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

До захисту допущено
Завідувач кафедри

_____ **Дмитро ЛАНДЕ**
(підпис)

«_____»_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра

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

на тему: Виявлення присутності та перебування APT-груп в Windows системах на основі їх атрибутів

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФБ-91
(шифр групи)

Снігур Антон Юрійович (підпис)
(прізвище, ім'я, по батькові)

Керівник к.т.н, доцент кафедри ІБ, Гальчинський Леонід Юрійович (підпис)
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові)

Рецензент к.т.н, доцент кафедри ММЗІ, Ніщенко Ірини Іванівни (підпис)
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові)

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

Київ – 2023 року

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

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

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Дмитро ЛАНДЕ
(підпис)

«___»_____2023 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти

Снігур Антон Юрійович
(прізвище, ім'я, по батькові)

1. Тема роботи Виявлення присутності та перебування APT-груп в Windows системах на основі їх атрибутів

керівник роботи Гальчинський Леонід Юрійович, к.т.н, доцент кафедри ІБ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «26» травня 2023 р. № 2028-С

2. Термін подання здобувачем вищої освіти роботи 14 червня 2023 р.

3. Вихідні дані до роботи Згідно з завданням дипломної роботи

4. Зміст роботи

Що таке APT загрози і тактика persistence, що вони використовують. Атрибуція загроз на основі індикаторів; Розробка системи виявлення присутності та перебування APT груп в windows системах на основі їх атрибутів; Вдосконалення системи за рахунок впровадження машинного навчання.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

Презентація

6. Дата видачі завдання 1 листопада 2022 року

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Формування теми дипломної роботи	01.10.2022– 31.10.2022	Виконано
2	Узгодження теми з дипломним керівником та отримання відповідного завдання	01.11.2022 – 7.11.2022	Виконано
3	Опрацювання інформаційних джерел для вирішення проблеми дипломної роботи, формування мети, об’єкту дослідження	8.11.2022 – 15.12.2022	Виконано
4	Детальний аналіз задачі та шляхів її вирішення	16.12.2022 – 31.01.2023	Виконано
5	Дослідження систем взаємодії та розбиття задачі на підзадачі та підсистеми	1.02.2023 – 7.02.2023	Виконано
6	Написання першого розділу	8.02.2023 – 28.02.2023	Виконано
7	Аналіз індикаторів компрометації, шляхів їх знаходження	1.03.2023 – 15.03.2023	Виконано
8	Робота над другим розділом	16.03.2023 – 31.03.2023	Виконано
9	Встановлення та програмування клієнт-серверної архітектури	1.04.2023 – 7.04.2023	Виконано
10	Продовження роботи над другим розділом та програмування алогритмів виявлення індикаторів	8.04.2023 – 30.04.2023	Виконано
11	Покращення продукту в контексті взаємодії, продуктивності та ефективності.	1.05.2023 – 28.05.2023	Виконано
12	Робота над третім розділом, дослідження перспективності та подальшої інтеграції.	29.05.2023 – 7.06.2023	Виконано

Здобувач вищої освіти _____ Антон СНІГУР
(підпис) **(Власне ім’я, ПРІЗВИЩЕ)**

Керівник роботи _____ Леонід ГАЛЬЧИНСЬКИЙ
(підпис) **(Власне ім’я, ПРІЗВИЩЕ)**

РЕФЕРАТ

Обсяг роботи 101 сторінок, 54 ілюстрацій, 26 джерел літератури

Мета дипломної роботи: дослідження присутності APT загроз в Windows та розробка системи по впровадженню методів їх виявлення на основі індикаторів компрометації.

Об’єкт дослідження: APT-загрози та їх присутність в Windows системах, детектування на основі індикаторів.

Предмет дослідження: методи пошуку та ідентифікації загроз на основі індикаторів компрометації

Наукова новизна отриманих результатів: полягає в розробці власної системи на основі клієнт-серверної моделі для виявлення загроз на основі статичних, динамічних та мережевих індикаторів й зручна взаємодія з системою через Телеграм бота.

Практичне значення: Результат даної роботи може бути використаний як на звичайних робочих станціях рядових співробітників для швидкого виявлення загроз так і спеціалістами в галузі кібербезпеки для автоматизації рутинних дій та пришвидшення реагування на наявні індикатори.

Ключові слова: APT, persistence, IOC, BIOC

ABSTRACT

The volume of work is 101 pages, 54 illustrations, 26 sources of literature.

The purpose of the thesis: study of the presence of APT threats in Windows and development of a system for implementing methods of their detection based on indicators of compromise.

Object of research: APT threats and their persistence in Windows systems, detection based on indicators.

The research subject: methods of searching and identifying threats based on indicators of compromise

The scientific novelty of the obtained results: the development of our own system based on the client-server model of communication for detecting threats based on static, dynamic and network indicators and user-friendly interaction with the system through the Telegram bot.

Practical significance: the result of this work can be used both on ordinary workstations of ordinary employees for quick detection of threats and by specialists in the field of cyber security to automate routine actions and speed up response to available indicators.

Keywords: APT, persistence, IOC, BIOC

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 7
Вступ 8
1 Що таке APT загрози і тактика persistence, що вони використовують. Атрибуція загроз на основі індикаторів 10
1.1 Що таке APT загрози? 10
1.2 Цикл проведення APT атаки 11
1.3 Persistence 12
1.4 Атрибуція загроз та IOC 13
1.5 Інструменти для виявлення загроз на основі їх індикаторів 17
Висновки до розділу 1 21
2 Розробка системи виявлення присутності та перебування APT груп в Windows системах на основі їх атрибутів 22
2.1 Планування системи та налаштування середовища 22
2.2 Визначення шуканих індикаторів та підключення клієнта 25
2.3 Детектування відомих загроз на основі статичних атрибутів (IOC) 28
2.4 BIOC правила 37
Висновки до розділу 2 42
3 Вдосконалення системи за рахунок впровадження машинного навчання 43
3.1 Опис проблеми та її вирішення з допомогою моделі машинного навчання 43
3.2 Пошук датасета та вибір моделі машинного навчання 44
3.3 Обробка датасета та підготовка для роботи 46
3.4 Перехоплення трафіка на машині клієнта 52
Висновки до розділу 3 61
Висновки 63
Перелік джерел посилань 65
Додаток А 68

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Інстанс (Instance) – віртуальний сервер або обчислювальний ресурс розташований у хмарі

ШПЗ – шкідливе програмне забезпечення

Metasploit - фреймворк для тестування на проникнення та експлуатації

Github – платформа для проектів з розробки програмного забезпечення

Парсинг – автоматичний процес обробки та збору даних

ОС – операційна система

Алерт – повідомлення або попередження, згенероване системою моніторингу

Інцидент – несанкціонована або деструктивна подія, яка ставить під загрозу безпеку, конфіденційність, цілісність або доступність комп'ютерних систем

Макрос – це набір певних інструкцій, що в об’днані в одну команду для автоматичного виконання

ВСТУП

Сьогодні неможливо уявити ситуацію де б інформаційні системи не полегшували роботу людям: лікарі використовують програми для швидкого діагностування хвороби та обробки величезної кількості даних; онлайн- банкінг широко розповсюджується по всьому світу для швидких платежів в один клік; підприємства, заводи, виробництва використовуються спеціальне програмне забезпечення для пришвидшення та автоматизації рутинних дій, що допомагає збільшити обсяги товару та його якості 1 зміцнити економіку країни. Навіть шкільне/університетське життя вже неможливо уявити без використовування гаджетів, IoT-систем і систем централізованого контролю – все це давно перебуває в нашому житті й активно продовжує розвиватись для нашого комфорту.

Проте... Як часто ми задумуємось, які ризики за собою несе експлуатація зловмисником таких систем? Ні одна система не є захищеною – питання злому базується лише в фінансах та наполегливості зловмисника. Згадаємо відому атаку шифрувальників на Colonial Pipeline унаслідок якої, вся трубопровідна система була зупинена на 5 днів та було оголошено надзвичайну ситуацію на півдні США. Атаки на медичний сектор, що призводять до зупинки або ж повної непрацездатності обладнання в результаті чого, збитки можуть рахуватись не лише фінансово, а й людськими життями. Все це – приклади просунутих постійних загроз (Advanced Persistent Threat – далі APT), які є великою проблемою інформаційних систем на теперішній час. APT атаки є формою зловмисної, навмисної та чіткої цільової атаки, вони використовують витончені та складні техніки і тактики.

Проте, вони завжди залишають “сліди”, по яким можна ідентифікувати загрозу й зупинити процес її перебування в систем Саме тому виявлення та усунення цих загроз є головним аспектом боротьби проти таких зловмисників.

Метою дипломної роботи є дослідження присутності APT загроз в Windows та розробка системи по впровадженню методів їх виявлення на основі індикаторів компрометації.

Об'єктом дослідження є застосування техніки Persistence для довготривалого доступу до комп'ютерної системи під час циклу атаки APT груп.

Наукова новизна отриманих результатів полягає в розробці власної системи на основі клієнт-серверної моделі для виявлення загроз на основі статичних, динамічних та мережевих індикаторів й зручна взаємодія з системою через Телеграм бота.

Практичне значення: Результат даної роботи може бути використаний як на звичайних робочих станціях рядових співробітників для швидкого виявлення загроз так і спеціалістами в галузі кібербезпеки для автоматизації рутинних дій та пришвидшення реагування на наявні індикатори.

Апробація результатів роботи: Подана заявка на авторську програму 19.06.2023. Номер заявки - c202304496.

1 ЩО ТАКЕ APT ЗАГРОЗИ І ТАКТИКА PERSISTENCE, ЩО ВОНИ ВИКОРИСТОВУЮТЬ. АТРИБУЦІЯ ЗАГРОЗ НА ОСНОВІ ІНДИКАТОРІВ

1.1 Що таке APT загрози?

APT(Advanced persistent threat)[1] – це високоорганізована, цілеспрямована і мотивована кіберзагроза, що керується висококваліфікованими й добре фінансованими зловмисниками протягом тривалого періоду часу. На відміну від традиційних кібератак, які зосереджені на порушенні працездатності або отриманні швидкого несанкціонованого доступу, APT характеризуються своєю прихованою природою, постійністю та довгостроковим перебуванням в інфраструктурі своїх цілей.

APT передусім націлені на організації, що мають високу цінність[2], зокрема державні установи, оборонні підприємства, постачальники критичної інфраструктури, науково-дослідницькі інститути або транснаціональні корпорації. Саме такі цілі володіють цінною інформацією, технологіями або ресурсами, що робить їх привабливими для зловмисників.

Також, APT часто асоціюються з суб’єктами загрози, які спонсоруються державою та мають значні ресурси, включаючи передові технічні можливості, фінансування і розвідувальну підтримку. Однак, вони також можуть здійснюватися організованими злочинними угрупованнями або хактивістами.

1.2 Цикл проведення APT атаки

Як і будь-які загрози в реальному світі, APT загрози в кіберпросторі складаються з певних етапів, через які зловмисники проходять, починаючи від початкової розвідки цілі, закінчуючи своєю кінцевою метою (компрометація жертви, викрадення таємних даних, шифрування інфраструктури, стирання всіх даних компанії). Unified Kill Chain[3] – це фраємворк, що є результатом об’єднання MITRE ATT&CK та Cyber Kill Chain фраємворків та описує 18 стадій реалізації атаки у вигляді впорядкованої схеми кроків, що їх зловмисник має проробити для успішної атаки на свою ціль. Реалізація цих етапів атаки залишається подібною для більшості атак, змінюються лише конкретні методи, інструменти і процедури.

Щоб проникнути всередину та отримати доступ до системи й даних, застосовуються наведені нижче етапи проведені атаки зловмисником:

- Початкова розвідка (Initial Reconnaissance)
- Початкова компрометація жертви (Initial Compromise)
- Закріплення присутності (Establish Foothold)
- Підвищення привілегій (Escalate Privileges)
- Внутрішня розвідка (Internal Reconnaissance)
- Бокове переміщення (Move Laterally)
- Підтримання присутності (Maintain Presence)
- Завершення атаки досягненням своїх цілей (Complete Mission)

Рисунок 1.1 – Ілюстрація циклу проведення APT атаки

1.3 Persistence

Як вже було сказано вищу, ***Persistence***[4] – це один з етапів проведення APT атаки. Це широко використовувана зловмисниками техніка, що полягає в приховувані своєї активності та перебування в системі. Після проведення початкової компрометації, зловмисник прагне забезпечити, щоб його присутність залишалася непоміченою і недоторканою, навіть якщо початкова точка входу буде виявлена і закрита. Persistence має вирішальне значення для зловмисників, оскільки вона дозволяє їм вести постійне спостереження, викрадати дані або досягати своїх цілей протягом тривалого періоду часу. За відсутності цього кроку, хакери втратять доступ до скомпрометованого середовища і тоді їхня здатність досягати своїх цілей буде суттєво ускладнена, саме тому це є одним із ключових та найскладніших етапів атаки.

Persistence передбачає створення механізмів, таких як бекдори, призупинені процеси, зворотна командна оболонка, що запускається в певний період часу, приховане шкідливе програмне забезпечення, що запускається одразу після включення системи, інструменти віддаленого доступу що дозволяють зловмиснику зберігати віддалений доступ.

Виявлення присутності загрози має вирішальне значення для кібербезпеки, оскільки дозволяє організаціям виявляти і зменшувати постійну присутність і діяльність зловмисників у своїх системах або мережах. Це дозволяє організаціям зрозуміти тактику і методи, які використовують зловмисники, покращити розвідку загроз і адаптувати засоби захисту, що сприяє ефективному реагуванню на інциденти, пришвидшуючи розслідування, відстеження діяльності зловмисників та розробку відповідних захисних рішень. Нарешті, це допомагає запобігти подальшій експлуатації, порушуючи доступ і контроль зловмисників, стримуючи їхнє перебування і захищаючи цільове середовище.

1.4 Атрибуція загроз та IOC

Атрибуція загроз[5] - це процес визначення або покладання відповідальності на окремих осіб, групи або організації, які стоять за кібератакою або зловмисною діяльністю. Він передбачає збір та аналіз різних типів інформації, включаючи технічні показники, поведінкові моделі, геополітичний контекст та розвідувальні дані з різних джерел, з метою визначення присутності атаки та її ймовірне походження. Атрибуція - це складне і відповідальне завдання, яке вимагає досвіду, всебічних доказів і співпраці між фахівцями з кібербезпеки, правоохоронними органами та розвідувальними управліннями.

Щодо самих атрибутів, то це специфічні характеристики або властивості загрози, шкідливого програмного забезпечення або цифрового артефакту, які використовуються для їх опису або класифікації. Ці атрибути надають цінну інформацію для аналізу, виявлення та пом'якшення кіберзагроз.

Після того, як зловмисник встановив присутність в цільовій системі, атрибути можуть з'являтися в різних формах, які вказують на проведення етапу persistence або триваючу зловмисну активність. Ось кілька прикладів:

- **Записи в реєстрі**: Зловмисники часто використовують реєстр Windows для закріплення. Вони можуть створювати й змінювати певні ключі та значення реєстру, щоб забезпечити виконання свого шкідливого коду або бекдорів при кожному завантаженні системи, при настанні певних подій (відкриття командної оболонки, запуск сервера і т.д). Ці записи реєстру слугують атрибутами, які можна ідентифікувати та проаналізувати для виявлення присутності загрози.
- **Модифікації файлової системи**: Під час приховування перебування, зловмисинки може вносити зміни у файлову систему, наприклад: встановлювати шкідливі файли або модифікувати існуючі. Ці модифіковані або новостворені файли виступають атрибутами, які можна знайти та проаналізувати для виявлення механізмів присутності і пов'язаної з ними поведінки.
- **Мережеві з'єднання**: Зловмисники можуть встановлювати постійні мережеві з'єднання для підтримки зв'язку та контролю над скомпрометованою системою. Атрибути, пов'язані з цими з'єднаннями, такі як IP-адреси, номери портів, протоколи зв'язку або незвичайні патерни трафіку, можна відстежувати і аналізувати, щоб виявити поточну зловмисну активність.
- **Заплановані процеси:** Зловмисники можуть створювати заплановані або відтерміновані процеси в цільовій системі для виконання шкідливих дій в певний час або інтервали часу. Ці заплановані завдання також слугують атрибутами, які можна дослідити, щоб виявити загрозу та потенційні індикатори компрометації. Запланований процес є одним із найскладнішим для виявлення атрибутів, оскільки потрібно відслідковувати стан процесів безперервно.

- **Породження нетипових процесів:** Породження процесів також є ознакою, яка вказує на наявність загрози та шкідливої активності в цільовій системі. Для засвоєння в системі, зловмисник може програмувати створення процесів для виконання свого шкідливого коду й збереження контролю над скомпрометованою системою.

Розглянемо один із вкрай поширених випадків проникнення в організацію[6] та приховування своєї присутності - фішинговий файл Office Word, що містить деякі зловмисні сценарії макросів. Після відкриття, програма Office може створити дочірній процес командної оболонки Windows, наприклад PowerShell, для запуску зловмисних скриптів. Після цього PowerShell може завантажувати шкідливі ZIP-файли, розархівовувати та інсталювати шкідливі програми для обходу антивірусних засобів. Зазвичай, таку атаку доволі складно виявити, оскільки в ній приймають участь повністю легальні та дозволені програми, що наявні на більшості корпоративних комп'ютерах (Microsoft Word, Powershell, Winrar, 7-Zip, Google Chrome).

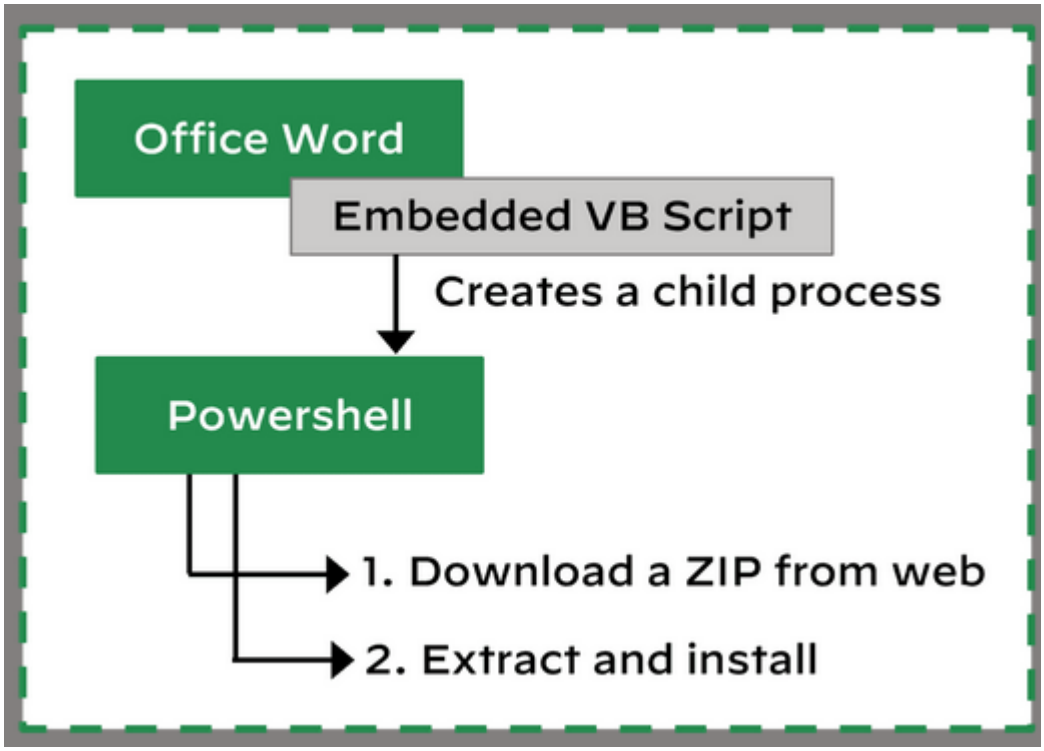


Рисунок 1.2 – Схема проникнення та засвоєння в організації популярних вірусів

Тому, існує необхідність ідентифікації таких індикаторів(атрибутів) та їх виявлення у вигляді сповіщення на єдиний центр управління для пришвидшеного реагування та знешкодження. Для початку розглянемо самі індикатори загроз та їх класифікацію:

1. IOC (Indicators of Compromise).

Статичний інформаційний показник або ознака наявності компрометації[7], що свідчить про присутність певної загрози та її перебування у систем. Широко використовуються для виявлення, ідентифікації та відслідковування загрози. Може включати в себе: IP-адреси, домени, хеш-суми файлів, назви файлів, записи реєстрів та інші.

2. BIOC (Behavioral Indicators of Competencies)

Це, в свою чергу, поведінковий або динамічний показник, що свідчить про наявність компрометації або підтвердження факту зловмисної діяльності в інформаційній системі[8]. На відміну від IOC – він базується на аналізі поведінки системи або користувачів, а не на конкретних артефактах або ознаках. Може включати в себе: створення нестандартних процесів, незвичайна активність користувача, використовування рідкісних програм.

3. Аномалії в мережевому трафіку

Аномальні або нестандартні відхилення в патернах звичайного, легального трафіку, що можуть свідчити про проблеми безпеки, зловмисну діяльність або атаку в мережі. Аналіз аномалій в трафіку є важливим компонентом безпеки, оскільки вся передача даних між комп’ютерними пристроями відбувається саме через мережу.

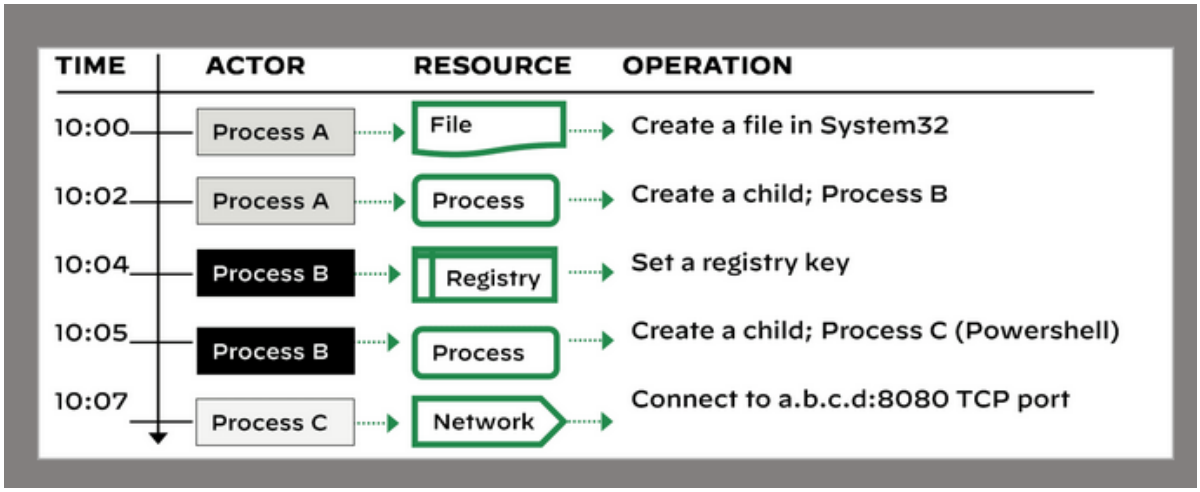


Рисунок 1.3 – Типові схеми зловмисників приховування свого перебування на основі IOC/BIOC та мережевих аномалій

Саме тому, виявлення IOC і BIOC в системах Windows дає критично важливу інформацію про потенційні загрози безпеці, допомагає в реагуванні на інциденти, покращує криміналістичні розслідування, підтримує механізми виявлення на основі сигнатур і дає організаціям можливість вживати активних заходів захисту для протидії приховування зловмисниками своєї присутності в системі. Використовуючи такі індикатори, команда безпеки може посилити свої стратегії захисту і знизити ризик компрометації або ж витоку даних.

1.5 Інструменти для виявлення загроз на основі їх індикаторів

Сьогодні кіберзагрози становлять величезну проблему як для самих систем, так і для людей – вони стають більше витонченими, просунутими та кваліфікованими. Відповідно, засоби безпеки теж повинні вдосконалюватись, щоб бути на крок попереду. В кібербезпеці вже давно відомо – жодна система не є захищеною, питання злому лише в фінансах та мотивації зловмисника. Тому, стало прийнятим вважати, що взлом системи може відбутись в будь-який момент, потрібно лише унеможливити його поширення та якомога сильніше зменшити подальші ризики та загрози. Одним із рішень в сфері кібербезпеки,

яке доволі добре справляється з попередженням загроз та їх виникнення є Cortex XDR, з яким мав нагоду попрацювати на моїй переддипломній практиці.

