КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

Фреймворк для автоматизованого виявлення компрометації автентифікаційних даних на основі аналізу шкідливого програмного забезпечення

Виконав студент 4-го курсу
Андрій НОСКОВ _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Тетяна КАРНАУХ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики

«____»_____2023 р., протокол № ____

Завідувач кафедри

Юрій КРАК _____

(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи 49 сторінок, 32 ілюстрації, 16 джерел посилань.

ФРЕЙМВОРК, КІБЕРРОЗВІДКА, КОМПРОМЕТАЦІЯ ОБЛІКОВИХ ДАНИХ, АНАЛІЗ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Об'єктом роботи є розробка фреймворку для автоматизованого виявлення компрометації облікових даних на основі аналізу шкідливого програмного забезпечення.

Предметом роботи є добування конфігурацій з проаналізованих зразків шкідливого програмного забезпечення, визначення методів ексфільтрації та вилучення викраденої інформації, а саме логінів та паролів (облікових даних).

Метою роботи є створення автоматизованого рішення, яке може ефективно виявляти та пом'якшувати наслідки інцидентів компрометації облікових даних.

Інструменти розробки: мова програмування Python, інтегроване середовище Microsoft Visual Studio Code, провайдер проаналізованих зразків шкідливого програмного забезпечення Hatching Triage API.

Результати роботи: повнофункціональний фреймворк, здатний отримувати конфігурації зі зразків шкідливого програмного забезпечення за допомогою онлайн-пісочниць, виявляти методи ексфільтрації, а також витягувати та зберігати викрадені облікові дані.

Розроблене програмне забезпечення може використовуватися організаціями та командами безпеки, які прагнуть посилити свій захист від інцидентів, пов'язаних з компрометацією облікових даних. Воно може бути особливо цінним для тих, хто займається реагуванням на інциденти, розвідкою загроз або операціями з безпеки, дозволяючи їм автоматизувати виявлення та пом'якшення наслідків компрометації облікових даних на основі аналізу шкідливого програмного забезпечення.

ЗМІСТ

С.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

СУБД — система управління базами даних

ШПЗ — шкідливе програмне забезпечення

API — Application Programming Interface, прикладний програмний інтерфейс

CLI — Command-line interface, інтерфейс командного рядка

EML — Electronic Mail Message, електронне поштове повідомлення

FTP — File Transfer Protocol, протокол передавання файлів

IMAP — Internet Message Access Protocol

IOC — Indicator of compromise, індикатор компрометації

IOT — Internet of Things, Інтернет речей

MITM — Man in the middle, атака «людина посередині»

RAT — Remote Access Trojan, троян для віддаленого доступу

SMTP — Simple Mail Transfer Protocol, протокол пересилання електронної пошти

SSL — Secure Sockets Layer, рівень захищених сокетів

URL — Uniform Resource Locator, єдиний вказівник на ресурс

ВСТУП

Оцінка сучасного стану об’єкта розробки. У сучасному ландшафті кібербезпеки компрометація облікових даних становить значну загрозу як для окремих осіб, так і для організацій. Зловмисники використовують різні методи для отримання несанкціонованого доступу до конфіденційної інформації, включаючи крадіжку облікових даних для входу в систему за допомогою атак на основі шкідливого програмного забезпечення. Оскільки кількість і складність таких атак продовжує зростати, існує нагальна потреба в ефективних механізмах виявлення та запобігання для захисту від компрометації облікових даних.

Актуальність роботи. Постійно зростаюча частота і складність атак з метою компрометації облікових даних вимагають розробки вдосконалених фреймворків, які можуть автоматизувати виявлення скомпрометованих облікових даних. Проект має на меті задовольнити цю критичну потребу, пропонуючи комплексний фреймворк для автоматизованого виявлення компрометації облікових даних на основі аналізу шкідливого програмного забезпечення. Використовуючи можливості методів аналізу шкідливого програмного забезпечення та онлайн-пісочниць, фреймворк спрямований на виявлення та вилучення викрадених облікових даних зі зразків шкідливого програмного забезпечення, отриманих з різних каналів.

Мета й завдання роботи. Мета кваліфікаційної роботи полягає у створенні фреймворку, який може ефективно виявляти та отримувати скомпрометовані облікові дані використовуючи конфігурації шкідливого програмного забезпечення з підтримкою декількох методів ексфільтрації. Автоматизуючи процес вилучення облікових даних, фреймворк має на меті спростити виявлення та пом'якшення наслідків інцидентів, пов'язаних з компрометацією облікових даних.

Для досягнення цієї мети поставлено такі завдання:

- Розробити архітектуру фреймворку
- Розробити модуль для добування конфігурацій шкідливого програмного забезпечення з різних джерел, таких як онлайн-пісочниці.
- Реалізувати модуль для отримання повідомлень з електронної пошти за допомогою протоколу IMAP.
- Розробити модуль аналізу та обробки видобутої інформації.

