейство протоколів IPSec і протокол L2TP.

У більшості випадків найбільш прийнятним є сполучення кінцевої крапки VPN і міжмережного екрана. Як правило, міжмережний екран використовує IPSec для з'єднання з вилученими системами, а до внутрішніх мереж передає незашифрований трафік. Розміщення кінцевої крапки VPN за міжмережного екраном означає, що VPN-трафік буде передаватися через міжмережний екран у зашифрованому виді, тобто міжмережний екран не буде мати можливість аналізувати вхідний або вихідний VPN-трафік, виконувати керування доступом, створювати логи, сканувати на віруси й т.п.

Однак сполучення міжмережного екрана й кінцевої крапки VPN має й свої негативні сторони. Одним з таких недоліків є висока вартість. Також, тому що VPN-трафік повинен бути зашифрований, те це істотно зменшує продуктивність міжмережного екрана. Виконання шифрування в апаратурі може істотно збільшити продуктивність.

Інтранет

Інтранетом називається мережа, що заснована на тих же протоколах і, отже, може виконувати ті ж самі сервіси й додатки, які використовуються в інтернеті, але без наявності з'єднання з інтернетом. Наприклад, мережа підприємства, побудована на сімействі протоколів TCP/IP, можна розглядати як інтранет.

Більшість організацій у цей час має деякий тип інтранет. У внутрішній мережі (інтранет) можуть бути створені ще менші інтранет, використовуючи внутрішні міжмережні екрани. Наприклад, можна захистити свою власну персональну мережу внутрішнім міжмережним екраном, і захищена мережа, що вийшла, може розглядатися як персональна інтранет.

Тому що інтранет використовує ті ж самі протоколи й прикладні сервіси, що й інтернет, багато роблем безпеки, що існують в інтернеті, також присутні в інтранет.

Екстранет

Термін екстранет застосовується до мережі, що логічно складає із трьох частин: два інтранет з'єднані між собою через інтернет з використанням VPN. Екстранет може бути визначена як business-to-business інтранет. Ця мережа дозволяє забезпечити обмежений, керований доступ вилучених користувачів за допомогою тої ж форми автентифікаціи й шифрування, які є в VPN.

Екстранет має ті ж самі характеристики, що й інтранет, за винятком того, що екстранет використовує VPN для створення захищених з'єднань через публічний інтернет. Метою інтранет є надання доступу до потенційно чутливої інформації вилученим користувачам або організаціям, але при цьому забороняючи доступ всім іншим зовнішнім користувачам і системам. Екстранет використовує протоколи TCP/IP і ті ж самі стандартні додатки й сервіси, які використовуються в інтернеті.

Компоненти інфраструктури: комутатори

Додатково до маршрутизаторів і міжмережним екранам, зв'язок між системами забезпечують такі інфраструктурні пристрої, як комутатори (switches).

Основними інфраструктурними пристроями є мережні комутатори. Ці пристрої функціонують на 2 рівні моделі OSI. Основна відмінність комутаторів полягає в тому, що вони передають пакети тільки потрібному адресатові, тому хости, приєднані до мережі за допомогою комутаторів, не можуть "підглядати" трафік один одного, і, як наслідок, комутатори краще підходять для створення

DMZ-мереж і оточень міжмережних екранів. Важливо помітити, що комутатори не повинні використовуватися для надання яких-небудь можливостей міжмережного екрана або забезпечення ізолювання трафіку поза оточенням міжмережного екрана, тому що комутатори не можуть запобігати можливі DoS-атаки.

Розташування серверів в DMZ-мережах

Де розташувати сервери при наявності міжмережних екранів, залежить від багатьох факторів, включаючи кількість DMZ, необхідність зовнішнього й внутрішнього доступу до серверів, розташованим в DMZ, інтенсивність трафіку й чутливість оброблюваних даних. Неможливі абсолютно універсальні рекомендації з розташування серверів, але основні принципи повинні бути наступними.

Варто ізолювати сервери таким чином, щоб успішні атаки на них не могли заподіяти збитку частини, що залишилася, мережі, зокрема, не слід розміщати зовні доступні сервери в захищеній мережі.

Варто захистити зовні доступні сервери за допомогою прикордонного маршрутизатора з можливостями пакетного фільтра.

Варто розмістити внутрішньо доступні сервери в DMZ-мережах, у яких забезпечується захист як від зовнішніх, так і від внутрішніх атак, оскільки звичайно ці сервери містять більше чутливі дані й до них потрібно більше обмежений доступ.

Визначимо деякі принципи розміщення серверів і систем. Хоча розміщення серверів визначається кожною організацією окремо, виходячи з конкретних вимог, всі зусилля повинні бути спрямовані на захист серверів як від зовнішніх, так і від внутрішніх погроз, і на ізоляцію успішних атак на сервери, щоб вони не впливали на частину, що залишилася, мережі.

Зовні доступні сервери

Зовні доступні HTTP-сервери, а також сервери каталогу або DNS-сервери, можуть бути розміщені в зовнішньої DMZ, тобто між прикордонним

маршрутизатором з функціями пакетного фільтра й основним міжмережним екраном. Прикордонний маршрутизатор може забезпечити деяке керування доступом і фільтрацію трафіку для серверів, а основний міжмережний екран – запобігти створенню з'єднань від серверів до внутрішніх систем, які можуть виникнути, якщо сервери будуть зламані. У випадку великого трафіку й сильної завантаженості серверів може використовуватися високошвидкісний прикордонний маршрутизатор з декількома приєднаними DMZ для ізолювання серверів в індивідуальних DMZ-мережах. Таким чином, якщо здійснюється DDo-атака на деякий сервер, інші сегменти мережі не постраждають.

VPN- і Dial-in-сервери

Ці сервери краще розмістити в зовнішньої DMZ, щоб їх трафік проходив у локальну мережу через основний міжмережний екран. Одна з можливих конфігурацій складається в сполученні VPN-сервера й міжмережного екрана, щоб вихідний трафік міг бути зашифрований після того, як він буде відфільтрований, і вхідний трафік може бути розшифрований і потім відфільтрований міжмережним екраном. Dial-in сервер повинен бути розміщений у зовнішньої DMZ по тимі ж причинам.

Внутрішні сервери

Внутрішньо доступні HTTP-сервери, SMTP-сервери й сервери каталогу можуть бути розміщені у внутрішньої DMZ, тобто між двома міжмережними екранами, основним і внутрішнім; при цьому внутрішній міжмережний екран відокремлює внутрішню DMZ від захищеної мережі. Розміщення цих систем у внутрішньої DMZ забезпечує оборону вглиб, захищаючи як від зовнішніх, так і від внутрішніх погроз. Якщо HTTP-проксі використовується для вихідного трафіку, розміщення цих систем у внутрішньої DMZ забезпечує більший захист від внутрішніх і зовнішніх погроз.

DNS-сервери

DNS є критичним сервісом у будь-якому оточенні, у якому використовується інтернет. У силу чутливої природи даного сервісу, з одного

боку, і надання їм інформації про зовні доступні сервіси, з іншої, необхідні спеціальні міри безпеки.

По-перше, внутрішні DNS-сервери повинні бути відділені від зовнішніх DNS-серверів. Наприклад, DNS-сервер, що доступний усьому світу, не повинен містити записів про системи, які не повинні бути доступні ззовні. Якщо такі записи про внутрішні системи є в зовнішньому DNS-сервері, це дасть можливість атакуючий визначити список цілей для атаки. Варто підтримувати окремо внутрішні й зовнішній DNS-сервери або використовувати технологію, відому як split DNS, що гарантує, що інформація про внутрішні системи ніколи не буде передана зовні.

По-друге, необхідно встановити дозволені типи доступу до DNS-сервера. Додаток DNS-сервера може виконуватися з використанням двох транспортних протоколів: клієнт звертається до сервера по протоколі UDP, а взаємодія двох серверів DNS, що виконують зонні пересилання, реалізовано з використанням TCP. Доступ до сервера DNS з використанням TCP повинен бути обмежений тільки для тих серверів DNS, які повинні виконувати зонні пересилання. Основний ризик, що існує при функціонуванні DNS, складається в модифікації переданої інформації. Наприклад, якщо сервер допускає неавтентифіковані запити й відповіді DNS, що атакує може модифікувати інформацію, у результаті чого мережний трафік буде переспрямований на інший хост. На рисунку показаний приклад топології мережі із двома DNS-серверами. Внутрішній DNS-сервер повинен бути зконфігурован для дозволу імен внутрішніх серверів, щоб внутрішні користувачі могли з'єднуватися з ними, всіма серверами в DMZ і інтернетом. Зовнішній DNS-сервер повинен забезпечувати дозвіл імен самого DNS-сервера й серверів у зовнішньої DMZ, але не у внутрішній мережі. Як результат, сервери в зовнішньої DMZ будуть видимі в інтернеті.

Планування й впровадження міжмережного екрана

Розглянемо проблеми, пов'язані із плануванням і впровадженням міжмережного екрана. Успішне розгортання міжмережного екрана може бути

досягнуто чітким, крок за кроком певним плануванням і наступною реалізацією цього плану. Використання для розгортання підходу, заснованого на етапах, може мінімізувати непередбачені проблеми й визначити потенційні підводні камені. Розглянемо основні етапи планування й впровадження міжмережного екрана:

Планування. На першому етапі необхідно визначити всі вимоги до міжмережного екрана, які необхідно виконати для реалізації політики безпеки.

Конфігурування. Другий етап включає всі аспекти конфігурування платформи, на якій виконується міжмережний екран. Це має на увазі установку апаратури й ПЗ, а також розробку правил.

Тестування. Наступний етап включає реалізацію й тестування прототипу в лабораторному або тестовому оточенні. Вихідна мета тестування складається в оцінці функціональностей, продуктивності, масштабованості й безпеці, а також у визначенні різних проблем, таких як інтероперабельність, з різними компонентами іншого ПЗ.

Розгортання. Після того, як тестування завершене, і всі проблеми вирішені, на наступному етапі варто розгортати міжмережний екран у реальних умовах.

Керування. Після того, як міжмережний екран розгорнуть, необхідно управляти їм протягом усього життєвого циклу його компонентів і вирішувати виникаючі проблеми.

Планування

На етапі планування здійснюється вибір міжмережного екрана й визначення, яку саме політику безпеки повинен реалізовувати міжмережний екран. Для цього необхідно оцінити можливі ризики:

– Ідентифікувати погрози й уразливості для кожної інформаційної системи.

– Визначити потенційний вплив і величину шкоди, що може нанести втрата конфіденційності, цілісності або доступності інформаційних активів організації або надаваних нею сервісів.

– Проаналізувати можливості керування обраним міжмережним екраном.

Основні принципи, яким необхідно випливати при плануванні розгортання міжмережного екрана:

Використовувати пристрої по призначенню. Наприклад, маршрутизатори призначені для виконання маршрутизації, вони не можуть виконувати складного фільтрування, що може означати надлишкове навантаження на процесор маршрутизатора. Також уважається, що міжмережні екрани не повинні надавати сервісів, які не ставляться до безпеки, такі як веб-сервер або поштовий сервер.

Створювати оборону в глиб. Це означає наявність декількох рівнів оборони. У цьому випадку при компрометації одного компонента атака може бути відбита іншим. У випадку міжмережних екранів це може бути реалізовано використанням декількох міжмережних екранів, один із яких розташований на границі мережного периметра, іншої на границі відділу із чутливою інформацією або на комп'ютері, що захищається. Щоб оборона вглиб була більше ефективною, міжмережні екрани повинні бути частиною загальної програми забезпечення безпеки, що включає й інші технології, такі як виявлення шкідливого коду й IDS.

Приділяти увагу внутрішнім погрозам. Якщо приділяти увагу винятково зовнішнім погрозам, то мережа стає відкритої для атак зсередини. Ці погрози можуть виходити не тільки безпосередньо від співробітників, але можуть виникнути в результаті того, що внутрішні хости інфіковано шкідливим кодом або якимось іншим способом скомпрометовані зовнішніми атакуючими. Важливо, щоб внутрішні системи були розташовані за внутрішнім міжмережним екраном, що фільтрує не тільки вхідний, але й вихідний трафік. Варто пам'ятати, що вираження "всі правила призначені для того, щоб їх порушувати" застосовне й до міжмережним екранів. У кожній мережі можуть існувати якісь унікальні вимоги й особливості, які вимагають унікальних рішень. При виборі міжмережного екрана варто проаналізувати потреби організації й можливості міжмережного екрана.

Призначення міжмережного екрана

Що повинне захищатися (периметр, внутрішні відділи, вилучений офіс, окремі хости, конкретні сервіси, мобільні клієнти й т.п.).

Які типи технологій міжмережних екранів найкраще підходять для трафіку, що повинен бути захищений (фільтрування пакетів, аналіз стану, прикладний міжмережний екран, шлюз прикладного проксі й т.п.).

Які додаткові можливості безпеки – такі як можливості виявлення проникнення, VPN, фільтрування вмісту – повинен підтримувати міжмережний екран.

Можливості керування міжмережним екраном

Які протоколи повинен підтримувати міжмережний екран для вилученого керування, наприклад, HTTP-поверх-SSL, SSH або доступ по послідовному порту.

Чи можуть бути відключені протоколи вилученого керування, якщо вони не прийнятні для організації й не відповідають її організаційній політиці.

Чи може вилучене керування бути обмежено певними інтерфейсами на міжмережному екрані й IP-адресами джерела, наприклад, що належать конкретної внутрішньої мережі.

Чи підтримує міжмережний екран централізоване керування декількома пристроями (не обов'язково тільки міжмережними екранами) від одного виробника.

Якщо централізоване керування можливо, чи виконується воно специфічним для виробника додатком або може виконуватися стандартними додатками, наприклад, через веб-інтерфейс.