Cortex XDR – це рішення в області кібербезпеки[9], розроблене компанією Palo Alto Networks. Cortex XDR поєднує дані з різних джерел, таких як кінцеві точки, мережеві пристрої, логи безпеки та хмарні сервіси, для забезпечення комплексного аналізу загроз і виявлення складних атак. Він використовує аналітичні алгоритми та штучний інтелект для ідентифікації та кореляції підозрілих активностей. Також, він відіграє важливу роль у захисті від сучасних постійних загроз (APT) та інших складних кіберзагроз. Cortex XDR використовує передові методи для виявлення індикаторів компрометації (IOC) і поведінкових індикаторів компрометації (BIOC), пов'язаних з APT. Засіб складається з хмарного екземпляра і агентів, встановлених на кінцевих точках організації локально або в хмарі, що забезпечує постійне спостереження за клієнтською машиною та надає командам безпеки дієві сповіщення та інформацію щодо виявленої загрози або ж підозрілої активності.

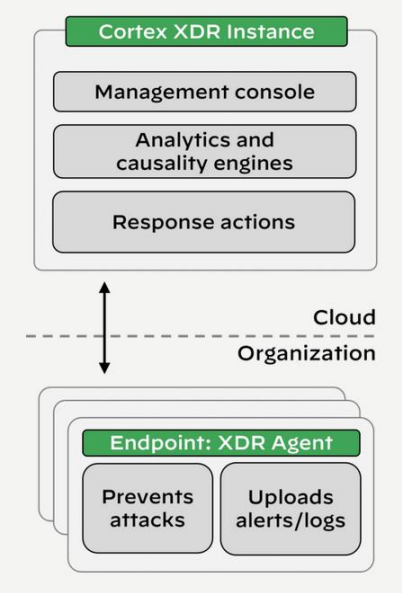


Рисунок 1.4 – Схема взаємодії Cortex XDR

Цей інструмент ідеально підходить для детектування присутності загроз на основі їх індикаторів, оскільки містить вбудовані механізми налаштування правил та політик для них.

- **Indicators of Compromise (IOCs).** Правила IOC у Cortex XDR визначають статичні властивості загроз, такі як: імена файлів, шляхи, хеші файлів, IP-адреси та імена доменів. Метою правил IOC є виявлення відомих простих шкідливих програм.

Рисунок 1.5 – IOC правило

- **Behavioral Indicators of Compromise (BIOCs).** Правила BIOC визначають поведінкові характеристики загроз, такі як: тактика, методи та процедури, що використовуються під час циклу атаки. Наприклад, під час атаки, ШПЗ може записувати файли, створювати процеси, відкривати мережеві порти та змінювати значення ключів реєстру - усе це залишає сліди, які складають особливості поведінки.

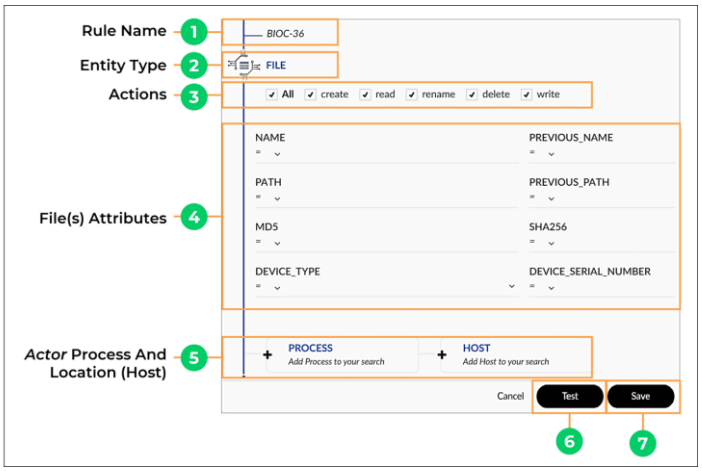


Рисунок 1.6 – BIOC правило Cortex XDR

- **Network Inspection.** Cortex XDR перехоплює мережеві пакети, що проходять через клієнтську машину (від\через\з) та застосовує свої механізми виявлення загроз до цього трафіка. Він використовує комбінацію виявлення на основі сигнатур, поведінкової аналітики та машинного

навчання для виявлення відомих і невідомих загроз, що проникають у мережу.

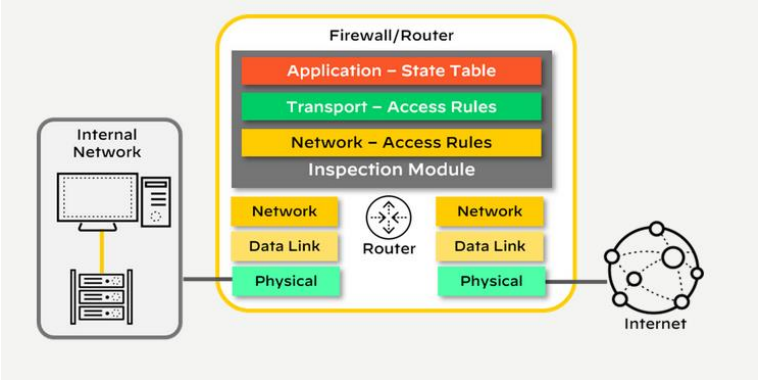


Рисунок 1.7 – Аналіз мережевого трафіку Cortex XDR

Проте, як і переваги, Cortex XDR має свої недоліки, серед яких:

- **Ресурсомісткість:** Cortex XDR вимагає спеціального обладнання або віртуальних машин для запуску своїх компонентів, таких як сервер управління, агенти та колектори. Ці вимоги до ресурсів можуть призвести до перевантаження існуючої інфраструктури, особливо в організаціях з обмеженими ресурсами або в невеликих середовищах.
- **Складність розгортання:** Розгортання цього засобу захисту у різноманітних IT-середовищах може бути складною і нетривіальною задачею. Проблеми сумісності зі старими системами, сегментація мережі та складні мережеві архітектури можуть створювати перешкоди на етапі розгортання.
- **Вартість**: Cortex XDR - це комерційне рішення, яке передбачає витрати на ліцензування. Структура цін може змінюватися залежно від розміру організації, моделі розгортання та необхідних функцій. Для деяких організацій вартість впровадження та підтримки Cortex XDR може бути суттєвим фактором.

Тому, під час цього я надихнувся та зайнявся розробкою власного продукту, що зможе виконувати такі ж дії, проте буде розрахований на

особисте та комфортне користування “з коробки”, яке повинне бути максимально простим навіть для звичайного користувача комп’ютера.

Висновки до розділу 1

В розділі було розглянуто поняття APT загроз та їх роль в сучасному кібепросторі, цикл проведення атаки ними, та один із етапів – persistence, виявлення якого є ключовим моментом.

Було досліджено, що одним із методів виявлення присутності таких загроз в системі є індикатори компрометації, що можуть бути як статичними (файл, хеш, запис в реєстрі) так і поведінковими (динамічними – створення процесу, відхилення в патернах трафіка).

Також було розглянуто один із передових засобів протидії просунутим кіберзагрозам – Cortex XDR та описані його базові особливості.

2 РОЗРОБКА СИСТЕМИ ВИЯВЛЕННЯ ПРИСУТНОСТІ ТА ПЕРЕБУВАННЯ APT ГРУП В WINDOWS СИСТЕМАХ НА ОСНОВІ ЇХ АТРИБУТІВ

2.1 Планування системи та налаштування середовища

Для створення власної системи виявлення загроз на основі індикаторів було вибрано клієнт-серверну модель, де комунікація між собою відбувається за допомогою сокетів. Система повинна бути зручною та “мати” мінімальні налаштування для ефективного користування з коробки – тому сервер впроваджено як центр управління в Telegram бота. Telegram – один із найпопулярніших месенджерів для обміну повідомленнями; він надає безкоштовний доступ до **Telegram API (Telegram Bot API)**[10] – потужного інструменту, який дозволяє розробникам створювати ботів та застосунки, які можуть надсилати та отримувати повідомлення, взаємодіяти з користувачами та виконувати різні дії на платформі цій платформі. Telegram надає офіційні бібліотеки для різних мов програмування серед яких **Python** – інтерпретована мова програмування з динамічною типізацією, яка й була вибрана для серверної частини.

Оскільки цільовою машиною було обрано Windows, тому доречно скористатись функціями **Windows API**[11], що надаються розробникам для гнучкої взаємодії з системою. Мова програмування клієнта - **C++,** яка є нативною мовою програмування для Windows, що означає, що код, написаний на C++, може безпосередньо взаємодіяти з операційною системою без будь-яких додаткових шарів або середовищ виконання. Поєднуючи Windows API з C++, розробники можуть використовувати потужність операційної системи Windows на повну, отримати доступ до специфічної для платформи функціональності, оптимізувати продуктивність і глибоко інтегруватися в

екосистему Windows. Таке поєднання надає широкі можливості для розробки надійних і багатофункціональних додатків для Windows.

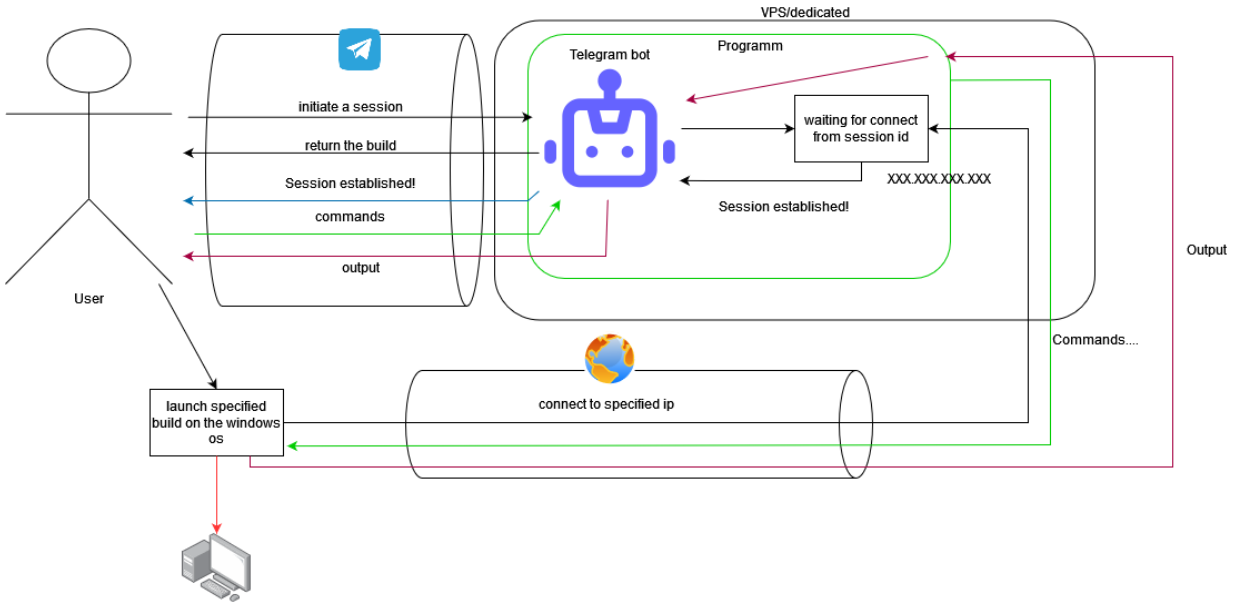


Рисунок 2.1 – Схема взаємодії користувача з системою

Сервер повинен працювати безперешкодно та автономно – тому прийняте рішення помістити серверну частину в хмарну інфраструктуру. Для цього орендовано інстанс в хмарному середовищі Amazon[12] та отримано публічну IP-адресу для взаємодії з ним.

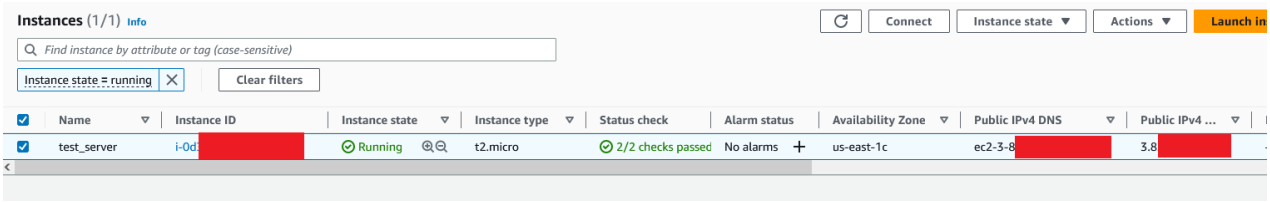


Рисунок 2.2 – Головне меню управління сервером на Amazon

Оскільки сервер має публічну адресу, потрібно подбати про його безпеку та обмежити мережевий доступ. Для цього налаштовано мережеві правила на доступ до сервера лише з клієнтської IP-адреси:

- 22 порт дозволений лише для адміністратора та відповідно управління сервера ним (через SSH)
- 65432 порт дозволений лише для доступу клієнтської машини через з’єднання сокетів до серверної частини для коректної роботи бота.

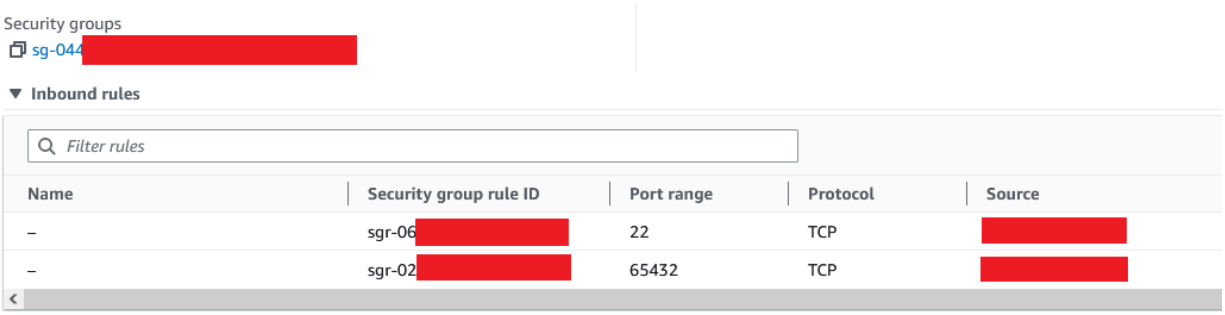


Рисунок 2.3 – Мережеві правила для сервера

Після налаштування середовища потрібно створити самого бота. Це можна зробити через @BotFather[13] - офіційний бот, наданий Telegram для управління та створення нових ботів на платформі Telegram. Він слугує інструментом конфігурації, який дозволяє користувачам створювати та керувати власними ботами, взаємодіючи з ним через чат-інтерфейс Telegram.

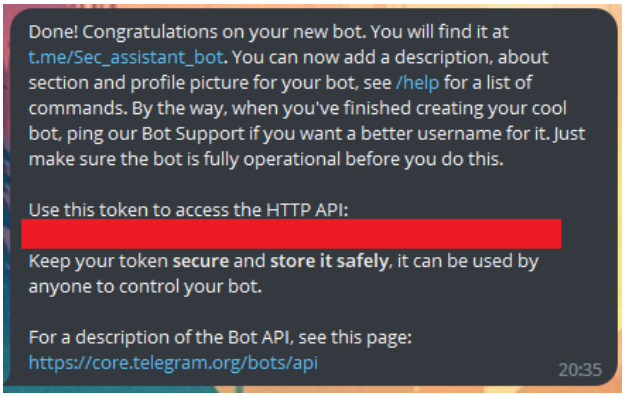


Рисунок 2.4 – Створення свого Телеграм бота

Проект самої системи взаємодії користувача виглядає наступним чином: одразу після встановлення користувач отримує відповідне повідомлення про успішне з’єднання своєї цільової машини з сервером та може приступати до вибору методу пошуку присутності загрози в його системі.

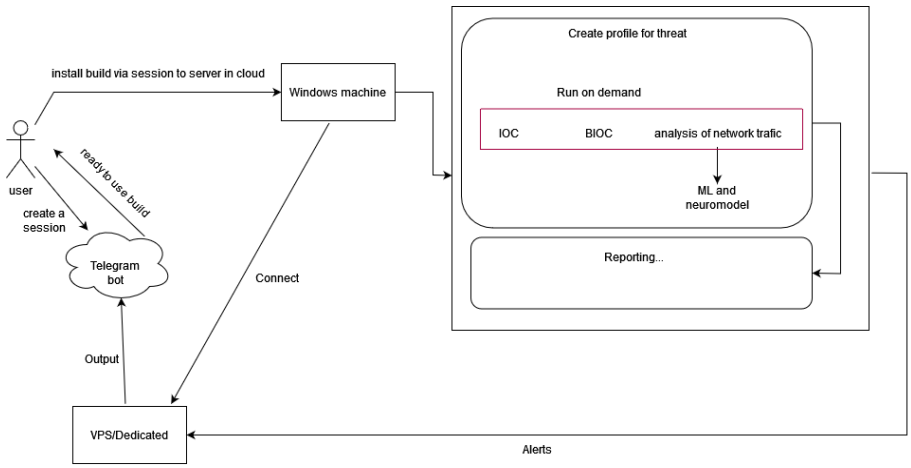
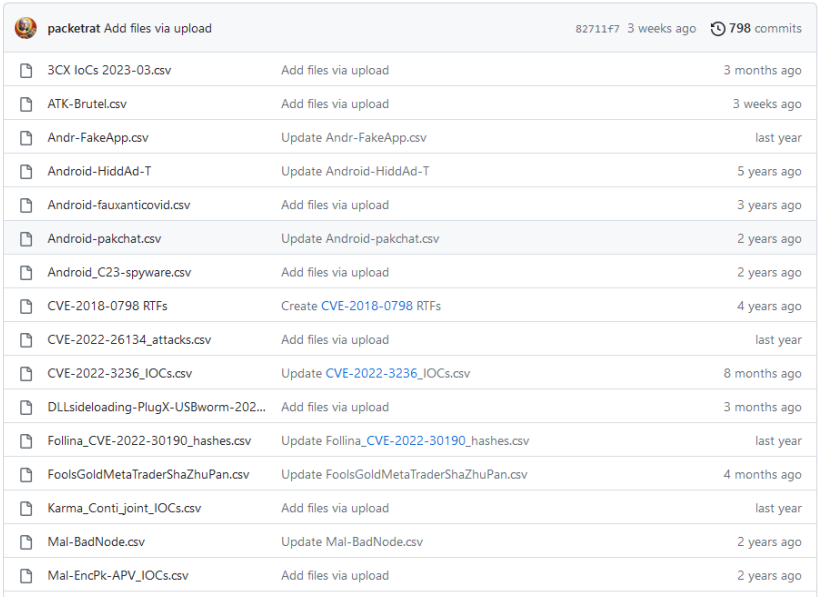


Рисунок 2.5 – Взаємодія користувача з системою

2.2 Визначення шуканих індикаторів та підключення клієнта

Як вже було сказано, після компрометації жертви зловмисники нерідко залишають свої “сліди” в системі, що може включати файли, хеші, записи в реєстрі й т.д. Однією із поширених технік є бекдори – несанкціонований доступ до комп’ютерної системи жертви за допомогою прихованих механізмів автентифікації. Він забезпечує доступ до системи, дозволяючи особам, які знають про нього, отримати контроль, маніпулювати даними або виконувати зловмисні дії. Зазвичай атака може включати в себе використання інструментів на проникнення, таких як Metasploit, для встановлення цих бекдорів.

Тому, для ідентифікації таких загроз компанії, орієнтовані на кібербезпеці компанії регулярно проводять збір та аналіз методів проникнення під час різноманітних атак. Після цього відбувається атрибуція цих загроз та публікація їх індикаторів у відкритий доступ для підвищення рівня розвідки загроз і допомоги у виявленні та запобіганні кібератакам. Одним із підходящих джерел для використанні у роботі було обрано Github репозиторій SophosLabs[14]:

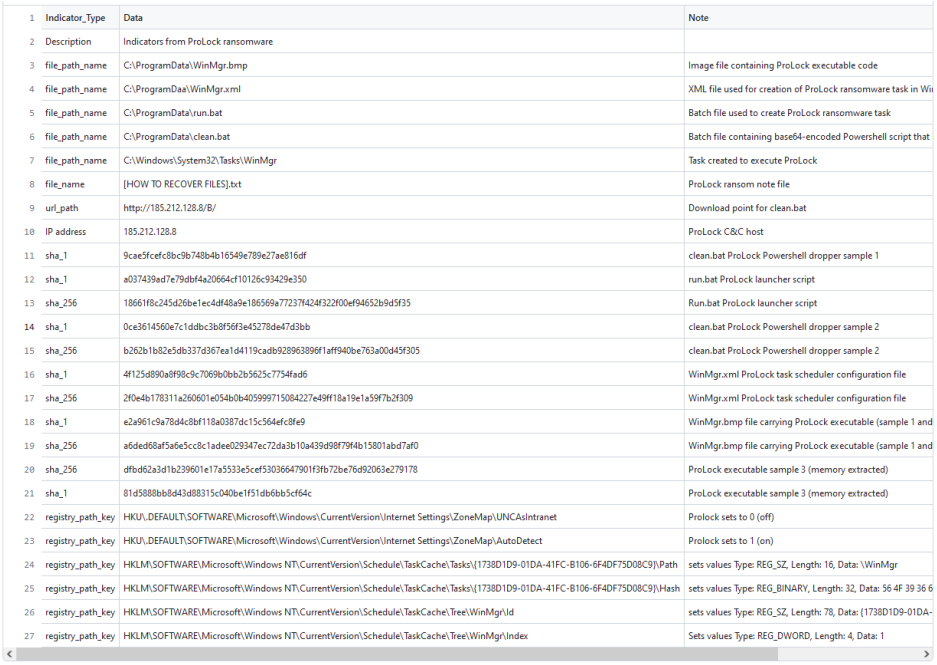


packetrat Add files via upload — 82711f7 3 weeks ago — 798 commits

3CX IoCs 2023-03.csv	Add files via upload	3 months ago
ATK-Brutel.csv	Add files via upload	3 weeks ago
Andr-FakeApp.csv	Update Andr-FakeApp.csv	last year
Android-HiddAd-T	Update Android-HiddAd-T	5 years ago
Android-fauxanticovid.csv	Add files via upload	3 years ago
Android-pakchat.csv	Update Android-pakchat.csv	2 years ago
Android_C23-spyware.csv	Add files via upload	2 years ago
CVE-2018-0798 RTFs	Create CVE-2018-0798 RTFs	4 years ago
CVE-2022-26134_attacks.csv	Add files via upload	last year
CVE-2022-3236_IOCs.csv	Update CVE-2022-3236_IOCs.csv	8 months ago
DLLsideloading-PlugX-USBworm-202...	Add files via upload	3 months ago
Follina_CVE-2022-30190_hashes.csv	Update Follina_CVE-2022-30190_hashes.csv	last year
FoolsGoldMetaTraderShaZhuPan.csv	Update FoolsGoldMetaTraderShaZhuPan.csv	4 months ago
Karma_Conti_joint_IOCs.csv	Add files via upload	last year
Mal-BadNode.csv	Update Mal-BadNode.csv	2 years ago
Mal-EncPk-APV_IOCs.csv	Add files via upload	2 years ago

Рисунок 2.6 - Github репозиторій індикаторів присутності популярних APT груп

Він містить всі типи популярних індикаторів загроз, які були описані й будуть використані в цій роботі, починаючи від файлових ідентифікаторів, закінчуючи мережевими:



	Indicator_Type	Data	Note
2	Description	Indicators from ProLock ransomware	
3	file_path_name	C:\ProgramData\WinMgr.bmp	Image file containing ProLock executable code
4	file_path_name	C:\ProgramDaa\WinMgr.xml	XML file used for creation of ProLock ransomware task in Wi
5	file_path_name	C:\ProgramData\run.bat	Batch file used to create ProLock ransomware task
6	file_path_name	C:\ProgramData\clean.bat	Batch file containing base64-encoded Powershell script that
7	file_path_name	C:\Windows\System32\Tasks\WinMgr	Task created to execute ProLock
8	file_name	[HOW TO RECOVER FILES].txt	ProLock ransom note file
9	url_path	http://185.212.128.8/B/	Download point for clean.bat
10	IP address	185.212.128.8	ProLock C&C host
11	sha_1	9cae5fcefc8bc9b748b4b16549e789e27ae816df	clean.bat ProLock Powershell dropper sample 1
12	sha_1	a037439ad7e79dbf4a20664cf10126c93429e350	run.bat ProLock launcher script
13	sha_256	18661f8c245d26be1ec4df48a9e186569a77237f424f322f00ef94652b9d5f35	Run.bat ProLock launcher script
14	sha_1	0ce3614560e7c1ddbc3b8f56f3e45278de47d3bb	clean.bat ProLock Powershell dropper sample 2
15	sha_256	b262b1b82e5db337d367ea1d4119cadb928963896f1aff940be763a00d45f305	clean.bat ProLock Powershell dropper sample 2
16	sha_1	4f125d890a8f98c9c7069b0bb2b5625c7754fad6	WinMgr.xml ProLock task scheduler configuration file
17	sha_256	2f0e4b178311a260601e054b0b405999715084227e49ff18a19e1a59f7b2f309	WinMgr.xml ProLock task scheduler configuration file
18	sha_1	e2a961c9a78d4c8bf118a0387dc15c564efc8fe9	WinMgr.bmp file carrying ProLock executable (sample 1 and
19	sha_256	a6ded68af5a6e5cc8c1adee029347ec72da3b10a439d98f79f4b15801abd7af0	WinMgr.bmp file carrying ProLock executable (sample 1 and
20	sha_256	dfbd62a3d1b239601e17a5533e5cef53036647901f3fb72be76d92063e279178	ProLock executable sample 3 (memory extracted)
21	sha_1	81d5888bb8d43d88315c040be1f51db6bb5cf64c	ProLock executable sample 3 (memory extracted)
22	registry_path_key	HKU\.DEFAULT\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\UNCAsIntranet	Prolock sets to 0 (off)
23	registry_path_key	HKU\.DEFAULT\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\AutoDetect	Prolock sets to 1 (on)
24	registry_path_key	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tasks\{1738D1D9-01DA-41FC-B106-6F4DF75D08C9}\Path	sets values Type: REG_SZ, Length: 16, Data: \WinMgr
25	registry_path_key	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tasks\{1738D1D9-01DA-41FC-B106-6F4DF75D08C9}\Hash	sets values Type: REG_BINARY, Length: 32, Data: 56 4F 39 36 6
26	registry_path_key	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\WinMgr\Id	sets values Type: REG_SZ, Length: 78, Data: {1738D1D9-01DA-
27	registry_path_key	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\WinMgr\Index	Sets values Type: REG_DWORD, Length: 4, Data: 1

Рисунок 2.7 – Індикатори ProLock ransomware[15]

При написані серверної частини – орієнтація була на простоту для користувача та зручність використання. Також немалу частину роботи було виділено на швидкість та синхронність дій. Саме меню бота, який виступає в

ролі сервера доволі просте та інтуїтивне – потрібно його запустити та отримати виконуваний файл, що потрібно встановити на цільову машину. Він є необхідним для зв’язку з сервером.

