Міністерство освіти і науки України
Харківський національний університет ім. В. Н. Каразіна
Факультет комп'ютерних наук
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

«Допущено до захисту»
В.о. завідувача кафедри БІСТ
Мелкозьорова О.М.

_____

« » 2024 р.

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему: «Методи та засоби безпеки інформаційних ресурсів в банківській системі»

оцінка « »

Голова ЕК

Лемешко О.В._____

Керівник к.т.н Мелкозьорова О.М.

Рецензент д.т.н. Краснобаєв В.А.

Виконавець: студентка групи КБ-42

_____ Шишкіна П.В.

Харків – 2024

РЕФЕРАТ

Дипломна робота викладена на 43 сторінках, вона містить 3 розділи, 26 рисунків, 1 таблицю, 19 джерел в переліку посилань.

Об’єктом розгляду є інформаційні ресурси в банківській системі.

Предметом дослідження є методи та засоби безпеки інформаційних ресурсів в банківській системі.

Метою роботи є порівняння систем виявлення вторгнень та програмна реалізація аналізу лог-файлів.

У першому розділі розглянуто загрози в банківській системі, наведені державні стандарти та постанови яким повинна відповідати банківська система та описані підходи для реалізації мережевої безпеки. У другому розділі обґрунтовано важливість використання систем виявлення вторгнень та їх види. Також наведено теоретичне порівняння Snort та Suricata. У третьому розділі представлено опис віртуальних машин за допомогою яких проводилося тестування IDS, шляхом симуляції syn-flood атаки, надсилання пакетів та сканування портів. Крім цього наведено покрокову реалізацію програми для аналізу та порівняння систем виявлення вторгнень.

Ключові слова: АТАКА, БАНКІВСЬКА СИСТЕМА, БЕЗПЕКА, ЗАГРОЗИ, ПОРІВНЯННЯ, СИСТЕМА ВИЯВЛЕННЯ ВТОРГНЕНЬ, ТЕСТУВАННЯ.

ABSTRACT

The diploma work consists of 43 pages, 3 chapters, 26 figures, 1 table, and 19 references.

The object of consideration is information resources in the banking system.

The subject of research is methods and means of information resources security in the banking system.

The purpose of the study is to compare intrusion detection systems and software implementation of log file analysis.

The first section discusses threats in the banking system, provides state standards and regulations that the banking system must comply with, and describes approaches to implementing network security. The second section substantiates the importance of using intrusion detection systems and their types. A theoretical comparison of Snort and Suricata is also provided. The third section describes the virtual machines used to test IDS by simulating a syn-flood attack, sending packets, and scanning ports. In addition, a step-by-step implementation of a program for analyzing and comparing intrusion detection systems is presented.

Keywords: ATTACK, BANKING SYSTEM, SECURITY, THREATS, COMPARISON, INTRUSION DETECTION SYSTEM, TESTING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ СКОРОЧЕНЬ І ТЕРМІНІВ

ДСТУ	Державний Стандарт України
НБУ	Національний Банк України
AES	Advanced Encryption Standard
DoS	denial of service atack
DSA	Digital Signature Algorithm
ECDSA	Elliptic Curve Digital Signature Algorithm
HIDS	Host-based intrusion detection system
ICMP	Internet Control Message Protocol
IPS	intrusion prevention system
IDS	intrusion detection system
NIDS	Network intrusion detection system
RSA	абревіатура від прізвищ Rivest, Shamir та Adleman
SHA	Secure Hash Algorithm
UDP	User Datagram Protocol
TCP	Transmission Control Protocol

ВСТУП

Інформаційні ресурси в банківській системі є критично важливими для забезпечення стабільності та безперебійного функціонування фінансових установ. З розвитком цифрових технологій і зростанням кіберзагроз, захист інформаційних ресурсів стає все більш актуальним завданням.

У сучасному світі банки стикаються з різноманітними видами загроз, включаючи фішингові атаки, шкідливе програмне забезпечення, крадіжку даних за допомогою кейлогерів та DDoS атаки. Ці загрози можуть призвести до значних фінансових втрат, підриву репутації та витоку конфіденційної інформації.

Робота присвячена порівняльному аналізу систем виявлення вторгнень в результаті одночасного надсилання тестового трафіку за допомогою віртуальних машин. Та подальшому аналізу отриманих результатів за допомогою створеної програмної реалізації.

Актуальність дослідження методів та засобів безпеки інформаційних ресурсів у банківській системі зумовлена необхідністю захисту клієнтських даних і фінансових транзакцій від несанкціонованого доступу та кібератак. У даній роботі розглядаються сучасні підходи до захисту інформації в банківських системах, зокрема, системи виявлення вторгнень (IDS) та їх порівняння.

1 ЗАГАЛЬНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Огляд загроз в банківській системі

Існує велике різноманіття загроз інформаційній системі банку. Так, деякі з атак спрямовані на отримання даних карток користувачів, як то фішингові посилання[1] (такі, що маскуються під банківські системи для крадіжки даних картки), або ж сповіщення з пропозицією вказати дані картки, для отримання начебто бонусів.

Так, на прикладі рисунку 1.1 можна побачити, що зловмисники можуть надавати посилання під маскуванням різних банків, проте, найчастіше, всі посилання ведуть на одну й ту саму форму, з нібито «заявою» на отримання грошей, в якій користувач може ввести свої банківські дані.

Рисунок 1.1 – Приклад злоякісного повідомлення[2]

Поширеним типом крадіжки банківських даних є використання кейлогерів[3]. Це тип шкідливих програм або пристроїв, що призначені для відстеження реєстрації кожного натискання на клавіатуру або екрану мобільного пристрою. Ця проблема наразі є дуже розповсюдженою, оскільки в сучасному світі дуже розповсюджений мобільний банкінг. Таким чином, якщо користувач не використовує автентифікацію за допомогою відбитку пальцю, то за допомогою кейлогеру можна вкрасти дані для заходу в банківський додаток.

Також, окрім витягування банківських даних таким чином можуть бути отримані особисті дані користувачів, які в майбутньому можуть бути використані в тому числі для крадіжки особистості.

Найбільшою веб небезпекою для самих банків, як і для більшості інтернет сервісів, є DDoS атака (тобто така, що надсилає дуже велику кількість однотипних запитів, для блокування пропускної здатності каналу. Найчастіше такі атаки працюють за схемою, зображеною на рисунку 1.2.

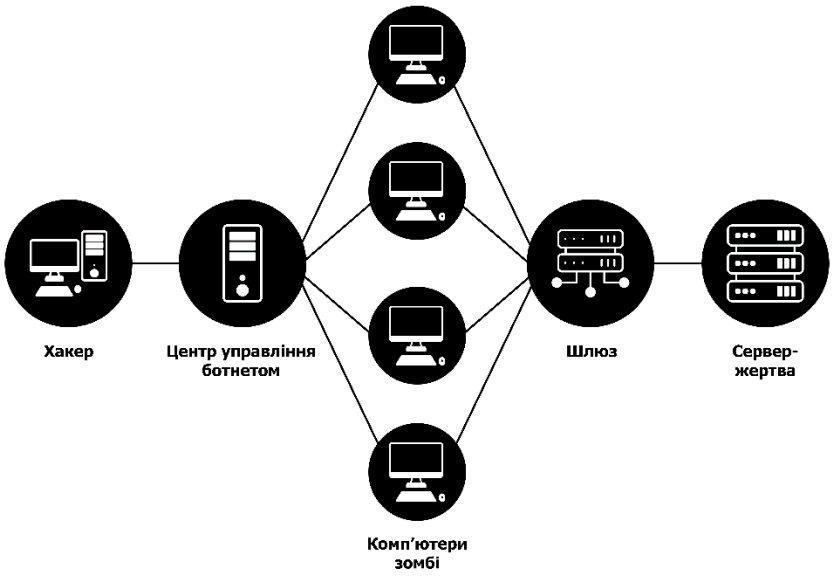

Рисунок 1.2 – Схема DDoS-атаки

Зловмисник, через певний сервер, до якого підключені «боти» (тобто пристрої, які автоматизовано надсилають велику кількість запитів. Це можуть

бути як стаціонарні пк, так і телефони. Іноді, за допомогою вірусів, такими «ботами» можуть бути навіть пк деяких взагалі не пов’язаних користувачів, оскільки їх пк можуть надсилати запити без згоди самих користувачів).

Оскільки наразі більшість діяльності банку з клієнтами відбувається шляхом комунікування різних сервісів через мережу інтернет (мобільні банківські застосунки, POS-термінали, банкомати, стаціонарні банківські термінали і т.д), така атака може заблокувати майже всю зовнішню банківську діяльність. Зазвичай при такій атаці внутрішня банківська мережа не страждає, оскільки вона має дуже обмежену кількість «точок входу» в мережу (тобто таких пристроїв, через які можна отримати доступ до мережі). Для протидії таким атакам, зазвичай, використовують разом декілька методів захисту, наприклад:

- Міжмережеві екрани, що дозволяють налаштувати правила фільтрації трафіку за декількома параметрами (IP адресами, портами, мережевими протоколами і т. д.)
- Системи виявлення вторгнень, котрі аналізують весь мережевий трафік в режимі реального часу та дозволяють виявити мережеві атаки і блокувати загрози шляхом переривання з’єднання.

Іноді хакери не мають на меті викрадення грошей, натомість вони шифрують дані, та потім вимагають гроші за розшифрування, і у випадку відмови – викладають дані у мережу.

Поширеними є також malware-атаки. Це загальна назва шкідливих програм, які створюються з метою нанесення шкоди комп’ютерам, мережам та даним. Це можуть бути як віруси, які прикріплюються до інших файлів, та розповсюджуються, наприклад, через електронну пошту, або ж шляхом завантаження з інтернету. Особливим підвидом вірусних програм є «троянські коні» – це програми, які приховуються під корисні або невразливі програми. Після їх встановлення вони можуть як збирати конфіденційні дані, так і надавати несанкціонований доступ до системи.

Аналіз шкідливого програмного забезпечення можна поділити на два типи: статичний і динамічний аналіз. Обидва методи показують функціональність досліджуваного шкідливого ПЗ, проте використовувані інструменти, час і необхідні навички відрізняються.

Статичний аналіз – це базовий аналіз двійкового коду, що допомагає зрозуміти функції шкідливого ПЗ. Поведінковий або динамічний аналіз вивчає поведінку шкідливого ПЗ під час встановлення, виконання та запуску.
Динамічний аналіз передбачає виконання шкідливого ПЗ для дослідження його поведінки та впливу на системні ресурси та мережу. Він виявляє технічні сигнатури, що підтверджують зловмисні наміри, і знаходить корисну інформацію, таку як доменні імена, шляхи до файлів, створені ключі реєстру, IP-адреси, додаткові файли, інсталяційні файли, бібліотеки DLL та пов'язані файли в системі або мережі.