Об’єкт, методи й засоби розроблення. Об’єктом розробки є фреймворк для автоматизованого виявлення компрометації облікових даних на основі аналізу шкідливого програмного забезпечення. Використовувались такі інструментальні засоби та бібліотеки: мова програмування Python, інтегроване середовище Microsoft Visual Studio Code, провайдер проаналізованих зразків шкідливого програмного забезпечення Hatching Triage API.

Можливі сфери застосування. Запропонована платформа для автоматизованого виявлення компрометації облікових даних на основі аналізу шкідливого програмного забезпечення має широке застосування в різних секторах і галузях. Її можуть використовувати команди кібербезпеки в організаціях для проактивного виявлення та пом'якшення наслідків інцидентів, пов'язаних з компрометацією облікових даних, таким чином захищаючи цінні активи та конфіденційну інформацію. Крім того, його можуть використовувати дослідники та аналітики з питань безпеки для аналізу та відстеження еволюції методів, що використовуються зловмисниками для компрометації облікових даних.

1 ПРЕДМЕТНА ОБЛАСТЬ

1.1 Превентивний пошук кіберзагроз

У сучасному світі, де технології глибоко увійшли в наше особисте та професійне життя, загроза кібератак стає все більш поширеною. Кіберзлочинці постійно вдосконалюють свої методи, націлюючись на приватних осіб, підприємства і навіть уряди за допомогою витончених і зловмисних дій. Як наслідок, необхідність превентивного пошуку кіберзагроз набула величезної актуальності для забезпечення безпеки та стабільності нашої цифрової інфраструктури.

Превентивний пошук кіберзагроз – це проактивне виявлення та зменшення потенційних кіберзагроз до того, як вони зможуть завдати значної шкоди. На відміну від реактивних підходів, які зосереджені на реагуванні на атаки після того, як вони сталися, превентивний пошук спрямований на передбачення та упереджувальне усунення вразливостей та ризиків. Завдяки постійному моніторингу, аналізу та оцінці цифрового ландшафту організації та приватні особи можуть бути на крок попереду кіберзлочинців і мінімізувати потенційний вплив атак.

Актуальність превентивного пошуку кіберзагроз неможливо переоцінити. З поширенням взаємопов'язаних пристроїв, Інтернету речей (IoT) та хмарних обчислень поверхня атак для кіберзлочинців розширилася в геометричній прогресії. Традиційних заходів безпеки, таких як брандмауери та антивірусне програмне забезпечення, вже недостатньо для боротьби з еволюціонуючим ландшафтом загроз. Вкрай важливо застосовувати проактивний підхід, який передбачає комплексну розвідку загроз, оцінку вразливостей і постійний моніторинг для виявлення та усунення потенційних слабких місць до того, як вони будуть використані.

Наслідки кібератак можуть призвести до фінансових/репутаційних втрат та компрометації конфіденційних даних, включаючи особисту інформацію та інтелектуальну власність. У таких секторах, як охорона здоров'я, фінанси та критична інфраструктура, наслідки успішної кібератаки можуть поширюватися на громадську та національну безпеку. Превентивний пошук дозволяє організаціям і приватним особам зміцнити свій захист, заздалегідь виявити потенційні загрози і вжити відповідних заходів для запобігання або пом'якшення наслідків атак.

1.2 Методи компрометації облікових даних

Методи компрометації облікових даних різноманітні і постійно розвиваються, оскільки кіберзлочинці застосовують все більш витончені технології для отримання несанкціонованого доступу до конфіденційної інформації. Шкідливе програмне забезпечення, зокрема, відіграє значну роль у викраденні облікових даних, надаючи зловмисникам цінні облікові дані для входу в систему, які можуть бути використані для подальших зловмисних дій.

Найпоширеніші методи компрометації облікових даних:

Кейлоггер (від англ. keylogger). Клавіатурні шпигуни — це тип шкідливого програмного забезпечення, яке записує натискання клавіш, що вводяться користувачем. ШПЗ перехоплює імена користувачів, паролі та іншу конфіденційну інформацію, яку вводить користувач, що дозволяє зловмисникам отримати доступ до облікових даних для входу в систему. Кейлоггери можуть працювати на різних рівнях, від рівня операційної системи до рівня браузера, що дозволяє їм перехоплювати облікові дані в різних додатках.

Фішинг (від англ. phishing). Фішингові атаки залишаються поширеним методом компрометації облікових даних. Заражені шкідливим програмним

забезпеченням електронні листи, веб-сайти або шкідливі посилання змушують користувачів вводити свої облікові дані в шахрайські форми для входу, які виглядають легітимно. Після введення облікові дані перехоплюються зловмисниками і використовуються для несанкціонованого доступу до цільових облікових записів.

Атака MITM. Під час атаки типу "людина посередині" ШПЗ перехоплює зв'язок між користувачем і сервером, на який спрямована атака. Перебуваючи між пристроєм користувача та цільовим сервером, ШПЗ перехоплює весь трафік, фактично підслуховуючи комунікацію з можливістю її модифікувати.