Продуктивність – звичайно особливо важливо для міжмережних екранів, розташованих на границі мережного периметра.

Яку пропускну здатність, максимальну кількість одночасно відкритих з'єднань, з'єднань у секунду може забезпечувати міжмережний екран.

Чи можливе балансування навантаження й резервування для гарантування високої відказостійкості.

Що є більше кращим – апаратний або програмний міжмережний екран.

Інтеграція

Потрібно чи для міжмережного екрана спеціалізована апаратура для коректної інтеграції в мережну інфраструктуру організації (специфічні вимоги підключення до електрики, специфічний тип мережного інтерфейсу (NIC), специфічні пристрої резервного копіювання й т.п.).

Чи необхідна для міжмережного екрана сумісність із іншими пристроями в мережі або сервісами, які забезпечують безпеку.

Чи існує інтероперабельність логів, створюваних міжмережним екраном, з існуючими системами керування балками.

Чи зажадає установка міжмережного екрана яких-небудь змін в інших сегментах мережі.

Фізичне оточення – звичайно розглядається для міжмережних екранів, установлених у мережі, але може також застосовуватися до централізованих компонентів міжмережних екранів для хостів.

Де буде фізично розташовуватися міжмережний екран і як буде гарантуватися його фізичний захист.

Чи досить силового кабелю, кондиціонерів, мережних з'єднань у тім місці, де буде розташовуватися міжмережний екран.

Персонал

Хто відповідає за керування міжмережним екраном.

Чи необхідно навчання системних адміністраторів перед впровадженням міжмережного екрана.

Майбутні потреби

Які можуть виникнути потреби в майбутньому (планується перехід на IPv6, очікувані вимоги до пропускної здатності, сумісність із регламентними вимогами й т.п.).

Також при покупці й впровадженні персональних міжмережних екранів і міжмережних екранів для хостів варто розглянути:

Чи задовольняють робочі станції й сервера мінімальним системним вимогам, які необхідні для функціонування міжмережного екрана.

Чи буде міжмережний екран сполучимо з іншим ПЗ забезпечення безпеки на робочій станції або сервері (наприклад, з ПЗ виявлення шкідливого коду).

Чи можливо централізоване керування міжмережним екраном і чи можуть політики, які забезпечують безпека організації, автоматично завантажуватися на клієнтські машини.

Чи можуть звіти про порушення з міжмережного екрана передаватися на центральний сервер.

Чи може блокуватися міжмережний екран ким-небудь, крім адміністратора, і чи може хто-небудь змінити установки міжмережного екрана.

Чи буде міжмережний екран конфліктовать із персональним міжмережним екраном, убудованим в ОС. Якщо так, то як легко перебороти цей конфлікт.

Конфігурування

Етап конфігурування включає всі аспекти конфігурування платформи міжмережного екрана. Це включає установку апаратури й ПЗ, конфігурування політик, конфігурування створення логів і оповіщень, інтегрування міжмережного екрана в мережну архітектуру.

Установка апаратури й ПЗ

Після того, як міжмережний екран обраний і придбаний, для програмного міжмережного екрана необхідно встановити апаратуру, ОС і лежаче в основі ПЗ. Далі, як для апаратних, так і для програмних міжмережних екранів варто встановити всі зміни й модифікації, передбачені виробником. На цьому етапі міжмережний екран повинен бути посилений, щоб зменшити ризик появи уразливостей і захистити систему від неавторизованого доступу. Також у цей момент варто встановити консоль вилученого доступу.

Під час інсталяції й конфігурування тільки адміністратор повинен мати можливість управляти міжмережним екраном. Всі інші сервіси керування міжмережним екраном, такі як SNMP, повинні бути заборонені. Вони повинні залишатися забороненими доти, поки вони явно не стануть потрібні. Якщо міжмережний екран підтримує окремі адміністративні облікові записи для адміністраторів з різними правами доступу, у цей момент треба зконфігурувати ці облікові записи.

Конфігурувати міжмережний екран треба в приміщенні, у якому відсутній неавторизований доступ.

Для аналізу проблем дуже важливо мати можливість порівняння логів декількох систем, тому внутрішні годинники кожного міжмережного екрана повинні бути коректно встановлені. Кращим способом зробити це є наявність авторизованого джерела часу, з яким будуть синхронізуватися всі годинники.

Конфігурування політики

Після того, як апаратура й ПЗ встановлені й забезпечена їхня безпека, необхідно створити політику міжмережного екрана. Деякі міжмережні екрани реалізують політику за допомогою явно заданих правил; інші вимагають конфігурування установок міжмережного екрана, з яких потім будуть створені внутрішні правила. Кінцевим результатом є набір правил, називаний ruleset, що описує функціонування міжмережного екрана. Послідовність правил у наборі істотно впливає на продуктивність міжмережного екрана. Більшість міжмережних екранів також дозволяють конфігурувати параметри, які задають оточення міжмережного екрана, наприклад, хто може переглядати або змінювати правила, де перебувають зовнішні DNS-сервера й сервера синхронізації часу.

Завдання цих параметрів також повинне відповідати політики безпеки організації. Трафік до цих систем повинен також оброблятися правилами міжмережного екрана. Для створення набору правил, по-перше, варто визначити типи трафіку (протоколи, адреси джерел і одержувачів і т.п.), які потрібні як розташованим у мережі додаткам, так і самому міжмережному екрану. Це

повинне включати протоколи, які необхідні самому міжмережному екрану (DNS, SNMP, NTP, створення логів і т.п.).

Деталі створення набору правил залежать від типу міжмережного екрана й специфічні для кожного продукту. Наприклад, багато які міжмережні екрани послідовно порівнюють трафік із правилами доти, поки не буде знайдена відповідність. Для таких міжмережних екранів правила, які відповідають найбільшій кількості трафіку, повинні розташовуватися якнайвище в списку, щоб збільшити продуктивність міжмережного екрана. Деякі міжмережні екрани можуть мати більше складні способи обробки набору правил, такі як спочатку перевірка правил "drop (deny)", а потім перевірка правил "allow".

Більшість міжмережних екранів дозволяє вводити коментар для кожного правила в наборі. Написання таких коментарів важливо, щоб надалі можна було згадати, чому те або інше правило було написано. Коментарі також корисні для перевірки набору правил. Всі зміни й відповідні коментарі повинні записуватися в балках керування конфігурацією.

Якщо потрібно, щоб трохи міжмережних екранів мали ті самі набори правил, то ці правила треба синхронізувати між цими міжмережними екранами. Звичайно спосіб виконати це залежить від виробника. Помітимо також, що міжмережні екрани можуть також мати політики, які залежать від їхнього розташування в мережі. Наприклад, організація може захотіти мати тільки один міжмережний екран, що функціонує як VPN-шлюз, а правила фільтрування для не-VPN трафіку будуть однакові на всіх міжмережних екранах. Отже, важливо мати можливість синхронізувати тільки ті правила, які є загальними для всіх міжмережних екранів.

Конфігурування створення логів і оповіщень

Наступним кроком є установка логів і оповіщень. Створення логів є критичним кроком для забезпечення відказостійкості й відновлення після збоїв, а також для гарантування, що на міжмережний екран установлений коректна конфігурація з погляду безпеки. Правильно створені логи також надають життєво

важливу інформацію при розслідуванні інцидентів, пов'язаних з безпекою. У кожному разі міжмережний екран повинен бути зконфігурован як для зберігання логів локально, так і для відсилання їх у централізовану інфраструктуру керування балками. Обмеженість ресурсів, можливості створення логів міжмережним екраном і інші ситуація можуть погіршувати можливість зберігання логів як локально, так і централізовано.

Рішення про те, що записувати в логи і як довго них зберігати може мінятися залежно від ситуації. Наприклад, адміністратор може вирішити записувати в балку всі дозволені вхідні з'єднання, тим самим він зможе проаналізувати надалі, що не був дозволений небажаний трафік. В іншому випадку він може не захотіти записувати в балку всі дозволені вхідні з'єднання, тому що їхня величезна кількість, і такі логи вимагають величезної кількості ресурсів. Аналогічно, деякі адміністратори не захочуть записувати в логи весь заборонений міжмережним екраном вхідний трафік, тому що кількість сканувань у їхній мережі дуже велико, але ніяких дій у відповідь на це не передбачається. Однак в інших випадках адміністратор може захотіти знать про сканування, тому що це може вказувати про потенційну атаку.

Якщо міжмережний екран підтримує можливість створення декількох адміністративних облікових записів з різними можливостями, варто створити одну або кілька адміністративних облікових записів з доступом по читанню до логів. Такі креденциали варто використовувати при виконанні завдань, пов'язаних з тільки читанням, таких як аудит і періодичний аналіз логів.

На додаток до конфігурування логів необхідно встановити оповіщення в реальному часі для повідомлення адміністраторів про важливі події, які відбулися на міжмережном екрані. Повідомлення можуть включати наступне:

Будь-яка модифікація або заборона правил міжмережного екрана.

Перезапуск системи, недостача дискового простору й інші події функціонування ОС.

Практично всі системи міжмережного екрана забезпечують ту або іншу функціональність створення логів. Як обговорювалося раніше, логи в прикладному проксі є більше об'ємними, чим логи пакетних фільтрів або міжмережних екранів з аналізом стану. Причина в тому, що прикладні фільтри аналізують більше число рівнів стека протоколів, чим пакетні фільтри.

Найбільше широко використовуваним додатком для створення логів є syslog в ОС UNIX. UNIX syslog призначений для централізованого створення логів, крім того він має багато опцій для їхньої перевірки й обробки. Дана програма створення логів доступна практично у всіх основних ОС, включаючи Windows NT, 2xxx і всі різновиди UNIX і Linux.

Після того, як логи міжмережного екрана будуть передані централізованому серверу логів, можуть використовуватися різні пакети для їхньої обробки. Логи, сумісні з syslog, можуть також подаватися як вхід у пакети аналізу проникнення й використовуватися для судового розслідування.

Ті міжмережні екрани, які не підтримують інтерфейсу syslog, повинні використовувати свою власну внутрішню функціональність створення логів. Залежно від платформи міжмережного екрана, існує багато інструментальних пакетів третіх фірм для підтримки й обробки логів.

Тестування

Перед розгортанням міжмережний екран повинен бути протестуван і оцінений, щоб гарантувати, що він працює коректно. Тестування повинне виконуватися в тестовій мережі, не з'єднаної з реальною мережею. При цьому така тестова мережа повинна максимально повторювати виробничу, включаючи топологію мережі й мережний трафік, що буде проходити через міжмережний екран. Тестування повинне перевіряти наступні параметри:

Підключення. Користувачі можуть установлювати й підтримувати з'єднання через міжмережний екран.

Набір правил. Дозволений трафік пропускається політикою, не дозволений трафік блокується. Перевірка набору правил повинна включати як перегляд їх вручну, так і тестування роботи правил.

Сумісність із додатками. Міжмережні екрани для хостів і персональні міжмережні екрани не перешкоджають і не впливають на роботу існуючих додатків. Це включає мережні взаємодії між компонентами додатка. Міжмережний екран, розташований у мережі, не впливає на додатки, що мають компоненти, які повинні взаємодіяти по мережі (наприклад, клієнт-серверне ПЗ).

Керування. Адміністратори можуть конфігурувати й управляти міжмережним екраном ефективно й безпечно.

Створення логів. Функція створення й керування балками відповідає політиці й стратегії організації.

Продуктивність. Забезпечується адекватна продуктивність при нормальному й піковому використанні. У багатьох випадках кращим способом протестувати продуктивність є використання генераторів симуляторів трафіку, які створять аналоги реальних характеристик очікуваного трафіку. Можна також виконати симуляцію DoS-атак. Тестування повинне включати перевірку, що різні додатки проходять через міжмережний екран, особливо ті, на які впливає пропускна здатність мережі або проблеми, пов'язані із затримкою.

Безпека реалізації. Реалізація міжмережного екрана сама може містити уразливості, які може використовувати атакуючий. Варто виконати оцінку уразливостей різних компонентів міжмережного екрана.

Інтероперабельність компонент. Компоненти міжмережного екрана повинні спільно коректно функціонувати. Це особливо важливо, якщо використовуються компоненти від різних виробників.

Синхронізація політики. Якщо трохи міжмережних екранів повинні виконувати синхронізовану політику або групу правил, то варто протестувати функціонування синхронізації в різних сценаріях (наприклад, якщо один або кілька вузлів перебувають у режимі off-line).

Додаткові можливості. Додаткові можливості, які будуть використовуватися міжмережним екраном, такі як VPN і виявлення шкідливого коду, повинні бути протестовані, щоб гарантувати їхню коректну роботу.

Розгортання

Після завершення тестування й рішення всіх проблем починається етап розгортання. Перед цим варто повідомити всіх користувачів, на яких може вплинути розгортання міжмережного екрана. Будь-які зміни, які необхідно зробити у зв'язку з розгортанням міжмережного екрана, повинні розглядатися як частина розгортання самого міжмережного екрана. Політика безпеки, обумовлена конфігурацією міжмережного екрана, повинна бути додана в загальну політику безпеки організації, і будь-які зміни в його конфігурації повинні інтегруватися із процесами керування конфігураціями в організації. Якщо розгортається трохи міжмережних екранів, необхідний послідовний підхід. Можливо також пілотне розгортання, що допомагає виявити й вирішити можливі проблеми й конфлікти.