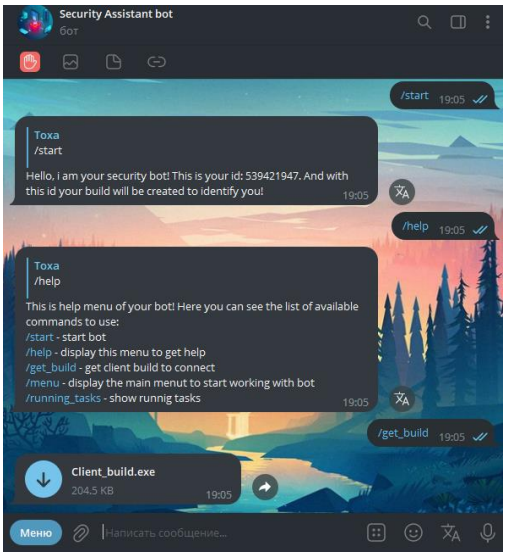


Рисунок 2.8 – Панель бота

Файл потрібно запустити на машині клієнта та очікувати з’єднання з сервером. Після успішного запуску та підключення –система покаже сповіщення у вигляді IP-адреси та порта підключеної цільової машини.

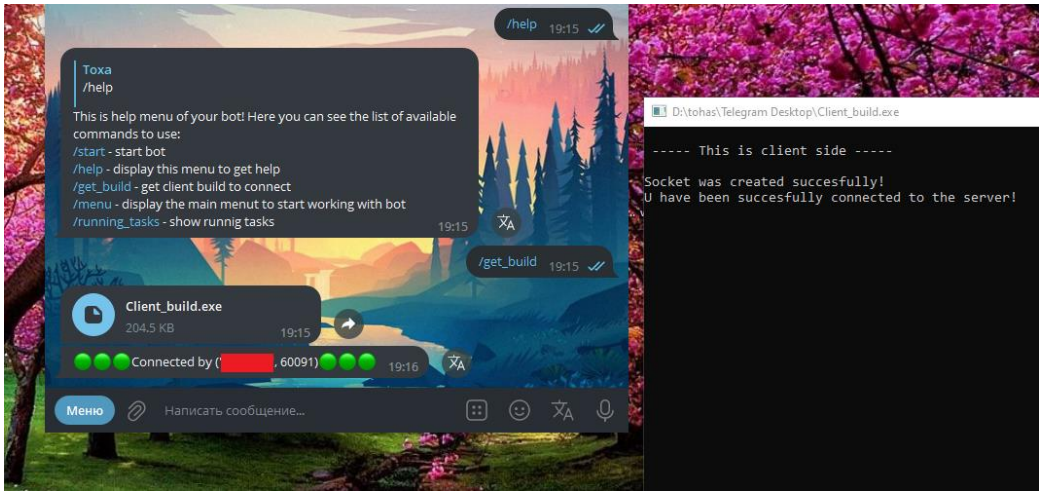


Рисунок 2.9 – Підключення клієнта до серверної частини

2.3 Детектування відомих загроз на основі статичних атрибутів (IOC)

Після успішного та безперешкодного підключення клієнта система видає доступ до головного меню:

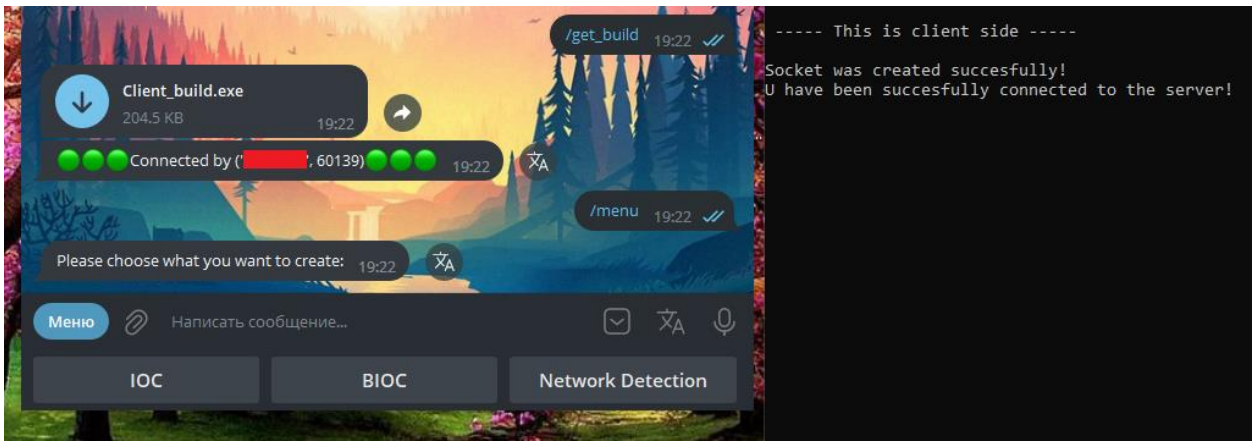


Рисунок 2.10 – Меню виявлення наявних індикаторів

Почнемо розгляд з найбільш простих, проте не менш важливих індикаторів – статичні індикатори (IOC). Вони мають на меті виявлення вже наявних та відомих загроз, що присутні в системі, відповідно, з розвитком витонченіших атак, покривають все менший й менший простір, через який загрози засвоюються в системі. Дані індикатори можуть з’явитись під час встановлення підозрілого та неперевіреного програмного забезпечення, переходження по сумнівним посиланням та перебування на сайтах, що містять в собі виконання шкідливого зразку коду або ж просто під час звичайного розархівування “на вигляд” легального архіву. Всі ці дії можуть містити в собі виконання небажаного коду, що призводить до приховування присутності зловмисника через файл, папку або ключі реєстру. Тому не варто недооцінювати цей вектор і починати розслідування та виявлення присутності загроз завжди з нього.

Через панель сервера можна натиснути відповідну кнопку та побачити що для IOC доступний пошук по імені файла, його шляху або ж іншим

послідовностям, що можуть виникнути при зверненні до нього; хешу файла; значень і ключів системного реєстру Windows.

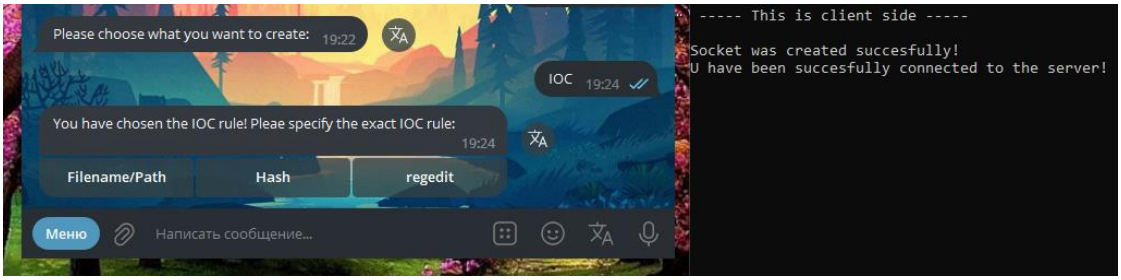


Рисунок 2.11 – Меню IOC

Можна виконати пошук тестового файлу в системі та переглянути результат роботи:

Рисунок 2.12 – Пошук по імені файлу

Після недовго проміжку часу система показує результат сканування по всій файловій системі:

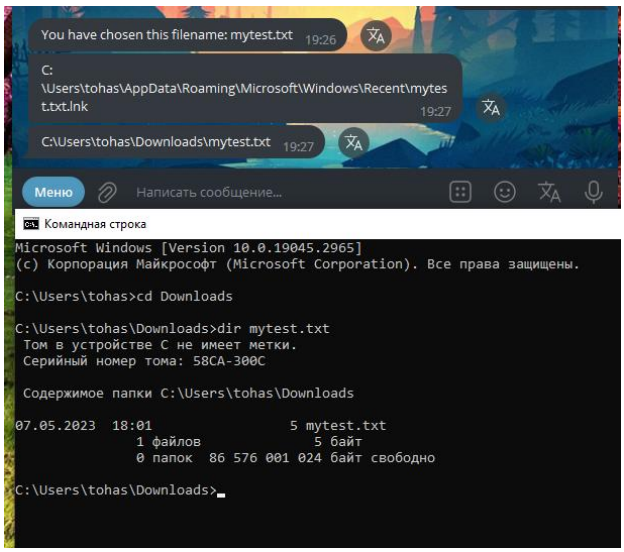


Рисунок 2.13 – Результати пошуку

Як бачимо файл справді існує в цільовій машині, а тому система його знайшла і вивела в панель бота повний шлях до нього. Варто відмітити, що для пошуку файла використовується рекурвисний обхід всієї файлової системи та функція ***strstr()***, що повертає вказівник на перше входження імені файлу в повний шлях кожного файлу на цільовій машині. Відпвідно, це дає змогу здійснювати пошук файла не лише по його імені, а й по його повному шляху в файловій системі:

```
else
{
    if (strstr(FindFileData.cFileName, lpFilePattern) != NULL)
    {
        cout << data_to_send << endl;
        SendDataToSocketConnectio(data_to_send);
        //return (lpFolder + string("\\") + FindFileData.cFileName);
        //cout << lpFolder << "\\" << FindFileData.cFileName << endl;
    }
}
```

Рисунок 2.14 – Фрагмент коду для пошуку файлу

Тому, перевірити наявність індикаторів можна не лише у виляді імені файлу, а також у вигляді директорій, та навіть частин його шляху, що є дуже зручним та ефективним при пошуку індикаторів з невідомими характеристиками.

Наступний статичний індикатор, що нам доступний – це пошук по хешу файла. Хеш файла - це унікальне буквено-цифрове значення, яке генерується шляхом застосування криптографічного алгоритму хешування до вмісту файлу. Він слугує цифровим відбитком або контрольною сумою даних файлу. Зазвичай використовуються такі поширені хеш-функції, як MD5, SHA-1 і SHA-256. Кожен файл має унікальне хеш-значення й навіть незначна зміна вмісту файлу призведе до кардинальної зміни хеш-значення. Ця властивість робить хеші корисними для ідентифікації та розрізнення файлів, незалежно від їхніх назв або розташування. Порівнюючи хеш файлу зі списком відомих шкідливих хешів, наш інструмент може виявляти файли, які відповідають відомим загрозам.

Після введення коректного формату SHA-256, користувач повинен дочекатись кінця сканування:

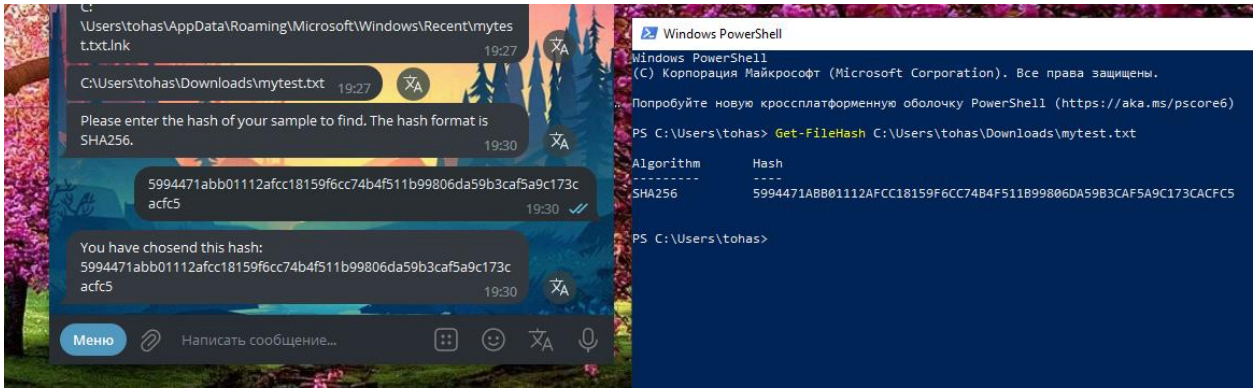


Рисунок 2.15 – Пошук по хешу файла

При реалізації цього функціоналу виникна доволі нетривіальна проблема - пошук хеша доволі трудомістка задача, оскільки потрібно обчислити хеші всіх наявних файлів у файловій системі й зрівняти їх із заданими, що займає близько години часу лише для одного значення. Такий проміжок часу є неприпустимим для реагування на присутні загрози, особливо тоді, коли кожна секунда оцінюється мега/гігабайтами викачаними данними з корпорацій, а тому потрібно було прискорити якомога більше цей процес.

Після тривалих роздумів впроваджено ефективніше рішення – багатопоточність. Для початку попередня функція проходження по файловій

системі була дещо модифікован й додано vector - контейнер динамічного масиву зі стандартної бібліотеки шаблонів (STL), що забезпечує гнучкий та ефективний спосіб зберігання та маніпулювання колекцією елементів. В цю змінну будуть записуватись повні шляхи до всіх файлів файлової системи.

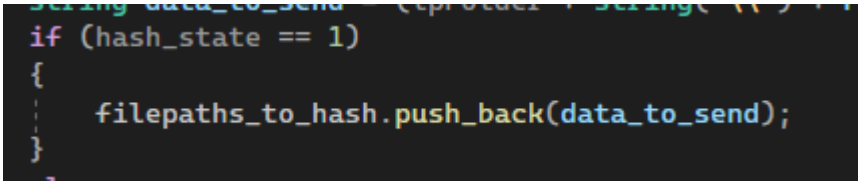


Рисунок 2.16 – Фрагмент коду, що відповідає за додавання повних шляхів до всіх файлів у вектор

Далі було створено дві функції – ThreadCreation() та ThreadFunc(). Перша з них створює два статичних масива розміром 64:

- hThreadClients - для утримання хендлів потоків, які будуть обчислювати хеші переданих їм файлів
- arr_index – для утримання порядкових номерів файлів, що будуть оброблятись попереднім потоком.

```
void ThreadCreation()
{
    HANDLE* hThreadClients = new HANDLE[num];
    DWORD* arr_index = new DWORD[num];
    cout << "inside thread func" << endl;
    for (int i = 0; i < num; i++)
    {
        arr_index[i] = i;

        hThreadClients[i] = CreateThread(NULL, 0, ThreadFunc, &arr_index[i], 0, 0);
    }
    cout << "before WaitForMultipleObjects" << endl;
    WaitForMultipleObjects(num, hThreadClients, TRUE, INFINITE);
    cout << "after WaitForMultipleObjects" << endl;
    delete[] hThreadClients;
    delete[] arr_index;
}



DWORD WINAPI ThreadFunc(LPVOID  lpParam)
{
    int value_id = (*(DWORD*)lpParam);
    while (value_id < filepaths_to_hash.size())
    {
        GetFileHash(filepaths_to_hash[value_id].c_str());
        value_id += num;
    }
    return 0;
}
```

Рисунок 2.17 – Реалізація багатопоточності при знаходженні файлу по його хешу

Після цього, за допомогою цилку for{}, кожному з хендлів присвоюється створення потоку та передається відповідний індекс елемента масиву, за якиим міститься шлях до файлу, що він повинен обробити (обчислити хеш). Відповідно, у другій функції ThreadFunc() (яка являється безпосередньо функцією потоку, що він виконує) потік приймає параметром змінну, яка вказує хеш якого саме файлу йому потрібно знайти. Далі за допомогою Windows Cryptographic API (CryptoAPI)[16] відбувається звертання до CSP (компонент, який забезпечує криптографічну функціональність ОС), створюється об’єкт хеша, із заданням відповідного алгоритма (в нашому випадку SHA-256), зчитується та хешується зміст файлу щоб у кінці зрівняти отриманий хеш з заданим через панель бота. Процес продовжується доти, доки всі файли у векторі не будуть пройдені та не обчислені відповідні їх хеші. Кожен потік незалежно від інших обчислює хеш для підмножини даних, що дозволяє

виконувати кілька обчислень одночасно. Це значно підвищує загальну продуктивність і колосально скорочує час очікування (у 64! рази), необхідний для обробки даних, що є достатньо хорошим рішенням при пошуку файлу по хешу обходячи абсолютно всю файлову систему.

Тому вже через 5 хвилин можна отримати результат, що свідчить про наявність файлу з таким хешем. Результат подається у вигляді повного шляху до файлу, для зручної ідентифікації та реагування загрози:

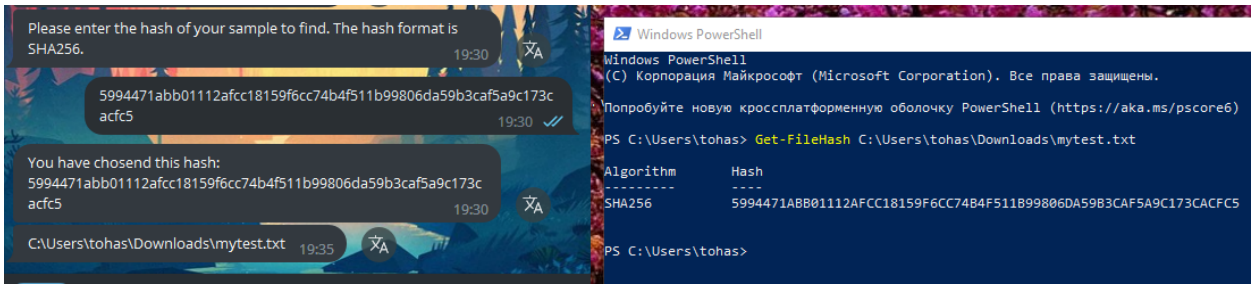


Рисунок 2.18 – Результати пошуку по хешу

Третій та останній статичний індикатор розробленій нашій програмі, по якому можна виявити присутність загрози – це реєстр Windows. Метою APT груп є встановлення постійного доступу до скомпрометованих систем, що дозволяє їм зберігати контроль і продовжувати свою шкідливу діяльність протягом тривалого періоду. Реєстр Windows є зручним і потужним засобом для досягнення такого доступу. Створюючи або змінюючи ключі та значення реєстру, APT групи можуть налаштувати систему на автоматичне виконання свого шкідливого коду під час запуску системи або при настанні певних подій. Це гарантує, що їхня присутність залишиться навіть після перезавантаження системи або застосування заходів безпеки. Також не варто забувати, що маніпулюючи налаштуваннями реєстру, вони можуть вимкнути або змінити функції безпеки, змінити конфігурацію системи або поведінку встановленого програмного забезпечення. Ці модифікації дозволяють їм обходити засоби захисту, уникати виявлення антивірусним програмним забезпеченням, що робить їх безкорисними в такому випадку, оскільки загрозу практично неможливо виявити.

Тому в програмі доступний пошук значень/ключів на підключеном клієнті. Наприклад, можна здійснити пошук по значенням, які залишає відомий та найнебезпечніший на сьогодні[17] вірус-шифрувальник Lockbit за собою в системі[18]:

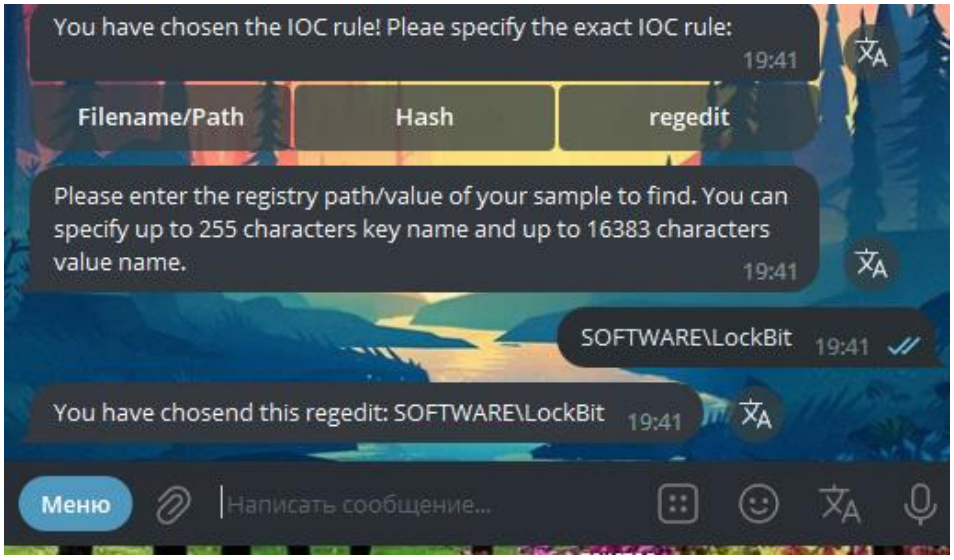


Рисунок 2.19 – Пошук по значенню в реєстрі

Алгоритм проходить по всіх ключах реєстру Windows:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_CURRENT_USER_LOCAL_SETTINGS
- HKEY_LOCAL_MACHINE
- HKEY_PERFORMANCE_DATA
- HKEY_PERFORMANCE_NLSTEXT
- HKEY_PERFORMANCE_TEXT
- HKEY_USERS

та шукає задані користувачем значення по всьому реєстру, використовуючи схожий алгоритм як і у випадку з файлами – є змога здійснювати пошук індикаторів йде не лише по його імені, а й по значенню його ключів, що знаходяться на рівень вище. Такий підхід дозволяє знаходити не тільки записи та їх значення, а й самі ключі, що є потужним рішенням, оскільки доволі часто

зловмисники маскуються і приховують свою присутність змінюючи локації своїх атрибутів.