В останній час дуже поширеною вразливістю стало зараження системи криптомайнерами, оскільки банківські сервери, потенційно, можуть обробляти дуже велику кількість даних. Використання криптомайнерів може призвести до спрощення DDoS-атаки на систему, оскільки вони дають велике навантаження на обчислювальні ресурси. Також, окрім цього, використання криптомайнерів може призвести до відкриття додаткових дірок в безпеці банківських систем. Даний вид загроз може бути також небезпечним оскільки криптомайнери можуть резервувати обчислювальні ресурси під свої потреби, таким чином не надаючи їх штатним програмним засобам.

Для забезпечення безпеки від криптомайнерів може допомогти як банальне встановлення антивірусного програмного забезпечення, моніторинг та аналіз трафіку в мережі, так і виявлення значного збільшення енергозатрат внаслідок зараження криптомайнерами.

Також небезпеку можуть створювати і працівники банку, які мають законний доступ до закритої інформації. Вони можуть навіть ненавмисно розголосити дані, пошкодити систему або ж підробити документи. Такі

загрози складко попередити та виявити, оскільки майже неможливо виявити який саме працівник може становити потенційну загрозу.

Призвести до вразливостей може і неправильна конфігурація сервері та мережевого обладнання. Слабкими місцями можуть бути слабкі паролі адміністратора або застаріле ПЗ. Хакери активно сканують мережі на наявність таких місць. Сюди також можна віднести і проблему передачі конфіденційних даних та трафіку без шифрування, оскільки таким чином створюється ризик перехоплення та витоку інформації.

Для захисту від наведених типів загроз потрібно застосовувати комплексні міри безпеки, такі як навчання персоналу та клієнтів, технічні рішення, тощо. Запорукою мінімізації ризиків кібератак є безперервний моніторинг внутрішніх систем, та систем взаємодії банку з зовнішніми системами.

Загалом, можна виділити такі основні типи інформаційних загроз банківській системі:

- Протиправне використання та збирання інформації;
- Порушення технологій опрацювання інформації;
- Несанкціонований доступ до інформації, що є в банківських установах і їх базах даних;
- Зловживання повноваженнями працівниками банку. [4]

За типом загроз можна виділити такі дві групи:

1) Випадкові загрози, тобто події, що не залежать від людини;
2) Навмисні загрози, тобто такі, що можуть реалізуватися учасниками процесу обробки інформації.

1.2 Принципи захисту інформації у банківській системі

Важливо розуміти, що захист від загроз не повинен бути одноразовим, це має бути стратегічна програма для забезпечення захисту інформації в форматі 24/7. Також не повинно бути ніяких «сліпих зон» у мережі, тобто проходження буть якого байта інформації повинно бути прозорим.

Відповідно до опису системи захисту інформації в правилах платіжної системи, платіжною організацією якої ж резидент, всі міжмережеві екрани, що використовуються в платіжній системі повинні працювати лише з вказаним діапазоном IP-адрес та портів, та перегляд їх налаштувань повинен здійснюватися не рідше одного разу на 3 місяці, що дозволить регулярно впроваджувати нові системи захисту, або швидко адаптуватися під нові типи атак.

В політиках безпеки банків вказуються вимоги до паролів, за якими відбувається автентифікація відповідальних осіб до інформаційних систем. Так, задля запобігання автентифікації людини під акаунтом посадової особи паролі повинні змінюватися щонайменше 1 раз на 60 днів, а також зберігатись на серверах у виглядах геш-функцій від паролю з додаванням «солі».

Для безпеки автентифікації користувачів, зазвичай, мобільні застосунки базуються або на двофакторній автентифікації, або ж, якщо телефон підтримує дані функції, то автентифікація відбувається шляхом використання відбитку пальців. Такий спосіб автентифікації дозволяє доступ до банківського рахунку тільки власнику мобільного пристрою.

1.3 Політики безпеки банків

Зазвичай, більшість банків будує свої політики безпеки на державних стандартах України з питань інформаційної безпеки:

- ДСТУ ISO/IEC 27000:2019 «Інформаційні технології. Методи захисту. Система управління інформаційною безпекою. Огляд і словник термінів»[5];
- ДСТУ ISO/IEC 27001:2015 «Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги»[6];
- ДСТУ ISO/IEC 27002:2015 «Інформаційні технології. Методи захисту. Звід практик щодо заходів інформаційної безпеки»[7];

Ці стандарти визначають вимоги до проектування, впровадження, підтримки та постійного вдосконалення системи управління інформаційною

безпекою з урахуванням обставин організації. Вимоги, наведені в цих стандартах, є загальними та можуть бути запроваджені для всіх організацій незалежно від типу, розміру та природи.

Згідно цих стандартів, має бути розроблено та впроваджено політику використання криптографічних засобів для захисту інформації. Також має бути розроблено систему управління ключами, яка має базуватися на погодженому наборі стандартів, процедур та методів безпеки щодо:

- Генерації ключів для різних криптографічних систем;
- Генерації та отримання сертифікатів відкритих ключів;
- Розподілу ключів серед призначених користувачів, включаючи те, як ключи повинні бути активовані після отримання;

Для зниження ймовірності компрометації, активації й деактивації даних, правила для ключів має бути визначено так, щоб ключі можна було застосовувати лише обмежений період часу, визначений відповідно до політики управління ключами.

Крім цього дані стандарти описують заходи щодо фізичної безпеки даних, безпеки комунікацій, розробки та підтримки інформаційних систем, тощо.

Також, для побудови політики безпеки банку зазвичай спираються на постанови НБУ:

- Постанови Правління НБУ №95 від 28.09.2017 року «Про затвердження Положення про організацію заходів із забезпечення інформаційної безпеки в банківській системі України»[8];
- Постанови Правління НБУ №4 від 16.01.2021 року «Про затвердження Положення про здійснення контролю за дотриманням банками вимог законодавства з питань інформаційної безпеки, кіберзахисту та електронних довірчих послуг»[9];
- Постанови Правління НБУ №178 від 12.08.2022 «Про затвердження Положення про організацію кіберзахисту в банківській системі України

та внесення змін до Положення про визначення об'єктів критичної інфраструктури в банківській системі України»[10].

В даних постановах більш детально описуються заходи із забезпечення інформаційної безпеки в банках, так, описано алгоритми, які банк застосовувати для криптографічного захисту. А саме, цей перелік складає:

1) Асиметричні алгоритми:
    a. Алгоритм Діффі – Геллмана для узгодження сеансових ключів шифрування;
    b. Алгоритм цифрового підпису (DSA) для цифрових підписів;
    c. Алгоритм Діффі – Геллмана на еліптичних кривих для узгодження сеансових ключів шифрування;
    d. Алгоритм цифрового підпису на еліптичних кривих (ECDSA) для цифрових підписів;
    e. алгоритм Ривест - Шаміра - Адлемана (RSA) для цифрових підписів і узгодження сеансових ключів шифрування або аналогічних ключів;
2) Алгоритми гешування, такі як SHA-224, SHA-256, SHA-384, SHA-521, «Купина» або більш криптостійкі;
3) Алгоритми симетричного шифрування, такі як :
    a. AES із використанням довжини ключа 128, 192, 256 біт або більше;
    b. Алгоритм криптографічного перетворення
    c. Алгоритм «Калина»[8]

Також постанова надає вимоги саме до використання даних алгоритмів, і для конфігурації мережі, розробки БД та роботи з БД, розміщення серверів, і т.д. Так з вимог до використання алгоритмів можна визначити:

- Для реалізації алгоритмів Діффі – Геллмана та цифрового підпису необхідно використовувати розмір модуля не менше ніж 2048 біт

- Для застосування алгоритму на еліптичних кривих потрібно використовувати еліптичні криві з ДСТУ 4145-2002
- Для застосування алгоритму RSA для цифрових підписів і ключів шифрування сеансу або аналогічних ключів потрібно використовувати розмір модуля не менше ніж 2048 біт, а також потрібно використовувати різні ключові пари для передавання ключів шифрування сеансу.[8]

Конфіденційність даних може бути досягнута за допомогою криптографії, що включає в себе процес шифрування та розшифрування. Шифрування даних полягає в захисті повідомлень шляхом перетворення їх у приховані тексти, а розшифрування – в процесі отримання оригінальних текстів з прихованих текстів. Ці процеси виконуються за допомогою певних ключів, і кожен алгоритм шифрування спрямований на ускладнення процесу розшифрування без наявності відповідного ключа. На рисунку 1.3 показано загальну ідею шифрування та дешифрування.[10]

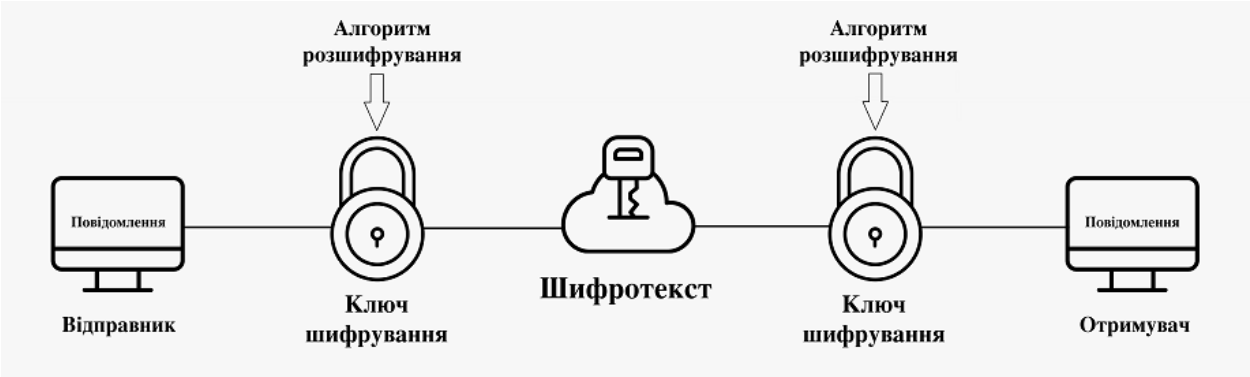

Рисунок 1.3 – Загальна ідея шифрування та дешифрування

Згідно постанови №95 банк повинен впровадити системи для виявлення несанкціонованого доступу до мережі (Intrusion Detection System, IDS) та системи для запобігання несанкціонованому доступу до мережі (Intrusion Prevention System, IPS) з метою захисту периметра своєї мережі.

1.4 Мережева безпека банківських систем

Мережева безпека критично важлива для банківських систем, оскільки вона забезпечує захист від кібератак, зберігає конфіденційність фінансових та

особистих даних клієнтів, і забезпечує безперебійну роботу банківських послуг. Банківські системи є привабливими цілями для кіберзлочинців через значні обсяги фінансових та особистих даних. В умовах зростання кіберзлочинності, надійна система безпеки допомагає уникнути шахрайства та фінансових втрат. Дотримання високих стандартів мережевої безпеки дозволяє банкам відповідати регуляторним вимогам, що запобігає накладанню штрафів і санкцій. Це також підтримує довіру клієнтів, які очікують, що їхні фінансові операції та дані будуть захищені, і сприяє загальній стабільності та надійності банківської діяльності.