Необхідно інтегрувати міжмережний екран з іншими елементами мережі, з якими міжмережний екран взаємодіє. Тому що звичайно міжмережні екрани розташовані в тих же сегментах мережі, що й маршрутизатори, міжмережний екран повинен бути інтегрований у структуру маршрутизації мережі. Це часто означає заміну маршрутизатора, що розташовувався в тому же самому місці в мережній топології, що й заміяючий його міжмережний екран. Це може також означати зміну таблиць маршрутизації на інших маршрутизаторах. Якщо елементи в мережі використовують динамічну маршрутизацію, також необхідно виконати відповідні зміни. Якщо міжмережний екран установлюється в системі із забезпеченням відказостійкості, то також може знадобитися зміна конфігурації.

Керування

Останній етап є найбільш тривалим, тому що він включає супровід архітектури, політик, ПЗ міжмережного екрана й інших компонентів, які були розгорнуті. Однією з типових дій при супроводі є відновлення й наступне тестування міжмережного екрана. Може також знадобитися змінювати правила

політик при виявленні нових погроз і зміні вимог, таких, наприклад, як установка нових додатків або хостів. Також важливо стежити за продуктивністю різних компонентів міжмережного екрана, щоб гарантувати, що потенційні проблеми з ресурсами будуть вчасно виявлені. Для визначення погроз повинні постійно проглядатися логи й оповіщення. Іншим важливим завданням є періодичне тестування для перевірки того, що правила міжмережного екрана обробляють трафіку так, як очікувалося. Також необхідно періодично виконувати резервне копіювання політик і наборів правил міжмережного екрана. Деякі міжмережні екрани можуть зберігати цю інформацію в декількох форматах, таких як бінарний формат, що використовується для конфігурування міжмережного екрана, і текстовий формат, що може бути проаналізований людиною. Якщо доступно кілька форматів, то резервне копіювання необхідно робити для всіх.

Зміни набору правил або політик міжмережного екрана впливають на безпеку, і цим варто управляти як частиною формального процесу керування конфігурацією. Багато хто міжмережні екрани виконують аудита змін як частина власного адміністративного інтерфейсу, але це не означає відстеження змін політики. Як мінімум необхідно зберігати зміни всіх рішень політики й наборів правил. Більшість міжмережних екранів передбачають обмеження, які можуть бути зроблені в наборі правил, наприклад, обмеження, пов'язані з адресами, з яких адміністратори можуть вносити такі зміни.

Інциденти безпеки

Не існує простої відповіді на питання, що таке інцидент безпеки.

У загальному випадку інцидентом безпеки є будь-яка подія, у якому неавторизований індивід одержує або намагається одержати доступ до комп'ютерних систем або ресурсів, на які в нього немає привілеїв. Серйозність інциденту може відрізнятися, і компанія сама дає строге визначення інциденту безпеки.

При високому рівні шкали строгості інцидентом безпеки може вважатися зондування мережі або системи, що може використовуватися для вивчення

топології мережі. Якщо таке зондування виконує неавторизований користувач, інцидент безпеки має місце. При великій кількості подібних подій більшість компаній воліють уважати, що дані події не є інцидентом безпеки.

При середньому рівні шкали строгості інцидентом безпеки можна вважати ту або іншу форму активних спроб одержання неавторизованого доступу до комп'ютерної системи. При низькому рівні шкали інцидентом може вважатися будь-яка успішна спроба одержання неавторизованого доступу до системи або ресурсів. Ці події потенційно можуть ліквідувати доступність ресурсів і, отже, є серйозними. При ідентифікації інциденту деякі організації можуть спробувати переслідувати порушника. У кожному разі, інцидент повинен бути зареєстрований.

По суті, визначення інциденту безпеки визначається політикою безпеки організації.

Міжмережні екрани є важливими елементами при визначенні інциденту безпеки – вони вказують на кореляцію подій. Кореляцію подій забезпечує той факт, що міжмережні екрани є рубежем, що повинні перетнути всі мережні атаки, щоб увійти в мережу. Це ставить міжмережний екран в унікальне положення по аналізі неавторизованої діяльності. Із цієї причини всі міжмережні екрани й інші системи, що створюють логи, такі, як IDS, повинні виконувати синхронізацію часу. Найбільш загальним механізмом для синхронізації часу є network time protocol (NTP).

Створення резервних копій міжмережних екранів

Створення й супровід резервних копій є ключовою крапкою в будь-якій політиці адміністрування міжмережного екрана. Для всіх міжмережних екранів повинне виконуватися щоденне створення резервних копій.

У принципі, на всіх міжмережних екранах завжди повинні створюватися повні резервні копії. В інкрементальних резервних копіях немає необхідності.

Звичайно буває важко реалізувати централізовану схему керування резервними копіями, тому що надання доступу до централізованого сервера

резервних копій, що звичайно розташований за міжмережним екраном, буде представляти великий ризик з погляду таємності резервних копій. Отже, більшість міжмережних екранів повинне використовувати свої власні пристрої архівування.

Також бажано (але не завжди можливо) використовувати міжмережні екрани, у яких всі критичні файлові системи розташовані на CDROM.

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. У багатьох пристроях для домашніх мереж, наприклад інтегрованих маршрутизаторах, часто є багатофункціональні програмні міжмережні екрани. Такі міжмережні екрани звичайно реалізують трансляцію мережних адрес (NAT), динамічний аналіз пакетів (SPI), а також фільтрацію по IP-адресах, додатках і веб-сайтах. Додатково вони підтримують функції DMZ.

Інтегрований маршрутизатор дозволяє настроїти примітивну DMZ для доступу до внутрішнього сервера з вузлів за межами мережі. Для цього сервер повинен мати статичну IP-адресу, що вказується в конфігурації DMZ. Інтегрований маршрутизатор ізолює трафік, що пересилається на зазначений IP-адрес. Цей трафік пропускається тільки на той порт комутатора, до якого підключений сервер. На всі інші вузли як і раніше поширюється захист міжмережного екрану.

При активації DMZ у найпростішому виді зовнішні вузли одержують доступ до всіх портів сервера, наприклад 80 (HTTP), 21 (FTP) і 110 (POP3 для електронної пошти).

Функція переадресації портів дозволяє настроїти більш строгу конфігурацію DMZ. У цьому випадку вказуються порти, які повинні бути доступні на сервері. Пропускається тільки трафік, спрямований на ці порти. Весь інший трафік виключається.

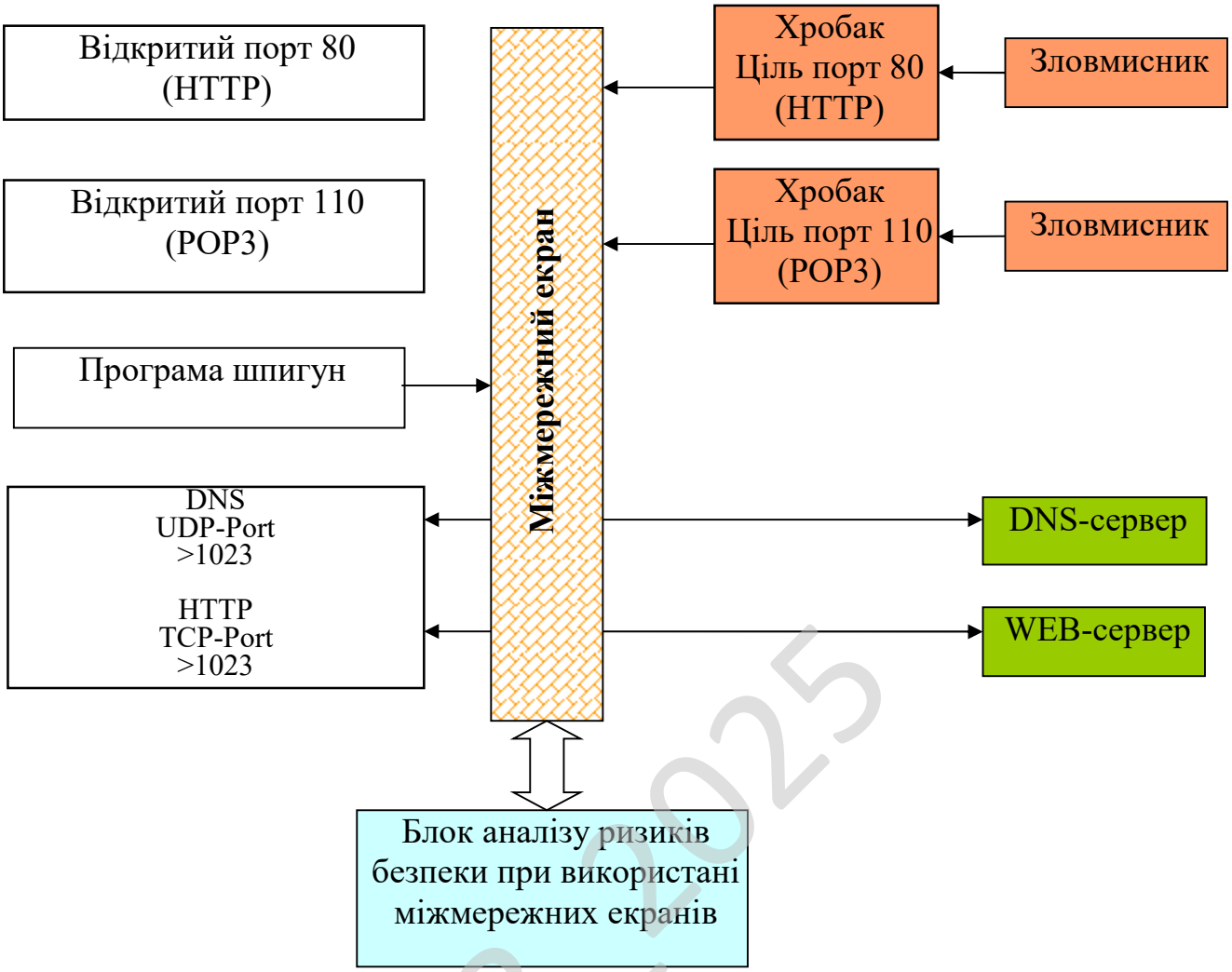

Рисунок 3.2 – Функціональна схема системи

Бездротова точка доступу в складі інтегрованого маршрутизатора часто вважається частиною внутрішньої мережі. Необхідно усвідомлювати, що при роботі точки доступу в незахищеному режимі всі користувачі, що підключилися до неї, одержують доступ до внутрішньої захищеної мережі без проходження міжмережного екрану. Зловмисники можуть у такий спосіб одержати доступ до внутрішньої мережі, минаючи всі засоби захисту.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі.

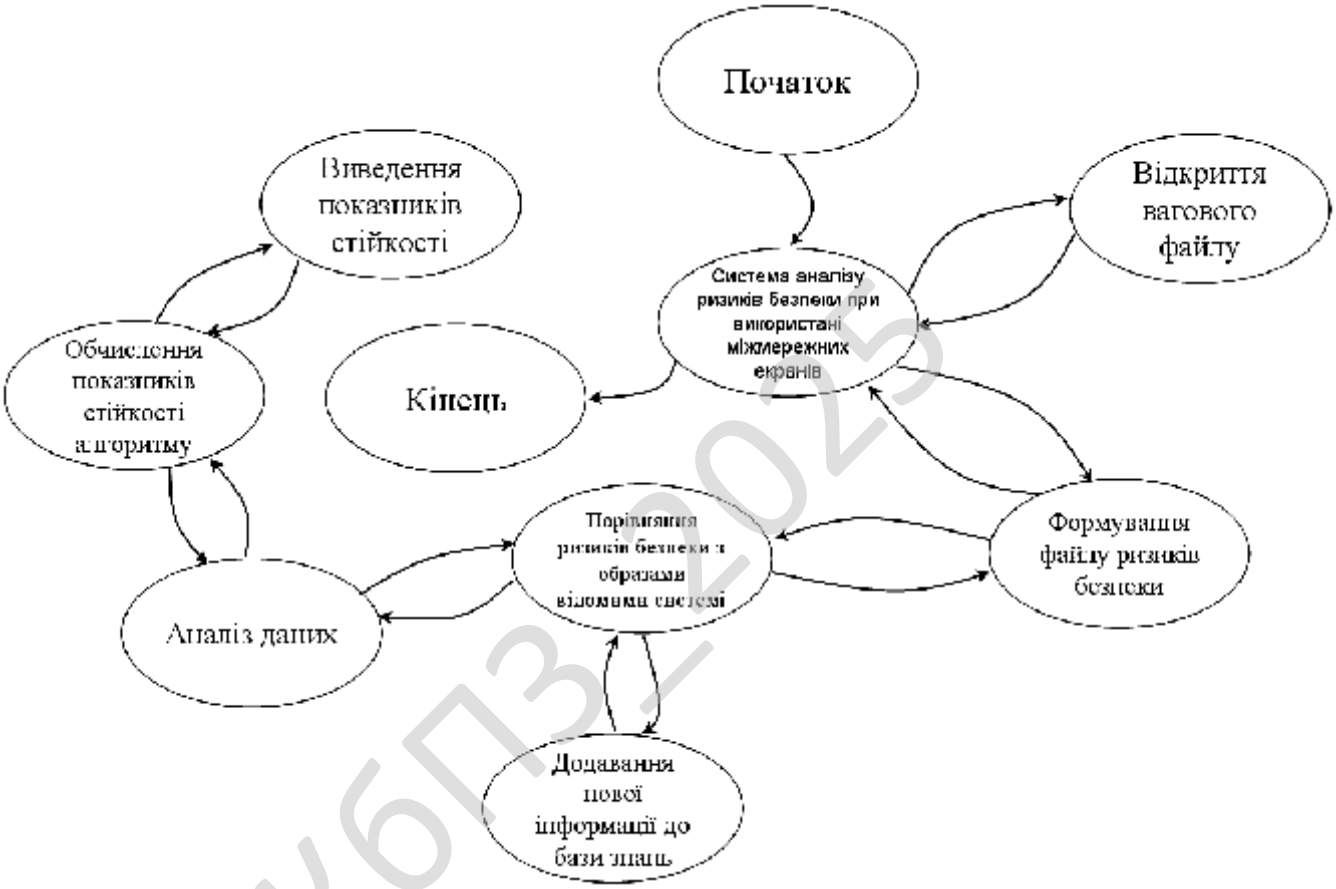

Рисунок 3.3 – Діаграма взаємодії процесів

Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи.

Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Під час роботи над бакалаврською дипломною роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю системи аналізу ризиків безпеки при використані міжмережних екранів.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем.

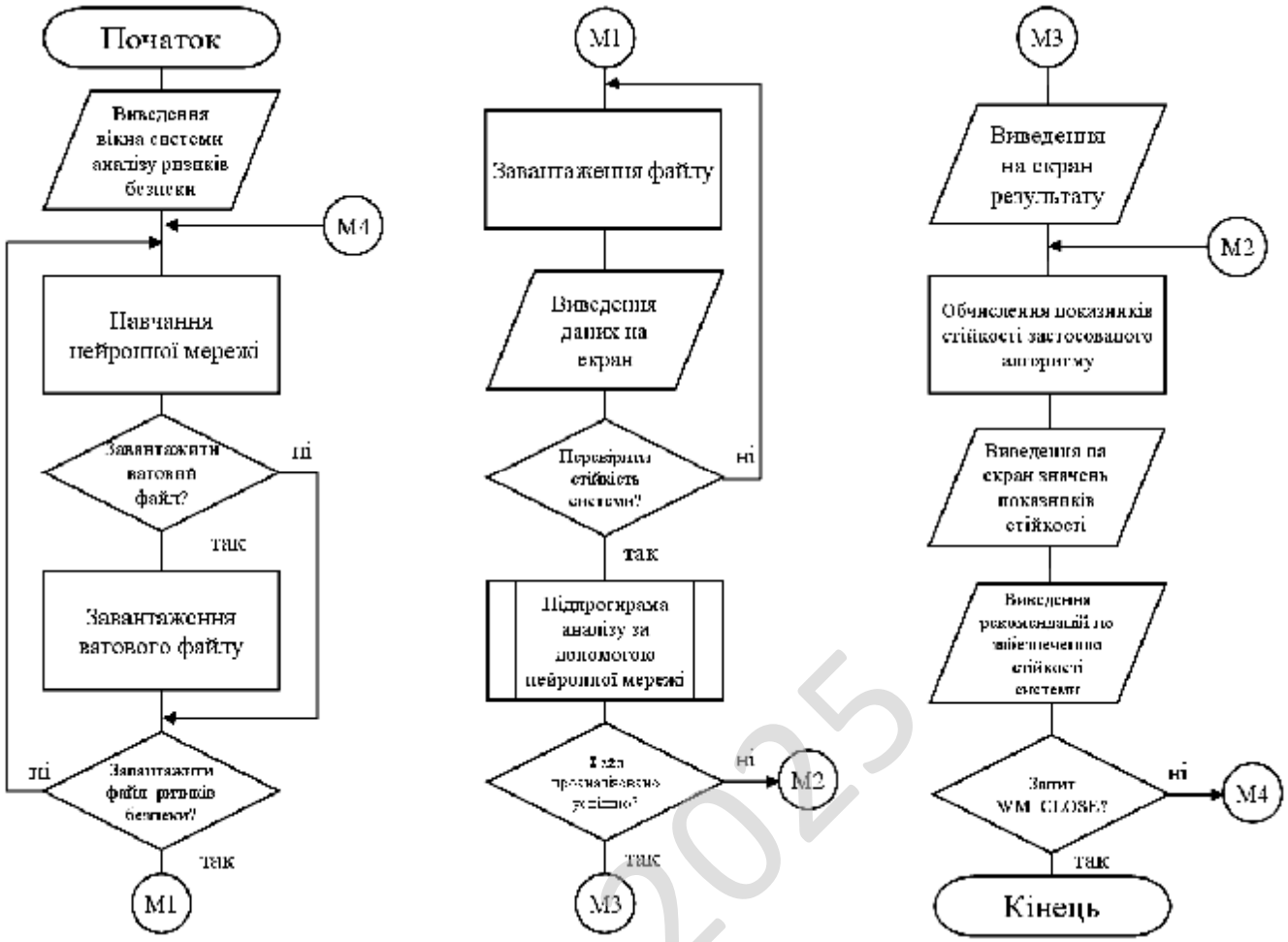

Рисунок 4.1 – Блок-схема основної програми

Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код. Основною причиною використання мови UML є спілкування розробників між собою.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки. Практично усі CASE-засоби (програми

автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

– Керівникам проектів, які керують розподілом завдань і контролем за проектом.

– Проектувальникам інформаційних систем які розробляють технічні завдання для програмістів.

– Бізнес-аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії.

– Програмістам які реалізовують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

Також при розробці бакалаврської дипломної роботи було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу); діаграма прецедентів (діаграми поведінки типу); Діаграма класів; Діаграма компонент; Діаграма об'єктів; Діаграма розгортання.

Діаграма діяльності. Це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій. Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів.

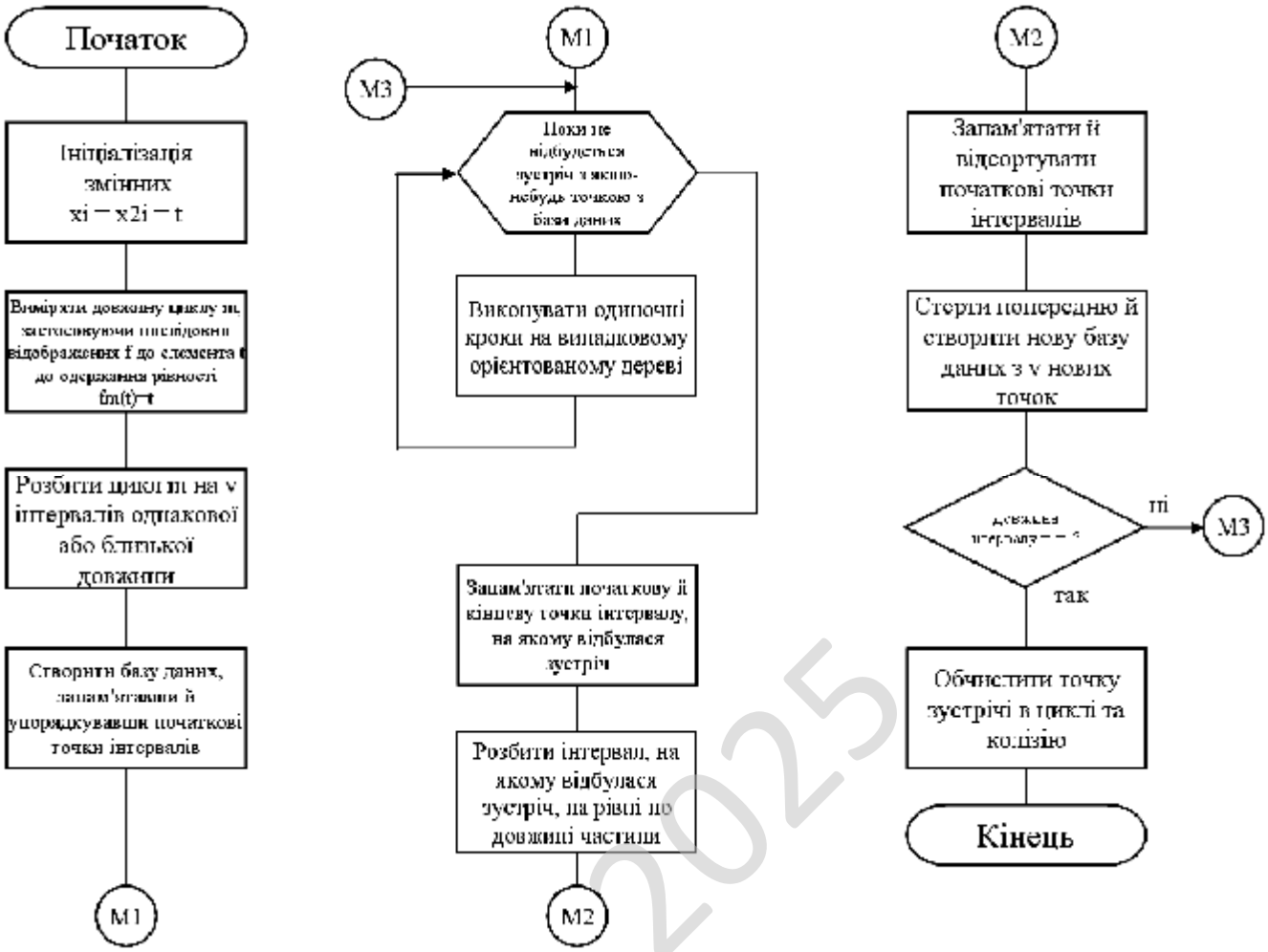

Рисунок 4.2 – Блок-схема роботи підпрограми

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Діаграма прецедентів це діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором.

При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

– асоціації (association relationship);

– включення (include relationship);

– розширення (extend relationship);

– узагальнення (generalization relationship).

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме – за допомогою відношень включення, розширення і узагальнення.

Відношення асоціації – одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Включення (include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його спеціальним випадком. При цьому відношенням залежності (dependency) є таке відношення між двома елементами моделі, при якому зміна одного елемента (незалежного) приводить до зміни іншого елемента (залежного).

Відношення розширення (extend) визначає взаємозв'язок базового варіанта використання з іншим варіантом використання, функціональна поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов.

Діаграма класів це статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення.

Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

В UML існують наступні типи зв'язків які використовуються у діаграмі класів: Асоціації; Агрегація; Композиція.

Асоціації це якщо між двома класами визначена асоціація, то можна переміщатися від об'єктів одного класу до об'єктів іншого. Цілком припустимі випадки, коли обидва кінці асоціації відносяться до одного і того ж класу. Це означає, що з об'єктом деякого класу дозволено зв'язати інші об'єкти з того ж класу. Асоціація, що зв'язує два класи, називається бінарної. Можна, хоча це рідко буває необхідним, створювати асоціації, що зв'язують відразу кілька класів. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

Асоціації може бути присвоєно ім'я, яке описує природу відносини. Зазвичай ім'я асоціації не вказується, якщо тільки ви не хочете явно задати для неї рольові імена або у вашій моделі настільки багато асоціацій, що виникає необхідність посилатися на них і відрізняти один від одного. Ім'я буде особливо корисним, якщо між одними і тими ж класами існує кілька різних асоціацій.

Клас, що бере участь в асоціації, грає в ній деяку роль. По суті, це "обличчя", яким клас, що знаходиться на одній стороні асоціації, звернений до класу з іншого її боку. Можна явно позначити роль, яку клас грає в асоціації.

Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано допомогою одного примірника асоціації. Це число називається кратністю (Multiplicity) ролі асоціації та записується або як вираз, значенням якого є діапазон значень, або в явному вигляді.

Вказуючи кратність на одному кінці асоціації, ви тим самим говорите, що на цьому кінці саме стільки об'єктів повинно відповідати кожному об'єкту на протилежному кінці. Кратність можна задати рівною одиниці (1), можна вказати діапазон: "нуль або одиниця" (0..1), "багато" (0 .. *), "одиниця або більше" (1 .. *). Дозволяється також вказувати певне число (наприклад, 3). За допомогою списку можна задати і більш складні кратності, наприклад 0. . 1, 3..4, 6 .. *, що означає "будь-яке число об'єктів, крім 2 і 5".

Агрегація це проста асоціація між двома класами відображає структурний відношення між рівноправними сутностями, коли обидва класу знаходяться на одному концептуальному рівні і ні один не є більш важливим, ніж інший. Але іноді доводиться моделювати відношення типу «частина/ціле», в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин).

Ставлення такого типу називають агрегацією; воно зараховане до відносин типу «має» (з урахуванням того, що об'єкт-ціле має кілька об'єктів-частин). Агрегація є окремим випадком асоціації і зображується у вигляді простої асоціації

з незафарбованим ромбом з боку «цілого». Графічно агрегація представляється порожнім ромбом на блоці класу, і лінією, яка від цього ромба до міститься класу.

Композиція це більш суворий варіант агрегації. Відома також як агрегація за значенням.

Композиція має жорстку залежність часу існування екземплярів класу контейнера та примірників містяться класів. Якщо контейнер буде знищений, то весь його вміст буде також знищено. Графічно представляється як і агрегація, але з зафарбовані ромбиком.

Діаграма компонент в UML це діаграма, на якій відображаються компоненти, залежності та зв'язки між ними.

Діаграма компонент відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись.

Модуль програмного забезпечення може бути представлено в якості компоненти. Деякі компоненти існують під час компіляції, деякі – під час компонування, а деякі під час роботи програми.

Діаграма компонент відображає лише структурні характеристики, для відображення окремих екземплярів компонент слід використовувати діаграму розгортування.

Компоненти об'єднуються разом використовуючи структурні зв'язки (assembly connector) щоб об'єднати інтерфейси двох компонент. Це ілюструє зв'язок типу «клієнт-сервер».

Структурна взаємодія – «зв'язок двох компонент, який передбачає, що один з них надає послуги, потрібні іншому компоненту».

При використанні діаграми компонент щоб показати внутрішню структуру компонента, клієнтські та серверні інтерфейси можуть утворювати пряме з'єднання з внутрішніми. Таке з'єднання називається з'єднанням делегації.

Діаграма об'єктів в UML це діаграма, що відображає об'єкти та їх зв'язки в певний момент часу. Діаграма об'єктів може розглядатись як окремий випадок

діаграми класів, на якій можуть бути представлені як класи, так і екземпляри (об'єкти) класів. Схожою за змістом є діаграма взаємодії (collaboration diagram).