Для проходження по реєстру використовується наступний алгоритм:

```
if (RegEnumValueA(hKey, i, achValue, &cchValue, NULL, NULL, NULL, NULL) == ERROR_SUCCESS)
{
    string str_to_check;
    if (cchValue == 0) // if name of subkey is empty
    {
        if ((signal_var == 0) && (i + 1 == cValues)) // if its the last empty value then c
        {
            str_to_check = key_val + "\\" + szFullPattern;
        }
    }
    else // if it is non empty value then simply  cout it without problems
    {
        str_to_check = key_val + "\\" + szFullPattern + achValue;
        signal_var = 1;
    }
    //cout << str_to_check << endl;
    if (strstr(str_to_check.c_str(), str_to_find.c_str()) != NULL) // check if this reg pa
    {
        string data_to_send = str_to_check + "\n";
        SendDataToSocketConnectio(data_to_send);
        cout << str_to_check << endl;
    }
    //memset(achValue, 0, sizeof(TCHAR) * MAX_VALUE_NAME);
}
```

Рисунок 2.20 – Фрагмент коду обходу реєстру Windows

Він дозволяє зменшити надмірні сповіщення, оскільки ключ може містити в собі ще декілька ключів, що лавинно зіблшує кількість записів та ускладнює аналіз загрози. За допомогою фукнції RegEnumValueA алгоритм проходить по кожному ключу і перевіряє чи пустий він, та виводить лише останнє значеня, що відповідає заданом. Функція для роботи з реєстром також є рекурсивними, що дозволяє ефективно розподілювати ресурси та час виконання – максимальний час пошуку займає до 1 хв.

2.4 BIOC правила

У той час як IOC фокусується на конкретних артефактах або патернах, пов'язаних з відомими загрозами, правила BIOC зосереджуються на поведінці та методах, що використовуються супротивниками, забезпечуючи проактивний і контекстно-орієнтований підхід до виявлення загроз. Оскільки вони не прив'язані до конкретних відомих індикаторів, вони можуть виявляти зловмисну діяльність, яка не описана традиційними підходами, що базуються на IOC. Відстежуючи підозрілу поведінку або незвичайні шаблони, правила BIOC можуть позначити потенційні загрози, які ще не були пов'язані з відомими IOC правилами і для яких ще заранння невідомі.

Після пошуку статичних можна перейти до більш цікавого – пошук динамічних. Для виявлення присутності доступні наступні поведінкові правила:

- **Process Creation** – моніторить створення певного процесу в клієнті
- **Process Generation** – моніторить породження певного процесу іншим (відношення parent -> child)

Стандартні засоби системи Windows такі як Диспетчер задач є не дуже корисними в ситуаціях виявлення витончених загроз, оскільки виконання певного процесу або ж його породження іншим може займати долі секунди, що не завжди кидається в очі при моніторингу з допомогою таких утиліт.

Саме тому, виявлення поведінкових загроз в розробленій програмі базується на постійному парсингу системного знімка процесів. Для цього було використано WinAPI функцію CreateToolhelp32Snapshot, яка створює дескриптор знімку процесів всієї системи. Отримавши цей дескриптор, ми можемо використовувати його разом з іншими функціями WinAPI, такими як Process32First і Process32Next, для ітераційного перегляду процесів у знімку. Ці функції дозволяють отримати інформацію про кожен процес, зокрема ідентифікатори процесів, ідентифікатори батьківських процесів, імена виконуваних файлів тощо. Перерахувавши всі процеси за допомогою такого

механізму, можна проаналізувати характеристики процесів і виявити потенційні механізми присутності зловмисних програм.

```
HANDLE hEvent = CreateEventA(NULL, TRUE, TRUE, NULL);
bioc_handles.push_back(hEvent);
PROCESSENTRY32 ProcInfo;
HANDLE PSnapshot;
vector<DWORD>ProcIDs;
while (WaitForSingleObject(hEvent,0) == WAIT_OBJECT_0)
{
    ProcInfo = { 0 };
    ProcInfo.dwSize = sizeof(ProcInfo);
    PSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    Process32First(PSnapshot, &ProcInfo);
    do
    {
        if ((string)ProcInfo.szExeFile == ProcessName && !vec_contain(ProcIDs, ProcInfo.th32ProcessID))
        {
            SYSTEMTIME time_struct;
            GetLocalTime(&time_struct);
            string data = "Process " + (string)ProcInfo.szExeFile + " with PID = " +
                to_string((int)ProcInfo.th32ProcessID) + " has been created!";
            SendDataToSocketConnectio(data);
            cout << data << endl;
            ProcIDs.push_back(ProcInfo.th32ProcessID);
        }
    } while (Process32Next(PSnapshot, &ProcInfo));
    CloseHandle(PSnapshot);
}
```

Рисунок 2.21 – Фрагмент коду проходження знімка процесів операційної системи

Варто відмітити, що парсинг цього знімку повинен відбуватись без зупинок, щоб не пропустити жоден новостворений процес, тому прийняте рішення виділити роботу поведінкових правил в окремий потік, щоб не заважати функціонуванню всієї клієнтської частини. На стороні сервера створюється файл Database.json, в який записуються всі створені й запущені динамічні правила для зручного управління ними:

```
data_from_file = read_data_from_database()

data_from_file["running_tasks"][proc_crt_gnrt_val] =  {
    "BIOC_rule":"PCRT",
    "Process_name": message.text,
    "time":time.strftime("%H:%M", time.localtime()),
}
write_data_to_database(data_from_file)
```

Рисунок 2.22 – База даних у форматі json для утримання поведінкових правил

Після цього, клієнту передається інформація про запуск нового поведінкового правила, з яким пов'язується виклик функції CreateEvent() – створення події у сигнальному стані, що відповідає за це правило та відображає його активний стан. Тому, в програмі є можливість використовувати паралельну обробку знімку процесів, з легкістю переглядати запущені правила та термінувати їх в разі необхідності.

Також передбачений механізм обробки повторних, вже відомих, процесів. Тобто, якщо програма вже знайшла бажаний процес, тоді його вже не потрібно відображати ще раз після повторних проходжень дескриптора знімку, оскільки цей процес рахується вже виявленим й алерт про нього вже надісланий.

Щодо Process Generation, то механізм доволі схожий – відбувається парсинг процесів наявних в системі. Підхід відрізняється від попереднього лише в додаванні циклу if() – здійснюється перевірка батьківського процесу, що породив заданий.

```
if ((string)ProcInfo.szExeFile == ChildProcessBIOC)
{
    char buffer[MAX_PATH];
    HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, ProcInfo.th32ParentProcessID);
    GetProcessImageFileNameA(hProcess, buffer, MAX_PATH);

    string ParentProcess = (string)buffer;
    size_t found = ParentProcess.find_last_of("\\");

    if (ParentProcess.substr(found + 1) == ParentProcessBIOC && !vec_contain(ProcIDs, ProcInfo.th32Proc
    {
        string data = "Process " + ParentProcess.substr(found + 1) + " with PID = " + to_string((int)Pr
        SendDataToSocketConnectio(data);
        cout << "Process " << ParentProcess.substr(found + 1) << " with PID = " <<
            ProcInfo.th32ParentProcessID << " create process " << ChildProcessBIOC << " with PID = " <<
        ProcIDs.push_back(ProcInfo.th32ProcessID);
        //return;
```

Рисунок 2.23 – Фрагмент коду, що відповідає за моніторинг породження процесів

Process Generation дуже корисний в тих випадках, коли зловмисник використовує інші програми для запуску ШПЗ, що буває складно виявити за допомогою стандартних наборів антивірусних засобів. Для демонстрації було проведено одну із розповсюджених атак проникнення в організацію[19] через фішинг та соціальну інженерію, використовуючи зловмисний макрос в Word документі. В ролі ШПЗ створено макрос, що запускає powershell:

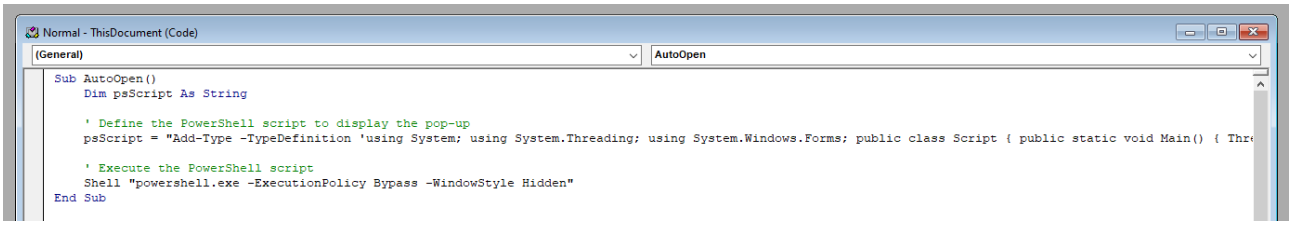


Рисунок 2.24 – Макрос для виклику консолі Powershell

Параметр -ExecutionPolicy Bypass використовується для обходу політики виконання, яка визначає рівень безпеки запуску сценаріїв PowerShell. Вказівка Bypass дозволяє виконувати сценарії незалежно від поточної політики виконання. Параметр -WindowStyle Hidden використовується для приховування вікна PowerShell під час виконання сценарію. Якщо вказати Hidden, вікно PowerShell не буде видимим для користувача під час виконання сценарію. Ці параметри зазвичай використовуються в сценаріях, де потрібно запустити сценарій без необхідності взаємодії з користувачем або видимості. Це дозволяє скрипту безшумно виконуватися у фоновому режимі без відображення будь-яких вікон або підказок.

Підпрограма Sub AutoOpen може бути використана для автоматизації певних процесів або додавання користувацької функціональності до документа Word. Вона позначає спеціальну секцію програми, яка виконується автоматично при відкритті документа Word, що дозволяє визначити набір інструкцій, які будуть виконані, одразу після того як користувач відкриє документ.

Поєднання -WindowStyle Hidden та Sub AutoOpen є частою тактикою зловмисників, так я дозволяє виконати скрипт без видимого вікна консолі та при здійсненні легальних дій – відкриття документа.

Після застосування цього коду до тестового документа можна налаштувати правило Process Generation для цільової системи виявлення:

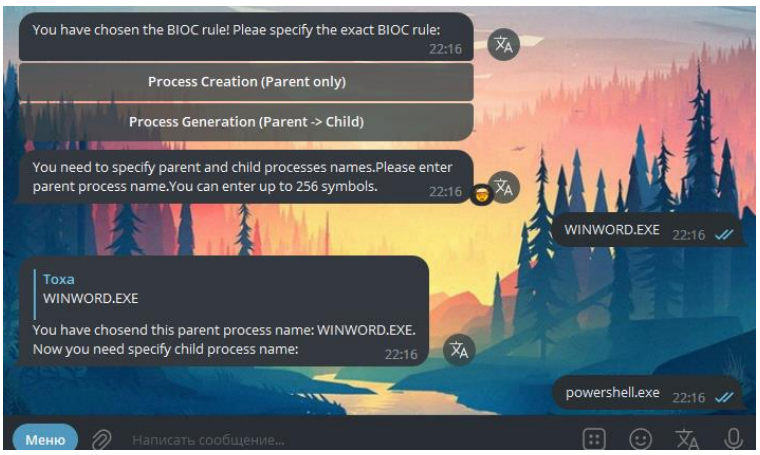


Рисунок 2.25 – BIOC Process Generation

Після запуску зловмисного файлу з ШПЗ програма видає відповідне повідомлення після парсингу знімку процесів:

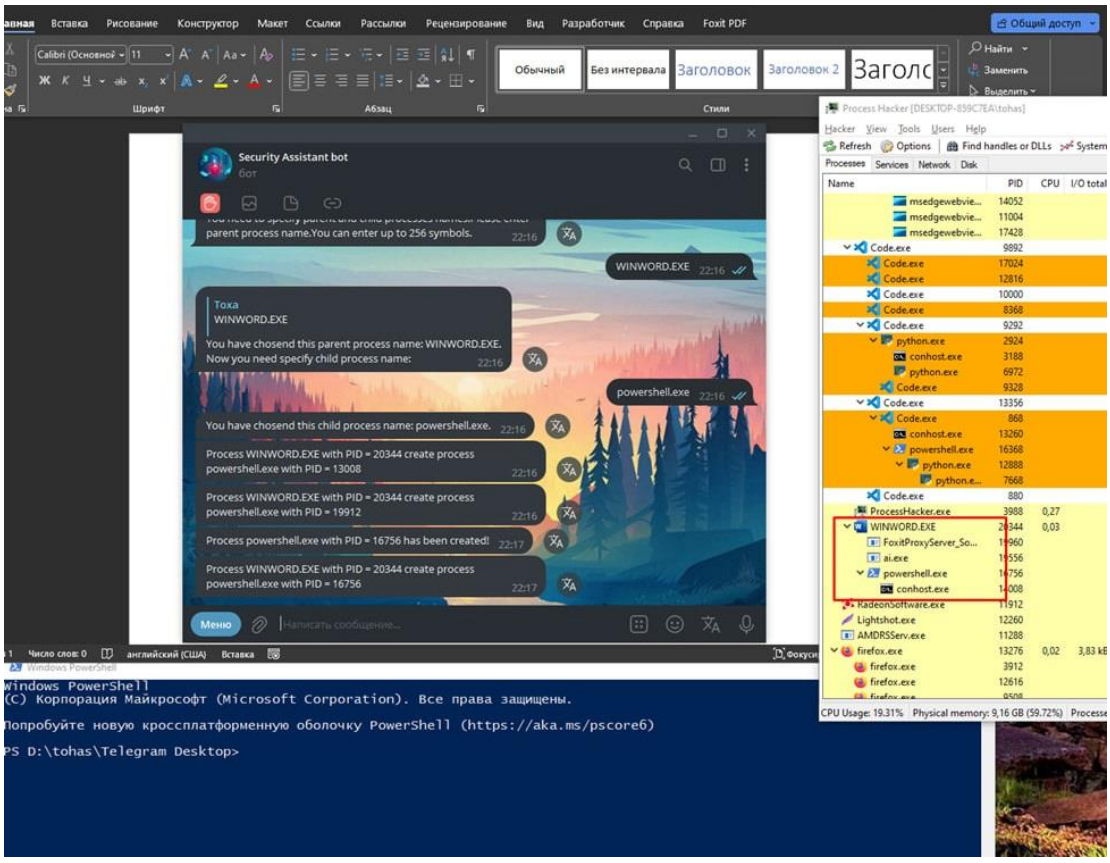


Рисунок 2.26 – Результати виявлення зловмисного породження процесів

Правило успішно спрацювало та надіслало відповідний алерт про це. Тому, навіть якщо поява вікна не буде помічена користувачем – клієнтська

частина сповістить про це на сервер, оскільки було виявлене породження процесів згідно заданого правила BIOC.

Висновки до розділу 2

В другому розділі було описано проектування та створення власної системи виявлення присутності загроз на основі їх статичних та поведінкових індикаторів (атрибутів).

Щодо моделі комунікації – обрано клієнт-серверну архітектуру з комунікацією через сокети. Сервером виступає віртуальний сервер в хмарному середовищі Amazon. Панель управління сервера реалізована через телеграм бот, мова програмування – Python. Клієнтська частина написана на низькорівневій мові програмування – C++ з використанням нативних функцій WinAPI для гнучкої та ефективної взаємодії з операційною системою.

Було розглянуто реалізацію статичних та поведінкових індикаторів детектування, продемонстровані приклади виявлення загроз на основі імені файлу, його шляху, хешу, записів/ключів реєстра Windows для зарання відомих загроз та поведінкові правила на основі створення та породження процесів, що допоможе знайти невідомі загрози.

3 ВДОСКОНАЛЕННЯ СИСТЕМИ ЗА РАХУНОК ВПРОВАДЖЕННЯ МАШИННОГО НАВЧАННЯ

3.1 Опис проблеми та її вирішення з допомогою моделі машинного навчання

У сучасному цифровому ландшафті кількість і складність кіберзагроз стрімко зростає. Зловмисники використовують все більш складні методи, щоб зламати захист мережі та отримати несанкціонований доступ до систем. Традиційним підходи до безпеки, які створюються в системі з допомогою статичних правил і сигнатур часто не вдається виявити нові або раніше невідомі атаки, що робить організації вразливими до кібератак, витоків даних та інших інцидентів безпеки.

Після закріплення в системі зловмисник зазвичай спілкується з нею зі своєї приватної мережі. Така мережева активність є вкрай непомітною для звичайного користувача, оскільки кількість пакетів, байтів та мережевих з’єднань за день може досягати колосальних чисел. Виникає необхідність в автоматичній обробці всіх цих з’єднань та виявленні підозрілих, які відхиляються від певних патернів.

Машинне навчання — це підмножина штучного інтелекту, яка зосереджена на розробці алгоритмів і моделей, що дозволяють комп’ютерам навчатися та робити прогнози чи рішення без явного програмування. Вона передбачає використання статистичних методів і математичних моделей для навчання комп’ютерів великими обсягами даних, що дозволяє їм ідентифікувати закономірності, розпізнавати аномалії та робити прогнози чи класифікацію. У кібербезпеці машинне навчання відіграє вирішальну роль у виявленні підозрілих мережевих підключень.

Алгоритми машинного навчання чудово виявляють аномалії або відхилення від нормальної поведінки. Навчаючи моделі на великих обсягах

даних мережевого трафіку, системи машинного навчання можуть навчитися розпізнавати закономірності та характеристики нормальної поведінки мережі. Зустрічаючи підозрілі мережеві підключення, які відхиляються від цих вивчених шаблонів, модель машинного навчання може позначати їх як потенційні загрози, навіть якщо конкретна атака раніше не була помічена. Ця можливість особливо цінна для виявлення зловмисних мережевих з‘єднань, що їх робить система, які буквально “зливаються” у величезній кількості легального трафіку.

Тому, для виявлення таких загроз, було прийняте рішення натренувати та застосувати одну із моделей машинного навчання для автоматизації та покращення захисту системи.

3.2 Пошук датасета та вибір моделі машинного навчання

Модель машинного навчання потребує датасет для навчання, оцінювання та налаштування своїх параметрів. Датасет слугує вхідними даними для алгоритмів машинного навчання і має вирішальне значення для того, щоб модель змогла навчитись і робити точні прогнози або приймати рішення. Датасет складається з вхідних даних (ознак) і відповідних вихідних або цільових значень (міток). За допомогою різноманітного та репрезентативного набору даних модель може вивчити основні закономірності, зв'язки та залежності між ознаками та міткою. Цей процес навчання дозволяє моделі узагальнювати і робити прогнози або приймати рішення на основі нових, небачених раніше даних.

На етапі навчання модель ітеративно налаштовує свої внутрішні параметри на основі вхідних даних і бажаних результатів. Порівнюючи передбачення моделі з фактичними мітками в датасеті, модель оновлює свої параметри, щоб мінімізувати різницю (похибку) між передбаченими і

фактичними значеннями. Цей процес оптимізації, який часто називають навчанням, дозволяє моделі з часом покращувати свою продуктивність.

CTU-13 [20] - це датасет мережевого трафіку, який часто використовується в дослідженнях кібербезпеки та експериментах з машинного навчання. Він був створений Чеським технічним університетом (Czech Technical University - CTU) в Празі і складається із перехопленого мережевого трафіку, зібраного з реального мережевого середовища. З офіційного сайту стало відомо, що датасет складається з трафіка різних джерел (пісочниці, ханіпоти, сайти), що був записаний в контрольованому середовищі. Також, деякий трафік було записано з інтерфейсу віртуальної машини, що заражена реальним ШПЗ. Мета цього датасета - надати широкий спектр різних типів мережевого трафіку, включаючи як звичайну, легальну, так і зловмисну діяльність. Цей датасет є стандартизованим еталоном для оцінки ефективності та точності різних підходів у виявленні різноманітних типів атак і аномальної поведінки мережі. Нормальний трафік в даному датасеті відповідає звичайній роботі пошти, серфінгу в інтернеті, гортанню соціальних мереж.

Ознайомившись з дослідженням[21], було прийняте рішення як алгоритм машинного навчання використовувати ***RandomForest***. Це алгоритм навчання в ансамблі, який поєднує кілька дерев рішень для прогнозування. Кожне дерево рішень навчається на різних підмножинах даних, і остаточний прогноз визначається шляхом агрегування прогнозів усіх окремих дерев. Цей ансамблевий підхід допомагає підвищити точність і надійність моделі. Алгоритм RandomForest був оцінений багатьма дослідженнями як алгоритм машинного навчання, який має багато видатних переваг порівняно з іншими алгоритмами класифікаційного машинного навчання. Результат прогнозується шляхом отримання більшості голосів від окремих дерев.

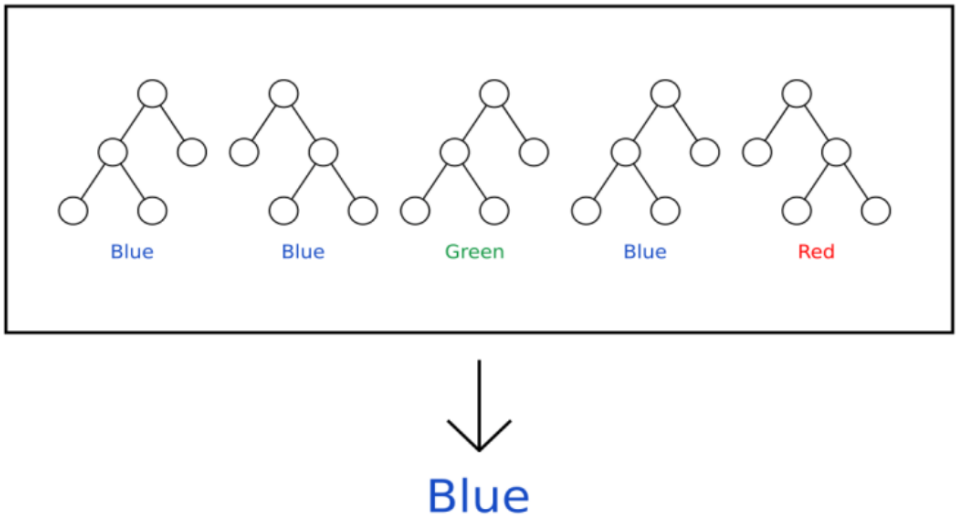


Рисунок 3.1 – Схематичне зображення алгоритму Random Forest

3.3 Обробка датасета та підготовка для роботи

Після скачування та зчитування датасета виявилось, що його розміри становлять 1925149 рядків та 15 колонок, кожна з яких описана в таблиці 3.1:

Таблиця 3.1- Опис полів датасета

Назва колонки	Опис
StartTime	Час початку атаки
Dur	Тривалість атаки (в секундах)
Proto	Протокол, що був використаний для передачі даних
SrcAddr	IP-адреса джерела
Sport	Порт джерела
Dir	Напрямок трафіку
DstAddr	IP-адреса приймача
Dport	Порт приймача

Кінець таблиці 3.1

State	Стан транзакції згідно з протоколом
sTos	Поле Tos (Type of Service) джерела
dTos	Поле Tos (Type of Service) приймача
TotPkts	Загальна кількість пакетів мережевого з'єднання
TotBytes	Загальна кількість байтів мережевого з'єднання
SrcBytes	Загальна кількість байтів мережевого з'єднання від джерела до приймача
Label	Три унікальні мітки трафіку: фоновий, ботнет і звичайний.