Програми для віддаленого доступу. RAT — це потужне шкідливе програмне забезпечення, яке надає зловмисникам повний контроль над зараженими системами. Маючи контроль над скомпрометованим пристроєм, зловмисники можуть відстежувати дії користувача, в тому числі перехоплювати введені ним облікові дані. RAT забезпечують постійне спостереження та викрадення даних, дозволяючи зловмисникам збирати облікові дані без відома користувача.

Підстановка облікових даних (від англ. Credential Stuffing). Метод передбачає використання викрадених облікових даних, отриманих з інших джерел, таких як витоки даних або чорні ринки. Шкідливе програмне забезпечення може автоматизувати процес тестування викрадених облікових даних на різних веб-сайтах або додатках, щоб отримати несанкціонований доступ до облікових записів користувачів. Наповнювачі облікових даних користуються перевагами осіб, які повторно використовують паролі на різних платформах.

Ще одним поширеним методом для компрометації облікових даних є використання стилерів.

Стилер (від англ. Stealer) [1] — це категорія шкідливих програм, спеціально розроблених для захоплення та викрадення облікових даних зі скомпрометованих систем. Основні цілі стилерів: веб-браузери, поштові

клієнти та інші програми, які зберігають облікові дані для входу в систему. Після встановлення на пристрій жертви зловмисники непомітно відстежують дії користувача, перехоплюють облікові дані та передають викрадені дані на віддалені командно-контрольні сервери, які контролюються зловмисниками, що дозволяє їм отримати широкий спектр облікових даних, включаючи імена користувачів, паролі та інші конфіденційні дані для автентифікації.

1.3 Ексфільтрація

Ексфільтрація [2] — процес несанкціонованої передачі даних зі скомпрометованої системи в зовнішнє місце, контрольоване зловмисниками.

Метою ексфільтрації є викрадення цінної інформації, наприклад, облікових даних, персональних даних або конфіденційної корпоративної інформації, для зловмисних цілей.

ШПЗ, що втілює різні методи ексфільтрації, для передачі викрадених даних зі скомпрометованих систем на зовнішні сервери використовує такі чотири найпоширеніші засоби:

- SMTP — стандартний протокол, який використовується для надсилання електронних листів.

ШПЗ може використовувати SMTP для ексфільтрації викрадених даних, створюючи та надсилаючи електронні листи на заздалегідь визначену адресу електронної пошти або віддалений сервер. Викрадені дані додаються до електронного листа або вбудовуються в його тіло.

- FTP — протокол, призначений для передачі файлів між клієнтом і сервером.

ШПЗ може використовувати FTP для завантаження викрадених даних на віддалений FTP-сервер. ШПЗ встановлює з'єднання з FTP-сервером, проходить автентифікацію за допомогою скомпрометованих облікових даних або

анонімного доступу, а потім передає викрадені дані на сервер.

- Бот Telegram [3]

Telegram — популярна платформа для обміну миттєвими повідомленнями, яка також підтримує взаємодію з ботами. ШПЗ може використовувати API ботів Telegram для встановлення каналу зв'язку зі шкідливим ботом. Викрадені дані можуть бути надіслані боту у вигляді повідомлень або завантажених файлів.

- Веб-шлюз

У даному випадку зловмисники для отримання та зберігання викрадених даних створюють веб-шлюз, який є шкідливим сервером або веб-сайтом. ШПЗ може бути запрограмоване на встановлення з'єднання з веб-шлюзом і надсилання викрадених даних за протоколами HTTP, HTTPS, DNS або SSH. Веб-шлюз виступає в ролі приймача, збираючи та зберігаючи викрадені дані для подальшого вилучення або подальшої експлуатації.

1.4 Методи аналізу шкідливого програмного забезпечення

Методи аналізу шкідливого програмного забезпечення мають вирішальне значення для розуміння поведінки, функціональності та потенційних ризиків, пов'язаних зі шкідливим програмним забезпеченням. Аналітики використовують різні методи, щоб дослідити зразки шкідливого програмного забезпечення та отримати уявлення про його можливості та наміри. Три найпоширеніші методи аналізу шкідливого програмного забезпечення – це статичний аналіз, динамічний аналіз і поведінковий аналіз.

Статичний аналіз передбачає вивчення шкідливого програмного забезпечення без його виконання, зосереджуючись на коді та структурі зразка. Аналітики перевіряють двійковий або вихідний код, витягують рядки і потенційні індикатори компрометації (IOC), а також виявляють будь-які

підозрілі або шкідливі шаблони. Метод допомагає ідентифікувати відомі сигнатури, розпізнавати поширені сімейства шкідливих програм і виявляти методи обфускації, які використовуються для уникнення виявлення. Статичний аналіз можна виконати за допомогою спеціалізованих інструментів і дизасемблерів, щоб глибше зрозуміти функціональність шкідливого програмного забезпечення.

Динамічний аналіз передбачає запуск шкідливого програмного забезпечення в контрольованому середовищі, наприклад, у віртуальній машині або пісочниці, для спостереження за його поведінкою. Аналітики відстежують взаємодію між шкідливим програмним забезпеченням і системою, перехоплюючи системні виклики, мережевий трафік і модифікації файлів. Метод дозволяє виявити поведінку під час виконання, наприклад, завантаження файлів, передачу команд і керування або модифікацію системи. Динамічний аналіз дає уявлення про фактичні дії шкідливого програмного забезпечення та його потенційний вплив на цільову систему.