Діаграми об'єктів не мають власної нотації. Оскільки діаграми класів можуть відображати об'єкти, то діаграма класів, на якій відображено лише об'єкти, та не відображено класи, може вважатись діаграмою об'єктів.

Діаграма об'єктів відображає об'єкти та зв'язки в певний момент роботи програми. Об'єкти можуть містити інформацію про власні значення а не про описання. Для відображення загальних шаблонів об'єктів та зв'язків, що можуть багаторазово створюватись під час роботи програми, слід використовувати діаграму взаємодії, яка може відображати характеристики об'єктів та зв'язків. Екземпляр діаграми взаємодії створює діаграму об'єктів.

Діаграма об'єктів не відображає еволюцію системи під час роботи. Натомість, слід використовувати діаграми взаємодії з повідомленнями, або діаграми послідовності.

Діаграма розгортання (deployment diagram) це діаграма в UML, на якій відображаються обчислювальні вузли під час роботи програми, компоненти, та об'єкти, що виконуються на цих вузлах. Компоненти відповідають представленню робочих екземплярів одиниць коду. Компоненти, що не мають представлення під час роботи програми на таких діаграмах не відображаються; натомість, їх можна відобразити на діаграмах компонент. Діаграма розгортання відображає робочі екземпляри компонент, а діаграма компонент, натомість, відображає зв'язки між типами компонент.

Розглянемо підпрограму запуску файрволу. Після того як створені чи відкриті правила фільтрації можна запустити файрвол.

Після запуску файрволу, програма починає фільтрувати пакети по вказаним правилам. Розглянемо код.

```
import json
import logging

# Налаштування логування для відстеження подій системи
logging.basicConfig(filename='firewall_risk_analysis.log',
```

```
level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
        class Firewall:
            # Клас для моделювання міжмережевого екрану
            def __init__(self, rules):
                # Ініціалізація об'єкта міжмережевого екрану з набором правил
                self.rules = rules
            def check_packet(self, packet):
                # Перевіряє пакет на відповідність правилам міжмережевого екрану
                for rule in self.rules:
        if all(packet.get(key) == value for key, value in rule.items()):
                        logging.info(f"Packet {packet} allowed by rule {rule}")
                        return "Allowed"
                logging.warning(f"Packet {packet} blocked by firewall")
                return "Blocked"

        class RiskAnalyzer:
            # Клас для аналізу ризиків безпеки
            def __init__(self, firewall):
                # Ініціалізує аналізатор ризиків з об'єктом міжмережевого екрану
                self.firewall = firewall

            def analyze_traffic(self, traffic_data):
                # Аналізує трафік та визначає ризики
                risk_report = {"total_packets": 0, "allowed": 0, "blocked": 0}
                for packet in traffic_data:
                    risk_report["total_packets"] += 1
                    result = self.firewall.check_packet(packet)
                    if result == "Allowed":
                        risk_report["allowed"] += 1
                    else:
                        risk_report["blocked"] += 1
                return risk_report

            def assess_risk_level(self, risk_report):
                # Оцінює рівень ризику на основі звіту про трафік
                blocked_ratio = risk_report["blocked"] /
                                risk_report["total_packets"]
                if blocked_ratio > 0.5:
                    return "High Risk"
                elif blocked_ratio > 0.2:
                    return "Medium Risk"
                return "Low Risk"
```

```
def load_firewall_rules(filename):
    # Завантажує правила міжмережевого екрану з файлу
    try:
        with open(filename, 'r') as file:
            rules = json.load(file)
            return rules
    except FileNotFoundError:
        logging.error("Firewall rules file not found")
        return []

def load_traffic_data(filename):
    # Завантажує дані трафіку з файлу
    try:
        with open(filename, 'r') as file:
            traffic_data = json.load(file)
            return traffic_data
    except FileNotFoundError:
        logging.error("Traffic data file not found")
        return []

def main():
    # Основна функція для запуску аналізу ризиків безпеки
    rules = load_firewall_rules("firewall_rules.json")
    traffic_data = load_traffic_data("traffic_data.json")

    firewall = Firewall(rules)
    risk_analyzer = RiskAnalyzer(firewall)

    risk_report = risk_analyzer.analyze_traffic(traffic_data)
    risk_level = risk_analyzer.assess_risk_level(risk_report)
    print("Risk Analysis Report:")
    print(json.dumps(risk_report, indent=4))
    print(f"Overall Risk Level: {risk_level}")

    logging.info("Risk analysis completed successfully")
if __name__ == "__main__":
    main()
```

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм Blowfish, який є симетричним алгоритмом шифрування, тобто таким, у якому ключ шифрування дорівнює ключу дешифрування. Він є мережею Фейштеля, у якій кількість ітерацій дорівнює 16. Довжина блоку дорівнює 64 бітам, ключ може мати будь-яку довжину в межах 448 біт. Хоча перед початком будь-якого шифрування виконується складна фаза ініціалізації, саме шифрування даних виконується досить швидко.

Алгоритм призначений в основному для додатків, у яких ключ міняється нечасто, до того ж існує фаза початкового рукостискання, під час якої відбувається автентифікація сторін і узгодження загальних параметрів і секретів. При реалізації на 32-бітних мікропроцесорах з більшим кешем даних Blowfish значно швидше DES. Алгоритм складається із двох частин: розширення ключа й шифрування даних. Розширення ключа перетворює ключ довжиною, принаймні, 448 біт у кілька масивів підключів загальною довжиною 4168 байт.

В основі алгоритму лежить мережа Фейштеля з 16 ітераціями. Кожна ітерація складається з перестановки, що залежить від ключа, і підстановки, що залежить від ключа й даних. Операціями є XOR і додавання 32-бітних слів.

Blowfish використовує велику кількість підключів. Ці ключі повинні бути обчислені заздалегідь, до початку будь-якого шифрування або дешифрування даних. Елементи алгоритму:

1. P – масив, що складається з вісімнадцяти 32-бітних підключів:

$$P_1, P_2, ..., P_{18}.$$

2. Чотири 32-бітних S-boxes с 256 входами кожний. Перший індекс означає номер S-box, другий індекс – номер входу.

$$S_{1,0}, S_{1,1}, \ldots S_{1,255};$$

$$S_{2,0}, S_{2,1}, \ldots S_{2,255};$$

$$S_{3,0}, S_{3,1}, \ldots S_{3,255};$$

$$S_{4,0}, S_{4,1}, \ldots S_{4,255};$$

Шифрування

Входом є 64-бітний елемент даних X, що ділиться на дві 32-бітні половини, X_l і X_r.

$$X_l = X_l \text{ XOR } P_i$$

$$X_r = F(X_l) \text{ XOR } X_r$$

$$\text{Swap } X_l \text{ and } X_r$$

Функція F

Розділити X_l на чотири 8-бітних елементи A, B, C, D.

$$F(X_l) = ((S_{1,A} + S_{2,B} \bmod 2^{32}) \text{ XOR } S_{3,C}) + S_{4,D} \bmod 2^{32}$$

Дешифрування відрізняється від шифрування тим, що P_i використовуються у зворотному порядку.

Генерація підключів

Підключи обчислюються з використанням самого алгоритму Blowfish.

1. Ініціалізувати перший P-масив і чотири S-boxes фіксовані рядки.

2. Виконати операцію XOR P_1 з першими 32 бітами ключа, операцію XOR P_2 із другими 32 бітами ключа й т.д. Повторювати цикл доти, поки весь P-масив не буде побітово складний з усіма бітами ключа. Для коротких ключів виконується конкатенація ключа із самим собою.

3. Зашифрувати нульовий рядок алгоритмом Blowfish, використовуючи підключи, описані в пунктах (1) і (2).

4. Замінити P_1 і P_2 виходом, отриманим на кроці (3).

5. Зашифрувати вихід кроку (3), використовуючи алгоритм Blowfish з модифікованими підключами.

6. Замінити P_3 і P_4 виходом, отриманим на кроці (5).

7. Продовжити процес, заміняючи всі елементи P-масиву, а потім всі чотири S-boxes, виходами відповідним чином модифікованого алгоритму Blowfish.

Для створення всіх підключів потрібна 521 ітерація.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс програмного забезпечення, розробленого у результаті виконання бакалаврської дипломної роботи.

Розроблене програмне забезпечення системи аналізу ризиків безпеки при використані міжмережних екранів складається з наступних функціональних блоків:

– Блоку правил обробки пакетів даних.

– Вікна додавання правил обробки пакетів даних.

– Вікно виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ.

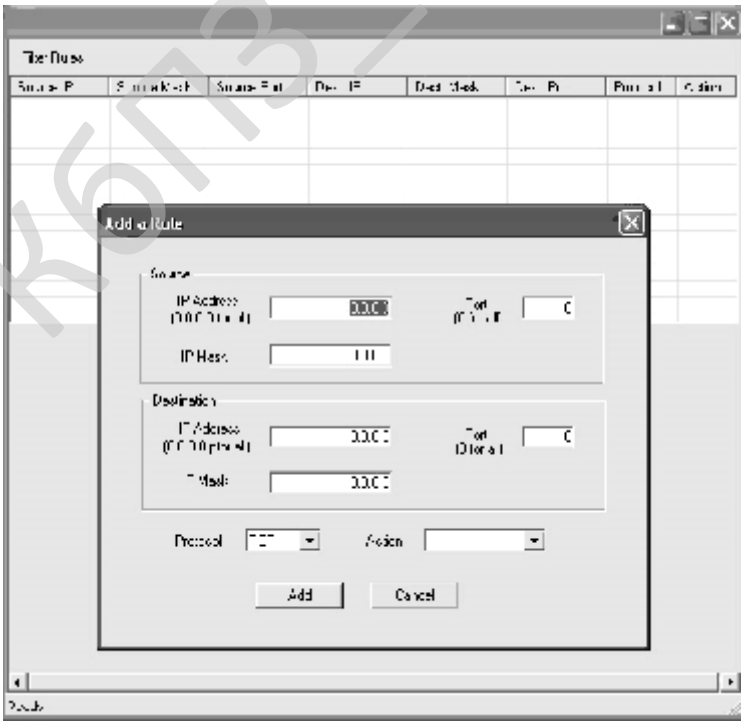

Рисунок 5.1 – Головне вікно розробленого ПЗ

Для перегляду короткої довідки про програму слід натиснути на основному вікні кнопку авторського права, після чого на екрані з’явиться вікно показане на рисунку 5.2.

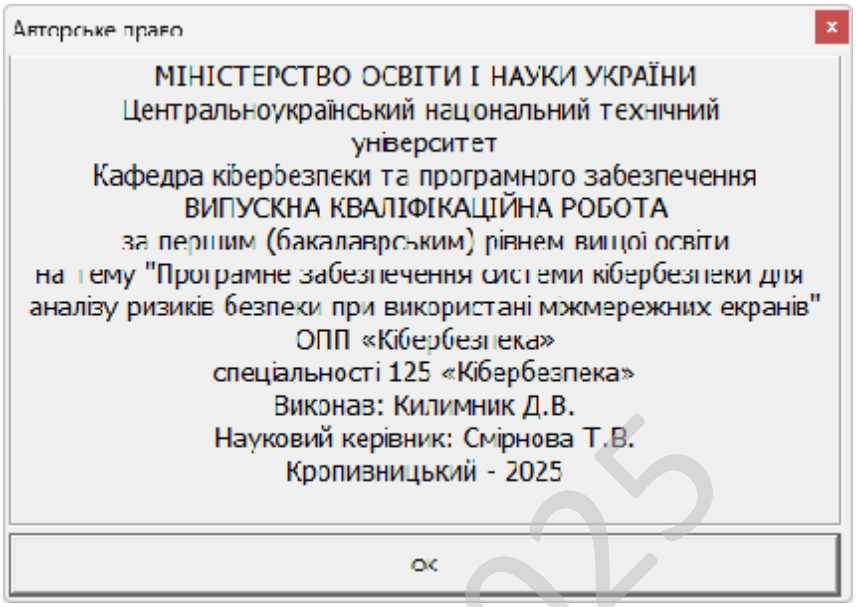

Рисунок 5.2 – Вікно розробника ПЗ

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування форматом білої засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

– Виконуються всі цикли (у межах їхніх кордонів та діапазонів).

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

– Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.

– У програмі можуть бути пропущені деякі маршрути.

– Не можна виявити помилки, поява яких залежить від даних.

Переваги тестування "білої скриньки" пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

– Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

– Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

– При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

– Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Обрано умови розповсюдження – Shareware.

Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання.

Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості.

Якщо після закінчення цього терміну користувач вирішить продовжити використання ПЗ, він зобов'язаний купити його (зареєструватися), заплативши авторові певну суму.

В іншому випадку користувач повинен припинити використання ПЗ та видалити його зі свого комп'ютера.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем для аналізу ризиків безпеки при використані міжмережних екранів.

– Досліджена система для аналізу ризиків безпеки при використані міжмережних екранів.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання для аналізу ризиків безпеки при використані міжмережних екранів.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних

екранів. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи кібербезпеки й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Blowfish.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alyssa Miller. Cybersecurity Career Guide. Manning Publications. 2022. 368 p.

2. Awais Rashid, Howard Chivers, George Danezis, Emil Lupu, Andrew Martin. CyBOK The Cyber Security Body of Knowledge. The National Cyber Security Centre. 2019. 854 p.