Існує чотири основні класифікації методів захисту безпеки, що використовуються для ідентифікації та запобігання загрозам і атакам у мережі:

- Превентивний підхід: складається з методів і прийомів, які дозволяють запобігти загрозам або атакам на цільову мережу. Основні превентивні методи, що використовуються в мережах, включають кілька ключових компонентів:
  - Механізми контролю доступу, такі як брандмауер, що є першою лінією захисту обмежуючи трафік, який може потраапити в мережу або залишити її.
  - Механізми контролю входу, такі як NAC (Network Access Control) і NAP (Network Access Protection) забезпечують додатковий рівень безпеки, перевіряючи відповідність пристроїв, що підключаються до мережі, певним вимогам безпеки.
  - Криптографічні програми, такі як IPSec і SSL забезпечують захист даних під час їх передачі по мережі.
  - Біометричні методи, такі як розпізнавання мови або обличчя.
- Реактивний підхід доповнює превентивний і зосереджується на нейтралізації атак і загроз, які не вдалося запобігти. Він є важливим для захисту мережі від таких атак, як DoS і DDoS. Реактивні методи включають системи моніторингу безпеки, такі як IDS (Intrusion Detection

System), SIMS (Security Information Management System), TRS (Threat Response System) і IPS (Intrusion Prevention System).

- Ретроспективний підхід аналізує причини атак у мережі. До нього відносяться:
  - Механізми пошуку несправностей, такі як аналізатори протоколів і монітори трафіку.
  - Методи криміналістики безпеки, такі як CSIRT (Computer Security Incident Response Team) і CERT (Computer Emergency Response Team).
- Проактивний підхід до мережевої безпеки банківської системи складається з методів і прийомів, спрямованих на прийняття рішень щодо протидії майбутнім атакам на цільову мережу. Він включає декілька ключових компонентів:
  - Розвідку загроз і оцінку ризиків для визначення ймовірних майбутніх загроз.
  - Методи, що сприяють впровадженню превентивних заходів безпеки та протидії потенційним інцидентам.[12]

2 СИСТЕМИ ВИЯВЛЕННЯ ВТОРГЕНЬ

2.1 Важливість використання IDS в банківських системах

Використання систем виявлення вторгнень в банківських системах важливе для виявлення нетипового мережевого трафіку, для виявлення, наприклад, підозрілі спроби підключення до бази даних. Також такі системи дозволяють виявляти більшість мережевих атак, таких як DDoS атаки, Smurf атаки, udp флуд і т.д, оскільки сканують мережевий трафік в потоковому режимі. Такі системи також мають можливість виявити несанкціоновану передачу даних за допомогою мережі. Найчастіше в банківський системах використовуються два типи систем виявлення вторгнень, а саме: мережеві, котрі встановлюються на мережевому рівні і аналізують весь трафік, що через неї проходить. Вони ефективні для виявлення атак, що напрямлені на мережеву інфраструктуру. Також такі системи дозволяють отримати інформацію про сканування портів. Другим типом встановлюваних систем виявлення вторгнень є хостові систем, що працюють на рівні окремих комп’ютерів чи серверів, відстежуючи події на цих пристроях. Вони корисні для виявлення атак, що напрямлені на окремі пристрої, наприклад спроба експлуатації вразливостей операційної системи певних серверів.

Виявлення вторгнень – це процес моніторингу подій, що відбуваються в комп'ютерній системі або мережі, та їх аналіз на наявність ознак вторгнень, які включають спроби порушення конфіденційності, цілісності або доступності даних, або обхід захисних механізмів комп'ютера або мережі.. Вторгнення здійснюються зловмисниками, які отримують доступ до систем через Інтернет, а також авторизованими користувачами, які намагаються отримати незаконні привілеї або зловживають наданими їм привілеями.

Виявлення вторгнень дозволяє організаціям захистити свої системи від зростаючих загроз, що виникають у зв'язку зі збільшенням кількості мережевих підключень та залежності від інформаційних систем. З урахуванням рівня і характеру сучасних загроз мережевій безпеці, фахівці з

безпеки повинні розглядати не питання про те, чи слід використовувати виявлення вторгнень, а скоріше – які функції і можливості виявлення вторгнень використовувати. IDS отримали визнання як необхідний елемент інфраструктури безпеки кожної організації з кількох вагомих причин:

1) Зменшення ризику проблемної поведінки шляхом підвищення ймовірності виявлення та покарання осіб, які здійснюють атаки або зловживають системою.
2) Виявлення атак і порушень безпеки, які не можуть бути зупинені іншими заходами безпеки.
3) Виявлення та запобігання підготовчим діям до атак, які часто проявляються у вигляді мережевих сканувань та інших спроб доступу
4) Документування поточних загроз для організації.
5) Виконання функцій контролю якості у процесах розробки та адміністрування системи безпеки, особливо для великих і складних підприємств.
6) Надання корисної інформації про поточні вторгнення, що допомагає покращити діагностику, відновлення та усунення причинних факторів.[13]

Системи виявлення вторгнень (IDS) — це програмне або апаратне забезпечення, яке відстежує мережу на наявність незвичних або зловмисних дій і сповіщає про це адміністраторів

Системи виявлення вторгнень (IDS) використовуються для виявлення різних атак.

1) Відмова в обслуговуванні (DoS) – атака, під час якої зловмисник відмовляє користувачам у доступі до послуг, переповнюючи обчислювальні ресурси чи ресурси пам’яті помилковими запитами, щоб вони не могли обслуговувати законні запити.
2) Зондування – атака, метою якої є отримання конфігурації цільової машини чи мережі.

3) User-to-root (U2R): ці атаки спрямовані на отримання адміністративного доступу до комп’ютера, де зловмисник має доступ на рівні користувача.
4) Remote-to-local (R2L) – це атака, під час якої користувачі надсилають пакети через Інтернет на комп’ютери, до яких вони не мають доступу, щоб виявити вразливі місця та використати привілеї. уразливості та використовує дозволи, які локальний користувач має на комп’ютері як локальний користувач. [14]

2.2 Мережеві IDS

Більшість комерційних систем виявлення вторгнень є мережевими (Network Intrusion Detection System:). Ці IDS виявляють атаки шляхом перехоплення та аналізу мережевих пакетів. Відстежуючи мережевий сегмент або комутатор, одна мережева IDS може контролювати мережевий трафік, який впливає на кілька хостів, підключених до цього сегмента мережі, тим самим захищаючи їх.

Мережеві IDS зазвичай складаються з набору спеціалізованих датчиків або хостів, розміщених у різних точках мережі. Ці пристрої моніторять мережевий трафік, проводячи локальний аналіз і повідомляючи про атаки на центральну консоль управління. Оскільки датчики спеціалізуються лише на запуску IDS, їх легше захистити від атак. Багато з цих датчиків працюють в "прихованому" режимі, що ускладнює зловмисникам виявлення їхньої присутності та місцезнаходження. Прикладами є Snort, CiscoSecure IDS та Dragon Enterasys.

2.3 Хостові IDS

HIDS — це система виявлення вторгнень для окремого комп’ютера, яка контролює безпеку цієї системи або комп’ютера для запобігання внутрішнім і зовнішнім атакам.

Внутрішня атака — атака, при якій система визначає, яка програма може отримати доступ до якого ресурсу та чи є вразливі місця в системі безпеки. Наприклад, текстовий процесор раптово отримує доступ до бази даних паролів

системи та починає її змінювати. При зовнішній атаці, HIDS аналізує пакети, які потрапляють в систему (комп’ютер) і залишають їх на її інтерфейсах. HIDS реагує, реєструючи подію та повідомляючи про це уповноважений орган. У HIDS для захисту від загроз встановлюються такі програми, як антивірус, шпигунське програмне забезпечення тощо.

Прикладами HIDS є Tripwire з відкритим кодом, адміністратор подій довіри aelita, програмне забезпечення ELM3.0 TNT, GFI LAN Guard S.E.L.M. [13]

2.4 Порівняння Snort та Suricata

Suricata система виявлення вторгнень що була представлена у 2009 році базується на правилах, що дозволяє визначати унікальні характеристики мережевого трафіку за допомогою простих визначень. При виконанні цих умов система запускає оповіщення та блокує або перериває зв'язок згідно з правилами. Suricata підтримує багатопотоковість, що дозволяє обробляти більше правил на тому ж обладнанні, забезпечуючи ефективність у мережах з великим обсягом трафіку. Завдяки цьому, навіть на звичайному обладнанні можна досягти швидкостей до 10 Гбіт/с без втрати ефективності роботи з правилами. Крім того, Suricata підтримує хешування та вилучення файлів. Suricata може працювати як на фізичних серверах, так і на віртуальних машинах в AWS, використовуючи нову функцію дзеркального відображення трафіку. Suricata також підтримує скрипти Lua, що дозволяє створювати складну та детальну логіку для виявлення сигнатур і виявлення складних загроз. Проект та код Suricata належать і підтримуються Open Information Security Foundation (OISF).[14]

Snort – це система виявлення та запобігання мережевим вторгненням з відкритим вихідним кодом. Створена у 1998 році Мартином Роешем. Система Snort здатна аналізувати мережевий трафік у режимі реального часу, перевіряючи протоколи та виявляючи різні види атак. Snort переважно перевіряє пакети на відповідність користувацьким правилам. Правила для Snort можна писати на будь-якій мові, вони мають зручну структуру, що

дозволяє легко їх читати та змінювати. Наприклад, при атаці переповнення буфера Snort може виявити атаку, зіставляючи її з відомими шаблонами, і вжити заходів для її запобігання. У сигнатурних IDS системах шаблонні атаки легко виявляються, але нові атаки можуть залишатися непоміченими. Snort долає це обмеження завдяки аналізу трафіку в реальному часі. Кожного разу, коли в мережу надходить пакет, Snort перевіряє поведінку мережі, і якщо виявляється зниження продуктивності, Snort припиняє обробку пакету, відкидає його та зберігає інформацію про нього у базі даних сигнатур.[15]

Порівняння можна побачити у таблиці 2.1.

Таблиця 2.1 – Порівняння Snort та Suricata

Критерії	Snort	Suricata
Рік випуску	1998	2009
Підтримувані платформи	Linux, Windows, MacOS	Linux, Windows, MacOS
Вартість	Безкоштовно	Безкоштовно
Функціональність	IDS, IPS, аналіз трафіку	IDS, IPS, аналіз трафіку, хешування, вилучення файлів
Підтримка багатопоточності	Немає	Є
Продуктивність	Висока, але обмежена відсутністю багатопоточності	Висока, може обробляти до 10 Гбіт/с
Підтримка	Широка спільнота користувачів та розробників, активна підтримка з боку Cisco.	Активна та зростаюча спільнота, підтримка з боку Open Information Security Foundation (OISF).

3 ПОРІВНЯННЯ СИСТЕМ ВИЯВЛЕННЯ ВТОРГНЕНЬ

3.1 Віртуальні машини

Для виконання порівняння було використано VirtualBox.