Поведінковий аналіз фокусується на розумінні загальної поведінки та намірів шкідливого програмного забезпечення. Він передбачає аналіз дій і взаємодій шкідливого програмного забезпечення протягом усього його життєвого циклу, від початкового зараження до виконання шкідливих дій. Спостерігаючи та документуючи поведінку шкідливого програмного забезпечення, аналітики можуть виявити специфічні методи, такі як перехоплення клавіатури, крадіжка облікових даних або витік даних. Поведінковий аналіз допомагає зрозуміти мету і потенційний вплив шкідливого програмного забезпечення, що сприяє розробці відповідних стратегій пом'якшення наслідків.

Для допомоги в аналізі шкідливого програмного забезпечення існує кілька онлайн-сервісів та інструментів. VirusTotal [4] – популярна онлайн-платформа, яка дозволяє аналітикам завантажувати зразки шкідливого програмного забезпечення та отримувати звіти про стан їх виявлення в різних

антивірусних системах . Вона надає швидкий огляд поширеності шкідливого програмного забезпечення та потенційних ризиків. Онлайн-пісочниці, такі як Hatching Triage [5], Cuckoo Sandbox [6] та CAPE Sandbox [7], пропонують контрольоване середовище для виконання зразків шкідливого програмного забезпечення та моніторингу їхньої поведінки в режимі реального часу.