Колонки, що не можуть бути перетворені в загальний числовий вид та не представляються змістовної інформації були видалені. У датасеті CTU-13 такі стовпці, як «StartTime», «SrcAddr», «sTos», «DstAddr» і «dTos» було вилучено, щоб зменшити кількість таких даних. Після перегляду колонки Proto, що містить протокол передачі даних було також видалено всі значень окрім tcp, udp та icmp – основних та часто використовуваних протоколів, що будуть тестуватись у дані роботі. В колонці State було видалено всі рідко-використовувані значення, що також не будуть корисним при роботі системи виявлення. Нарешті, колонки портів джерела та приймача були очищені від пустих (NaN) значень в датасеті, що може спричиняти невизначеності та помилки під час тренування моделі.

```
data = data.drop(["StartTime", "DstAddr", "SrcAddr", "sTos", "dTos"], axis=1)

data = data[data["Proto"].str.contains("ipnip|llc|unas|udt|pim|ipx/spx|ipv6-icmp|ipv6|igmp|rtp|rtcp|arp") == False]
data = data[data["Dir"].str.contains("\?") == False]

data = data[data["State"].str.contains("UNK|URNPRO|DCE|URO|URF|SRC|RSP|URN|URFIL|URHPRO") == False]

data = data.dropna(subset=['Sport', 'Dport'])
```

Рисунок 3.2 – Видалення непотрібних даних з датасета

Як відомо, протокол ICMP не використовує порти для свого зв'язку, натомість використовує спеціальні коди, зазначені в заголовку пакета. Тому, в колонці Sport та Dport, що відповідають ICMP протоколу містились нечислові значення, що були надалі замінені на 0. Після цього самі колонки Sport та Dport були конвертовані в числовий тип даних int.

```
data.loc[data['Dport'].str.contains('0x'), 'Dport'] = 0
data.loc[data['Sport'].str.contains('0x'), 'Sport'] = 0
data['Sport'] = data['Sport'].astype('int')
data['Dport'] = data['Dport'].astype('int')
```

Рисунок 3.3 – Очистка колонок, що містять значення портів

Як вже раніше сказано, колонка State відповідає за стан з'єднання. Вона містить значення, що відповідають встановленим прапорам під час передачі пакетів або встановленню та контролю сесії з'єднання (наприклад TCP 3-way handshake). Оскільки, деякі значення протоколів що не є вкрай поширеними були очищені, тому потрібно видалити й їх значення станів з'єднання. Згідно дослідження[22], були залишені тільки основі прапори, роз'яснення яких наведено в таблиці 3.3.2:

Таблиця 3.2 – Опис станів протоколів

Значення	Опис
CON	SYN + ACK прапори в TCP з'єднанні
FIN	Нормальне завершення з'єднання TCP (FINISH)
INT	Протоколи без збереження стану (udp)
REQ	SYN прапор в TCP з'єднанні
RST	Примусове завершення з'єднання TCP (RESET)
ACC	ACK прапор в TCP з'єднанні.
URP	Unreachable port (ICMP тип 3, код 3)

Кінець таблиці 3.2

ECO	Echo Request (ICMP тип 8, код 0)
ECR	Echo Reply (ICMP тип 0, код 0)
RED	Redirect for icmp (ICMP тип 5, код 1-4)
TXD	Time Exceeded (ICMP тип 11)

Модель машинного навчання працює тільки з числовими даними, тому датасет повинен містити дані тільки такого типу. В попередніх прикладах було розглянуто, що колонки, які містять протокол, напрям та стан мають лише строковий тип даних, що не дасть коректно працювати алгоритму. Для вирішення даної ситуації застосовано ***OrdinalEncoder*** —техніка попередньої обробки даних, яка використовується в машинному навчанні для кодування нечислових змінних у числове представлення.

```
encoder = OrdinalEncoder()
data[["Proto", "Dir", "State"]] = encoder.fit_transform(data[["Proto", "Dir", "State"]])
```

Рисунок 3.4 – Кодування даних датасету у числовий вид

Метод ***fit_transform*** поєднує кроки підгонки та перетворення даних і використовується на заданих даних для перетворення у їх закодовані представлення.

Нарешті, можна приступити до обробки останньої колонки – Label. Після перегляду її вмісту стало відомо, що вона також містить багато строкових типів даних, тому їх потрібно перетворити в числовий, проте перед цим потрібно згадати її призначення – мітка для вердикту даних.

```
data.Label.value_counts()
[8] ✓ 0.0s
```

flow=To-Background-UDP-CVUT-DNS-Server	756280
flow=Background-UDP-Established	673288
flow=Background-TCP-Established	147196
flow=Background-UDP-Attempt	42312
flow=Background-TCP-Attempt	37105
flow=From-Botnet-V54-TCP-Attempt	22463
flow=Background-Established-cmpgw-CVUT	20527
flow=From-Normal-V54-Stribrek	20299
flow=Background	12813
flow=From-Botnet-V54-UDP-DNS	7523
flow=To-Background-CVUT-Proxy	6659
flow=From-Botnet-V54-TCP-Attempt-SPAM	6617
flow=To-Background-CVUT-WebServer	6196
flow=Background-UDP-NTP-Established-1	5822
flow=Background-Attempt-cmpgw-CVUT	2987
flow=From-Normal-V54-Grill	2955
flow=From-Normal-V54-Jist	814
flow=From-Botnet-V54-TCP-Not-Encrypted-SMTP-Private-Proxy-1	614

Рисунок 3.5 – Перегляд колонки Label в датасет

Відповідно, вона повинна приймати лише значення 0 (легальний трафік) та 1 (підозрілий трафік). Для цього застосовано функцію для перевірки вмісту колонки – якщо рядок містить значення “Botnet”, то це підозрілий трафік, в протилежному випадку – легальний:

```python
def crete_bool_label(input_flow):
    if "Botnet" in input_flow:
        return 1
    else:
        return 0

data["Label"] = data["Label"].apply(crete_bool_label)
```

Рисунок 3.6 – Функція перетворення мітки (y) в числовий тип

Після повторного перегляду датасету видно, що в ньому присутні лише необхідні для тренування моделі колонки, та лише числові дані:

	Dur	Proto	Sport	Dir	Dport	State	TotPkts	TotBytes	SrcBytes
0	1.307281	1.0	14652	0.0	443	8.0	6	366	186
1	2571.371094	2.0	50169	1.0	13363	1.0	4	272	152
2	16.174158	1.0	51006	0.0	80	4.0	10	1562	442
3	0.140191	1.0	4337	0.0	80	4.0	10	1663	777
4	1.347865	1.0	3061	0.0	443	8.0	6	366	186

Рисунок 3.7 – Датасет після обробки

Коли кількість певних значень датасета в порівняні з іншим значенням є дуже висока виникає інша проблема - незбалансований датасет. Це створює додаткову проблему для машинного навчання, оскільки впливає на здатність моделі вивчати параметри та робити точні прогнози. Переглянувши кількість класів Label значень, можна помітити, що датасет має таку проблему:

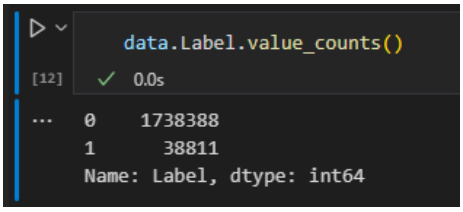


Рисунок 3.8 – Незбалансований датасет

Для вирішення цієї проблеми застосовано техніку – **Undersampling** – штучне видалення вибірки з класу більшості до прирівняння до кількості меншості. Це робиться випадковим чином й тому створюється більш збалансоване представлення даних, зменшуючи кількість домінантного класу.

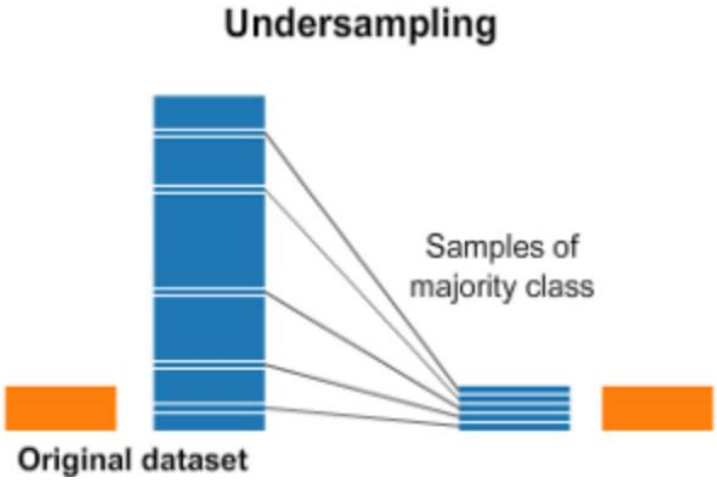


Рисунок 3.9 – Графічне зображення техніки Undersampling

Після повторного перегляду датасета та кількості кількість класів Label програма видає однакові результати, що свідчить про повну збалансованість програми:

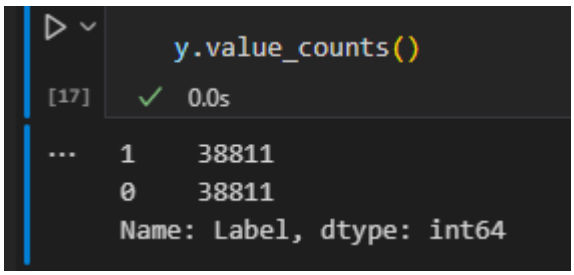


Рисунок 3.10 – Збалансований датасет після застосування техніки Undersampling

3.4 Перехоплення трафіка на машині клієнта

Після підготовки та обробки датасета й тренування моделі потрібно подбати про находження до неї даних для винесення вердикту на основі параметрів. Для цього необхідно реалізувати перехоплення та парсинг мережевого трафіку клієнтської машини та надсилання його на серверну частину для винесення вердикту.

Оскільки мова програмування клієнтської частини C++, тому вибір впав на бібліотеку з доволі широким функціоналом - Npcap. Npcap — це бібліотека для перехоплення пакетів і аналізу мережі для операційних систем Windows. Це проект із відкритим вихідним кодом, який забезпечує розширені можливості захоплення пакетів і призначений для заміни старішої бібліотеки WinPcap. Npcap пропонує покращені можливості захоплення пакетів порівняно зі стандартними механізмами захоплення пакетів Windows. Він надає такі функції, як loopback перехоплення, перехоплення Wi-Fi 802.11 і підтримує розширену фільтрацію пакетів. Npcap оптимізовано для високопродуктивного захоплення та аналізу пакетів. Він забезпечує високу швидкість і ефективність захоплення мережевих пакетів, дозволяючи обробляти великі обсяги мережевого трафіку.

Проте, перед тим як перехоплювати мережеві дані комп’ютерної системи, потрібно ідентифікувати активний адаптер, через який йде передача пакетів в

інтернет. В сучасних робочих машинах таких адаптерів може бути доволі багато, що ускладнює завдання (VPN інтерфейс, інтерфейс для віртуальних машин, безпровідний інтерфейс, інтерфейс Ethernet та ін.). Тому, згідно знайденого дослідження[23] розроблено нехитрий алгоритм автоматичної ідентифікації активного адаптера, що використовується. Спочатку потрібно отримати список всіх адаптерів – для цього використано функцію GetAdaptersAddresses(). Це функція, яка надається Windows API та отримує інформацію про всі мережеві адаптери, встановлені в системі. Використовуючи її, можна отримати достатньо детальну інформацію про мережеві адаптери, таку як: IP-адреса, MAC-адреса, MTU, IfIndex, OperationStatus, Flags, Ім’я та іншу. Маючи всю цю інформацію, можна написати алгоритм для визначення активного адаптера:

1. Вилучення всіх неактивних адптерів (тобто ті, що виключені й не використовуються в даний момент) – для цього OperStatus адаптера повинен бути рівний 1.
2. Вилучення адаптера Loopback інтерфейс IfType якого = 24
3. Проходження по масиву всіх адаптерів та знаходження того, IfIndex якого дорівнює мінімальному значенню з наявних – це і буде активний адаптер

```
//find the index of adapter
while (pCurrAddresses)
{
    if ((pCurrAddresses->OperStatus == 1) && (pCurrAddresses->IfType != 24))
    { // The interface is up and able to pass packets; not a software loopback network interface.

        if (pCurrAddresses->IfIndex < minIfIndex)
        {
            minIfIndex = pCurrAddresses->IfIndex;
            NetAdapterName = "\\Device\\NPF_" + (string)pCurrAddresses->AdapterName;
            NetAdapterDesc = pCurrAddresses->Description;

            IP_ADAPTER_UNICAST_ADDRESS* unicast = pCurrAddresses->FirstUnicastAddress;
            sockaddr_in* addr = (sockaddr_in*)unicast->Address.lpSockaddr;
            inet_ntop(AF_INET, &(addr->sin_addr), ipAddress, INET_ADDRSTRLEN);
        }
    }
    pCurrAddresses = pCurrAddresses->Next;
}
```

Рисунок 3.11 – Алгоритм пошуку активного адаптера

Алгоритм протестований у трьох різних ситуаціях (з безпровідним з'єднанням, з Ethernet з'єднанням та з VPN з'єднанням) й всюди показав правильний адаптер. Після ідентифікації адаптера отримуємо його фізичне ім'я з поля AdapterName та можемо використовувати для перехоплення трафіку, що йде через цей адаптер.

Ідентифікувавши та отримавши ім'я активного адаптера, можна приступити до перехоплення та парсингу мережевого трафіка. При обробці датасета, було згадано, що для перехоплення визначені основні три протоколи: ICMP, TCP, UDP. Щоб з легкістю отримувати доступ і маніпулювати різними значеннями заголовків цих протоколів визначимо структури ip_header, tcp_header та udp_header

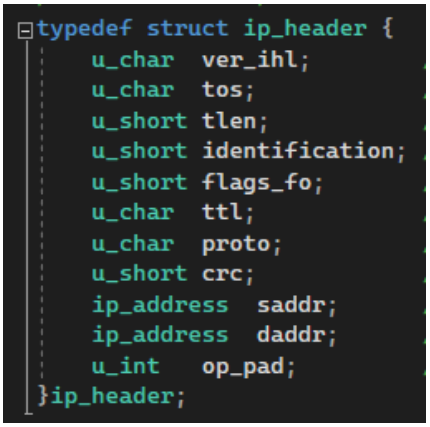


Рисунок 3.12 – Структура ip заголовка

Фрагмент коду на Рисунку 3.4.2 визначає структуру під назвою ip_header, яка представляє структуру заголовка пакета IPv4. Ця структура дозволяє аналізувати та витягувати інформацію з IP-заголовка захоплених пакетів. Отримавши доступ до полів у структурі, є можливість отримати такі деталі, як IP-адреси джерела та призначення, тип протоколу, довжина пакета тощо. Ця інформація є дуже важливою для аналізу мережевого трафіку.

Подібним чином були визначені структури для TCP та UDP заголовків:

```
typedef struct udp_header {
    u_short sport;
    u_short dport;
    u_short len;
}udp_header;

typedef struct tcp_header {
    u_short sport;
    u_short dport;
    u_int seq_num;
    u_int ack_num;
    u_char offset_reserved;
    u_char flags;
} tcp_header;
```

Рисунок 3.13 – Структури TCP та UDP заголовків

Тепер маючи доступ до полів заголовків наших протоколів, можемо приступити до перехоплення цих даних та надсилання їх для перевірки на сервер. Перехопивши пакет даних та обрахувавши позицію IP заголовка (припускаємо, що довжина Ethernet заголовка дорівнює 14 байтам) потрібно отримати значення в полі proto цього заголовка. У цьому полі вказується числове значення протоколу верхнього рівня, який отримуватиме корисне навантаження пакета на вузлі призначення. Наприклад, якщо це поле містить десяткове значення 17, тоді інтернет-рівень вузла призначення передає корисне навантаження на протокол UDP. Для протоколу TCP це значення дорівнює 6, ICMP – 1. Це дає інформацію про те, який протокол верхнього рівня очікувати й яку заранння визначену структур даних використати для коректного відображення її полів в неї.

```
if (ih->proto == 6) { ... }

else if (ih->proto == 17) { ... }

else if (ih->proto == 1) { ... }
```

Рисунок 3.14 – Визначення протокола верхнього рівня за полем IP заголовка

Після того, отримано інформацію, який протокол для передачі даних використовується – програма застосовує відповідний заголовок та збирає його

поля для передачі на сервер. Для ICMP та UDP протоколів цей процес так і відбувається, проте не для TCP.

TCP — це протокол, орієнтований на з‘єднання, що означає, що він встановлює надійне та впорядковане з’єднання між відправником і одержувачем перед передачею даних. UDP, з іншого боку, не має з’єднання, тобто не встановлює перевірене з’єднання між відправником і одержувачем. Для встановлення такого надійного з’єднання використовується трьохстороннє рукостискання – трьох-етапний метод, який вимагає, щоб клієнт і сервер обмінювалися пакетами SYN і ACK (підтвердження) до початку фактичної передачі даних. Це стандартна процедура встановлення з’єднання для цього протоколу, тому вона виконується кожного при встановленні TCP підключення. Саме тому, щоб перехопити з’єднання TCP, необхідно відстежити його трьохстороннє рукостискання, оскільки це початковий процес.

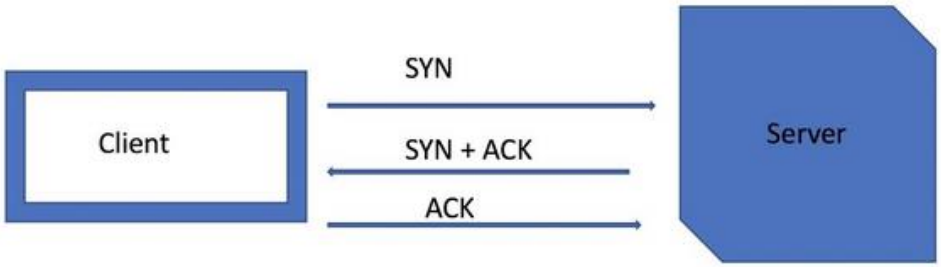


Рисунок 3.15 – Трьохстороннє рукостискання TCP

Для цього створена нова окрема структура даних, що міститиме дані про поточне TCP з’єднання та буде утримувати його стан. В її складі, окрім інших даних, міститься три булевих змінних syn_rcvd, syn_ack_rcvd та ack_rcvd, що відповідатимуть за отримані прапори під час трьохстороннього рукостискання.

```
typedef struct tcp_connection_struct{
    u_short sport;
    u_short dport;

    bool syn_rcvd;
    bool syn_ack_rcvd;
    bool ack_rcvd;
    clock_t syn_time;
    clock_t syn_ack_time;
    float syn_ack_time_sec;
    clock_t ack_time;
    float ack_time_sec;
    float total_duration;
    int total_bytes;
    int total_packets;
}tcp_connection_struct;
```

Рисунок 3.16 – Структура даних для утримання поточного TCP з‘єднання

При вирішенні цієї задачі виникла проблема зберігання таких з’єднань та утримування їхнього стану незалежно від інших. Це було вирішено з допомогою unordered_map — класа-контейнера, наданого стандартною бібліотекою C++, який реалізує асоціативний масив зі структурою даних пари ключ-значення. Він є частиною стандартної бібліотеки шаблонів C++ (STL) і доступний з C++11. Він використовується для зберігання з’єднань TCP, оскільки забезпечує ефективний спосіб пов’язати унікальний ключ (у моєму випадку ключ з’єднання) зі значенням (у моєму випадку структурою TCP_connection). Ключ з’єднання, який є комбінацією IP-адреси джерела, порту джерела, IP-адреси призначення та порту призначення, служить унікальним ідентифікатором для кожного TCP-з’єднання. Відповідно, використовуючи unordered_map, гарантується, що ключі унікальні, запобігаючи повторюваним записам. Саме тому, це дозволяє незалежно зберігати стан кожного такого з’єднання, що допомагає уникнути помилок та втрат даних під час перехоплення пакетів.

Для того щоб правильно записати наше з’єднання в структуру проводиться перевірка наявних бітів в його заголовку:

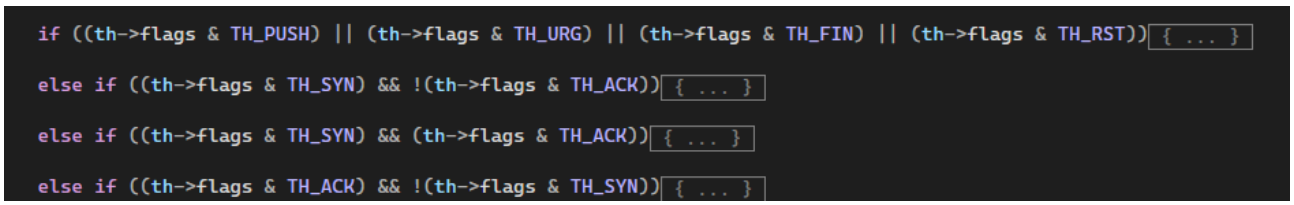


Рисунок 3.17 - Ідентифікація бітів з’єднання TCP

Й, при наявності бітів SYN, SYN+ACK, ACK програма змінює булеві змінні в структурі tcp_connection_state для фіксації цього з’єднання. Після того як відбулось успішне встановлення TCP рукостискання – починається фіксація полів даних, що передаються їх коректне зчитування.

Після відповідної класифікації дані кодуються та відправляються на сервер, де вони у зворотному напрямку декодуються та парсяться для передачі в модель машинного навчання для передбачення. Вже натренована модель проганяє дані через свій алгоритм та отримує вердикт стосовно цих мережевих даних і якщо він свідчить про шкідливе з’єднання – надсилає сповіщення про це в телеграм боті у вигляді ір-адреси й портів джерела та отримувача.

Для демонстрації роботи інспектування мережевого трафіку було проведено експеримент на віртуальній машині Windows 10 x64 зараженою шкідливими зразками програмного забезпечення “ботнетів” Rbot, DonBot, Virut й “троянів” Yakes/Dridex та Miuref. Всі вони є давно відомими шкідливими програмами, що використовуються в поєднанні з іншими в складних атаках APT груп. Для коректності спрацювань та відсутності блокувань зі сторони інших захисних рішень, у системі відключено всі наявні налаштування Windows Defender та Windows Defender Firewall.

Після завантаження на цільову машину та підключення до сервера клієнтської програми розпочинається інспектування мережевого трафіку.