3. Loren Kohnfelder. Designing Secure Software. No Starch Press. 2022. 332 p.

4. Samir Kumar Rakshit. Ethical Hacker's Penetration Testing Guide. BPB Online. 2022. 509 p.

5. Corey J. Ball. Hacking APIs. No Starch Press. 2022. 353 p.

6. Kevin Beaver. Hacking for Dummies. John Wiley & Sons. 2022. 419 p.

7. Mark S. Merkow. Practical Security for Agile and DevOps. CRC Press. 2022. 236 p

8. Derek Fisher. Application Security Program Handbook. Manning Publications. 2021. 155 p.

9. Cameron Wyatt PH.D. Kali Linux Tutorial. Independently published. 2021. 60 p.

10. Alex Matrosov, Eugene Rodionov, Sergey Bratus. Rootkits and Bootkits. No Starch Press. 2019. 450 p.

11. Lakhno, V., Malyukov, V., Smirnov, O., Bebeshko, B., Chubaievskiy, V., Zhumadilova, M., Malyukova, I., Smirnov, S. «Multifactorial Model for Targeted Attacks Counteracting Within the Framework of a Multi-Step Quality Game with Fuzzy Information». *8th International Symposium on Intelligent Informatics, ISI 2023*, 2025. vol 389. pp 377-389. Springer, Singapore.

12. Kuznetsov, O., Frontoni, E., Kryvinska, N., Chevardin, V., Smirnov, O. «Wireless Network Encryption Stream Ciphers, Computational Modeling, and Security

Analysis». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 379–402.

13. Kuznetsov, O., Frontoni, E., Kryvinska, N., Smirnov, O., Imoize, G.L. «Computational Modeling of Enhanced Spread Spectrum Codes for Asynchronous Wireless Communication». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 403–447.

14. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

15. Akhalaia, G., Iavich, M., Iashvili, G., Prysiazhnyy, D., Smirnova, T. «Secure Encrypted Connection on Georgian Website». *CEUR Workshop Proceedings,* 2023, 3550, pp. 313-320.

16. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56

17. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

18. Kuznetsov, O., Kandiy, S., Frontoni, E., Smirnov, O. «Trade-offs in Post-Quantum Cryptography: A Comparative Assessment of BIKE, HQC, and Classic McEliece». *CEUR Workshop Proceedings,* Volume 3504, 2023, pp. 1-11.

19. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» *CEUR Workshop Proceedings,* Volume 3187, 2022,

20. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». *In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile*

Networks. Lecture Notes on Data Engineering and Communications Technologies, vol 131. 2023. Springer, Singapore. pp. 21-34.

21. Smirnov O.A., Al-Oraiqat A.M., Ulichev O.S., Meleshko Ye.V., Al-Rawashdeh H.S., Polishchuk L.I. «Modeling strategies for information influence dissemination in social networks». *Journal of Ambient Intelligence and Humanized Computing* Volume 13, Issue 5. Springer, Cham. 2022, pp. 2463-2477.

22. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». *11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021,* Cracow, Poland, 22-25 September 2021. P. 414-418

23. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». *4 IEEE International Conference on Advanced Information and Communication Technologies (AICT) - 2021*, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

24. Smirnov O., Kuznetsov A., Girzheva O., Kiian A., Nakisko O., Kuznetsova T. «Advanced Code-Based Electronic Digital Signature Scheme». *2020 IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020,* Kharkiv, 6 October 2020-9 October 2020, P. 358-362.

25. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova K. «Data hiding scheme based on spread sequence addressing». *CEUR Workshop Proceedings* Volume 2805, 2020, Pages *44-58.*

26. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». *International Journal of Computing*; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

27. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings*. Volume 2740, 2020, Pages 102-114.

28. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

29. Smirnov O., Kuznetsov A., Arischenko A., Chepurko I., Onikiychuk A., Kuznetsova T. «Pseudorandom sequences for spread spectrum image steganography». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 122-131.

30. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 1-14.

31. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions». *Lecture Notes in Networks and Systems*, vol 152. Springer, Cham. 2021, pp 66-84.

32. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

33. Smirnov O., Kuznetsov A., Kiian A., Babenko V., Perevozova I.,Chepurko I. «New Approach to the Implementation of Post-Quantum Digital Signature Scheme». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 166-171.

34. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

35. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In:

Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

36. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

37. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». *International Journal of Computer Network and Information Security (IJCNIS)*. Vol. 12, No. 3, 2020. PP.33-43.

38. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 646-660.

39. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

40. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

41. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, Vol 2588, P. 215-227, 2019.

42. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019.

43. Smirnov, O., Kuznetsov, A., Kiian, A., Gorbenko, Y., Cherep, O., Bexhter L. «Code-based Pseudorandom Generator for the Post-Quantum Period», *2019 IEEE International Conference on Advanced Trends in Information Theory (IEEE ATIT 2019).* 18.12.19-20.12.19 Kyiv Ukraine. P. 204 – 209.

44. Smirnov, O., Kuznetsov, A., Nariezhnii, O., Stelnyk, S., Kokhanovska, T., Kuznetsova T., «Side Channel Attack on a Quantum Random Number Generator», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18 - 21 September 2019. P.713-718.

45. Kuznetsova, T., «Code-Based Schemes for Post-Quantum Digital Signatures», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019;* Metz; France; 18-21 September 2019. P. 707-712.

46. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18-21 September 2019. P.701-706.

47. Smirnov, O., Hu, Z., Vasiliu, Y., Sydorenko, V., Polishchuk, Y., «Abstract Model of Eavesdropper and Overview on Attacks in Quantum Cryptography Systems», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18-21 September 2019. P.399-405.

48. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019*/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

49. Smirnov, O., Kuznetsov, A., Kiian, A., Babenko, B., Zhosan, H., Prokopovych-Tkachenko, D., «Soft Decoding Method for Turbo-Productive Codes»,

2019 3rd International Conference on Advanced Information and Communications Technologies, AICT 2019, Lviv, Ukraine, 2-6 July, 2019, P. 129-134.

50. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

51. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

52. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 51-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи кібербезпеки з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 87 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 4.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Смірнова Т.В.

Програмне забезпечення системи кібербезпеки для аналізу ризиків безпеки при використані міжмережних екранів

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 20

Літера: РП

Кропивницький – 2025 року

Основна програма

```
#!/usr/bin/env python3
#Ініціалізація модуля random для генерації випадкових чисел
import random
#Імпорт модуля time для роботи з часом
import time
#Імпорт модуля datetime для роботи з датою та часом
import datetime
#Імпорт модуля threading для роботи з потоками
import threading
#Імпорт модуля queue для обробки запитів
import queue
#Ініціалізація глобального конфігураційного списку IP міжмережних екранів
FIREWALL_IPS = ["192.168.1.1", "192.168.1.2", "10.0.0.1"]
#Встановлення порогу сповіщення для ризику
ALERT_THRESHOLD = 5
#Визначення класу для симуляції міжмережного екрана
class Firewall:
#Конструктор класу для ініціалізації IP та логів
    def __init__(self, ip):
#Призначення IP адреси міжмережного екрана
        self.ip = ip
#Ініціалізація списку логів
        self.log_entries = []
#Створення об’єкту блокування для потокобезпечної роботи
        self.lock = threading.Lock()
#Функція для генерації запису логу подій
    def generate_log_entry(self):
#Отримання поточного часу
        now = datetime.datetime.now()
#Випадковий вибір типу події з-поміж ALLOW, DENY, DROP
        event_type = random.choice(["ALLOW", "DENY", "DROP"])
#Генерація випадкового IP-адреси джерела
        source_ip = ".".join(str(random.randint(1, 255)) for _ in range(4))
#Призначення IP адреси міжмережного екрана як адреси призначення
        destination_ip = self.ip
#Випадковий вибір порту для події
        port = random.choice([80, 443, 22, 8080, 21])
#Випадкове визначення рівня серйозності події від 1 до 10
        severity = random.randint(1, 10)
#Створення словника з деталями події
        log_entry = {
            "timestamp": now,
            "event_type": event_type,
            "source_ip": source_ip,
            "destination_ip": destination_ip,
            "port": port,
            "severity": severity
        }
#Повернення згенерованого запису
        return log_entry
#Функція для додавання запису до логів
    def add_log_entry(self, entry):
#Забезпечення потокобезпечного доступу до списку логів
        with self.lock:
#Додавання запису до списку логів
            self.log_entries.append(entry)
#Функція для отримання копії логів
    def get_logs(self):
#Забезпечення потокобезпечного доступу до списку логів
        with self.lock:
#Створення копії логів для безпечного використання
            logs_copy = self.log_entries.copy()
#Повернення копії логів
        return logs_copy
#Функція для очищення списку логів
    def clear_logs(self):
#Забезпечення потокобезпечного доступу до списку логів
```

```
        with self.lock:
#Очищення списку логів
            self.log_entries = []
#Кінець класу Firewall

#Визначення класу для аналізу ризиків
class RiskAnalyzer:
#Конструктор класу для ініціалізації списку міжмережних екранів
    def __init__(self, firewall_instances):
#Збереження списку об’єктів міжмережних екранів
        self.firewall_instances = firewall_instances
#Функція для аналізу логів з метою визначення ризику
    def analyze_logs(self):
#Ініціалізація змінної для підрахунку ризикового балу
        risk_score = 0
#Ініціалізація списку для детального звіту
        detailed_report = []
#Прохід по всіх міжмережних екранах
        for fw in self.firewall_instances:
#Отримання логів поточного екрана
            logs = fw.get_logs()
#Перебір усіх записів логу
            for log in logs:
#Якщо тип події DENY, підвищення ризику з коефіцієнтом 1.5
                if log["event_type"] == "DENY":
#Обчислення приросту ризику
                    increase = log["severity"] * 1.5
#Додавання приросту до загального ризикового балу
                    risk_score += increase
#Додавання запису в детальний звіт
                    detailed_report.append("Risk increased by " + str(increase)
+ " due to DENY event from " + log["source_ip"])
#Якщо тип події DROP, підвищення ризику з коефіцієнтом 2.0
                elif log["event_type"] == "DROP":
#Обчислення приросту ризику
                    increase = log["severity"] * 2.0
#Додавання приросту до загального ризикового балу
                    risk_score += increase
#Додавання запису в детальний звіт
                    detailed_report.append("Risk increased by " + str(increase)
+ " due to DROP event from " + log["source_ip"])
#Якщо тип події ALLOW, підвищення ризику з коефіцієнтом 0.5
                else:
#Обчислення приросту ризику
                    increase = log["severity"] * 0.5
#Додавання приросту до загального ризикового балу
                    risk_score += increase
#Додавання запису в детальний звіт
                    detailed_report.append("Risk increased by " + str(increase)
+ " due to ALLOW event from " + log["source_ip"])
#Повернення загального ризикового балу та детального звіту
        return risk_score, detailed_report
#Функція для виконання пошукових запитів по логах
    def query_firewall(self, query_str):
#Ініціалізація списку результатів пошуку
        results = []
#Прохід по всіх міжмережних екранах
        for fw in self.firewall_instances:
#Отримання логів поточного екрана
            logs = fw.get_logs()
#Перебір усіх записів логу
            for log in logs:
#Перевірка наявності рядка запиту у типі події або в IP джерела
                if query_str in log["event_type"] or query_str in
log["source_ip"]:
#Додавання запису до результатів
                    results.append(log)
#Повернення списку результатів
        return results
```

```
#Кінець класу RiskAnalyzer

#Визначення класу для обробки запитів користувача
class QueryHandler:
#Конструктор класу для ініціалізації об’єкта аналізу ризиків
    def __init__(self, risk_analyzer):
#Призначення об’єкта аналізу ризиків до локальної змінної
        self.risk_analyzer = risk_analyzer
#Функція для обробки запиту користувача
    def handle_query(self, query):
#Отримання результатів запиту через аналіз логів
        result_logs = self.risk_analyzer.query_firewall(query)
#Повернення отриманих результатів
        return result_logs
#Кінець класу QueryHandler

#Визначення класу для детального логування системних подій
class DetailedLogger:
#Конструктор класу для ініціалізації списку записів логування
    def __init__(self):
#Ініціалізація списку логів
        self.logs = []
#Функція для додавання нового запису в журнал
    def log(self, message):
#Отримання поточного часу для запису
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
#Формування рядка з повідомленням
        log_message = "[" + timestamp + "] " + message
#Додавання запису в список логів
        self.logs.append(log_message)
#Вивід повідомлення в консоль
        print(log_message)
#Функція для отримання всіх записів журналу
    def get_all_logs(self):
#Повернення копії списку логів
        return self.logs.copy()
#Кінець класу DetailedLogger

#Функція для симуляції генерації логів міжмережного екрана в окремому потоці
def simulate_firewall_logging(firewall_instance, stop_event, logger):
#Цикл, що виконується до отримання сигналу зупинки
    while not stop_event.is_set():
#Генерація запису логу
        log_entry = firewall_instance.generate_log_entry()
#Додавання запису до логів міжмережного екрана
        firewall_instance.add_log_entry(log_entry)
#Логування події генерації запису
        logger.log("Generated log entry at firewall " + firewall_instance.ip + "
with event " + log_entry["event_type"])
#Очікування випадкового часу перед наступною генерацією
        time.sleep(random.uniform(0.1, 0.5))
#Кінець функції simulate_firewall_logging

#Функція для періодичного виконання аналізу ризиків
def periodic_risk_analysis(risk_analyzer, interval, stop_event, logger):
#Цикл, що виконується до отримання сигналу зупинки
    while not stop_event.is_set():
#Виконання аналізу логів
        risk_score, report = risk_analyzer.analyze_logs()
#Логування поточного ризикового балу
        logger.log("Current Risk Score: " + str(round(risk_score, 2)))
#Логування деталей звіту аналізу ризиків
        for line in report:
            logger.log("Detail: " + line)
#Перерва на заданий інтервал часу
        time.sleep(interval)
#Кінець функції periodic_risk_analysis

#Функція для обробки запитів користувача в окремому потоці
```

```
def query_processing_thread(query_handler, query_queue, stop_event, logger):
#Цикл обробки запитів до отримання сигналу зупинки
    while not stop_event.is_set():
#Спроба отримати запит з черги з таймаутом
        try:
            query = query_queue.get(timeout=1)
#Обробка отриманого запиту
            results = query_handler.handle_query(query)
#Логування результатів запиту
            logger.log("Query '" + query + "' returned " + str(len(results)) + "
results.")
#Перебір та логування кожного результату
            for res in results:
                logger.log("Result: " + str(res))
#Позначення завдання як виконаного
            query_queue.task_done()
#Обробка виключення у випадку порожньої черги
        except queue.Empty:
            continue
#Кінець функції query_processing_thread

#Функція для генерації додаткових симуляційних запитів користувача
def simulate_user_queries(query_queue, stop_event, logger):
#Цикл симуляції запитів до отримання сигналу зупинки
    simulated_queries = ["DENY", "DROP", "ALLOW", "192", "10"]
#Виконання кількох запитів
    for _ in range(10):
#Перевірка на сигнал зупинки
        if stop_event.is_set():
            break
#Випадковий вибір запиту з переліку
        sample_query = random.choice(simulated_queries)
#Додавання запиту до черги
        query_queue.put(sample_query)
#Логування відправленого запиту
        logger.log("Simulated user query added: " + sample_query)
#Перерва між запитами
        time.sleep(random.uniform(3, 7))
#Кінець функції simulate_user_queries

#Функція для періодичної очистки логів міжмережних екранів для уникнення
переповнення пам’яті
def periodic_log_cleanup(firewall_instances, interval, stop_event, logger):
#Цикл виконання очищення логів до отримання сигналу зупинки
    while not stop_event.is_set():
#Логування початку процедури очищення логів
        logger.log("Starting periodic log cleanup...")
#Перебір усіх міжмережних екранів
        for fw in firewall_instances:
#Очищення логів поточного міжмережного екрана
            fw.clear_logs()
#Логування очищення логів для поточного міжмережного екрана
            logger.log("Cleared logs for firewall " + fw.ip)
#Перерва на заданий інтервал часу
        time.sleep(interval)
#Кінець функції periodic_log_cleanup

#Головна функція для ініціалізації та запуску системи
def main():
#Створення об’єкту детального логгера
    logger = DetailedLogger()
#Логування старту головної функції
    logger.log("Starting Cybersecurity Risk Analysis System...")
#Створення списку об’єктів міжмережних екранів
    firewalls = []
#Перебір IP адрес для створення об’єктів міжмережних екранів
    for ip in FIREWALL_IPS:
#Створення об’єкту міжмережного екрана для поточної IP адреси
        fw = Firewall(ip)
```

```
#Додавання об’єкту до списку міжмережних екранів
        firewalls.append(fw)
#Логування створення міжмережних екранів
        logger.log("Initialized Firewall with IP: " + ip)
#Кінець створення об’єктів міжмережних екранів

#Створення об’єкту аналізатора ризиків
    analyzer = RiskAnalyzer(firewalls)
#Логування створення аналізатора ризиків
    logger.log("RiskAnalyzer initialized with " + str(len(firewalls)) + "
firewalls.")
#Створення об’єкту для обробки запитів користувача
    query_handler = QueryHandler(analyzer)
#Логування створення обробника запитів
    logger.log("QueryHandler initialized.")
#Створення об’єкту події зупинки для керування потоками
    stop_event = threading.Event()
#Логування створення події зупинки
    logger.log("Stop event created for thread management.")

#Створення списку потоків для симуляції генерації логів
    fw_threads = []
#Перебір кожного міжмережного екрана для запуску окремого потоку генерації логів
    for fw in firewalls:
#Створення потоку для симуляції логування для поточного міжмережного екрана
        t = threading.Thread(target=simulate_firewall_logging, args=(fw,
stop_event, logger))
#Позначення потоку як демон
        t.daemon = True
#Запуск потоку симуляції логування
        t.start()
#Додавання потоку до списку потоків
        fw_threads.append(t)
#Логування запуску потоків генерації логів
        logger.log("Started logging thread for firewall " + fw.ip)
#Кінець запуску потоків для міжмережних екранів

#Створення та запуск потоку для періодичного аналізу ризиків
    analysis_thread = threading.Thread(target=periodic_risk_analysis,
args=(analyzer, 10, stop_event, logger))
#Позначення потоку як демон
    analysis_thread.daemon = True
#Запуск потоку аналізу ризиків
    analysis_thread.start()
#Логування запуску потоку аналізу ризиків
    logger.log("Risk analysis thread started.")
#Кінець створення потоку аналізу ризиків

#Створення черги для обробки запитів користувача
    query_queue = queue.Queue()
#Логування створення черги запитів
    logger.log("Query queue initialized.")
#Створення та запуск потоку для обробки запитів користувача
    query_thread = threading.Thread(target=query_processing_thread,
args=(query_handler, query_queue, stop_event, logger))
#Позначення потоку як демон
    query_thread.daemon = True
#Запуск потоку обробки запитів
    query_thread.start()
#Логування запуску потоку обробки запитів
    logger.log("Query processing thread started.")
#Кінець створення потоку для запитів користувача

#Створення та запуск потоку для періодичного очищення логів
    cleanup_thread = threading.Thread(target=periodic_log_cleanup,
args=(firewalls, 30, stop_event, logger))
#Позначення потоку як демон
    cleanup_thread.daemon = True
#Запуск потоку очищення логів
```

```
    cleanup_thread.start()
#Логування запуску потоку очищення логів
    logger.log("Log cleanup thread started.")
#Кінець створення потоку очищення логів

#Запуск симуляції запитів користувача в окремому потоці
    user_query_thread = threading.Thread(target=simulate_user_queries,
args=(query_queue, stop_event, logger))
#Позначення потоку як демон
    user_query_thread.daemon = True
#Запуск потоку симуляції запитів користувача
    user_query_thread.start()
#Логування запуску симуляції запитів користувача
    logger.log("User query simulation thread started.")
#Кінець створення потоку симуляції запитів

#Головний цикл роботи системи, який очікує сигналу зупинки
    try:
#Цикл, що триває протягом роботи системи
        while True:
#Періодичне очікування з можливістю переривання
            time.sleep(1)
#Обробка переривання користувача (Ctrl+C)
    except KeyboardInterrupt:
#Логування отримання сигналу зупинки
        logger.log("KeyboardInterrupt received. Stopping simulation...")
#Кінець блоку обробки переривання
    finally:
#Встановлення сигналу зупинки для всіх потоків
        stop_event.set()
#Логування встановлення сигналу зупинки
        logger.log("Stop event set. Waiting for threads to finish...")
#Очікування завершення потоків симуляції логів
        for t in fw_threads:
            t.join(timeout=1)
#Очікування завершення потоку аналізу ризиків
        analysis_thread.join(timeout=1)
#Очікування завершення потоку обробки запитів
        query_thread.join(timeout=1)
#Очікування завершення потоку очищення логів
        cleanup_thread.join(timeout=1)
#Очікування завершення потоку симуляції запитів користувача
        user_query_thread.join(timeout=1)
#Логування завершення всіх потоків
        logger.log("All threads have been terminated. Exiting system.")
#Кінець блоку finally
#Кінець головної функції main

#Вхідна точка програми
if __name__ == '__main__':
#Виклик головної функції для запуску системи
    main()
```