VirtualBox – програмне забезпечення розробляється компанією Oracle та є потужним рішення для віртуалізації x86 та AMD64/Intel64. підтримує роботу на хостах Windows, Linux, Macintosh і Solaris, а також забезпечує сумісність з великою кількістю гостьових операційних систем, таких як Windows (від NT 4.0 до Windows 10), DOS/Windows 3.x, Linux (версії 2.4, 2.6, 3.x і 4.x), Solaris і OpenSolaris, OS/2 і OpenBSD.

У VirtualBox було створено дві віртуальні машини з операційними системами Linux. Далі буде наведений опис кожної з цих машин.

Перша віртуальна машина ubuntu:

- Операційна система: Ubuntu 24.04 LTS.
- Процесор: 2 ядра
- Оперативна пам'ять: 2048 MB
- Дисковий простір: 25 GB
- Відеопам’ять: 16 MB
- Адаптер 1: Intel PRO/1000 MT Desktop (мережевий мост, Qualcomm Atheros QCA9377 Wireless Network Adapter)
- IP-адреса: 192.168.1.4

Друга віртуальна машина (ubuntu1):

- Операційна система: Ubuntu 22.04 LTS.
- Процесор: 3 ядра
- Оперативна пам'ять: 3000 MB
- Дисковий простір: 25 GB
- Відеопам’ять: 24 MB
- Адаптер 1: Intel PRO/1000 MT Desktop (мережевий мост, Qualcomm Atheros QCA9377 Wireless Network Adapter)

- IP-адреса: 192.168.1.3

Для порівняння було обрано системи виявлення вторгнень, що описані другому розділі, а саме:

- Snort
- Suricata

Ці системи були встановлені на першу віртуальну машину(ubuntu), що можна побачити на рисунку 3.1.

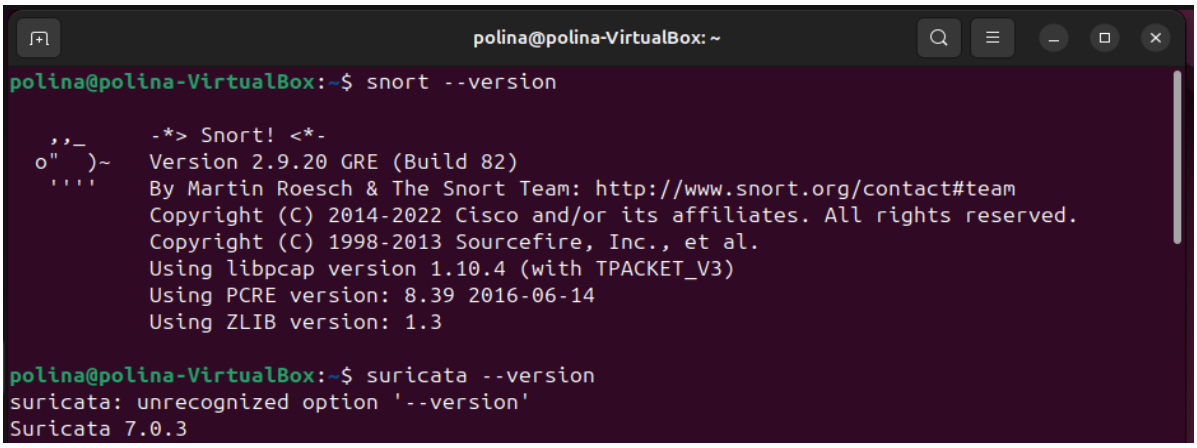

Рисунок 3.1 – Демонстрація версій встановлених програм

На другу віртуальну машину (ubuntu1) були встановлені утиліти nmap та hping3 для виконання атак. Встановлені версії цих утиліт можна побачити на рисунку 3.2.

Рисунок 3.2 – Демонстрація версій встановлених утиліт

3.2 Опис використовуваних утиліт для тестування

3.2.1 Утиліта hping3

hping3 — це мережевий інструмент, здатний надсилати спеціальні пакети TCP\IP і відображати цільові відповіді, як це робить інструмент ping із

відповідями ICMP. За замовчуванням протокол по якому працює hping3 — TCP, проте він також має можливість надсилання пакетів інших протоколів, таких як:

- -0 — режим, у якому hping3 надсилатиме Ip заголовок із даними, доданими в ручну за допомогою інших аргументів командного рядка, таких як --signature, --file та --ipproto.
- -1 — режим, у якому hping3 надсилає exo-запити ICMP.
- -2 — режим, у якому hping3 надсилатиме udp пакети на цільовий хост

За допомогою цієї утиліти можна протестувати поведінку IDS систем при простому надсиланні пакетів, та при флуд атаках, оскільки інструмент також має аргумент командного рядка —flood, що дозволяє надсилати пакети у режимі флуд атаки.[17]

3.2.2 Утиліта nmap

nmap — інструмент з відкритим кодом для дослідження мережі та аудиту безпеки. Його розроблено для швидкого сканування як великих мереж, тка і окремих хостів. Nmap використовує необроблені IP-пакети, щоб визначити, які хости доступні в мережі, які послуги пропонують ці хости, які операційні системи вони використовують, які типи пакетних фільтрів використовуються та десятки інших характеристик. Незважаючи на те, що Nmap зазвичай використовується для аудита безпеки, багато системних і мережевих адміністраторів вважають його корисним для рутинних задач.

Результатом використання Nmap є список просканованих цілей із додатковою інформацією про кожну залежно від використовуваних опцій. Ключовою серед цієї інформації є «цікава таблиця портів». У цій таблиці перелічено номер порту та протокол, назву служби та стан. Стан або open, filtered, closed або unfiltered. Опис стану портів:

- Open означає, що програма на цільовій машині прослуховує підключення/пакети на цьому порту.

- Filtered означає, що брандмауер, фільтр або інша мережева перешкода блокує порт, тому Nmap не може визначити, чи це open або closed.
- Closed порти не мають програми, які їх слухають, хоча вони можуть відкритися будь-коли.
- Порти класифікуються як unfiltered коли вони реагують на зонди Nmap, але Nmap не може визначити, чи вони відкриті чи закриті.
- Nmap повідомляє про комбінації станів open|filtered iclosed|filtered коли він не може визначити, який із двох станів описує порт.

Таблиця портів також може включати деталі версії програмного забезпечення, якщо надходить запит на визначення версії. Коли надходить запит на сканування IP-протоколу (-sO), Nmap надає інформацію про підтримувані IP-протоколи, а не порти прослуховування.[18]

3.3 Створення тестового трафіку

Перед початком тестування було одночасно запущено IDS Snort та Suricata на першій віртуальній машині за допомогою наступних команд.

Для запуску Snort:

```
sudo systemctl start snort
```

Для запуску Snort:

```
sudo systemctl start suricata
```

Для правильного порівняння IDS систем Snort та Suricata, необхідно створити тестовий мережевий трафік. Для цього спочатку була запущена друга віртуальна машина. Після цього за допомогою інструмента hping3 було створено симуляцію SYN-флуд атаки. Для цього було вказано спеціальний флаг -S, що означає те, що буде надсилатися TCP пакети з позначкою SYN, також вказана цільова ip-адресу, та порт. На рисунку 3.3 зображено виклик даного інструменту.

```
polina1@ubuntu1:~$ sudo hping3 --flood -S 192.168.1.4 -p 80
HPING 192.168.1.4 (enp0s3 192.168.1.4): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
```

Рисунок 3.3 — Запуск симуляції syn-flood атаки

Також, для дослідження було виконано надсилання UDP-пакетів, та ICMP пакетів. На рисунках 3.4 — 3.5 зображено надсилання таких типів пакетів.

```
polina1@ubuntu1:~$ sudo hping3 -2 192.168.1.4 -p 80 -c 100
HPING 192.168.1.4 (enp0s3 192.168.1.4): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from ip=192.168.1.4 name=UNKNOWN
status=0 port=1865 seq=0
ICMP Port Unreachable from ip=192.168.1.4 name=UNKNOWN
status=0 port=1866 seq=1
ICMP Port Unreachable from ip=192.168.1.4 name=UNKNOWN
status=0 port=1867 seq=2
ICMP Port Unreachable from ip=192.168.1.4 name=UNKNOWN
status=0 port=1868 seq=3
ICMP Port Unreachable from ip=192.168.1.4 name=UNKNOWN
status=0 port=1869 seq=4
^C
--- 192.168.1.4 hping statistic ---
5 packets transmitted, 5 packets received, 0% packet loss
```

Рисунок 3.4 — Надсилання udp пакетів

```
polina1@ubuntu1:~$ sudo hping3 -1 192.168.1.4 -c 100
HPING 192.168.1.4 (enp0s3 192.168.1.4): icmp mode set, 28 headers + 0 data bytes
len=46 ip=192.168.1.4 ttl=64 id=2615 icmp_seq=0 rtt=8.4 ms
len=46 ip=192.168.1.4 ttl=64 id=2712 icmp_seq=1 rtt=3.0 ms
len=46 ip=192.168.1.4 ttl=64 id=3468 icmp_seq=2 rtt=1.8 ms
len=46 ip=192.168.1.4 ttl=64 id=3666 icmp_seq=3 rtt=12.8 ms
len=46 ip=192.168.1.4 ttl=64 id=4439 icmp_seq=4 rtt=8.0 ms
^C
--- 192.168.1.4 hping statistic ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 1.8/6.8/12.8 ms
```

Рисунок 3.5 — Надсилання icmp пакетів

Також, системи IDS вміють розпізнавати сканування портів на хості, що може в деяких випадках інтерпретуватись як дослідження системи для подальшої атаки. Сканування портів цільового хосту зображено на рисунку 3.6.