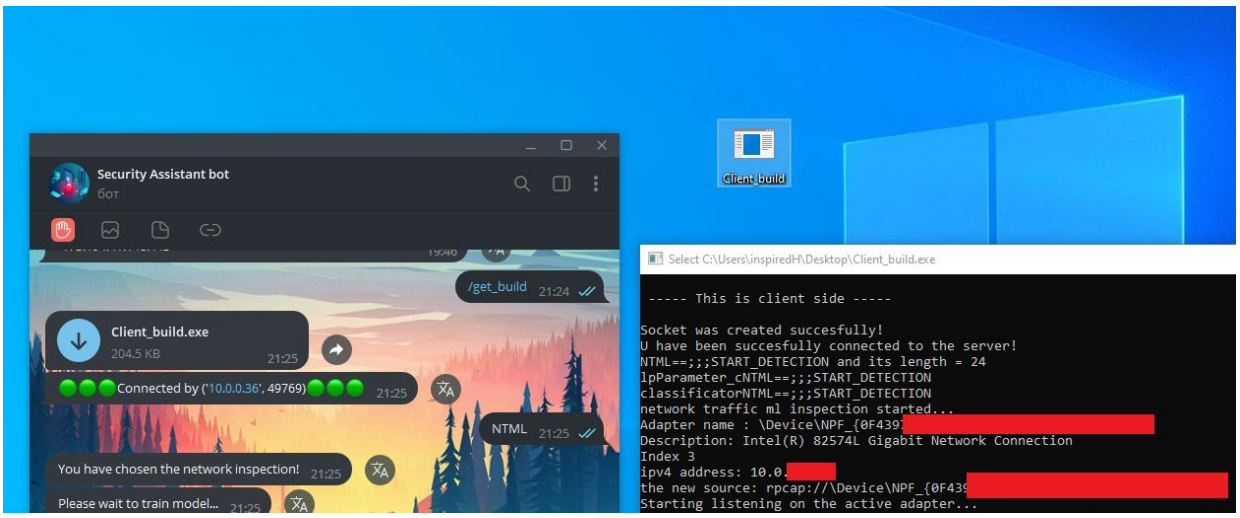


Рисунок 3.18 –Початок мережевої інспекції

На рисунку 3.18 показано, що програма автоматично вибрала правильний адаптер для перехоплення трафіку й почала прослуховування на вихідні та вхідні з’єднання до машини. Після короткого проміжку часу в телеграм боті з’явилось сповіщення про підозрілий трафік. Підозріла IP-адреса повністю співпадає з тією, що вказана в сервісі VirusTotal для запущеного зразку:

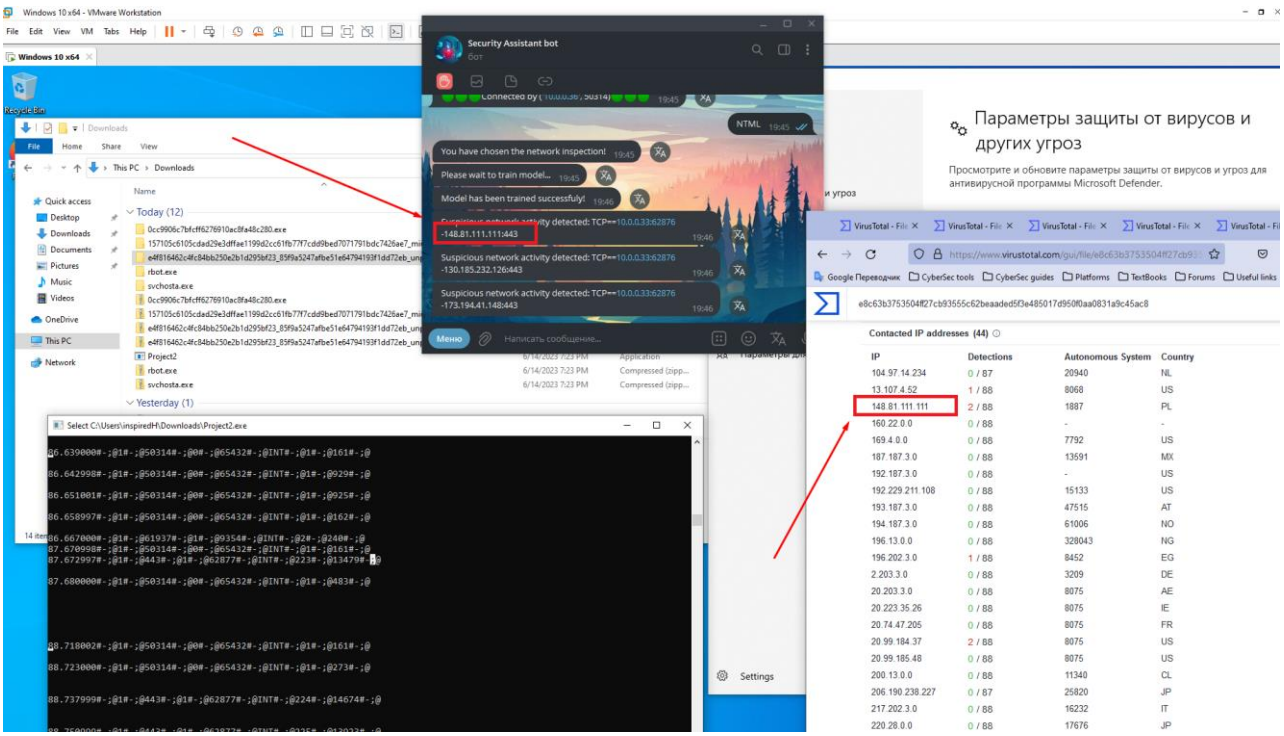


Рисунок 3.19 – Перевірка першої зловмисної адреси[24]

Також, програма знайшла ще декілька зловмисних з’єднань, а тому перевірено і їх. Як виявилось, всі вони відповідають зловмисним зразкам, що були запущені у віртуальному середовищі:

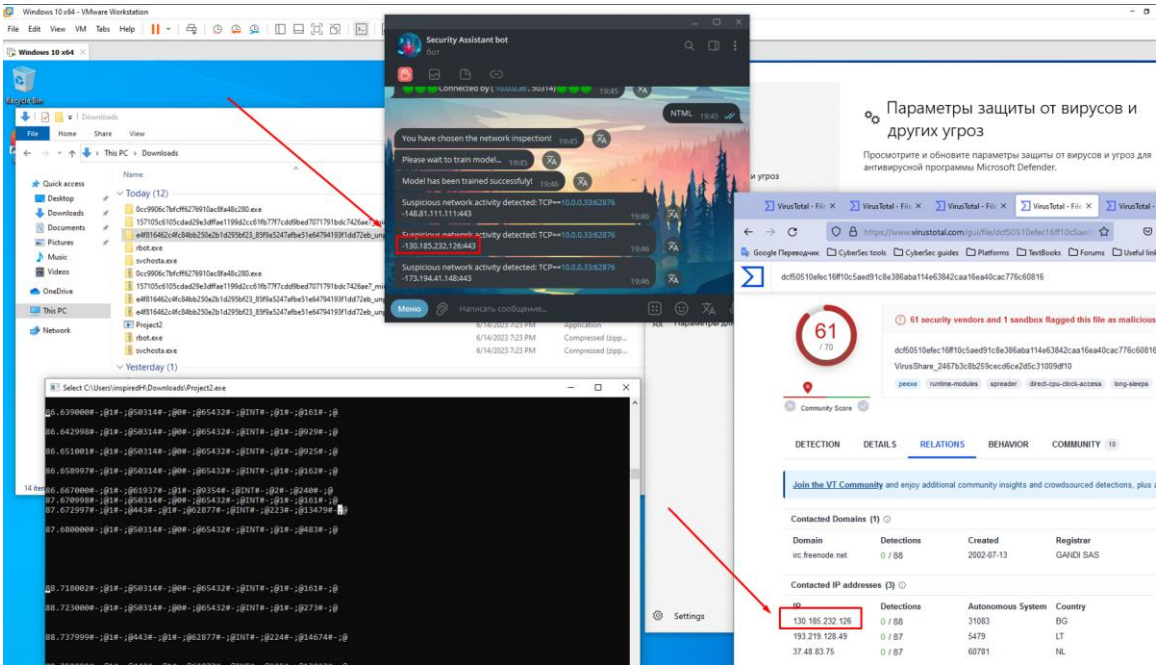


Рисунок 3.20 – Перевірка другої зловмисної адреси[25]

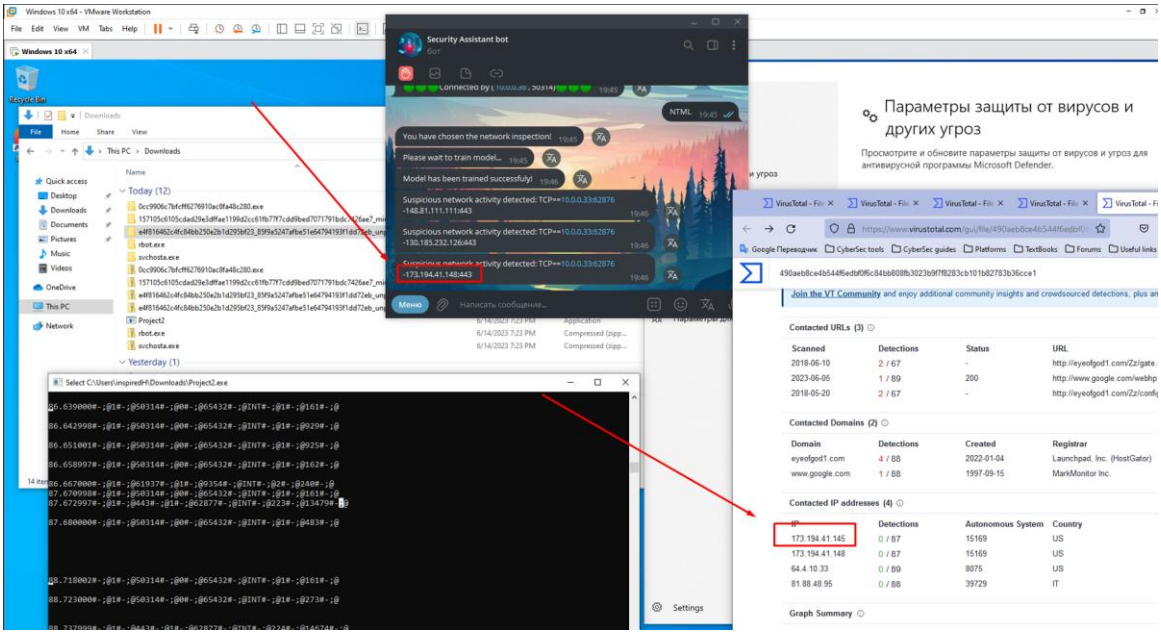


Рисунок 3.21 - Перевірка третьої зловмисної адреси[26]

Всі виявлені адреси є в переліку адрес, з якими контактує зловмисна програма під час свого виконання, а тому можна зробити висновок, що

програма відпрацювала коректно та видала правильні аналізи, що свідчать про наявність шкідливого ПЗ у робочій цільовій машині.

Висновки до 3 розділу

В третьому розділі було проведено вдосконалення власного програмного продукту для виявлення присутності загроз.

Інспектування мережевого трафіку має вирішальне значення у виявлення техніки persistence використаної зловмисниками, оскільки він служить цінним джерелом інформації про дії, що відбуваються в мережі. Коли хакер отримує несанкціонований доступ до системи або мережі, він зазвичай закріплює своє перебування, щоб підтримувати контроль і виконувати свої зловмисні цілі протягом тривалого періоду. Крім того, вони можуть спробувати викрасти конфіденційні дані, відправивши їх на свої контрольовані сервери, що є сильною загрозою для великих компаній. Саме тому, моніторинг мережевого трафіку може допомогти виявити передачу даних у невідомі або неавторизовані місця призначення, сигналізуючи про потенційні спроби викрадання даних і наявність постійної присутності загрози.

Для вирішення цієї проблеми була реалізована програмна модель машинного навчання, що пришвидшить процес виявлення мережевих індикаторів загроз на основі відхилення від певних поведінкових патернів. У своїй роботі я обрав алгоритм RandomForest, який був навчений на датасеті що містить записи легального та шкідливого мережевого трафіку – CTU-13. Цей датасет є добре відомим та популярним у сфері кібербезпеки. Він був зібраний і наданий Чеським Технічним Університетом (CTU) у Празі для дослідницьких цілей. Датасет містить дані про мережевий трафік різних зразків шкідливого програмного забезпечення та ботнетів. Після навчання алгоритм може визначати відхилення або аномалії від нормальної поведінки та робити

висновки на наявність постійних загроз, про що відповідно сповіщає на сервер. RandomForest може працювати з великими обсягами даних і ефективно їх обробляти, що робить його ідеальним варіантом для аналізу мережевого трафіку в режимі реального часу.

На клієнті були введені деякі нові впровадження такі як:

- Підключено бібліотеку Npcap (для перехоплення трафіка)
- Впроваджено алгоритм визначення активного адаптера та його відкриття для збору даних
- Додано алгоритм ідентифікації й класифікації транспортних протоколів й кодування отриманих даних для подальшого їх передавання на сервер

В кінці роботи було проведено експеримент, що підтверджує правильність та працездатність роботи – виявлено зловмисні з’єднання шкідливих програм.

ВИСНОВКИ

Persitence — це концепція, яка часто використовується групами Advanced Persistent Threat (APT). Групи APT — це високопрофесійні та добре забезпечені зловмисники, які проводять цілеспрямовані та довгострокові атаки проти конкретних організацій чи організацій. Групи APT прагнуть підтримувати довгостроковий доступ до цільових систем або мереж. Закріплюючись в системі, вони можуть забезпечити безперервний неавторизований доступ, навіть якщо початкові точки входу або скомпрометовані облікові записи було виявлено та виправлено.

В дипломній роботі було розглянуто детально такі поняття к атрибуція загроз, атрибути загроз, IOC, BIOC, аномалії в мережевому трафіку. IOC – це індикатор компрометації. Це артефакти або докази, які свідчать про наявність або виникнення інциденту або злому системи. BIOC – це Behavioral Indicators of Compromise. На відміну від традиційних IOC, які зосереджуються на конкретних артефактах або атрибутах, BIOC зосереджені на аналізі шаблонів дій, поведінки або аномалій, які демонструють системи або користувачі. Аномалія трафіку спостерігається у вигляді нетипової або аномальної поведінки, яка відбувається в мережевому трафіку. Вона передбачає відхилення від очікуваних або нормальних шаблонів мережевої комунікації, які визначені в датасеті. Також було розглянуто передовий програмний засіб в області виявлення та запобіганню загрозам – Cortex XDR.

В другому розділі було спроектовано та побудовано модель власної системи на основі виявлення присутності загроз у Windows. Система працює на основі клієнт-серверної архітектури, сервером виступає Телеграм бот, що вагомо пришвидшує взаємодію. Впроваджено основні правила IOC та BIOC, для знаходження присутніх загроз. Продемонстровано їх виконання та результат роботи.

В третьому розділі було представлено покращення системи на основі впровадження в неї моделі машинного навчання. Для навчання цієї моделі було

знайдемо за застосовано датасет CTU-13, який є передовим датасетом в області машинного навчання в кібербезпеці та містить записаний мережевий трафік з різних комп’ютерних системи, заражених різноманітними видами шкідливого програмного забезпечення. Моделі машинного навчання часто можуть виявляти підозрілу активність на ранній стадії, до того, як відбудеться повномасштабна атака. Це дозволяє організаціям вживати профілактичних заходів для пом’якшення потенційних загроз і запобігання серйозній шкоді або втраті даних. Відповідно організації можуть покращити захист в сфері кіберпростору, точніше та оперативніше визначати загрози та зменшити вплив потенційних кібератак.

\

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. ADVANCED PERSISTENT THREATS (APT): AN AWARENESS REVIEW [Електронний ресурс] - https://www.abacademies.org/articles/Advanced-persistent-threats-apt-an-awareness-review-1533-3604-21-6-202.pdf
2. Big Game Hunting [Електронний ресурс] - https://www.crowdstrike.com/cybersecurity-101/cyber-big-game-hunting/
3. The Unified Kill Chain [Електронний ресурс] - https://www.unifiedkillchain.com/assets/The-Unified-Kill-Chain.pdf
4. What is an Advanced Persistent Threat? APT Definition [Електронний ресурс] - https://www.digitalguardian.com/blog/what-advanced-persistent-threat-apt-definition
5. Attribution of Advanced Persistent Threats: How to Identify the Actors Behind Cyber-Espionage [Електронний ресурс] - https://dokumen.pub/attribution-of-advanced-persistent-threats-how-to-identify-the-actors-behind-cyber-espionage-1st-ed-9783662613122-9783662613139.html
6. Data Talks: Following a Trail of Confusion: PowerShell in Malicious Office Documents [Електронний ресурс] - https://threatresearch.ext.hp.com/powershell-malicious-office-documents/
7. From IOC to TTP: How Attack Chains Have Evolved [Електронний ресурс] - https://www.cyberseer.net/wp-content/uploads/2018/11/Cyberseer-UK-Sec-Show-From-IOC-to-TTP-How-Attack-Chains-Have-Evolved.pdf
8. Emerging Challenges in Security Operations [Електронний ресурс] - https://www.coherentia.it/public/coherentia/PDF/Cortex022021Customer.pdf
9. CORTEX XDR [Електронний ресурс] - https://www.paloguard.com/datasheets/cortex-xdr.pdf
10. Telegram Bot API [Електронний ресурс] - https://core.telegram.org/bots/api
11. Windows API [Електронний ресурс] - https://learn.microsoft.com/en-us/windows/win32/apiindex/windows-api-list

12. Free Cloud Computing Services - AWS Free Tier [Електронний ресурс] - https://aws.amazon.com/free/
13. BotFather Telegram [Електронний ресурс] - https://t.me/botfather
14. Sophos-originated indicators-of-compromise from published reports [Електронний ресурс] - https://github.com/sophoslabs/IoCs
15. Prolock ransomware IOC [Електронний ресурс] - https://github.com/sophoslabs/IoCs/blob/master/Ransomware-ProLock.csv
16. Windows Cryptographic API (CryptoAPI) [Електронний ресурс] - https://learn.microsoft.com/en-us/windows/win32/seccng/cng-portal
17. Ransomware in France, April 2022–March 2023 – Malwarebytes [Електронний ресурс] - https://www.malwarebytes.com/blog/threat-intelligence/2023/04/ransomware-review-france
18. LockBit IOC [Електронний ресурс] - https://github.com/sophoslabs/IoCs/blob/master/Ransomware-LockBit
19. Cyber criminals use Microsoft PowerShell in ransomware attacks [Електронний ресурс] - https://www.computerweekly.com/news/450280067/Cyber-criminals-use-Microsoft-PowerShell-in-ransomware-attacks
20. The CTU-13 Dataset [Електронний ресурс] - https://www.stratosphereips.org/datasets-ctu13
21. Detecting APT Attacks Based on Network Traffic Using Machine Learning [Електронний ресурс] - https://journals.riverpublishers.com/index.php/JWE/article/view/5577
22. Network Traffic Based Botnet Detection Using Machine Learning [Електронний ресурс] - https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1917&context=etd_projects
23. Getting the Physical (MAC) address of a Network Interface Card and finding out if it is the primary adapter on a multi-homed system [Електронний

ресурс] - https://www.codeproject.com/Articles/13421/Getting-the-Physical-MAC-address-of-a-Network-Inte

24. e4f816462c4fc84bb250e2b1d295bf23_85f9a5247afbe51e64794193f1dd72eb_unpacked.exe [Електронний ресурс] - https://www.virustotal.com/gui/file/e8c63b3753504ff27cb93555c62beaaded5f3e485017d950f0aa0831a9c45ac8/details
25. VirusShare_2467b3c8b259cecd6ce2d5c31009df10 [Електронний ресурс] - https://www.virustotal.com/gui/file/dcf50510efec16ff10c5aed91c8e386aba114e63842caa16ea40cac776c60816/details
26. 5b1e1e909a6efca6cabc0fad8a0458a6.virobj [Електронний ресурс] - https://www.virustotal.com/gui/file/490aeb8ce4b544f6edbf0f6c84bb808fb3023b9f7f8283cb101b82783b36cce1/relations

ДОДАТОК А

Код клієнта:

Source.cpp

```
#include "TrafficIntercept.h"

DWORD WINAPI ClassficationFucntion(void* classificator_param)
{
    char* classificator_c = (char*)classificator_param;
    string classificator = (string)classificator_c;
    string param_to_do = classificator.substr(0, 4);
    cout << "classificator" << classificator << endl;
    if (param_to_do == "IOCR")
    {
        param_to_do = classificator.substr(9, 4);
        if (param_to_do == "FILE")
        {
            cout << "FILE HAS BEEN CHOSEN " << endl;
            SearchFile("C:", classificator.substr(18, string::npos).c_str(), 0);
            /*static char word[] = { 0 };
            strcpy(word, classificator.substr(18, string::npos).c_str());
            hThread = CreateThread(nullptr, 0, timeConsumingAction, (void*)word, 0,
nullptr);
            CloseHandle(hThread);*/
        }
        else if (param_to_do == "HASH")
        {
            cout << "HASH HAS BEEN CHOSEN " << endl;
            SearchFile("C:", "", 1);
            hStruct.hash_value = classificator.substr(18, string::npos);
            ThreadCreation();
        }
        else
        {
            cout << "REGEDIT HAS BEEN CHOSEN " << endl;
            cout << "reg string " << classificator.substr(18, string::npos) << endl;
            cout << "len reg str " << classificator.substr(18, string::npos).length()
<< endl;

            FindReg(classificator.substr(18, string::npos));
        }
    }

    else if (param_to_do == "BIOC")
    {
        param_to_do = classificator.substr(9, 4);
        string ActionToDo = classificator.substr(classificator.rfind(shablon) +
shablon.length(), string::npos);
        int index_to_delete = (stoi)(ActionToDo.substr(6));
        if (param_to_do == "PCRT")
        {
            //string ActionToDo = classificator.substr(classificator.rfind(shablon) +
shablon.length(), string::npos);
            if (ActionToDo.substr(0, 6) == "create")
            {
                cout << "PROCESS CREATION HAS BEEN CHOSEN " << endl;
                CheckProcessCreation(classificator.substr(18,
classificator.find_last_of(shablon) - 22));
            }
            else
            {

                ResetEvent(::bioc_handles[index_to_delete]);
```

```
                cout << "DELETE PROCESS CREATION HAS BEEN CHOSEN " << endl;
            }
        }
        else if (param_to_do == "PGNR")
        {
            if (ActionToDo.substr(0, 6) == "create")
            {
                string temp_str = classificator.substr(classificator.find(shablon, 18)
+ 5);
                string child = temp_str.substr(0, temp_str.find(shablon));
                string parent = classificator.substr(18, classificator.find(shablon,
18) - 18);

                cout << "PROCESS GENERATION HAS BEEN CHOSEN " << endl;
                ProcessEnum(child, parent);
            }
            else
            {
                ResetEvent(::bioc_handles[index_to_delete]);
                cout << "DELETE PROCESS GENERATION HAS BEEN CHOSEN " << endl;
            }
        }

    }

    else if (param_to_do == "NTML")
    {
        param_to_do = classificator.substr(9, 5);
        if (param_to_do == "START")
        {
            cout << "network traffic ml inspection started..." << endl;
            string AdaptName = IdentifyActiveNetAdapter();
            OpenAndInterceptTraffic(AdaptName);
        }
        else
        {
            int index_to_delete = stoi(&classificator.back());
            ResetEvent(::bioc_handles[index_to_delete]);
            cout << "you hve stopped traffic ml inspection started..." << endl;
        }
    }

    else
    {
        cout << "Incorrect parameter! Please try again!" << endl;
    }
    return 0;
}


void StartConnection(int port, string ipv4address)
{
    WORD WinSockVersion = MAKEWORD(2, 2);
    cout << endl << " ----- This is client side -----" << endl << endl;
    WSADATA wsaData;
    if (WSAStartup(WinSockVersion, &wsaData) != 0)
    {
        cout << "Error during WINsock initializing :" << WSAGetLastError() << endl;
        exit(EXIT_FAILURE);
    }

    SOCKADDR_IN addr;
    addr.sin_addr.s_addr = inet_addr(ipv4address.c_str());
    addr.sin_port = htons(port);
    addr.sin_family = AF_INET;
```

```
    hStruct.SocketConnection = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0,
WSA_FLAG_OVERLAPPED);
    if (hStruct.SocketConnection == INVALID_SOCKET)
    {
        cout << "Error during creating Socket = " << WSAGetLastError() << endl;

    }
    else { cout << "Socket was created succesfully! " << endl; }
    if (WSAConnect(hStruct.SocketConnection, (SOCKADDR*)&addr, sizeof(addr), NULL,
NULL, NULL, NULL) == 0)
    {
        cout << "U have been succesfully connected to the server! " << endl;
    }
    else {
        cout << "Error during connecting to the server! " << endl; exit(EXIT_FAILURE);
    }

    WSABUF wsaBuf;
    char data[128] = { 0 };
    wsaBuf.len = sizeof(data);
    wsaBuf.buf = data;

    DWORD ReceivedData = 0, Flags = 0;
    while (true)
    {
        if (WSARecv(hStruct.SocketConnection, &wsaBuf, 1, &ReceivedData, &Flags, NULL,
NULL) == 0)
        {
            //cout << "Data has been received!" << endl;
            string text = (string)wsaBuf.buf;

            cout << text << " and its length = " << text.length() << endl;
            char data[128] = { 0 };
            wsaBuf.len = sizeof(data);
            wsaBuf.buf = data;

            static char lpParameter_c[] = { 0 };
            strcpy(lpParameter_c, text.c_str());
            cout << "lpParameter_c" << lpParameter_c << endl;
            HANDLE hThread = CreateThread(nullptr, 0, ClassficationFucntion,
(void*)lpParameter_c, 0, nullptr);
            CloseHandle(hThread);
        }
        else
        {
            cout << "Error during receiving data = " << WSAGetLastError() << endl;
            break;
        }
        //break;

    }
    shutdown(hStruct.SocketConnection, 2);
    closesocket(hStruct.SocketConnection);
    WSACleanup();
}


int main()
{

    int portValue = 65432;

    string ipv4addr = "";
    StartConnection(portValue, ipv4addr);
```

```
    return 0;
}
```