Файл Firewall.py

```
#!/usr/bin/env python3
import random
import time
import datetime
import threading
import queue
import re
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import json
import sys
from sklearn.linear_model import LogisticRegression
import numpy as np

FIREWALL_IPS = ["192.168.1.1", "192.168.1.2", "10.0.0.1"]
ALERT_THRESHOLD = 5

class Firewall:
    def __init__(self, ip):
        self.ip = ip
        self.log_entries = []
        self.lock = threading.Lock()
    def generate_log_entry(self):
        now = datetime.datetime.now()
        event_type = random.choice(["ALLOW", "DENY", "DROP"])
        source_ip = ".".join(str(random.randint(1, 255)) for _ in range(4))
        destination_ip = self.ip
        port = random.choice([80, 443, 22, 8080, 21])
        severity = random.randint(1, 10)
        log_entry = {"timestamp": now, "event_type": event_type, "source_ip":
source_ip, "destination_ip": destination_ip, "port": port, "severity": severity}
        return log_entry
    def add_log_entry(self, entry):
        with self.lock:
            self.log_entries.append(entry)
    def get_logs(self):
        with self.lock:
            logs_copy = self.log_entries.copy()
        return logs_copy
    def clear_logs(self):
        with self.lock:
            self.log_entries = []

class RiskAnalyzer:
    def __init__(self, firewall_instances):
        self.firewall_instances = firewall_instances
    def analyze_logs(self):
        risk_score = 0
        detailed_report = []
        for fw in self.firewall_instances:
```

```
            logs = fw.get_logs()
            for log in logs:
                if log["event_type"] == "DENY":
                    increase = log["severity"] * 1.5
                    risk_score += increase
                    detailed_report.append("Risk increased by " + str(increase)
+ " due to DENY event from " + log["source_ip"])
                elif log["event_type"] == "DROP":
                    increase = log["severity"] * 2.0
                    risk_score += increase
                    detailed_report.append("Risk increased by " + str(increase)
+ " due to DROP event from " + log["source_ip"])
                else:
                    increase = log["severity"] * 0.5
                    risk_score += increase
                    detailed_report.append("Risk increased by " + str(increase)
+ " due to ALLOW event from " + log["source_ip"])
        return risk_score, detailed_report
    def query_firewall(self, query_str):
        results = []
        for fw in self.firewall_instances:
            logs = fw.get_logs()
            for log in logs:
                if query_str in log["event_type"] or query_str in
log["source_ip"]:
                    results.append(log)
        return results
    def extended_query(self, pattern):
        results = []
        regex = re.compile(pattern)
        for fw in self.firewall_instances:
            logs = fw.get_logs()
            for log in logs:
                if regex.search(log["event_type"]) or
regex.search(log["source_ip"]):
                    results.append(log)
        return results

class QueryHandler:
    def __init__(self, risk_analyzer):
        self.risk_analyzer = risk_analyzer
    def handle_query(self, query):
        result_logs = self.risk_analyzer.query_firewall(query)
        return result_logs
    def handle_extended_query(self, pattern):
        result_logs = self.risk_analyzer.extended_query(pattern)
        return result_logs

class DetailedLogger:
    def __init__(self):
        self.logs = []
    def log(self, message):
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

```
        log_message = "[" + timestamp + "] " + message
        self.logs.append(log_message)
        print(log_message)
    def get_all_logs(self):
        return self.logs.copy()

class RealTimeNotifier:
    def __init__(self, smtp_server, smtp_port, username, password, from_addr,
to_addr):
        self.smtp_server = smtp_server
        self.smtp_port = smtp_port
        self.username = username
        self.password = password
        self.from_addr = from_addr
        self.to_addr = to_addr
    def send_notification(self, subject, body):
        msg = MIMEMultipart()
        msg['From'] = self.from_addr
        msg['To'] = self.to_addr
        msg['Subject'] = subject
        msg.attach(MIMEText(body, 'plain'))
        server = smtplib.SMTP(self.smtp_server, self.smtp_port)
        server.starttls()
        server.login(self.username, self.password)
        text = msg.as_string()
        server.sendmail(self.from_addr, self.to_addr, text)
        server.quit()

class Dashboard:
    def __init__(self, risk_analyzer):
        self.risk_analyzer = risk_analyzer
    def display(self):
        risk_score, report = self.risk_analyzer.analyze_logs()
        print("\n----- DASHBOARD -----")
        print("Current Risk Score: " + str(round(risk_score, 2)))
        print("Recent Events:")
        logs = []
        for fw in self.risk_analyzer.firewall_instances:
            logs.extend(fw.get_logs())
        logs = sorted(logs, key=lambda x: x["timestamp"], reverse=True)
        for log in logs[:10]:
            print(str(log))
        print("----------------------\n")
    def run(self, interval):
        while True:
            self.display()
            time.sleep(interval)

class AttackPredictor:
    def __init__(self):
        self.model = LogisticRegression()
        self.trained = False
    def prepare_data(self, firewalls):
```

```
        X = []
        y = []
        for fw in firewalls:
            logs = fw.get_logs()
            for log in logs:
                feature = [log["severity"], log["port"]]
                X.append(feature)
                label = 1 if log["event_type"] in ["DENY", "DROP"] else 0
                y.append(label)
        if len(X) == 0:
            X = [[0, 0]]
            y = [0]
        return np.array(X), np.array(y)
    def train(self, firewalls):
        X, y = self.prepare_data(firewalls)
        self.model.fit(X, y)
        self.trained = True
    def predict(self, log_entry):
        if not self.trained:
            return 0
        feature = np.array([[log_entry["severity"], log_entry["port"]]])
        prediction = self.model.predict(feature)
        return prediction[0]

class AuditLogger:
    def __init__(self, audit_log_file):
        self.audit_log_file = audit_log_file
        self.lock = threading.Lock()
    def log_change(self, change_description):
        with self.lock:
            entry = {"timestamp": datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S"), "change": change_description}
            with open(self.audit_log_file, "a") as f:
                f.write(json.dumps(entry) + "\n")
    def get_audit_logs(self):
        with self.lock:
            try:
                with open(self.audit_log_file, "r") as f:
                    lines = f.readlines()
                    logs = [json.loads(line.strip()) for line in lines]
                    return logs
            except FileNotFoundError:
                return []

def simulate_firewall_logging(firewall_instance, stop_event, logger, notifier,
predictor, audit_logger):
    while not stop_event.is_set():
        log_entry = firewall_instance.generate_log_entry()
        firewall_instance.add_log_entry(log_entry)
        risk_score, _ = RiskAnalyzer([firewall_instance]).analyze_logs()
        if risk_score > ALERT_THRESHOLD:
            subject = "Alert from Firewall " + firewall_instance.ip
```