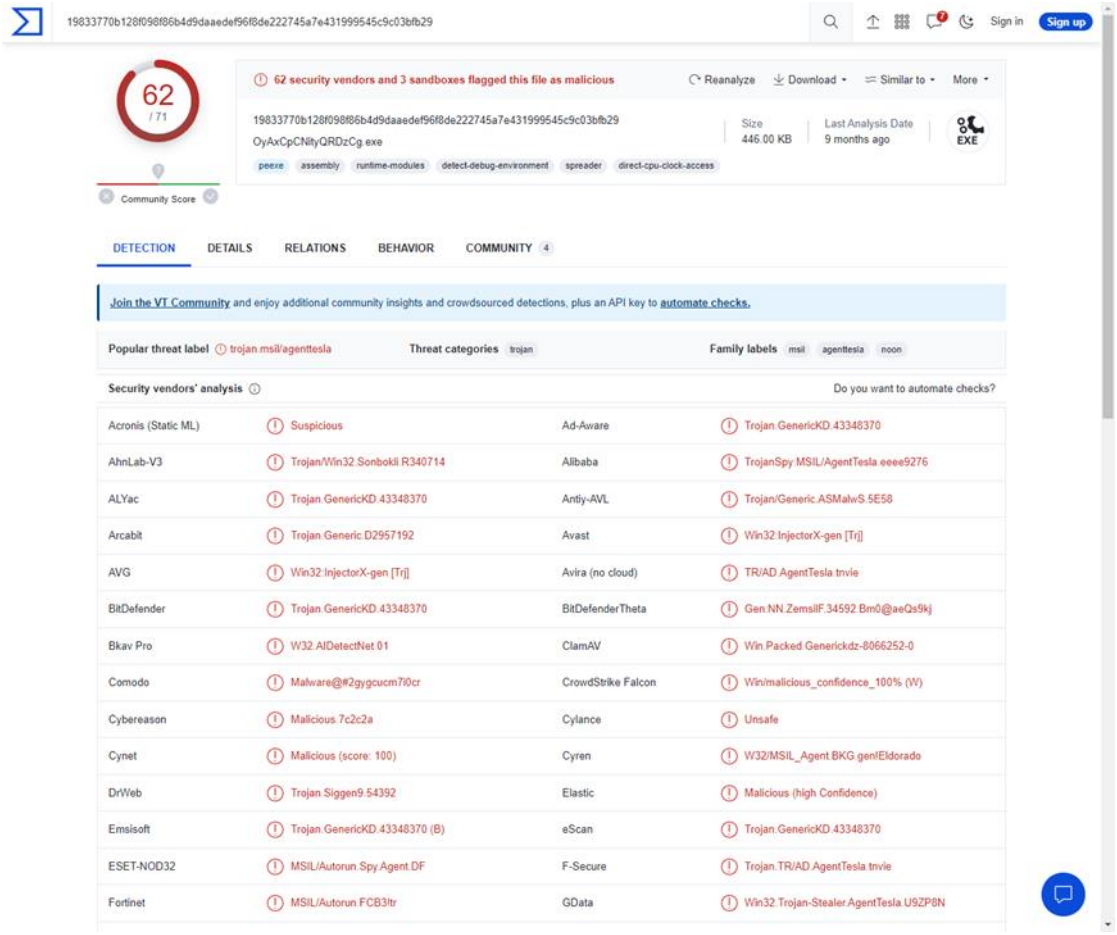

Рисунок 1 – Приклад звіту з VirusTotal

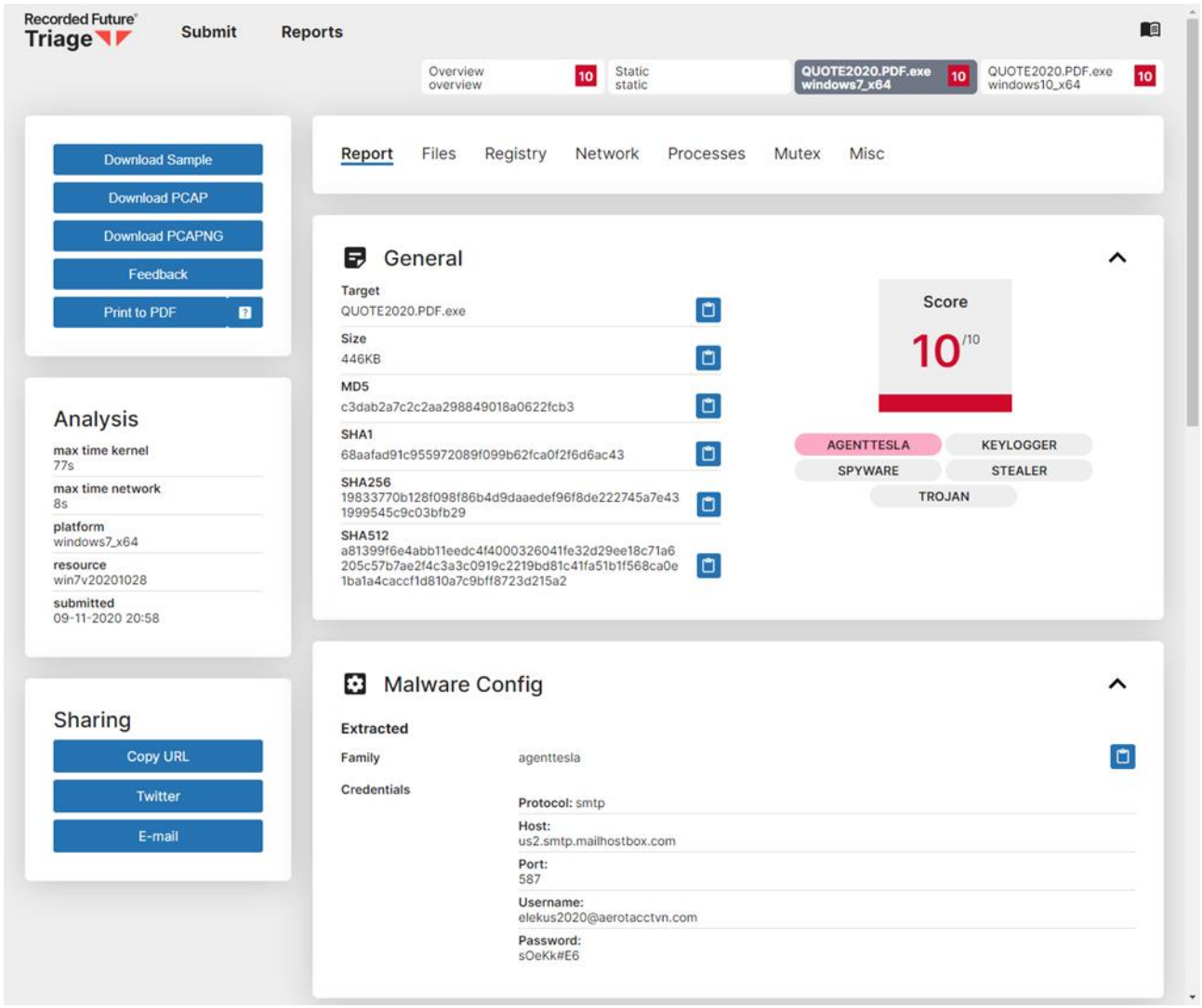

Рисунок 2 – Приклад звіту з онлайн-пісочниці Hatching Triage

2 АРХІТЕКТУРА РОЗРОБЛЮВАНОГО ФРЕЙМВОРКУ

2.1 Огляд архітектурних рішень

Фреймворк розроблений як CLI-додаток, написаний на мові Python. Архітектура фреймворку базується на монолітному дизайні, в якому кожен компонент об'єднаний в єдиний додаток. Така стратегія спрощує розгортання та управління, оскільки весь набір необхідних функцій міститься в єдиній кодовій базі.

Основний цикл виконання, який відповідає за координацію виконання завдань і нагляд за робочим процесом, знаходиться в центрі фреймворку. Цикл взаємодіє з компонентами та базою даних для отримання завдань, обробляє їх відповідно до останнього часу виконання та заданих інтервалів.

Фреймворк використовує модульну структуру, де кожен компонент фокусується на конкретних завданнях та обов'язках. Такий модульний підхід покращує ремонтопридатність і дозволяє легко розширювати систему в майбутньому. Компоненти слабко пов'язані між собою, що дозволяє незалежну розробку та тестування кожного модуля.