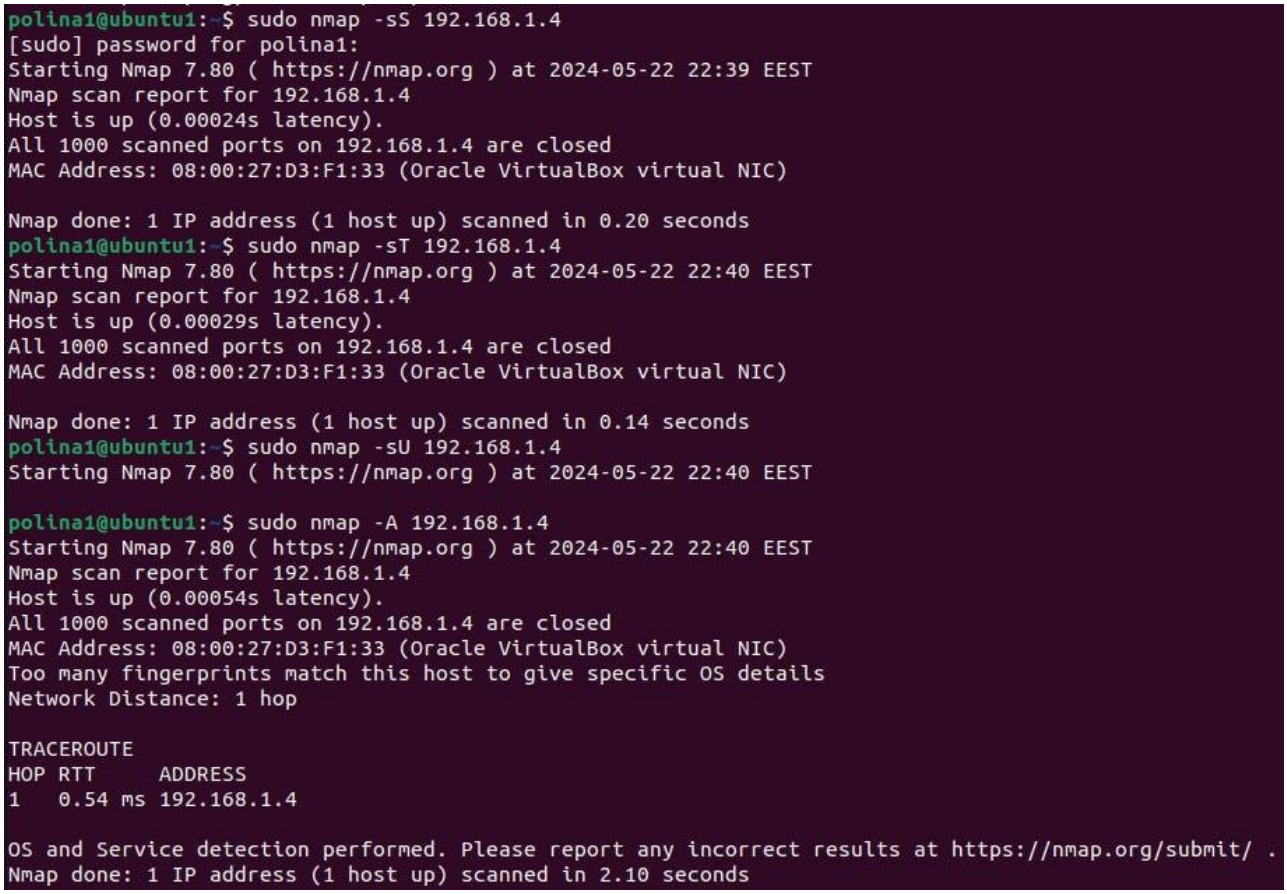

Рисунок 3.6 — Сканування портів утилітою nmap

3.4 Створення програмної реалізації

Програмна реалізація створювалась на мові програмування python.

Python — об'єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Вбудовані високорівневі структури даних разом із динамічною типізацією та динамічним зв'язуванням роблять її ідеальною для швидкої розробки додатків, а також для використання як мови сценаріїв або для інтеграції існуючих компонентів. [19]

Перед початком роботи log файл Snort для зручності був перенесений до домашньої папки та за допомогою команди

```
sudo snort -r snort.log.1716405355 > testsnort.txt
```

перетворений з бінарного формату до текстового.

У випадку з Suricata був використаний файл fast.log, що також був перенесений до домашньої папки з папки /var/log/suricata

Для початку необхідно імпортувати необхідні бібліотеки, а також визначити перелік змінних, необхідних для роботи застосунку. Для роботи потрібен буде лише імпорт однієї бібліотеки, ipaddress, що буде дозволяти з’ясувати, чи IP адреса у нас зараз, чи ні. Також далі треба описати деякий перелік змінних, а саме:

- ips — змінна, що міститиме в собі загальний список IP-адрес, що потрапили до лог файлу.
- protocols — змінна, що міститиме в собі загальний список протоколів, що потрапили до лог файлу.
- sent_ips_protocols_map_count — змінна, що містить в собі кількість надісланих протоколів згрупованих за ip адресами.
- reached_ips_protocols_map_count — змінна, що містить в собі кількість отриманих протоколів згрупованих за ip адресами.
- protocols_count — змінна, що містить в собі загальну кількість того чи іншого протоколу в лог файлі.
- ips_sent_count — кількість надісланих пакетів з ip адреси.
- ips_reached_count — кількість отриманих пакетів на ip адресу.
- last_sent_ip та last_reached_ip — останні прочитані ip адреси, ці змінні необхідні для коректного підрахунку пакетів в лог файлі snort.
- is_ip_got — допоміжна змінна, що вказує на те, чи отримали вже ip адресу з блоку тексту в лог файлі, чи ні.
- enabled_protocols — список протоколів, які будуть прочитані з логу.

Ініціалізація наведених змінних зображена на рисунку 3.7.

```
from ipaddress import ip_address

ips = []
protocols = []

sent_ips_protocols_map_count = dict([])
reached_ips_protocols_map_count = dict([])
protocols_count = dict([])
ips_sent_count = dict([])
ips_reached_count = dict([])

last_sent_ip = ''
last_reached_ip = ''
is_ip_got = False

enabled_protocols = [
    'TCP',
    'UDP',
    'ICMP'
]
```

Рисунок 3.7 — Ініціалізація допоміжних змінних

Далі необхідно реалізувати функцію, що буде скидати допоміжні змінні до початкового значення, що б використати їх знову. Реалізацію даної функції зображено на рисунку 3.8.

```
def reinit_global_vars():
    global ips
    global protocols
    global sent_ips_protocols_map_count
    global reached_ips_protocols_map_count
    global protocols_count
    global ips_sent_count
    global ips_reached_count
    global last_sent_ip
    global last_reached_ip
    global is_ip_got
    global enabled_protocols
    ips = []
    protocols = []

    sent_ips_protocols_map_count = dict([])
    reached_ips_protocols_map_count = dict([])
    protocols_count = dict([])
    ips_sent_count = dict([])
    ips_reached_count = dict([])

    last_sent_ip = ''
    last_reached_ip = ''
    is_ip_got = False
```

Рисунок 3.8 — Функція для скидання допоміжних змінних до початкового стану

Далі, перед реалізацією функцій парсингу даних з лог файлу, необхідно оглянути структуру даних файлів. На рисунках 3.9 та 3.10 видно структуру записів для TCP пакету, а на рисунках 3.11 та 3.12 — для ICMP пакетів.

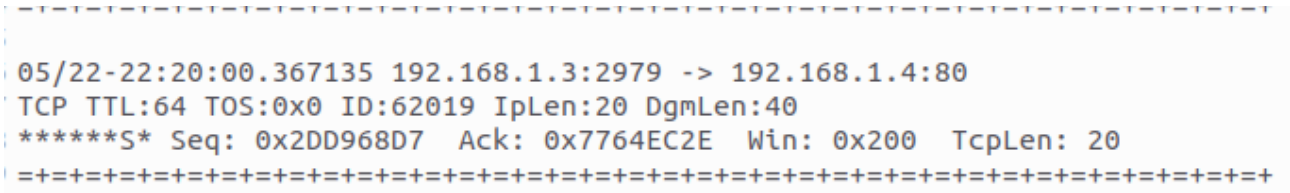

Рисунок 3.9 — Запис для TCP пакету в лог файлі snort

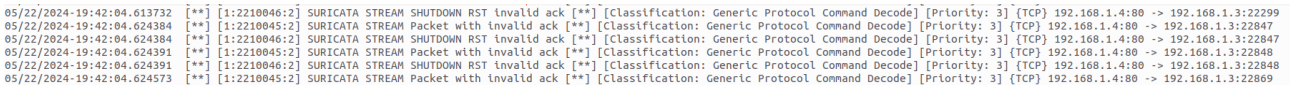

Рисунок 3.10 — Декілька записів для TCP пакету в лог файлі suricata

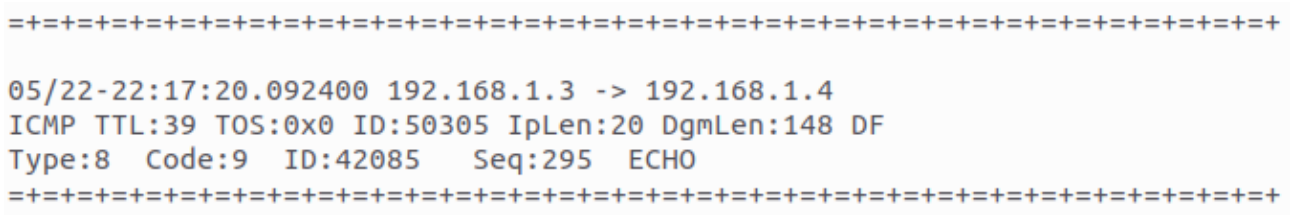

Рисунок 3.11 — Запис для ICMP пакету в лог файлі snort

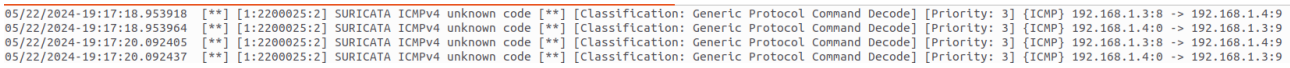

Рисунок 3.12 — Декілька записів для ICMP пакету в лог файлі suricata

Як видно з рисунків 3.9 – 3.12, застосунки мають різну структуру лог файлів, тож для кожного окремого інструменту буде реалізований власний набір функцій для парсингу ip адрес, протоколів та суміжних даних.

Для парсингу ip адрес в лог файлі snort, необхідно в поточному рядку спочатку розбити його по символу пробілу, що б знайти дві записані ip адреси в рядку. Далі необхідно окремо порахувати кількість записів до кожної ip адреси як у випадку надсилання пакетів, так і у випадку отримання пакетів. Релазація цієї функції зображена на рисунку 3.13.

```
def parse_ip(line):
    global last_sent_ip
    global last_reached_ip
    global is_ip_got
    is_source_ip = True
    arr = line.split(' ')
    for line_part in arr:
        try:
            ip_string = line_part.split(':')[0]
            ip = ip_address(ip_string)
            if is_source_ip:
                if ip_string in ips_sent_count:
                    ips_sent_count[ip_string] += 1
                else:
                    ips_sent_count[ip_string] = 1
                last_sent_ip = ip_string
                is_source_ip = False
            else:
                if ip_string in ips_reached_count:
                    ips_reached_count[ip_string] += 1
                else:
                    ips_reached_count[ip_string] = 1
                last_reached_ip = ip_string
            if ip_string not in ips:
                ips.append(ip_string)
            is_ip_got = True
        except ValueError:
            continue
        except TypeError:
            continue
```

Рисунок 3.13 — функція для парсингу ip адрес з лог файлів snort

Для лог файлів suricata дана функція буде дещо відрізнятися, оскільки він має трохи іншу структуру. Також в даному файлі на кожен надісланий TCP