TrafficIntercept.h

```
	#include "functions.h"

	#define WORKING_BUFFER_SIZE 15000
	#define FREE(x) HeapFree(GetProcessHeap(), 0, (x))
	#define MAX_TRIES 3
	#pragma comment(lib, "IPHLPAPI.lib")

	/* 4 bytes IP address */
	typedef struct ip_address {
		u_char byte1;
		u_char byte2;
		u_char byte3;
		u_char byte4;
	}ip_address;

	/* IPv4 header */
	typedef struct ip_header {
		u_char ver_ihl;		// Version (4 bits) + Internet header length (4
bits)
		u_char tos;		// Type of service
		u_short tlen;			// Total length
		u_short identification; // Identification
		u_short flags_fo;	// Flags (3 bits) + Fragment offset (13 bits)
		u_char ttl;		// Time to live
		u_char proto;		// Protocol
		u_short crc;		// Header checksum
		ip_address	saddr;	// Source address
		ip_address	daddr;	// Destination address
		u_int	op_pad;			// Option + Padding
	}ip_header;

	/* UDP header*/
	typedef struct udp_header {
		u_short sport;			// Source port
```

```
        u_short dport;                  // Destination port
        u_short len;              // Datagram length
    }udp_header;


    typedef struct tcp_header {
        u_short sport;
        u_short dport;
        u_int seq_num;
        u_int ack_num;
        u_char offset_reserved;
        u_char flags;
    } tcp_header;

    typedef struct tcp_connection_struct{
        u_short sport;
        u_short dport;

        bool syn_rcvd;
        bool syn_ack_rcvd;
        bool ack_rcvd;
        clock_t syn_time;
        clock_t syn_ack_time;
        float syn_ack_time_sec;
        clock_t ack_time;
        float ack_time_sec;
        float total_duration;
        int total_bytes;
        int total_packets;
    }tcp_connection_struct;

    unordered_map <string, tcp_connection_struct> tcp_structs;



    /* prototype of the packet handler */
    void packet_handler(u_char* param, const struct pcap_pkthdr* header, const
u_char* pkt_data);

    char ipAddress[INET_ADDRSTRLEN];


    string IdentifyActiveNetAdapter()
    {
```

```
            PIP_ADAPTER_ADDRESSES  pAddresses, pCurrAddresses;
            ULONG outBufLen = WORKING_BUFFER_SIZE;
            DWORD dwRetVal = 0;
            ULONG Iterations = 0;
            string  NetAdapterName;

            wstring NetAdapterDesc;
            do
            {
                  pAddresses = (IP_ADAPTER_ADDRESSES*)HeapAlloc(GetProcessHeap(), 0,
(outBufLen));
                  if (pAddresses != NULL) {
                        dwRetVal = GetAdaptersAddresses(AF_INET,
GAA_FLAG_INCLUDE_PREFIX, NULL, pAddresses, &outBufLen);
                        if (dwRetVal == ERROR_BUFFER_OVERFLOW)
                        {
                              FREE(pAddresses);
                              pAddresses = NULL;
                        }
                        else
                        {
                              break;
                        }
                  }
                  Iterations++;
            } while ((dwRetVal == ERROR_BUFFER_OVERFLOW) && (Iterations <
MAX_TRIES));

            if (dwRetVal == NO_ERROR)
            {
                  // If successful, output information from the data we received
                  pCurrAddresses = pAddresses;
                  int minIfIndex = 1000;

                  //find the index of adapter
                  while (pCurrAddresses)
                  {
                        if ((pCurrAddresses->OperStatus == 1) && (pCurrAddresses-
>IfType != 24))
                        //if ((pCurrAddresses->OperStatus == 1))
                        { // The interface is up and able to pass packets; not a
software loopback network interface.

                              if (pCurrAddresses->IfIndex < minIfIndex)
```

```
                                {
                                        minIfIndex = pCurrAddresses->IfIndex;
                                        NetAdapterName = "\\Device\\NPF_" +
(string)pCurrAddresses->AdapterName;
                                        NetAdapterDesc = pCurrAddresses->Description;

                                        IP_ADAPTER_UNICAST_ADDRESS* unicast =
pCurrAddresses->FirstUnicastAddress;
                                        sockaddr_in* addr = (sockaddr_in*)unicast-
>Address.lpSockaddr;
                                        inet_ntop(AF_INET, &(addr->sin_addr), ipAddress,
INET_ADDRSTRLEN);
                                }
                        }
                        pCurrAddresses = pCurrAddresses->Next;
                }
                cout << "Adapter name : " << NetAdapterName << endl;
                wcout << "Description: " << NetAdapterDesc << endl;
                cout << "Index " << minIfIndex << endl;
                cout << "ipv4 address: " << (ipAddress) << endl;

        }
        else
        {
                cout << "Call to GetAdaptersAddresses failed with error!" << endl;
        }

        if (pAddresses) FREE(pAddresses);

        return NetAdapterName;
}

HANDLE hEvent_for_network_detection;

void OpenAndInterceptTraffic(string AdaptName)
{
        char source[PCAP_BUF_SIZE];
        char errbuf[PCAP_ERRBUF_SIZE];
        if (pcap_createsrcstr(source, PCAP_SRC_IFLOCAL, NULL, NULL,
AdaptName.c_str(), errbuf) == 0) //sucess
        {
                //the complete source string according to the new format(e.g.
'rpcap://1.2.3.4/eth0').
                cout << "the new source: " << source << endl;
```

```
                  pcap_t*  AdapterHandle = pcap_open(source, 65536,
PCAP_OPENFLAG_PROMISCUOUS, 1000, NULL, errbuf);
                  if (AdapterHandle != NULL)
                  {
                        hEvent_for_network_detection = CreateEventA(NULL, TRUE, TRUE,
NULL);
                        bioc_handles.push_back(hEvent_for_network_detection);
                        cout << "Starting listening on the active adapter..." <<
endl;
                        pcap_loop(AdapterHandle, 0, packet_handler,
(u_char*)AdapterHandle);

                        //int res;
                        //struct pcap_pkthdr* header;
                        //const u_char* pkt_data;
                        //struct tm ltime;
                        //char timestr[16];
                        //time_t local_tv_sec;
                        //while ((res = pcap_next_ex(AdapterHandle, &header,
&pkt_data)) >= 0) {
                        //    if (res == 0)
                        //          /* Timeout elapsed */
                        //          continue;
                        //    /* convert the timestamp to readable format */
                        //    local_tv_sec = header->ts.tv_sec;
                        //    localtime_s(&ltime, &local_tv_sec);
                        //    strftime(timestr, sizeof timestr, "%H:%M:%S", &ltime);
                        //    printf("%s,%.6d len:%d\n", timestr, header-
>ts.tv_usec, header->len);
                        //}
                        //if (res == -1) {
                        //    printf("Error reading the packets: %s\n",
pcap_geterr(AdapterHandle));
                        //}
                  }
                  else
                  {
                        cout << "Error occured during opening the adapter: " <<
errbuf << endl;
                  }
            }
      }

      int udp_tot_pkts = 0;
```

```
	int udp_tot_bytes = 0;


	int icmp_tot_pkts = 0;
	int icmp_tot_bytes = 0;
	/* Callback function invoked by libpcap for every incoming packet */
	void packet_handler(u_char* param, const struct pcap_pkthdr* header, const
u_char* pkt_data)
	{
		if (WaitForSingleObject(hEvent_for_network_detection, 0) ==
WAIT_OBJECT_0)
		{
			struct tm ltime;
			char timestr[16];
			ip_header* ih;
			udp_header* uh;
			tcp_header* th;
			u_int ip_len;
			u_short sport, dport;
			time_t local_tv_sec;
			//(VOID)(param);


			/* retireve the position of the ip header */
			ih = (ip_header*)(pkt_data + 14); //length of ethernet header

			/* retireve the position of the udp header */
			ip_len = (ih->ver_ihl & 0xf) * 4; // multiply ihl by 0000 1111
(oxf)
			string direction;

			string source_ip = to_string(ih->saddr.byte1) + "." +
				to_string(ih->saddr.byte2) + "." +
				to_string(ih->saddr.byte3) + "." +
				to_string(ih->saddr.byte4);

			string destination_ip = to_string(ih->daddr.byte1) + "." +
				to_string(ih->daddr.byte2) + "." +
				to_string(ih->daddr.byte3) + "." +
				to_string(ih->daddr.byte4);

			direction = (source_ip == ipAddress) ? "0" : "1";
			string data_to_send;
			string ip_data_to_send = "";
```

```
                  if (ih->proto == 6)
                  {

                        th = (tcp_header*)((u_char*)ih + ip_len);

                        string connection_key = source_ip + ":" + to_string(ntohs(th-
>sport)) + "-" +
                              destination_ip + ":" + to_string(ntohs(th->dport));

                        string reversed_connection_key = destination_ip + ":" +
to_string(ntohs(th->dport)) + "-" +
                              source_ip + ":" + to_string(ntohs(th->sport));

                        if ((th->flags & TH_PUSH) || (th->flags & TH_URG) || (th-
>flags & TH_FIN) || (th->flags & TH_RST))
                        {
                              tcp_connection_struct connection;
                              string s_port, d_port;
                              string state = "INT";
                              if (direction == "0") //i am source
                              {
                                    connection = tcp_structs[connection_key];

                                    s_port = to_string(ntohs(th->sport));
                                    d_port = to_string(ntohs(th->dport));
                                    if ((th->flags & TH_FIN) || (th->flags &
TH_RST))
                                    {
                                          tcp_structs.erase(connection_key);
                                          state = "FIN";
                                    }
                              }
                              else if (direction == "1")
                              {
                                    connection =
tcp_structs[reversed_connection_key];
                                    s_port = to_string(ntohs(th->sport));
                                    d_port = to_string(ntohs(th->dport));
                                    if ((th->flags & TH_FIN) || (th->flags &
TH_RST))
                                    {

      tcp_structs.erase(reversed_connection_key);
                                          state = "FIN";
```

```
                                        }
                                } // i am not source
                                else {
                                        return;
                                }



                                connection.total_duration = ((float)(clock() -
connection.syn_time)) / CLOCKS_PER_SEC;
                                connection.total_bytes += header->caplen;
                                connection.total_packets += 1;
                                data_to_send = to_string(connection.total_duration) +
shablon2
                                        + to_string(TCP_PROTO) + shablon2
                                        + s_port + shablon2
                                        + direction + shablon2
                                        + d_port + shablon2
                                        + state + shablon2
                                        + to_string(connection.total_packets) + shablon2
                                        + to_string(connection.total_bytes) + shablon2;

                        }

                        else if ((th->flags & TH_SYN) && !(th->flags & TH_ACK))
                        {
                                //printf("SYN packet detected.\n");
                                tcp_connection_struct connection;
                                connection.sport = ntohs(th->sport);
                                connection.dport = ntohs(th->dport);
                                connection.syn_rcvd = true;
                                connection.syn_ack_rcvd = false;
                                connection.ack_rcvd = false;
                                connection.syn_time = clock();
                                connection.total_bytes = header->caplen;
                                connection.total_packets = 1;
                                tcp_structs[connection_key] = connection;

                                data_to_send = "0" + shablon2
                                        + to_string(TCP_PROTO) + shablon2
                                        + to_string(ntohs(th->sport)) + shablon2
                                        + direction + shablon2
                                        + to_string(ntohs(th->dport)) + shablon2
                                        + "REQ" + shablon2
```

```
                                    + to_string(connection.total_packets) + shablon2
                                    + to_string(connection.total_bytes) + shablon2;
                              //SendDataToSocketConnectio();
                        }

                        else if ((th->flags & TH_SYN) && (th->flags & TH_ACK))
                        {
                              //printf("SYN-ACK packet detected.\n");
                              if (tcp_structs.contains(reversed_connection_key))
                              {
                                    tcp_connection_struct connection =
tcp_structs.at(reversed_connection_key);
                                    connection.syn_ack_rcvd = true;
                                    connection.syn_ack_time = clock();
                                    connection.total_bytes += header->caplen;
                                    connection.syn_ack_time_sec =
((float)(connection.syn_ack_time - connection.syn_time)) / CLOCKS_PER_SEC; // get time
in miliseconds for syn ack
                                    data_to_send =
to_string(connection.syn_ack_time_sec) + shablon2
                                          + to_string(TCP_PROTO) + shablon2
                                          + to_string(ntohs(th->dport)) + shablon2
                                          + direction + shablon2
                                          + to_string(ntohs(th->sport)) + shablon2
                                          + "CON" + shablon2
                                          + "2" + shablon2
                                          + to_string(connection.total_bytes) +
shablon2;

                                    tcp_structs.at(reversed_connection_key) =
connection;
                              }
                        }

                        else if ((th->flags & TH_ACK) && !(th->flags & TH_SYN))
                        {
                              if (tcp_structs.contains(connection_key))
                              {
                                    //printf("ACK packet detected.\n");
                                    tcp_connection_struct connection =
tcp_structs[connection_key];
                                    connection.ack_rcvd = true;
                                    connection.ack_time = clock();
```

```
                                    connection.ack_time_sec =
((float)(connection.ack_time - connection.syn_time)) / CLOCKS_PER_SEC;
                                    connection.total_bytes += header->caplen;
                                    connection.total_packets += 1;
                                    tcp_structs.at(connection_key) = connection;

                                    if (tcp_structs[connection_key].syn_rcvd &&
tcp_structs[connection_key].syn_ack_rcvd && tcp_structs[connection_key].ack_rcvd) {
                                          //printf("TCP 3-way handshake
complete.\n");
                                          //printf("Connection: %s\n",
connection_key.c_str());
                                          data_to_send =
to_string(connection.ack_time_sec) + shablon2
                                                + to_string(TCP_PROTO) + shablon2
                                                + to_string(ntohs(th->sport)) +
shablon2
                                                + direction + shablon2
                                                + to_string(ntohs(th->dport)) +
shablon2
                                                + "ACC" + shablon2
                                                +
to_string(connection.total_packets) + shablon2
                                                + to_string(connection.total_bytes)
+ shablon2;

                                    }
                              }
                        }
                        //SendDataToSocketConnectio(data_to_send);
                        ip_data_to_send = "TCP=="+connection_key;
                        cout << data_to_send << endl;

                  }
                  else if (ih->proto == 17)
                  {
                        cout << "UDP" << endl;
                        uh = (udp_header*)((u_char*)ih + ip_len);
                        sport = ntohs(uh->sport);
                        dport = ntohs(uh->dport);
                        string connection_key = source_ip + ":" + to_string(ntohs(uh-
>sport)) + "-" +
                              destination_ip + ":" + to_string(ntohs(uh->dport));
                        udp_tot_pkts += 1;
```

```
		udp_tot_bytes += header->caplen;
		data_to_send = "0" + shablon2
			+ to_string(UDP_PROTO) + shablon2
			+ to_string(ntohs(uh->sport)) + shablon2
			+ direction + shablon2
			+ to_string(ntohs(uh->dport)) + shablon2
			+ "INT" + shablon2
			+ to_string(udp_tot_pkts) + shablon2
			+ to_string(udp_tot_bytes) + shablon2;
		cout << data_to_send << endl;
		ip_data_to_send = "UDP==" + connection_key;
		//SendDataToSocketConnectio(data_to_send);

	}
	else if (ih->proto == 1)
	{
		cout << "ICMP " << endl;
		icmp_tot_pkts += 1;
		icmp_tot_bytes += header->caplen;
		string connection_key = source_ip + ":" + destination_ip;
		if (direction == "0") // if i am source
		{
			data_to_send = "0" + shablon2
				+ to_string(ICMP_PROTO) + shablon2
				+ "0" + shablon2
				+ direction + shablon2
				+ "0" + shablon2
				+ "ECO" + shablon2
				+ to_string(icmp_tot_pkts) + shablon2
				+ to_string(icmp_tot_bytes) + shablon2;
		}
		else
		{
			data_to_send = "0" + shablon2
				+ to_string(ICMP_PROTO) + shablon2
				+ "0" + shablon2
				+ direction + shablon2
				+ "0" + shablon2
				+ "ECR" + shablon2
				+ to_string(icmp_tot_pkts) + shablon2
				+ to_string(icmp_tot_bytes) + shablon2;
		}
		ip_data_to_send = "ICMP==" + connection_key;
		cout << data_to_send << endl;
```

```
                  }
                  if (data_to_send != "")
                  {
                        data_to_send = data_to_send + to_string(header->caplen) +
shablon2  + ip_data_to_send + shablon2;
                        SendDataToSocketConnectio(data_to_send);
                  }
            }
            else
            {
                  pcap_t* adapterHandle = (pcap_t*)param;
                  cout << "ml stopeed" << endl;
                  pcap_breakloop(adapterHandle);
            }
      }
```

Function.h

```
#pragma once
#define WIN32_LEAN_AND_MEAN
#pragma comment(lib,"ws2_32.lib")
#include <windows.h>
#include <string>
#include <stdio.h>
#include <iostream>
#include <winsock2.h>
#include <msi.h>
#include <bcrypt.h>
#include <vector>
#include <TlHelp32.h>
#include "psapi.h"
#include <iphlpapi.h>
#include <unordered_map>

#define TH_FIN      0x01
#define TH_SYN      0x02
#define TH_RST      0x04
#define TH_PUSH     0x08
#define TH_ACK      0x10
#define TH_URG      0x20
```

```
#define ICMP_PROTO 0
#define TCP_PROTO 1
#define UDP_PROTO 2


#pragma warning(disable:4996)
#pragma comment (lib, "bcrypt.lib")

//#pragma comment (linker, "wpcap.lib")
#include <pcap.h>
//#include <wpcap.dll>


#define BUFFER_SIZE 4096

#define BUFSIZE 1024
#define MD5LEN  16

#define MAX_KEY_LENGTH 255
#define MAX_VALUE_NAME 16383

using namespace std;

vector <string> filepaths_to_hash;
int num = 64;

string shablon = "==;;;";
string shablon2 = "#-;@";

struct Hash_struct
{
    string hash_value;
    SOCKET SocketConnection;
    HANDLE RegFile;
};
Hash_struct hStruct;



void SendDataToSocketConnectio(string DataToSend)
{
    //cout << DataToSend << endl;
    WSABUF wsaBuf = {0};
```

```
    wsaBuf.len = (DataToSend.length());
    wsaBuf.buf = (char*)DataToSend.c_str();
    if (WSASend(hStruct.SocketConnection, &wsaBuf, 1, &wsaBuf.len, 0, NULL, NULL) !=
0)
    {
        cout << "error occured" << endl;
    }
}



// ---------- SEARCH FUNCTIONS ----------


void SearchFile(LPCSTR lpFolder, LPCSTR lpFilePattern, int hash_state)
{
    char szFullPattern[BUFSIZE];
    strcpy_s(szFullPattern, BUFSIZE, lpFolder);
    strcat_s(szFullPattern, BUFSIZE, "\\*");
    WIN32_FIND_DATAA  FindFileData;
    HANDLE hFind = FindFirstFileA(szFullPattern, &FindFileData);

    if (hFind == INVALID_HANDLE_VALUE)
    {
        return ;
    }

    do
    {
        if (strcmp(FindFileData.cFileName, ".") == 0 || strcmp(FindFileData.cFileName,
"..") == 0)
        {
            continue;
        }
        char szFilePath[BUFSIZE];
        strcpy_s(szFilePath, BUFSIZE, lpFolder);
        strcat_s(szFilePath, BUFSIZE, "\\");
        strcat_s(szFilePath, BUFSIZE, FindFileData.cFileName);

        if (FindFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
        {
            SearchFile(szFilePath, lpFilePattern, hash_state);
        }
        else
```

```
        {
            string data_to_send = (lpFolder + string("\\") + FindFileData.cFileName);
            if (hash_state == 1)
            {
                filepaths_to_hash.push_back(data_to_send);
            }
            else
            {
                if (strstr(FindFileData.cFileName, lpFilePattern) != NULL)
                {
                    cout << data_to_send << endl;
                    SendDataToSocketConnectio(data_to_send);
                    //return (lpFolder + string("\\") + FindFileData.cFileName);
                    //cout << lpFolder << "\\" << FindFileData.cFileName << endl;
                }
            }
        }
    } while (FindNextFileA(hFind, &FindFileData));

    FindClose(hFind);
}


// ---------- SEARCH FUNCTIONS ----------


// ---------- REGEDIT FUNCTIONS ----------


HKEY predefined_keys[9] = {
    HKEY_CLASSES_ROOT,
    HKEY_CURRENT_CONFIG,
    HKEY_CURRENT_USER,
    HKEY_CURRENT_USER_LOCAL_SETTINGS,
    HKEY_LOCAL_MACHINE,
    HKEY_PERFORMANCE_DATA,
    HKEY_PERFORMANCE_NLSTEXT,
    HKEY_PERFORMANCE_TEXT,
    HKEY_USERS
};


string predefined_keys_strings[9] = {
    "HKEY_CLASSES_ROOT",
```

```
    "HKEY_CURRENT_CONFIG",
    "HKEY_CURRENT_USER",
    "HKEY_CURRENT_USER_LOCAL_SETTINGS",
    "HKEY_LOCAL_MACHINE",
    "HKEY_PERFORMANCE_DATA",
    "HKEY_PERFORMANCE_NLSTEXT",
    "HKEY_PERFORMANCE_TEXT",
    "HKEY_USERS"
};


void QueryKey(HKEY hKey, TCHAR  lpFolder[], string key_val, int index ,string
str_to_find)
{
    string szFullPattern = lpFolder;
    if (szFullPattern != "")
    {
        szFullPattern += string("\\");
    }

    TCHAR achKey[MAX_KEY_LENGTH];   // buffer for subkey name
    DWORD cSubKeys = 0;               // number of subkeys
    DWORD cValues;              // number of values for key
    TCHAR achValue[MAX_VALUE_NAME];
    DWORD    cchClassName = MAX_PATH;

    // Get the class name and the value count.
    DWORD  retCode = RegQueryInfoKeyA(hKey, NULL, NULL, NULL, &cSubKeys, NULL, NULL,
&cValues, NULL, NULL, NULL, NULL);

    // Enumerate the subkeys values, until RegEnumKeyEx fails.
    if (cValues)
    {
        BOOL signal_var = 0; // signal var to detect empty name values
        for (int i = 0, retCode = ERROR_SUCCESS; i < cValues; i++)
        {
            DWORD cchValue = MAX_VALUE_NAME;

            if (RegEnumValueA(hKey, i, achValue, &cchValue, NULL, NULL, NULL, NULL) ==
ERROR_SUCCESS)
            {
                string str_to_check;
                if (cchValue == 0) // if name of subkey is empty
                {
```

```
                    if ((signal_var == 0) && (i + 1 == cValues)) // if its the last
empty value then cout it
                    {
                        str_to_check = key_val + "\\" + szFullPattern;
                    }
                }
                else // if it is non empty value then simply  cout it without problems
                {
                    str_to_check = key_val + "\\" + szFullPattern + achValue;
                    signal_var = 1;
                }
                //cout << str_to_check << endl;
                if (strstr(str_to_check.c_str(), str_to_find.c_str()) != NULL) //
check if this reg path contain out string
                {
                    string data_to_send = str_to_check + "\n";
                    SendDataToSocketConnectio(data_to_send);
                    cout << str_to_check << endl;
                }
                //memset(achValue, 0, sizeof(TCHAR) * MAX_VALUE_NAME);
            }
        }
    }

    if (cSubKeys > 0) // emumerate subkeys values
    {
        for (int i = 0; i < cSubKeys; i++)
        {
            DWORD cbName = MAX_KEY_LENGTH;
            if (RegEnumKeyEx(hKey, i, achKey, &cbName, NULL, NULL, NULL, NULL) ==
ERROR_SUCCESS) // if success
            {
                string temp_pass = szFullPattern + achKey;
                HKEY hTestKey1;
                if (RegOpenKeyExA(predefined_keys[index], temp_pass.c_str(), 0,
KEY_READ, &hTestKey1) == ERROR_SUCCESS) //open it
                {
                    QueryKey(hTestKey1, const_cast<LPSTR>(temp_pass.c_str()), key_val,
index, str_to_find); // cal this function again and again.....
                }
                RegCloseKey(hTestKey1);//close HKEY
            }
            //memset(achValue, 0, sizeof(TCHAR) * MAX_VALUE_NAME);
        }
```

```
    }
    else if (cSubKeys <= 0 && cValues <= 0) // if there is no "childs" (key values
values) and no subkey then cout its path
    {
        string temp_pattern = (key_val + "\\" + szFullPattern);
        if (strstr(temp_pattern.c_str(), str_to_find.c_str()) != NULL)
        {
            cout << temp_pattern << endl;
            string data_to_send = temp_pattern + "\n";
            SendDataToSocketConnectio(data_to_send);
            //couner += 1;
        }
        //cout << "wvalue " << "HKEY_CURRENT_USER\\" + szFullPattern << endl;

    }
}


void FindReg(string reg_ioc)
{
    //TCHAR key_string[MAX_PATH] = TEXT("SOFTWARE\\Intel\\PSIS");
    //LPSTR key_string = "SOFTWARE\\Intel\\PSIS";
    //HKEY hTestKey;
    //RegOpenKeyExA(HKEY_LOCAL_MACHINE, key_string, 0, KEY_READ, &hTestKey);
    //QueryKey(hTestKey, key_string, "HKEY_LOCAL_MACHINE", reg_ioc);
    //RegCloseKey(hTestKey);
    //hStruct.RegFile = CreateFileA(("Regedit_IOC_"+reg_ioc).c_str(), GENERIC_WRITE |
GENERIC_READ, FILE_SHARE_WRITE | FILE_SHARE_READ, NULL, CREATE_ALWAYS, NULL, 0);
    for (int i = 0; i < sizeof(predefined_keys) / 8; i++)
    {
        //LPCSTR key_string = "SOFTWARE\\Python";
        //LPCSTR key_string = "AppEvents\\Schemes";
        //LPCSTR key_string = "";
        TCHAR key_string[MAX_PATH] = TEXT("");;
        HKEY hTestKey;
        if (RegOpenKeyExA(predefined_keys[i],
            NULL,
            0,
            KEY_READ,
            &hTestKey) == ERROR_SUCCESS
            )
        {
            QueryKey(hTestKey, key_string, predefined_keys_strings[i], i, reg_ioc);
        }
```

```
        RegCloseKey(hTestKey);
    }
}


// ---------- REGEDIT FUNCTIONS ----------


// ---------- HASH FUNCTIONS ----------


string GetFileHash(LPCSTR filename)
{
    HANDLE hFile = CreateFileA(filename, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL);
    HCRYPTPROV hProv = 0;
    CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_AES, CRYPT_VERIFYCONTEXT);
    HCRYPTHASH hHash = 0;
    CryptCreateHash(hProv, CALG_SHA_256, 0, 0, &hHash);
    BOOL bResult = FALSE;
    BYTE rgbFile[10240];
    DWORD cbRead = 0;
    while (bResult = ReadFile(hFile, rgbFile, 10240, &cbRead, NULL))
    {
        if (0 == cbRead)
        {
            break;
        }
        CryptHashData(hHash, rgbFile, cbRead, 0);
    }
    BYTE rgbHash[64];
    DWORD cbHash = 64;
    char rgbDigits[] = "0123456789abcdef";
    CryptGetHashParam(hHash, HP_HASHVAL, rgbHash, &cbHash, 0);
    string ResultHashString;

    for (DWORD i = 0; i < cbHash; i++)
    {
        ResultHashString += rgbDigits[rgbHash[i] >> 4];
        ResultHashString += rgbDigits[rgbHash[i] & 0xf];
        //cout << (rgbDigits[rgbHash[i] >> 4]);
        //cout << (rgbDigits[rgbHash[i] & 0xf]);
    }
    //cout << "ResultHashString" << ResultHashString << endl;
```

```
    if (ResultHashString == hStruct.hash_value)
    {
        string data_to_send = filename;
        SendDataToSocketConnectio(data_to_send);
        cout << filename << endl;
    }
    //cout << ResultHashString << endl;
    //delete[] rgbFile;
    CryptDestroyHash(hHash);
    CryptReleaseContext(hProv, 0);
    CloseHandle(hFile);
    return  ResultHashString;
    //cout << ResultHashString << endl;
}


DWORD WINAPI ThreadFunc(LPVOID  lpParam)
{
    int value_id = (*(DWORD*)lpParam);
    while (value_id < filepaths_to_hash.size())
    {
        GetFileHash(filepaths_to_hash[value_id].c_str());
        value_id += num;
    }
    return 0;
}


void ThreadCreation()
{
    HANDLE* hThreadClients = new HANDLE[num];
    DWORD* arr_index = new DWORD[num];
    cout << "inside thread func" << endl;
    for (int i = 0; i < num; i++)
    {
        arr_index[i] = i;

        hThreadClients[i] = CreateThread(NULL, 0, ThreadFunc, &arr_index[i], 0, 0);
    }
    cout << "before WaitForMultipleObjects" << endl;
    WaitForMultipleObjects(num, hThreadClients, TRUE, INFINITE);
    cout << "after WaitForMultipleObjects" << endl;
    delete[] hThreadClients;
    delete[] arr_index;
```

```
}



// ---------- HASH FUNCTIONS ----------


// ---------- BIOC FUNCTIONS ----------

template <typename T>
bool vec_contain(vector<T> vec, const T &elem)
{
    bool res = false;
    if (!vec.empty())
    {
        for (auto& x : vec)
        {
            if (x == elem)
            {
                res = true;
                break;
            }
        }
    }
    return res;
}


static vector<HANDLE>bioc_handles;


void ProcessEnum(string ChildProcessBIOC, string ParentProcessBIOC)
{
    HANDLE hEvent = CreateEventA(NULL, TRUE, TRUE, NULL);
    if (hEvent!=NULL)
    {
        bioc_handles.push_back(hEvent);
        PROCESSENTRY32 ProcInfo;
        HANDLE PSnapshot;
        vector<DWORD>ProcIDs;
        while ((WaitForSingleObject(hEvent, 0) == WAIT_OBJECT_0))
        {
            ProcInfo = { 0 };
            ProcInfo.dwSize = sizeof(ProcInfo);
```

```
            PSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
            Process32First(PSnapshot, &ProcInfo);
            do
            {
                if ((string)ProcInfo.szExeFile == ChildProcessBIOC)
                {
                    char buffer[MAX_PATH];
                    HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE,
ProcInfo.th32ParentProcessID);
                    GetProcessImageFileNameA(hProcess, buffer, MAX_PATH);

                    string ParentProcess = (string)buffer;
                    size_t found = ParentProcess.find_last_of("\\");

                    if (ParentProcess.substr(found + 1) == ParentProcessBIOC &&
!vec_contain(ProcIDs, ProcInfo.th32ProcessID))
                    {
                        string data = "Process " + ParentProcess.substr(found + 1) + "
with PID = " + to_string((int)ProcInfo.th32ParentProcessID) + " create process " +
ChildProcessBIOC + " with PID = " + to_string((int)ProcInfo.th32ProcessID);
                        SendDataToSocketConnectio(data);
                        cout << "Process " << ParentProcess.substr(found + 1) << "
with PID = " <<
                            ProcInfo.th32ParentProcessID << " create process " <<
ChildProcessBIOC << " with PID = " << ProcInfo.th32ProcessID << endl;
                        ProcIDs.push_back(ProcInfo.th32ProcessID);
                        //return;
                    }
                    CloseHandle(hProcess);
                }
            } while (Process32Next(PSnapshot, &ProcInfo));
            CloseHandle(PSnapshot);
        }
    }


}


void CheckProcessCreation(string ProcessName)
{
    HANDLE hEvent = CreateEventA(NULL, TRUE, TRUE, NULL);
    bioc_handles.push_back(hEvent);
    PROCESSENTRY32 ProcInfo;
```

```
    HANDLE PSnapshot;
    vector<DWORD>ProcIDs;
    while (WaitForSingleObject(hEvent,0) == WAIT_OBJECT_0)
    {
        ProcInfo = { 0 };
        ProcInfo.dwSize = sizeof(ProcInfo);
        PSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
        Process32First(PSnapshot, &ProcInfo);
        do
        {
            if ((string)ProcInfo.szExeFile == ProcessName && !vec_contain(ProcIDs,
ProcInfo.th32ProcessID))
            {
                SYSTEMTIME time_struct;
                GetLocalTime(&time_struct);
                string data = "Process " + (string)ProcInfo.szExeFile + " with PID = "
+
                    to_string((int)ProcInfo.th32ProcessID) + " has been created!";
                SendDataToSocketConnectio(data);
                cout << data << endl;
                ProcIDs.push_back(ProcInfo.th32ProcessID);
            }
        } while (Process32Next(PSnapshot, &ProcInfo));
        CloseHandle(PSnapshot);
    }
}


// ---------- BIOC FUNCTIONS ----------
```