Ключові компоненти фреймворку включають:

- Екстратор конфігурацій ШПЗ: відповідає за отримання конфігурацій ШПЗ з онлайн-пісочниці.
- Збирачі даних, або грабери: відповідають за отримання скомпрометованих даних з різних джерел в необробленому вигляді, тобто журналів, або логів створених ШПЗ як звіт про свою роботу.
- Аналізатор: аналізує необроблені дані, шукає шаблони, що вказують на потенційно скомпрометовані автентифікаціні дані (наприклад, логіни, паролі, URL-адреси) у необроблених даних.

Використання SQLite [8] в якості СУБД для фреймворку спрощує процес розробки, забезпечуючи легке розгортання та управління в різних системах. Портативність та автономність SQLite полегшує спільне використання та розповсюдження фреймворку без необхідності додаткових програмних залежностей від баз даних.

Фреймворк взаємодіє із зовнішніми сервісами завдяки використанню офіційних або популярних бібліотек, спеціально розроблених для кожного сервісу. Наприклад, при взаємодії з онлайн-пісочницею Hatching Triage використовується — офіційний клієнт API, що надається платформою [9].

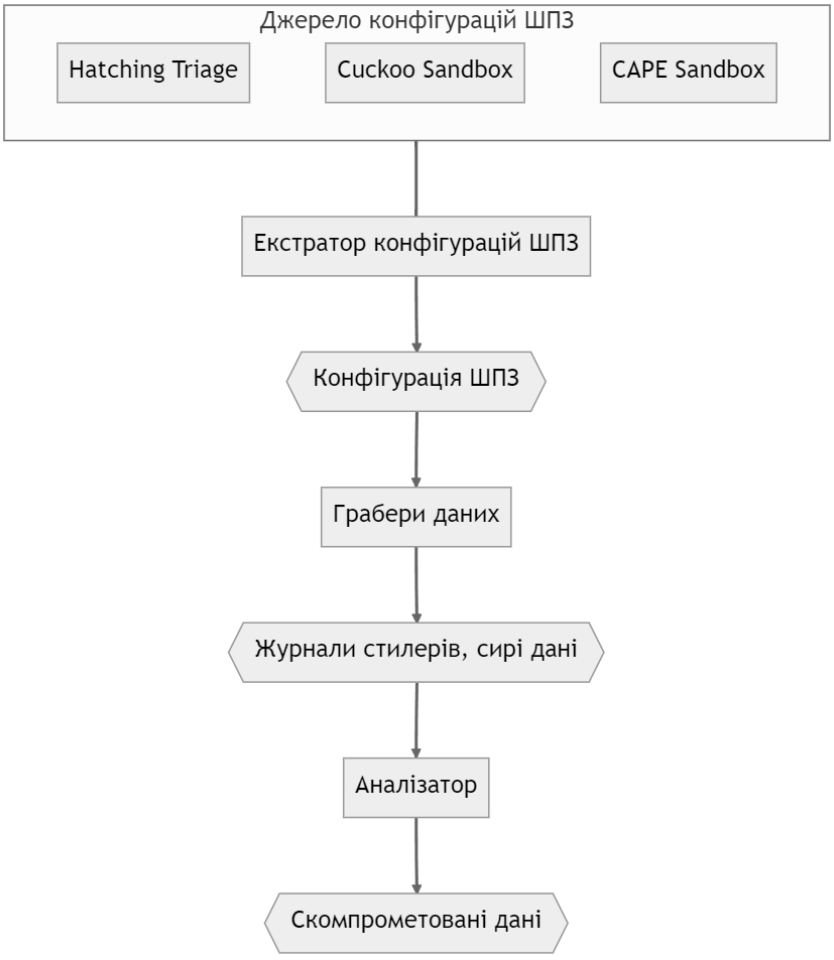

Рисунок 3 – Потік даних фреймворку

2.2 Використання онлайн-пісочниці Hatching Triage

У фреймворку використовується онлайн-пісочниця Hatching Triage як основне джерело для вилучення конфігурацій шкідливого програмного забезпечення. Хоча фреймворк не обмежується використанням виключно Hatching Triage, даний сервіс пропонує кілька переваг, які роблять його цінним вибором.

Однією з ключових переваг інтеграції Hatching Triage у фреймворк є можливість делегувати роботу з аналізу шкідливого програмного забезпечення зовнішньому сервісу. Використовуючи можливості Hatching Triage, фреймворк знімає з себе завдання запуску зразків шкідливого ПЗ в середовищі пісочниці та вилучення їхніх конфігурацій, що дозволяє фреймворку зосередитися на обробці та ефективному використанні отриманих даних.

Hatching Triage пропонує широкий вибір екстракторів (рис. 4), які відповідальні за виявлення та вилучення конфігураційних даних з різних типів ШПЗ (рис. 5), включаючи кейлоггери, стилери та трояни віддаленого доступу.