пакет записується також і відповідь на нього, тож потрібно при підрахунку враховувати лише один запис на один TCP пакет. Тож для цього до функції додаються ще деякі перевірки. Реалізація цієї функції зображена на рисунку 3.14.

```
def parse_suricata_ip(line):
    allowed_line = False
    global last_sent_ip
    global last_reached_ip
    global is_ip_got
    is_source_ip = True
    arr = line.split(' ')
    for line_part in arr:
        try:
            if 'ICMP' in line_part or 'Packet' in arr:
                allowed_line = True
            if allowed_line:
                ip_string = line_part.split(':')[0]
                ip = ip_address(ip_string)
                if is_source_ip:
                    if ip_string in ips_sent_count:
                        ips_sent_count[ip_string] += 1
                    else:
                        ips_sent_count[ip_string] = 1
                    last_sent_ip = ip_string
                    is_source_ip = False
                else:
                    if ip_string in ips_reached_count:
                        ips_reached_count[ip_string] += 1
                    else:
                        ips_reached_count[ip_string] = 1
                    last_reached_ip = ip_string
                if ip_string not in ips:
                    ips.append(ip_string)
                is_ip_got = True
        except ValueError:
            continue
        except TypeError:
            continue
```

Рисунок 3.14 — Функція для парсингу ip адрес для лог файлу suricata

Для подальшої роботи також необхідно окремо парсити всі пакети, що наявні в файлах. Для реалізації цього функціоналу у файлі логів snort, необхідно розбивати рядок по символу пробілу, потім для того, аби не дублювати записи про деякі пакети, необхідно перевіряти, що це саме рядок з інформацією про пакет, і достатньо для цього перевірити наявність напису TTL в рядку, тобто позначки про час життя пакету. Після цього вже відбувається просте збереження протоколів до загального списку, а також підрахунок їх кількості. Реалізацію даної функції зображено на рисунку 3.15.

```
def parse_snort_protocol(line):
    global last_sent_ip
    global last_reached_ip
    arr = line.split(' ')
    protocol = arr[0]
    allowed_protocol = False
    for elem in arr:
        if 'TTL' in elem and protocol in enabled_protocols:
            allowed_protocol = True
            break
    if allowed_protocol:
        if protocol not in protocols:
            protocols.append(protocol)
        parse_snort_ips_protocols(protocol)
        if protocol in protocols_count:
            protocols_count[protocol] += 1
        else:
            protocols_count[protocol] = 1
```

Рисунок 3.15 — Функція для парсингу протоколів з файлу логів snort

Для парсингу протоколів з файлу логів suricata потрібно трохи модифікувати функцію, оскільки в файлі інший формат запису, тож додаються нові перевірки. Реалізація даної функції зображена на рисунку 3.16.

```
def parse_suricata_protocol(line):
    arr = line.split(' ')
    allowed_protocol = False
    protocol = ''
    for elem in arr:
        if '{' in elem:
            protocol_str = elem.split('{')[1].split('}')[0]
            if protocol_str == 'ICMP' or 'Packet' in arr:
                allowed_protocol = True
                protocol = protocol_str
                break
    if allowed_protocol:
        if protocol not in protocols:
            protocols.append(protocol)
        parse_suricata_ips_protocols(line, protocol)
        if protocol in protocols_count:
            protocols_count[protocol] += 1
        else:
            protocols_count[protocol] = 1
```

Рисунок 3.16 — Функція, для парсингу протоколів з файлу логів suricata

Далі, потрібно ще реалізувати дві функції, що будуть підраховувати, скільки і яких пакетів було надіслано або отримано тою чи іншою ip адресою.

Функції parse_suricata_ips_protocols і parse_snort_ips_protocols призначені для аналізу та обліку протоколів, які використовуються між IP-адресами, в логах Suricata і Snort відповідно. Обидві функції оновлюють глобальні словники, які зберігають інформацію про кількість пакетів, відправлених і отриманих від різних IP-адрес за кожним протоколом.

Функція з рисунку 3.17 Ця функція аналізує рядок лога Suricata і оновлює лічильники відправлених і отриманих пакетів за IP-адресами і протоколами.

Спочатку функція ділить рядок на частини. Далі обробляє кожну частину рядка, намагаючись знайти і розпізнати IP-адреси. Визначає, чи є IP-адреса джерелом або одержувачем. Оновлює відповідні словники sent_ips_protocols_map_count і reached_ips_protocols_map_count з урахуванням протоколу.

```
def parse_suricata_ips_protocols(line, protocol):
    is_source_ip = True
    arr = line.split(' ')
    for line_part in arr:
        try:
            ip_string = line_part.split(':')[0]
            ip = ip_address(ip_string)
            if is_source_ip:
                if ip_string not in sent_ips_protocols_map_count:
                    sent_ips_protocols_map_count[ip_string] = dict([])
                if ip_string in sent_ips_protocols_map_count:
                    if protocol in sent_ips_protocols_map_count[ip_string]:
                        sent_ips_protocols_map_count[ip_string][protocol] += 1
                    else:
                        sent_ips_protocols_map_count[ip_string][protocol] = 1
                is_source_ip = False
            else:
                if ip_string not in reached_ips_protocols_map_count:
                    reached_ips_protocols_map_count[ip_string] = dict([])
                if ip_string in reached_ips_protocols_map_count:
                    if protocol in reached_ips_protocols_map_count[ip_string]:
                        reached_ips_protocols_map_count[ip_string][protocol] += 1
                    else:
                        reached_ips_protocols_map_count[ip_string][protocol] = 1
        except ValueError:
            continue
        except TypeError:
            continue
```

Рисунок 3.17 — Отримання даних за ip адресами для файлу логів suricata

Функція з рисунку 3.18 виконує аналогічне завдання для логів Snort. Вона оновлює лічильники відправлених і отриманих пакетів за IP-адресами і протоколами.

Спочатку перевіряє, чи встановлені останні відправлена й отримана IP-адреси (last_sent_ip і last_reached_ip).

Далі оновлює словники sent_ips_protocols_map_count і reached_ips_protocols_map_count з урахуванням протоколу

```
def parse_snort_ips_protocols(protocol):
    if not last_sent_ip or not last_sent_ip:
        return False
    if last_sent_ip not in sent_ips_protocols_map_count:
        sent_ips_protocols_map_count[last_sent_ip] = dict([])
    if last_reached_ip not in reached_ips_protocols_map_count:
        reached_ips_protocols_map_count[last_reached_ip] = dict([])
    if last_sent_ip in sent_ips_protocols_map_count:
        if protocol in sent_ips_protocols_map_count[last_sent_ip]:
            sent_ips_protocols_map_count[last_sent_ip][protocol] += 1
        else:
            sent_ips_protocols_map_count[last_sent_ip][protocol] = 1

    if last_reached_ip in reached_ips_protocols_map_count:
        if protocol in reached_ips_protocols_map_count[last_reached_ip]:
            reached_ips_protocols_map_count[last_reached_ip][protocol] += 1
        else:
            reached_ips_protocols_map_count[last_reached_ip][protocol] = 1
```

Рисунок 3.18 — Отримання даних за ip адресами для файлу логів snort

На рисунку 3.19 зображена функція, що призначена для читання та аналізу логів Snort. Вона обробляє лог-файл, витягує IP-адреси, протоколи і підраховує різні статистики.

Послідовність виконання функції:

- Відкриває лог-файл Snort (testsnort.txt) для читання.
- Читає файл по рядках.
- Для кожного рядка викликає функцію parse_ip для вилучення IP-адрес. Викликає функцію parse_snort_protocol для аналізу протоколів.
- Після обробки всіх рядків, виводить зібрані дані:
  - Усі IP-адреси (ips).

- Кількість відправлених пакетів по IP (ips_sent_count).
- Кількість отриманих пакетів по IP (ips_reached_count).
- Усі виявлені протоколи (protocols).
- Кількість пакетів за протоколами (protocols_count).Кількість відправлених пакетів за протоколами для кожного IP (sent_ips_protocols_map_count).
- Кількість отриманих пакетів за протоколами для кожного IP (reached_ips_protocols_map_count).

```
def parse_snort_log():
    snort_log = open('testsnort.txt', 'r')
    global last_sent_ip
    global last_reached_ip
    global is_ip_got

    while True:
        is_ip_got = False
        line = snort_log.readline()
        if not line:
            break
        line_str = line.strip()
        if len(line_str) == 0:
            continue
        parse_ip(line_str)
        if is_ip_got:
            continue
        parse_snort_protocol(line_str)
        last_sent_ip = ''
        last_reached_ip = ''
    snort_log.close()
    print('ALL IPs: ', ips)
    print('COUNT OF SENT PACKAGES BY IP: ', ips_sent_count)
    print('COUNT OF REACHED PACKAGES BY IP: ', ips_reached_count)
    print('ALL DETECTED PROTOCOLS: ', protocols)
    print('PROTOCOLS COUNT BY PROTOCOL NAME: ', protocols_count)
    print('SENT PROTOCOLS COUNT BY IP: ', sent_ips_protocols_map_count)
    print('REACHED PROTOCOLS COUNT BY IP: ', reached_ips_protocols_map_count)
```

Рисунок 3.19 – Функція для читання та аналізу логів Snort

Функція, що зображена на рисунку 3.20 призначена для читання й аналізу логів Suricata. Вона виконує аналогічні завдання, що і parse_snort_log, але з урахуванням особливостей логів Suricata.

Послідовність виконання функції:

- Відкриває лог-файл Suricata (fast.log) для читання.
- Читає файл порядково.
- Для кожного рядка викликає функцію parse_suricata_ip для вилучення IP-адрес. Викликає функцію parse_suricata_protocol для аналізу протоколів.
- Після обробки всіх рядків, виводить зібрані дані аналогічно функції parse_snort_log.

```
def parse_suricata_log():
    suricata_log = open('fast.log', 'r')
    while True:
        line = suricata_log.readline()
        if not line:
            break
        line_str = line.strip()
        if len(line_str) == 0:
            continue
        parse_suricata_ip(line_str)
        parse_suricata_protocol(line_str)
    print('ALL IPs: ', ips)
    print('COUNT OF SENT PACKAGES BY IP: ', ips_sent_count)
    print('COUNT OF REACHED PACKAGES BY IP: ', ips_reached_count)
    print('ALL DETECTED PROTOCOLS: ', protocols)
    print('PROTOCOLS COUNT BY PROTOCOL NAME: ', protocols_count)
    print('SENT PROTOCOLS COUNT BY IP: ', sent_ips_protocols_map_count)
    print('REACHED PROTOCOLS COUNT BY IP: ', reached_ips_protocols_map_count)

    suricata_log.close()
```

Рисунок 3.20 – Функція для читання та аналізу логів Suricata

У функції main викликаються функції, що зображені на рисунках 3.19 та 3.20. Весь код програми можна побачити у Додатку А.