```
            body = "Risk score " + str(risk_score) + " exceeded threshold.
Event: " + log_entry["event_type"]
            try:
                notifier.send_notification(subject, body)
            except Exception as e:
                pass
        prediction = predictor.predict(log_entry)
        if prediction == 1:
            audit_logger.log_change("Predicted attack event on firewall " +
firewall_instance.ip + " with event " + log_entry["event_type"])
        time.sleep(random.uniform(0.1, 0.5))

def periodic_risk_analysis(risk_analyzer, interval, stop_event, logger):
    while not stop_event.is_set():
        risk_score, report = risk_analyzer.analyze_logs()
        logger.log("Current Risk Score: " + str(round(risk_score, 2)))
        for line in report:
            logger.log("Detail: " + line)
        time.sleep(interval)

def query_processing_thread(query_handler, query_queue, stop_event, logger):
    while not stop_event.is_set():
        try:
            query = query_queue.get(timeout=1)
            results = query_handler.handle_query(query)
            logger.log("Query '" + query + "' returned " + str(len(results)) + "
results.")
            for res in results:
                logger.log("Result: " + str(res))
            query_queue.task_done()
        except queue.Empty:
            continue

def extended_query_processing_thread(query_handler, query_queue, stop_event,
logger):
    while not stop_event.is_set():
        try:
            pattern = query_queue.get(timeout=1)
            results = query_handler.handle_extended_query(pattern)
            logger.log("Extended Query '" + pattern + "' returned " +
str(len(results)) + " results.")
            for res in results:
                logger.log("Extended Result: " + str(res))
            query_queue.task_done()
        except queue.Empty:
            continue

def simulate_user_queries(query_queue, stop_event, logger):
    simulated_queries = ["DENY", "DROP", "ALLOW", "192", "10"]
    for _ in range(10):
        if stop_event.is_set():
            break
        sample_query = random.choice(simulated_queries)
```

```
        query_queue.put(sample_query)
        logger.log("Simulated user query added: " + sample_query)
        time.sleep(random.uniform(3, 7))

def simulate_extended_user_queries(query_queue, stop_event, logger):
    simulated_patterns = [r"DENY", r"DROP", r"ALLOW", r"^\d{1,3}", r"^10"]
    for _ in range(5):
        if stop_event.is_set():
            break
        sample_pattern = random.choice(simulated_patterns)
        query_queue.put(sample_pattern)
        logger.log("Simulated extended query added: " + sample_pattern)
        time.sleep(random.uniform(5, 10))

def periodic_log_cleanup(firewall_instances, interval, stop_event, logger):
    while not stop_event.is_set():
        logger.log("Starting periodic log cleanup...")
        for fw in firewall_instances:
            fw.clear_logs()
            logger.log("Cleared logs for firewall " + fw.ip)
        time.sleep(interval)

def main():
    logger = DetailedLogger()
    logger.log("Starting Cybersecurity Risk Analysis System with Extended
Functionalities...")
    firewalls = []
    for ip in FIREWALL_IPS:
        fw = Firewall(ip)
        firewalls.append(fw)
        logger.log("Initialized Firewall with IP: " + ip)
    analyzer = RiskAnalyzer(firewalls)
    logger.log("RiskAnalyzer initialized with " + str(len(firewalls)) + "
firewalls.")
    query_handler = QueryHandler(analyzer)
    logger.log("QueryHandler initialized.")
    notifier = RealTimeNotifier("smtp.example.com", 587, "user@example.com",
"password", "from@example.com", "to@example.com")
    dashboard = Dashboard(analyzer)
    predictor = AttackPredictor()
    predictor.train(firewalls)
    audit_logger = AuditLogger("audit.log")
    stop_event = threading.Event()
    logger.log("Stop event created for thread management.")
    fw_threads = []
    for fw in firewalls:
        t = threading.Thread(target=simulate_firewall_logging, args=(fw,
stop_event, logger, notifier, predictor, audit_logger))
        t.daemon = True
        t.start()
        fw_threads.append(t)
        logger.log("Started logging thread for firewall " + fw.ip)
```

```
    analysis_thread = threading.Thread(target=periodic_risk_analysis,
args=(analyzer, 10, stop_event, logger))
    analysis_thread.daemon = True
    analysis_thread.start()
    logger.log("Risk analysis thread started.")
    query_queue = queue.Queue()
    logger.log("Query queue initialized.")
    query_thread = threading.Thread(target=query_processing_thread,
args=(query_handler, query_queue, stop_event, logger))
    query_thread.daemon = True
    query_thread.start()
    logger.log("Query processing thread started.")
    extended_query_queue = queue.Queue()
    logger.log("Extended query queue initialized.")
    extended_query_thread =
threading.Thread(target=extended_query_processing_thread, args=(query_handler,
extended_query_queue, stop_event, logger))
    extended_query_thread.daemon = True
    extended_query_thread.start()
    logger.log("Extended query processing thread started.")
    cleanup_thread = threading.Thread(target=periodic_log_cleanup,
args=(firewalls, 30, stop_event, logger))
    cleanup_thread.daemon = True
    cleanup_thread.start()
    logger.log("Log cleanup thread started.")
    dashboard_thread = threading.Thread(target=dashboard.run, args=(15,))
    dashboard_thread.daemon = True
    dashboard_thread.start()
    logger.log("Dashboard thread started.")
    user_query_thread = threading.Thread(target=simulate_user_queries,
args=(query_queue, stop_event, logger))
    user_query_thread.daemon = True
    user_query_thread.start()
    logger.log("User query simulation thread started.")
    extended_user_query_thread =
threading.Thread(target=simulate_extended_user_queries,
args=(extended_query_queue, stop_event, logger))
    extended_user_query_thread.daemon = True
    extended_user_query_thread.start()
    logger.log("Extended user query simulation thread started.")
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        logger.log("KeyboardInterrupt received. Stopping simulation...")
    finally:
        stop_event.set()
        logger.log("Stop event set. Waiting for threads to finish...")
        for t in fw_threads:
            t.join(timeout=1)
        analysis_thread.join(timeout=1)
        query_thread.join(timeout=1)
        extended_query_thread.join(timeout=1)
```

```
        cleanup_thread.join(timeout=1)
        user_query_thread.join(timeout=1)
        extended_user_query_thread.join(timeout=1)
        dashboard_thread.join(timeout=1)
        logger.log("All threads have been terminated. Exiting system.")
if __name__ == '__main__':
    main()
```

Файл ExportReportingSystem.py

```
import random
import time
import datetime
import threading
import queue
import csv
import json
import socket
import statistics

class ThreatIntelligenceModule:
    def __init__(self):
        self.threats = []
        self.last_fetched = None
    def fetch_threats(self):
        self.threats = ["123.45.67.89", "98.76.54.32", "192.0.2.1",
"203.0.113.5"]
        self.last_fetched = datetime.datetime.now()
        return self.threats
    def is_malicious(self, ip):
        if not self.threats:
            self.fetch_threats()
        return ip in self.threats

class ExportReportingSystem:
    def export_csv(self, logs, filename):
        with open(filename, 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(["timestamp", "event_type", "source_ip",
"destination_ip", "port", "severity"])
            for log in logs:
                writer.writerow([log["timestamp"], log["event_type"],
log["source_ip"], log["destination_ip"], log["port"], log["severity"]])
    def export_json(self, logs, filename):
        with open(filename, 'w') as f:
            json.dump(logs, f, default=str)
    def export_report(self, logs, filename):
        with open(filename, 'w') as f:
            for log in logs:
                f.write(str(log) + "\n")

class SIEMIntegration:
    def __init__(self, siem_host, siem_port):
        self.siem_host = siem_host
        self.siem_port = siem_port
    def send_to_siem(self, log_entry):
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.connect((self.siem_host, self.siem_port))
```

```
            s.sendall(json.dumps(log_entry, default=str).encode())
            s.close()
        except:
            pass

class AccessControl:
    def __init__(self):
        self.users = {"admin": {"password": "admin123", "role": "admin"},
"analyst": {"password": "analyst123", "role": "analyst"}, "viewer": {"password":
"viewer123", "role": "viewer"}}

        self.permissions = {"admin": ["read", "write", "delete"], "analyst":
["read", "write"], "viewer": ["read"]}
    def authenticate(self, username, password):
        user = self.users.get(username)
        if user and user["password"] == password:
            return True

        return False
    def check_permission(self, username, permission):
        user = self.users.get(username)
        if user:
            role = user["role"]
            return permission in self.permissions.get(role, [])
        return False

class BehavioralAnomalyDetector:
    def __init__(self):
        self.severity_history = []
        self.window_size = 10
        self.threshold_factor = 2

    def add_log(self, log_entry):
        self.severity_history.append(log_entry["severity"])
        if len(self.severity_history) > self.window_size:
            self.severity_history.pop(0)

    def detect_anomaly(self, log_entry):
        if len(self.severity_history) < self.window_size:
            return False
        avg = statistics.mean(self.severity_history)
        stdev = statistics.stdev(self.severity_history)
        if stdev == 0:
            stdev = 1
        if abs(log_entry["severity"] - avg) > self.threshold_factor * stdev:
            return True
        return False

class Firewall:
    def __init__(self, ip):
        self.ip = ip
```

```
        self.log_entries = []
        self.lock = threading.Lock()
    def generate_log_entry(self):
        now = datetime.datetime.now()
        event_type = random.choice(["ALLOW", "DENY", "DROP"])
        source_ip = ".".join(str(random.randint(1, 255)) for _ in range(4))
        destination_ip = self.ip
        port = random.choice([80, 443, 22, 8080, 21])
        severity = random.randint(1, 10)

        log_entry = {"timestamp": now, "event_type": event_type, "source_ip":
source_ip, "destination_ip": destination_ip, "port": port, "severity": severity}
        return log_entry

    def add_log_entry(self, entry):
        with self.lock:
            self.log_entries.append(entry)
    def get_logs(self):
        with self.lock:
            logs_copy = self.log_entries.copy()
        return logs_copy
    def clear_logs(self):
        with self.lock:
            self.log_entries = []

class RiskAnalyzer:
    def __init__(self, firewall_instances):
        self.firewall_instances = firewall_instances
    def analyze_logs(self):
        risk_score = 0
        detailed_report = []

        for fw in self.firewall_instances:
            logs = fw.get_logs()
            for log in logs:
                if log["event_type"] == "DENY":
                    increase = log["severity"] * 1.5
                    risk_score += increase

                    detailed_report.append("Risk increased by " + str(increase)
+ " due to DENY event from " + log["source_ip"])
                elif log["event_type"] == "DROP":
                    increase = log["severity"] * 2.0
                    risk_score += increase
                    detailed_report.append("Risk increased by " + str(increase)
+ " due to DROP event from " + log["source_ip"])
                else:
                    increase = log["severity"] * 0.5
                    risk_score += increase

                    detailed_report.append("Risk increased by " + str(increase)
+ " due to ALLOW event from " + log["source_ip"])
```

```
        return risk_score, detailed_report

def simulate_firewall_logging(firewall_instance, stop_event, threat_intel, siem,
anomaly_detector):
    while not stop_event.is_set():
        log_entry = firewall_instance.generate_log_entry()
        firewall_instance.add_log_entry(log_entry)
        if threat_intel.is_malicious(log_entry["source_ip"]):
            pass
        siem.send_to_siem(log_entry)
        anomaly_detector.add_log(log_entry)
        if anomaly_detector.detect_anomaly(log_entry):
            pass
        time.sleep(random.uniform(0.1, 0.5))

def periodic_risk_analysis(risk_analyzer, interval, stop_event):
    while not stop_event.is_set():
        risk_score, report = risk_analyzer.analyze_logs()
        print("Current Risk Score: " + str(round(risk_score, 2)))
        for line in report:
            print("Detail: " + line)
        time.sleep(interval)

def main():
    threat_intel = ThreatIntelligenceModule()
    export_system = ExportReportingSystem()
    siem = SIEMIntegration("localhost", 514)
    access_control = AccessControl()
    anomaly_detector = BehavioralAnomalyDetector()
    firewalls = []
    for ip in ["192.168.1.1", "192.168.1.2", "10.0.0.1"]:
        fw = Firewall(ip)
        firewalls.append(fw)
    risk_analyzer = RiskAnalyzer(firewalls)
    stop_event = threading.Event()
    fw_threads = []

    for fw in firewalls:
        t = threading.Thread(target=simulate_firewall_logging, args=(fw,
stop_event, threat_intel, siem, anomaly_detector))
        t.daemon = True
        t.start()
        fw_threads.append(t)
    analysis_thread = threading.Thread(target=periodic_risk_analysis,
args=(risk_analyzer, 10, stop_event))
    analysis_thread.daemon = True
    analysis_thread.start()
    time.sleep(30)
    for fw in firewalls:
        logs = fw.get_logs()
```

```
        export_system.export_csv(logs, fw.ip.replace('.', '_') + "_logs.csv")
        export_system.export_json(logs, fw.ip.replace('.', '_') + "_logs.json")
        export_system.export_report(logs, fw.ip.replace('.', '_') +
"_report.txt")

    username = "admin"
    password = "admin123"
    if access_control.authenticate(username, password):
        if access_control.check_permission(username, "delete"):
            print("User " + username + " has delete permission.")
        else:
            print("User " + username + " does not have delete permission.")
    else:

        print("Authentication failed for " + username)
    stop_event.set()
    for t in fw_threads:
        t.join(timeout=1)
    analysis_thread.join(timeout=1)
if __name__ == '__main__':
    main()
```