Крім того, Hatching Triage надає детальні звіти про проаналізоване шкідливе програмне забезпечення. Ці звіти дають уявлення про поведінку, мережеву активність та інші характеристики шкідливого програмного забезпечення. Фреймворк може використовувати цю інформацію для кращого розуміння можливостей шкідливого програмного забезпечення, методів його проникнення та потенційних цілей. Детальні звіти сприяють всебічному розумінню конфігурації шкідливого програмного забезпечення та допомагають у розробці ефективних контрзаходів.

Ще однією перевагою використання Hatching Triage є простота використання, що забезпечується доступним API. API забезпечує безперешкодну інтеграцію з фреймворком, що дозволяє автоматично надсилати зразки шкідливого програмного забезпечення для аналізу та отримувати

витягнуті дані про конфігурацію. Цей спрощений процес усуває необхідність ручного втручання і зменшує складність інтеграції Hatching Triage у фреймворк.

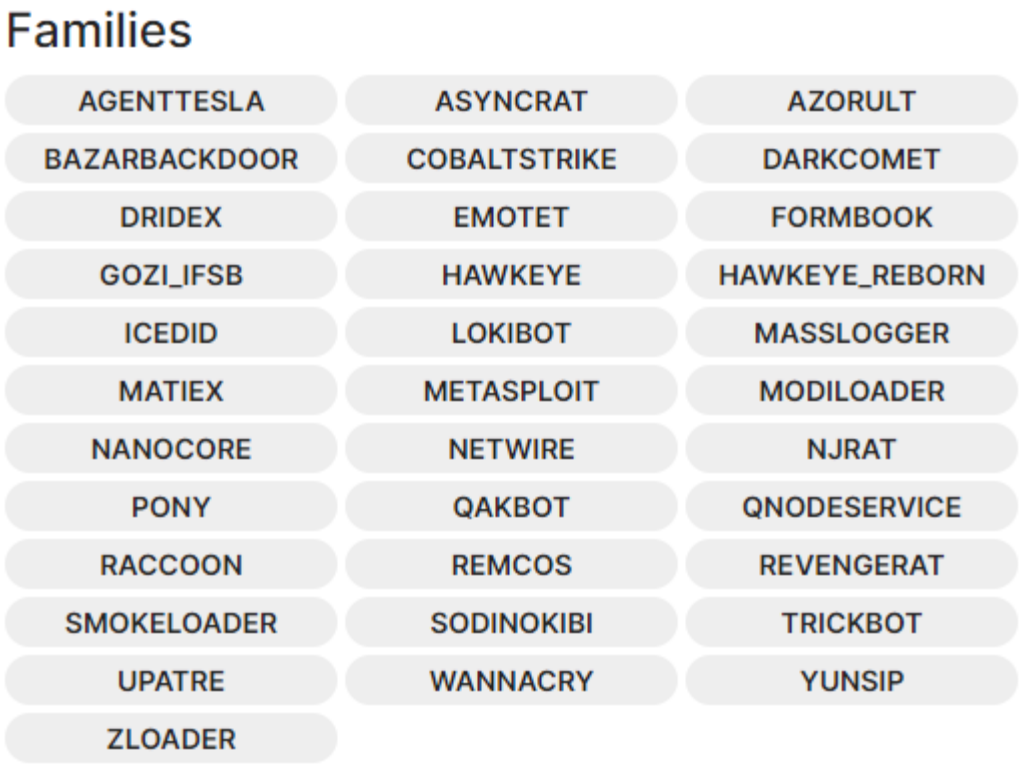

Рисунок 4 – Список підтримуваних екстракторів [10]

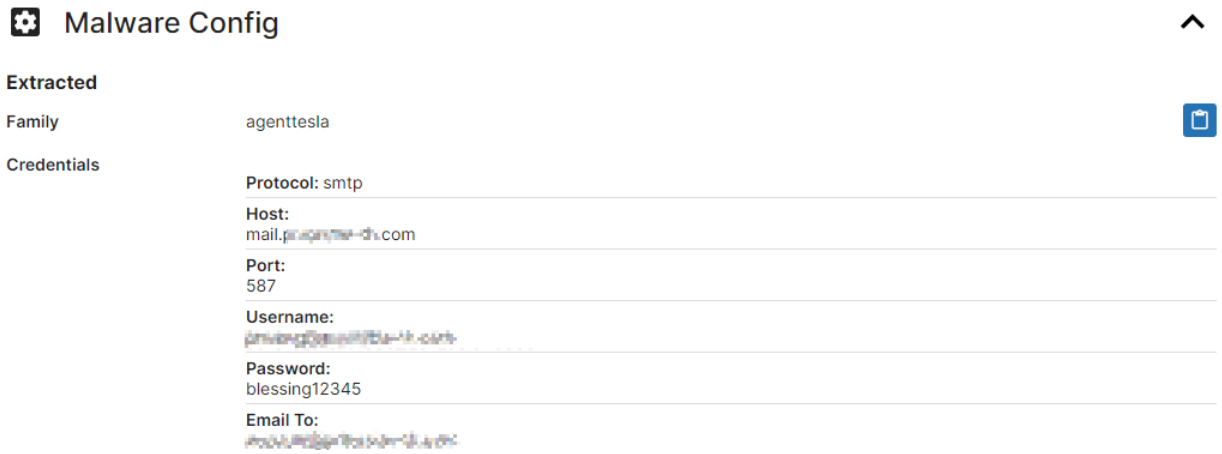

Рисунок 5 – Приклад конфігурації ШПЗ [11]

2.3 Аналіз необроблених даних

На початкових етапах процесу аналізу даних виникла проблема розбору сирих даних, отриманих з різних стилерів. Спочатку була прийнята стратегія створення окремих аналізаторів для кожного стилера, в результаті чого було розроблено понад тридцять окремих аналізаторів. Однак такий підхід виявився ненадійним і непередбачуваним, незважаючи на те, що він давав змогу визначити конкретне ШПЗ, відповідальне за викрадення облікових даних.