3.5 Огляд результатів виконання програми

В результаті виконання програми буде виведено інформацію, що зображена на рисунку 3.21.

```
SNORT INFO:
ALL IPs:  ['192.168.1.3', '192.168.1.4']
COUNT OF SENT PACKAGES BY IP:  {'192.168.1.3': 2422, '192.168.1.4': 29}
COUNT OF REACHED PACKAGES BY IP:  {'192.168.1.4': 2422, '192.168.1.3': 29}
ALL DETECTED PROTOCOLS:  ['TCP', 'ICMP', 'UDP']
PROTOCOLS COUNT BY PROTOCOL NAME:  {'TCP': 2393, 'ICMP': 38, 'UDP': 20}
SENT PROTOCOLS COUNT BY IP:  {'192.168.1.3': {'TCP': 2393, 'UDP': 20, 'ICMP': 9}, '192.168.1.4': {'ICMP': 29}}
REACHED PROTOCOLS COUNT BY IP:  {'192.168.1.4': {'TCP': 2393, 'UDP': 20, 'ICMP': 9}, '192.168.1.3': {'ICMP': 29}}
------------------
SURICATA INFO:
ALL IPs:  ['192.168.1.3', '192.168.1.4']
COUNT OF SENT PACKAGES BY IP:  {'192.168.1.3': 4, '192.168.1.4': 3140}
COUNT OF REACHED PACKAGES BY IP:  {'192.168.1.4': 4, '192.168.1.3': 3140}
ALL DETECTED PROTOCOLS:  ['ICMP', 'TCP']
PROTOCOLS COUNT BY PROTOCOL NAME:  {'ICMP': 8, 'TCP': 3136}
SENT PROTOCOLS COUNT BY IP:  {'192.168.1.3': {'ICMP': 4}, '192.168.1.4': {'ICMP': 4, 'TCP': 3136}}
REACHED PROTOCOLS COUNT BY IP:  {'192.168.1.4': {'ICMP': 4}, '192.168.1.3': {'ICMP': 4, 'TCP': 3136}}
```

Рисунок 3.21 — Результат виконання програми

З даного результату можна наглядно побачити наступні дані:

- Загальний список ip адрес, що брали участь в обміні даними в файлах логів.
- Кількість надісланих пакетів з кожної з наявних ip адрес
- Кількість отриманих пакетів кожною ip адресою.
- Кількість всіх протоколів, що були помічені в файлах логів.
- Кількість згадувань кожного з протоколів
- Кількість надісланих кожного з протоколів з кожної ip адреси.
- Кількість отриманих кожного протоколу на кожну з ip адрес.

З Рисунку 3.21 видно, що дані дещо відрізняються. По перше, Suricata не помітила обмін UDP пакетами, що в свою чергу також призвело до зменшення помічених ICMP пакетів, оскільки при кожному з обмінів UDP пакетами також відбувався обмін ICMP пакетами. Також, Suricata помітила більшу кількість TCP пакетів, оскільки працювала трохи більший час.

На основі отриманих даних під час виконання програми було створено графіки, що зображені на рисунках 3.22 та 3.23.

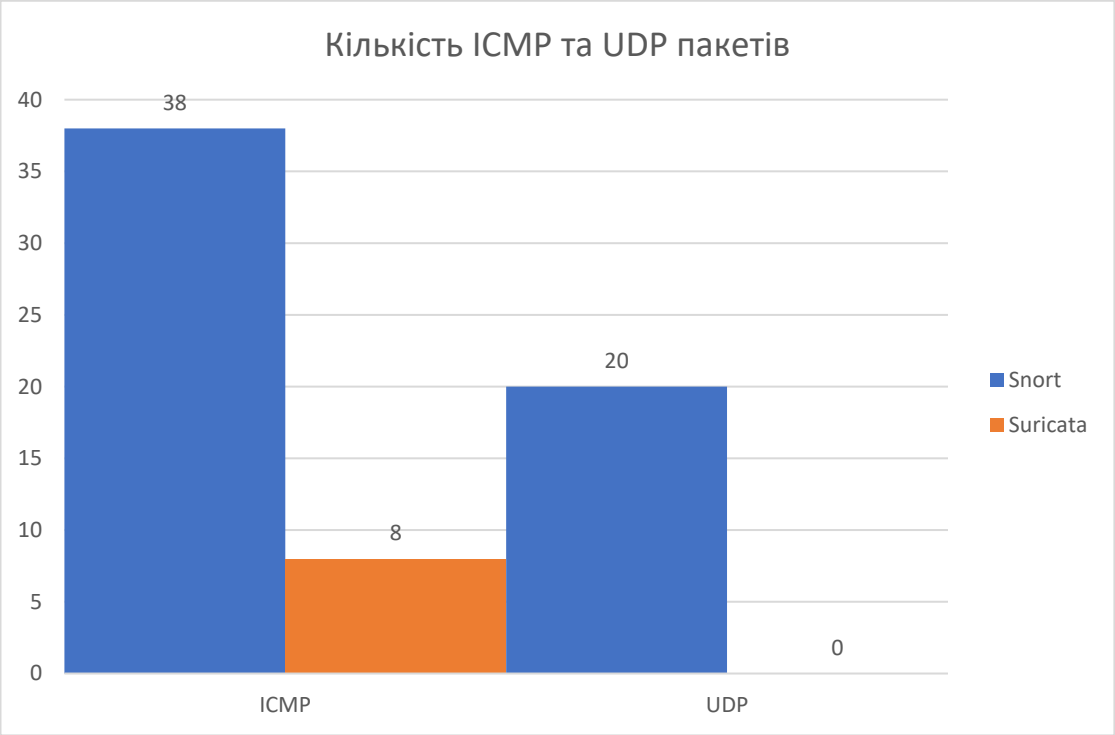

Рисунок 3.22 – Кількість розпізнаних ICMP та UDP пакетів

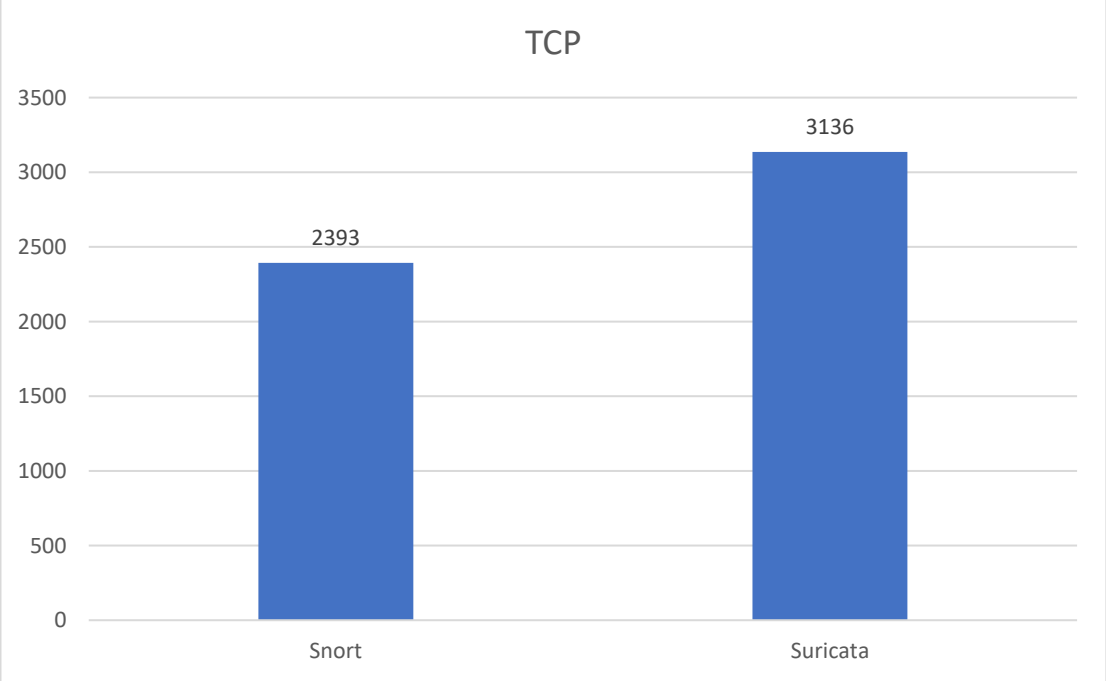

Рисунок 3.23 – Кількість розпізнаних TCP пакетів

З рисунку 3.22 можна побачити, що кількість зафіксованих ICMP пакетів виявлених Snort є значно більшою ніж виявлених Suricata. А пакети UDP не виявлені Suricata зовсім. Однак у випадку з TCP пакетами навпаки більшу кількість виявила IDS Suricata

ВИСНОВОК

У дипломній роботі розглядаються загрози для банківських систем, що можуть призвести до значних фінансових втрат, шкоди репутації та розкриття конфіденційної інформації. Для ефективної боротьби з цими загрозами необхідний комплексний підхід, який включає технічний захист і організаційні заходи, такі як навчання співробітників і клієнтів.

Також детально розглянуто системи виявлення вторгнень (IDS) як ключовий елемент забезпечення кібербезпеки банківських систем. Представлено теоретичне порівняння двох популярних IDS: Snort і Suricata. Обидві системи мають свої переваги: Snort відома своєю простотою та широкою підтримкою, тоді як Suricata характеризується високою продуктивністю та здатністю обробляти багатопотокові дані.

Практичне тестування IDS було проведено на віртуальних машинах, симулюючи різні типи атак, такі як атаки syn-flood, надсилання пакетів і сканування портів. Результати тестування показують, що обидві системи ефективні у виявленні загроз, але кожна має свої особливості в боротьбі з певними типами атак.

У цій роботі також розроблено програму для аналізу та порівняння систем виявлення вторгнень, яка може автоматизувати процес оцінки ефективності IDS. Програма містить інструменти для збору та аналізу лог-файлів, що значно спрощує процес моніторингу та виявлення загроз у реальному часі.

Порівняння систем виявлення вторгнень та програмна реалізація аналізу лог-файлів можуть бути важливими етапами у вдосконаленні системи безпеки інформаційних ресурсів у банківській системі. Запропоновані підходи та рішення можуть бути використані для підвищення рівня безпеки в банківському секторі та спрощення аналізу даних отриманих в результаті роботи IDS.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. All About Phishing Scams & Prevention: What You Need to Know. Режим доступу: https://www.kaspersky.com/resource-center/preemptive-safety/phishing-prevention-tips Дата звернення: 01.04.2024
2. ФЕЙК: Приватбанк, Монобанк та Ощадбанк нарахують кожному користувачу 3000 грн. Рисунок. Режим доступу: https://voxukraine.org/fejk-pryvatbank-monobank-ta-oshhadbank-narahuyut-kozhnomu-korystuvachu-3000-grn
3. International Symposium on Electronic Imaging. The strange world of keyloggers - an overview, Part I Дата звернення: 01.04.2024
4. O. Izmailova, H. Krasovska, K. Krasovska, V. Zaslavskyi. Assessing the Variety of Expected Losses upon the Materialisation of Threats to Banking Information Systems. 2020 p. C. 93-94. Дата звернення: 15.03.2024
5. Інформаційні технології. Методи захисту. Системи керування інформаційною безпекою. Огляд і словник термінів. ДСТУ ISO/IEC 27000:2019 – [Чинний від 2019-11-01]. Дата звернення: 17.03.2024
6. Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги. ДСТУ ISO/IEC 27001:2015 – [Чинний від 2017-01-01]. Дата звернення: 17.03.2024
7. Інформаційні технології. Методи захисту. Звід практик щодо заходів інформаційної безпеки. ДСТУ ISO/IEC 27002:2015 – [Чинний від 2017-01-01]. Дата звернення: 17.03.2024
8. Постанова Правління НБУ №95 «Про затвердження Положення про організацію заходів із забезпечення інформаційної безпеки в банківській системі України – [Чинна від 2017-09-28] Режим доступу: https://zakon.rada.gov.ua/laws/show/v0095500-17#Text Дата звернення: 20.03.2024
9. Постанова Правління НБУ №4 «Про затвердження Положення про здійснення контролю за дотриманням банками вимог законодавства з