Код Сервера

Server_bot_functions.py

```
from telebot import *
from telebot.types import ReplyKeyboardMarkup, KeyboardButton
from telebot.types import InlineKeyboardButton, InlineKeyboardMarkup
import socket
import time
import json
import re
from threading import Thread
import network_detection

running_tasks_db = "Database.json"

def write_data_to_database(data_to_be_written):
    with open(running_tasks_db, "w") as f_db:
        json.dump(data_to_be_written, f_db)

def read_data_from_database():
    with open(running_tasks_db) as json_file:
        data = json.load(json_file)
    return data

HOST = ""

PORT = 65432

sample = "==;;;"
sample2 = "#-;@"

proc_crt_gnrt_val = 0

splitted_data = []
API_TOKEN = “”
bot = telebot.TeleBot(API_TOKEN)
Keybutton1 = KeyboardButton("IOC")
Keybutton2 = KeyboardButton("BIOC")
Keybutton3 = KeyboardButton("NTML")
keyboard = ReplyKeyboardMarkup(one_time_keyboard=True,
resize_keyboard=True).row(Keybutton1, Keybutton2, Keybutton3)

inlButton11 = InlineKeyboardButton(text="Filename/Path", callback_data="filename")
inlButton12 = InlineKeyboardButton(text="Hash", callback_data="hash")
inlButton13 = InlineKeyboardButton(text="regedit", callback_data="regedit")
keyboard_inline_IOC = InlineKeyboardMarkup().add(inlButton11, inlButton12,
inlButton13)

inlButton21 = InlineKeyboardButton(text="Process Creation (Parent only)",
callback_data="process_create")
inlButton22 = InlineKeyboardButton(text="Process Generation (Parent -> Child)",
callback_data="process_generate")
keyboard_inline_BIOC = InlineKeyboardMarkup().add(inlButton21).add(inlButton22)

def return_running_tasks_keybord(tasks):
    keyboard = InlineKeyboardMarkup()

    for i in (tasks['running_tasks']):
        element_from_table = tasks['running_tasks'][str(i)]
        record = f"{element_from_table['BIOC_rule']}:
{element_from_table['Process_name']} created at {element_from_table['time']}"
        keyboard.add(InlineKeyboardButton(text=record,
callback_data="stop_proc_"+str(i)))
    return keyboard
```

```
def bot_creation():
    #bot.send_message(539421947, "шо ти голова")
    @bot.message_handler(commands=['start'])#start bot
    def handle_command(message):
        bot.reply_to(message, "Hello, i am your security bot! This is your id: " +
str(message.chat.id) + ". And with this id your build will be created to identify
you!")

    @bot.message_handler(commands=['get_build'])#get client build to run
    def handle_command(message):
        global user_id__to_send_file
        user_id__to_send_file = message.chat.id
        fileHandle = open('Client_build.exe' , 'rb')
        bot.send_document(user_id__to_send_file, fileHandle)
        fileHandle.close()

    @bot.message_handler(commands=['running_tasks'])#get running tasks
    def handle_command(message):
        tasks = read_data_from_database()
        if tasks["running_tasks"]:
            keyboard_inline_rtasks = return_running_tasks_keybord(tasks)
            bot.reply_to(message, "Hey, here is a list of your running tasks!👇",
reply_markup=keyboard_inline_rtasks)

        else:
            bot.reply_to(message, "You haven't created any task  yet")

    @bot.message_handler(commands=['help'])#help menu
    def handle_command(message):
        bot.reply_to(message, """This is help menu of your bot! Here you can see the
list of available commands to use:
/start - start bot
/help - display this menu to get help
/get_build - get client build to connect
/menu - display the main menut to start working with bot
/running_tasks - show runnig tasks""")

    @bot.message_handler(commands=['menu'])#main menu
    def handle_menu(message):
        bot.send_message(message.chat.id, "Please choose what you want to
create:",reply_markup=keyboard)

    # handle all messages, echo response back to users
    @bot.message_handler(content_types=['text'])
    def button_answer(message):
        if message.text == "IOC":
            bot.send_message(message.chat.id, "You have chosen the IOC rule! Pleae
specify the exact IOC rule:", reply_markup=keyboard_inline_IOC)
        elif message.text == "BIOC":
            bot.send_message(message.chat.id, "You have chosen the BIOC rule! Pleae
specify the exact BIOC rule:", reply_markup=keyboard_inline_BIOC)
            #bot.send_message(message.chat.id, "You have not specified IOC rule((((")
        elif message.text == "NTML":
            bot.send_message(message.chat.id, "You have chosen the network
inspection!")
            if network_detection.ml_counter:
                #network_detection.make_a_prediction(model, splitted_data)
                bot.send_message(message.chat.id, "Model has been already trained.
Ready to make a prediction!")
            else:
                bot.send_message(user_id__to_send_file, f"Please wait while model is
training...")
                global model
                model = network_detection.prepare_model_for_prediction()
```

```
                bot.send_message(message.chat.id, "Model has been trained
successfuly!")
            conn.send(("NTML" + sample + "START_DETECTION").encode())
            global proc_crt_gnrt_val
            data_from_file = read_data_from_database()

            data_from_file["running_tasks"][proc_crt_gnrt_val] =  {
                "BIOC_rule":"NTML",
                "Process_name": "network_inspection",
                "time":time.strftime("%H:%M", time.localtime()),
            }
            write_data_to_database(data_from_file)
            proc_crt_gnrt_val += 1


        else:
            bot.reply_to(message, "There is no such option! Please try again!")

#text = ["filename", "hash", "regedit"])
    @bot.callback_query_handler(func=lambda call:True)
    def create_ioc_rule_for_machine(call):
        if call.message:
            if "stop_proc" in call.data:
                print(call.data[-1])
                conn.send(("BIOC==;;;"+"PCRT"+ sample + "empty_data" + sample +
"delete" + call.data[-1]).encode())
                data = read_data_from_database()
                del data["running_tasks"][call.data[-1]]
                write_data_to_database(data)
                bot.answer_callback_query(call.id, text="Successfuly deleted this
rule!", show_alert=True)

                keyboard_inline_rtasks =
return_running_tasks_keybord(read_data_from_database())
                bot.edit_message_reply_markup(call.message.chat.id,
call.message.message_id, reply_markup=keyboard_inline_rtasks)
            else:
                if call.data == "filename":
                    #bot.answer_callback_query(call.id, show_alert=1, text="You have
chosen the filename!")
                    bot.send_message(call.message.chat.id, "Please enter the
filename/path of your sample to find. "\
                                                           "The max filepath in Windows
is 256 characters. You can enter any filename, relative/full path.",
                                                           reply_markup=keyboard)

                elif call.data == "hash":
                    #bot.answer_callback_query(call.id, text="You have chosen the hash
value!")
                    bot.send_message(call.message.chat.id, "Please enter the hash of
your sample to find. "\
                                                           "The hash format is SHA256.",
                                                           reply_markup=keyboard)

                elif call.data == "regedit":
                    #bot.answer_callback_query(call.id, text="You have chosen the
regedit value!")
                    bot.send_message(call.message.chat.id, "Please enter the registry
path/value of your sample to find. "\
                                                           "You can specify up to 255
characters key name and up to 16383 characters value name.",reply_markup=keyboard)

                elif call.data == "process_create":
                    bot.send_message(call.message.chat.id, "Please specify the process
name to trace."\
```

```
                                                                "You can enter up to 256
symbols.")

                elif call.data == "process_generate":
                    bot.send_message(call.message.chat.id, "You need to specify parent
and child processes names."\
                                                           "Please enter parent process
name."
                                                           "You can enter up to 256
symbols.")

                bot.register_next_step_handler(call.message,
do_action_based_on_inline_button, call.data)

    def do_action_based_on_inline_button(message, selected_option):
        ioc_pattern = "IOCR==;;;"
        bioc_pattern = "BIOC==;;;"

        if selected_option == "filename":
            if len(message.text) <= 260 : #and re.match('^[a-zA-
Z]:\\[\\\\\S\\|\\*\\S]?.*$', message.text) HZ NADO LI.....
                bot.send_message(message.chat.id, f"You have chosen this filename:
{message.text}")
                conn.send((ioc_pattern+"FILE==;;;"+message.text).encode()+b'\x00')
            else:
                bot.send_message(message.chat.id, f"The max filepath in Windows is 250
characters! Please specify the correct string!")

                bot.register_next_step_handler(message,
do_action_based_on_inline_button, selected_option)

        elif selected_option == "hash":
            if  re.match("^([A-Fa-f0-9]+){16}$", message.text): # check the hash
length and maybe a-zA-Z0-9   len(message.text)==64 and
                bot.send_message(message.chat.id, f"You have chosend this hash:
{message.text}")
                conn.send((ioc_pattern+"HASH==;;;"+message.text).encode())
            else:
                bot.send_message(message.chat.id, f"Please enter the correct hash type
(SHA256)!")
                bot.register_next_step_handler(message,
do_action_based_on_inline_button, selected_option)

        elif selected_option == "regedit":
            # check the regedit...... hz kak)))
            print(message.text)
            bot.send_message(message.chat.id, f"You have chosend this regedit:
{message.text}")
            conn.send((ioc_pattern+"REGD==;;;"+message.text).encode())

        elif selected_option == "process_create" :
            if len(message.text) <= 256 and re.search("^(\w|\s)+[.](\w)+$",
message.text): #{1,10} check length like this {1,10}
                global proc_crt_gnrt_val
                bot.send_message(message.chat.id, f"You have chosend this process
name: {message.text}")
                conn.send((bioc_pattern+"PCRT"+ sample + message.text + sample +
"create" + str(proc_crt_gnrt_val)).encode())

                data_from_file = read_data_from_database()

                data_from_file["running_tasks"][proc_crt_gnrt_val] =  {
                    "BIOC_rule":"PCRT",
                    "Process_name": message.text,
                    "time":time.strftime("%H:%M", time.localtime()),
```

```
                }
                write_data_to_database(data_from_file)
                proc_crt_gnrt_val += 1
                # data =  {
                #     "BIOC_index": proc_crt_gnrt_val,
                #     "BIOC_rule":"PCRT",
                #     "Process_name":message.text,
                #     "time":time.strftime("%H:%M", time.localtime()),
                # }
                # data_from_file = read_data_from_database(running_tasks_db)
                # data_from_file["running_tasks"].append(data)
                # write_data_to_database(running_tasks_db, data_from_file)


            else:
                bot.send_message(message.chat.id, f"Please specify the correct process
name!")
                bot.register_next_step_handler(message,
do_action_based_on_inline_button, selected_option)

        elif selected_option == "process_generate":
            if len(message.text) <= 256 and re.search("^(\w|\s)+[.](\w)+$",
message.text):
                #handle the child prcess name
                msg = bot.reply_to(message, f"You have chosend this parent process
name: {message.text}. \nNow you need specify child process name:")
                bot.register_next_step_handler(msg, child_proccess, message.text,
bioc_pattern)
            else:
                bot.send_message(message.chat.id, f"Please specify the correct parent
process name!")
                bot.register_next_step_handler(message,
do_action_based_on_inline_button, selected_option)

    def child_proccess(msg, parent_process_name, bioc_pattern):
            if len(msg.text) <= 256 and re.search("^(\w|\s)+[.](\w)+$", msg.text):
                global proc_crt_gnrt_val
                bot.send_message(msg.chat.id, f"You have chosend this child process
name: {msg.text}.")
                conn.send((bioc_pattern + "PGNR==;;;" + parent_process_name + sample +
msg.text + sample + "create" + str(proc_crt_gnrt_val)).encode())#send data to
client?????

                data_from_file = read_data_from_database()
                data_from_file["running_tasks"][proc_crt_gnrt_val] =  {
                    "BIOC_rule":"PGNR",
                    "Process_name": parent_process_name+" generate "+msg.text,
                    "time":time.strftime("%H:%M", time.localtime()),
                }
                write_data_to_database(data_from_file)
                proc_crt_gnrt_val += 1

            else:
                msg2 = bot.reply_to(msg, f"Please specify the correct child process
name!")
                bot.register_next_step_handler(msg2, child_proccess, bioc_pattern)

    bot.infinity_polling()



def parse_thaffic_data(data):
    global splitted_data
    #np.where(encoder.categories_[2]=="FIN")
    splitted_data.extend(data.split(sample2))
```

```
    splitted_data = list(filter(None, splitted_data))
    print(splitted_data)
    while len(splitted_data)>=10: #while length more than 8
        try:
            X = network_detection.pd.DataFrame({'Dur': [float(splitted_data[0])],
                                    'Proto': [int(splitted_data[1])],
                                    "Sport":[int(splitted_data[2])],
                                    "Dir":[float(splitted_data[3])],
                                    "Dport":[int(splitted_data[4])],

"State":[network_detection.np.where(network_detection.encoder.categories_[2]==splitted
_data[5])[0][0]],
                                    "TotPkts":[int(splitted_data[6])],
                                    "TotBytes":[float(splitted_data[7])],
                                    "SrcBytes":[float(splitted_data[8])] })
            predicted_value = network_detection.make_a_prediction(model, X)
            print(splitted_data[9])
            if predicted_value:
                bot.send_message(user_id__to_send_file, f"Suspicious network activity
detected: {splitted_data[9]}")
            #delete first 8 elements
            del splitted_data[:10]
        except:
            del splitted_data[0]



def waiting_for_connection():
    try:
        global conn
        conn, addr = socket_handle.accept()
        print(f"Connected by {addr}")
        bot.send_message(user_id__to_send_file, f"🟢🟢🟢Connected by {addr}🟢🟢🟢")
        while True:
            data = conn.recv(10240)
            if not data:
                print("no data and exit")
                break
            if sample2 in data.decode():
                parse_thaffic_data(data.decode())
            else:
                #print("")
                bot.send_message(user_id__to_send_file, data.decode())
    except OSError:
        bot.send_message(user_id__to_send_file, f"🛑🛑🛑Disconnected by
{addr}🛑🛑🛑")
        return
        #print("OS Error")

def create_connection():
    global socket_handle
    socket_handle = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    socket_handle.bind((HOST, PORT))
    socket_handle.listen()
    connection_thread = Thread(target=waiting_for_connection).start()

if __name__ == "__main__":
    default_data_in_db =  {
        "running_tasks": {},
    }
    write_data_to_database(default_data_in_db)
    create_connection()
    bot_thread = Thread(target=bot_creation).start()
```

network_detection.py

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder
from sklearn.utils import resample
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

pd.options.display.max_rows = None

data = pd.read_csv("capture20110815-3.binetflow", low_memory=False)

ml_counter = 0

def cleaning_data():
    global data
    data = data.drop(["StartTime", "DstAddr", "SrcAddr", "sTos", "dTos"], axis=1)

    data = data[data["Proto"].str.contains("ipnip|llc|unas|udt|pim|ipx/spx|ipv6-
icmp|ipv6|igmp|rtp|rtcp|arp") == False]
    data = data[data["Dir"].str.contains("\?") == False]

    data =
data[data["State"].str.contains("UNK|URNPRO|DCE|URO|URF|SRC|RSP|URN|URFIL|URHPRO") ==
False]

    data = data.dropna(subset=['Sport', 'Dport'])
    data.loc[data['Dport'].str.contains('0x'), 'Dport'] = 0
    data.loc[data['Sport'].str.contains('0x'), 'Sport'] = 0
    data['Sport'] = data['Sport'].astype('int')
    data['Dport'] = data['Dport'].astype('int')

    data["State"] = data["State"].replace(['FSPA_FSPA'], 'FIN')
    data["State"] = data["State"].replace(['S_RA'], 'RST')
    data["State"] = data["State"].replace(['SA_'], 'ACC')
    data["State"] = data["State"].replace(['S_'], 'REQ')
    data["State"] = data["State"].replace(['URH'], 'ECR')

    data = data[data["State"].str.contains("_") == False]
    global encoder
    encoder = OrdinalEncoder()
    data[["Proto", "Dir", "State"]] = encoder.fit_transform(data[["Proto", "Dir",
"State"]])

    # data['TotBytes_over_TotPkts'] = data['TotBytes'] / data['TotPkts']
    # data['TotPkts_over_Dur'] =  data['TotPkts'] / data['Dur']
    # data['TotBytes_over_Dur'] =  data['TotBytes'] / data['Dur']

    data = data.dropna()
    def crete_bool_label(input_flow):
        if "Botnet" in input_flow:
            return 1
        else:
            return 0

    data["Label"] = data["Label"].apply(crete_bool_label)
    return data
```

```
def slicing(data):
    df_majority = data[data['Label'] == 0]
    df_minority = data[data['Label'] == 1]

    undersampled_majority = resample(df_majority,
                                     replace=False,  # Set to False for undersampling
                                     n_samples=len(df_minority),  # Match the number of
samples in the minority class
                                     random_state=42)  # Set random shuffle

    undersampled_df = pd.concat([undersampled_majority, df_minority])
    undersampled_df = undersampled_df.sample(frac=1,
random_state=42).reset_index(drop=True)
    y = undersampled_df.Label
    undersampled_df = undersampled_df.drop(["Label"], axis=1)
    return undersampled_df, y

def train_dataset(x, y):

    X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
random_state=44)
    rf_model = RandomForestClassifier(n_estimators=50, max_features="auto",
random_state=44)
    rf_model.fit(X_train, y_train)

    return rf_model

def make_a_prediction(model, x_data_to_predict):
    prediction = model.predict(x_data_to_predict)# Calculate the absolute errors
    print("predidction: ", prediction[0])
    return prediction[0]

def prepare_model_for_prediction():
    global ml_counter
    dataset = cleaning_data()
    X_data, y_label = slicing(dataset)
    ml_model = train_dataset(X_data, y_label)
    ml_counter = 1
    return ml_model
```