питань інформаційної безпеки, кіберзахисту та електронних довірчих послуг» – [Чинна від 2021-01-16] Режим доступу: https://zakon.rada.gov.ua/laws/show/v0004500-21#Text Дата звернення: 24.03.2024

10. Постанова Правління НБУ №178 «Про затвердження Положення про організацію кіберзахисту в банківській системі України та внесення змін до Положення про визначення об'єктів критичної інфраструктури в банківській системі України» – [Чинна від 2022-08.12] Режим доступу: https://bank.gov.ua/ua/legislation/Resolution_12082022_178 Дата звернення: 20.03.2024
11. International Journal of Communication Networks and Information Security. Mohammed N. Alenezi, Haneen Alabdulrazzaq, Nada Q. Mohammad. Symmetric Encryption Algorithms: Review and Evaluation study. С 256. Дата звернення: 15.04.2024
12. Платформа Checkpoint. Режим доступу: https://checkpointpublic.litmoseu.com Дата звернення: 25.04.2024
13. Rebecca Bace. Intrusion Detection Systems. / Rebecca Bace, Peter Mell – NIST Special Publication on Intrusion Detection Systems. С 5-17. Дата звернення: 18.04.2024
14. Abhishek Pharate. Classification of Intrusion Detection System. / Abhishek Pharate, Harsha Bhat, Vaibhav Shilimkar, Nalini Mhetre – International Journal of Computer Applications. 2015р. Дата звернення: 19.04.2024
15. IDS Suricata. Режим доступу: https://suricata.io/ Дата звернення: 14.05.2024
16. IDS Snort. Режим доступу: https://www.snort.org/ Дата звернення: 14.05.2024
17. Hping3. Tool documentation. Режим доступу: https://www.kali.org/tools/hping3/ Дата звернення: 19.05.2024
18. Nmap. Режим доступу: https://nmap.org/ Дата звернення: 19.05.2024

19."A Python Book: Beginning Python, Advanced Python, and Python Exercises". 2012р. Режим доступу: https://docs.huihoo.com/python/2.4.4/ref/front.html Дата звернення: 19.05.2024

ДОДАТОК А

Лістинг А.1 – Код функції для зчитування інформації з лог-файлів

```
from ipaddress import ip_address

ips = []
protocols = []

sent_ips_protocols_map_count = dict([])
reached_ips_protocols_map_count = dict([])
protocols_count = dict([])
ips_sent_count = dict([])
ips_reached_count = dict([])

last_sent_ip = ''
last_reached_ip = ''
is_ip_got = False

enabled_protocols = [
    'TCP',
    'UDP',
    'ICMP'
]


def reinit_global_vars():
    global ips
    global protocols
    global sent_ips_protocols_map_count
    global reached_ips_protocols_map_count
    global protocols_count
    global ips_sent_count
    global ips_reached_count
    global last_sent_ip
    global last_reached_ip
    global is_ip_got
    global enabled_protocols
    ips = []
    protocols = []

    sent_ips_protocols_map_count = dict([])
    reached_ips_protocols_map_count = dict([])
    protocols_count = dict([])
    ips_sent_count = dict([])
    ips_reached_count = dict([])

    last_sent_ip = ''
    last_reached_ip = ''
    is_ip_got = False
```

```
def parse_snort_protocol(line):
    global last_sent_ip
    global last_reached_ip
    arr = line.split(' ')
    protocol = arr[0]
    allowed_protocol = False
    for elem in arr:
        if 'TTL' in elem and protocol in enabled_protocols:
            allowed_protocol = True
            break
    if allowed_protocol:
        if protocol not in protocols:
            protocols.append(protocol)
        parse_snort_ips_protocols(protocol)
        if protocol in protocols_count:
            protocols_count[protocol] += 1
        else:
            protocols_count[protocol] = 1


def parse_ip(line):
    global last_sent_ip
    global last_reached_ip
    global is_ip_got
    is_source_ip = True
    arr = line.split(' ')
    for line_part in arr:
        try:
            ip_string = line_part.split(':')[0]
            ip = ip_address(ip_string)
            if is_source_ip:
                if ip_string in ips_sent_count:
                    ips_sent_count[ip_string] += 1
                else:
                    ips_sent_count[ip_string] = 1
                last_sent_ip = ip_string
                is_source_ip = False
            else:
                if ip_string in ips_reached_count:
                    ips_reached_count[ip_string] += 1
                else:
                    ips_reached_count[ip_string] = 1
                last_reached_ip = ip_string
            if ip_string not in ips:
                ips.append(ip_string)
            is_ip_got = True
        except ValueError:
            continue
        except TypeError:
            continue
```

```
def parse_suricata_ip(line):
    allowed_line = False
    global last_sent_ip
    global last_reached_ip
    global is_ip_got
    is_source_ip = True
    arr = line.split(' ')
    for line_part in arr:
        try:
            if 'ICMP' in line_part or 'Packet' in arr:
                allowed_line = True
            if allowed_line:
                ip_string = line_part.split(':')[0]
                ip = ip_address(ip_string)
                if is_source_ip:
                    if ip_string in ips_sent_count:
                        ips_sent_count[ip_string] += 1
                    else:
                        ips_sent_count[ip_string] = 1
                    last_sent_ip = ip_string
                    is_source_ip = False
                else:
                    if ip_string in ips_reached_count:
                        ips_reached_count[ip_string] += 1
                    else:
                        ips_reached_count[ip_string] = 1
                    last_reached_ip = ip_string
                if ip_string not in ips:
                    ips.append(ip_string)
                is_ip_got = True
        except ValueError:
            continue
        except TypeError:
            continue


def parse_snort_ips_protocols(protocol):
    if not last_sent_ip or not last_sent_ip:
        return False
    if last_sent_ip not in sent_ips_protocols_map_count:
        sent_ips_protocols_map_count[last_sent_ip] = dict([])
    if last_reached_ip not in reached_ips_protocols_map_count:
        reached_ips_protocols_map_count[last_reached_ip] =
dict([])
    if last_sent_ip in sent_ips_protocols_map_count:
        if protocol in
sent_ips_protocols_map_count[last_sent_ip]:
            sent_ips_protocols_map_count[last_sent_ip][protocol]
+= 1
        else:
            sent_ips_protocols_map_count[last_sent_ip][protocol]
= 1
```

```
    if last_reached_ip in reached_ips_protocols_map_count:
        if protocol in
reached_ips_protocols_map_count[last_reached_ip]:

reached_ips_protocols_map_count[last_reached_ip][protocol] += 1
        else:

reached_ips_protocols_map_count[last_reached_ip][protocol] = 1


def parse_snort_log():
    snort_log = open('testsnort.txt', 'r')
    global last_sent_ip
    global last_reached_ip
    global is_ip_got

    while True:
        is_ip_got = False
        line = snort_log.readline()
        if not line:
            break
        line_str = line.strip()
        if len(line_str) == 0:
            continue
        parse_ip(line_str)
        if is_ip_got:
            continue
        parse_snort_protocol(line_str)
        last_sent_ip = ''
        last_reached_ip = ''
    snort_log.close()
    print('ALL IPs: ', ips)
    print('COUNT OF SENT PACKAGES BY IP: ', ips_sent_count)
    print('COUNT OF REACHED PACKAGES BY IP: ',
ips_reached_count)
    print('ALL DETECTED PROTOCOLS: ', protocols)
    print('PROTOCOLS COUNT BY PROTOCOL NAME: ', protocols_count)
    print('SENT PROTOCOLS COUNT BY IP: ',
sent_ips_protocols_map_count)
    print('REACHED PROTOCOLS COUNT BY IP: ',
reached_ips_protocols_map_count)


def parse_suricata_ips_protocols(line, protocol):
    is_source_ip = True
    arr = line.split(' ')
    for line_part in arr:
        try:
            ip_string = line_part.split(':')[0]
            ip = ip_address(ip_string)
            if is_source_ip:
```

```
                if ip_string not in
sent_ips_protocols_map_count:
                    sent_ips_protocols_map_count[ip_string] =
dict([])
                if ip_string in sent_ips_protocols_map_count:
                    if protocol in
sent_ips_protocols_map_count[ip_string]:

sent_ips_protocols_map_count[ip_string][protocol] += 1
                    else:

sent_ips_protocols_map_count[ip_string][protocol] = 1
                is_source_ip = False
            else:
                if ip_string not in
reached_ips_protocols_map_count:
                    reached_ips_protocols_map_count[ip_string] =
dict([])
                if ip_string in reached_ips_protocols_map_count:
                    if protocol in
reached_ips_protocols_map_count[ip_string]:

reached_ips_protocols_map_count[ip_string][protocol] += 1
                    else:

reached_ips_protocols_map_count[ip_string][protocol] = 1
        except ValueError:
            continue
        except TypeError:
            continue


def parse_suricata_protocol(line):
    arr = line.split(' ')
    allowed_protocol = False
    protocol = ''
    for elem in arr:
        if '{' in elem:
            protocol_str = elem.split('{')[1].split('}')[0]
            if protocol_str == 'ICMP' or 'Packet' in arr:
                allowed_protocol = True
                protocol = protocol_str
                break
    if allowed_protocol:
        if protocol not in protocols:
            protocols.append(protocol)
        parse_suricata_ips_protocols(line, protocol)
        if protocol in protocols_count:
            protocols_count[protocol] += 1
        else:
            protocols_count[protocol] = 1
```

```
def parse_suricata_log():
    suricata_log = open('fast.log', 'r')
    while True:
        line = suricata_log.readline()
        if not line:
            break
        line_str = line.strip()
        if len(line_str) == 0:
            continue
        parse_suricata_ip(line_str)
        parse_suricata_protocol(line_str)
    print('ALL IPs: ', ips)
    print('COUNT OF SENT PACKAGES BY IP: ', ips_sent_count)
    print('COUNT OF REACHED PACKAGES BY IP: ',
ips_reached_count)
    print('ALL DETECTED PROTOCOLS: ', protocols)
    print('PROTOCOLS COUNT BY PROTOCOL NAME: ', protocols_count)
    print('SENT PROTOCOLS COUNT BY IP: ',
sent_ips_protocols_map_count)
    print('REACHED PROTOCOLS COUNT BY IP: ',
reached_ips_protocols_map_count)

    suricata_log.close()


def main():
    print('SNORT INFO:')
    parse_snort_log()
    print('------------------')
    reinit_global_vars()
    print('SURICATA INFO:')
    parse_suricata_log()


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