Щоб оптимізувати процес аналізу даних і підвищити загальну продуктивність, фокус був зміщений у бік розробки універсального аналізатору. Такий підхід дозволив консолідувати зусилля, що призвело до створення рішення, здатного ефективно обробляти більшість популярних стилів. Це усунуло необхідність підтримувати та оновлювати численні специфічні аналізатори, оскільки один адаптивний тепер міг обробляти різні формати логів.

При аналізі необроблених даних, отриманих від стилерів, було виявлено декілька ключових моментів, які є важливим в розумінні алгоритму дій щодо їх аналізу:

1. Структура логів
2. Формати логів
3. Структура логів з обліковими даними

Більшість стилерів, часто написаних на C# для швидкого розроблення, мають схожу структуру логів. Цю схожість можна пояснити тим, що некваліфіковані розробники роблять “copy-paste” коду для створення ШПЗ з метою отримання швидкого прибутку. Така стандартизована структура полегшує процес аналізу, оскільки можна виявити загальні закономірності.

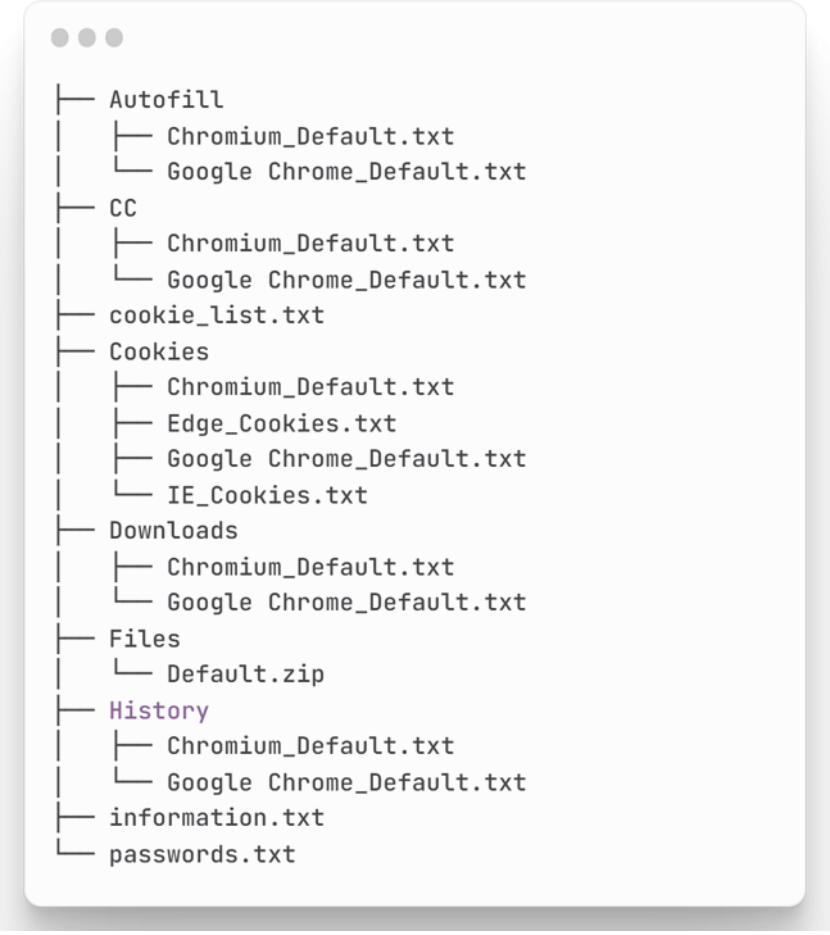

Рисунок 6 – Приклад структури логів зі стилерів Raccoon, Anubis

Одним із поширених форматів звіту стилерів є HTML-файл, що містить викрадені облікові дані та основну інформацію про скомпрометованого користувача, таку як ім'я користувача, ім'я хоста, IP-адресу, часову мітку та дані операційної системи. Інший популярний формат — zip-архів, який може містити безліч додаткової інформації, викраденої у жертви, включаючи файли cookie, форми автоматичного заповнення в браузерах, списки встановленого програмного забезпечення, скріншоти робочого столу, паролі та профілі до крипто-гаманців.

Рисунок 7 – Приклад HTML-логу зі стилера AgentTesla

У викрадених облікових даних файл логу паролів часто складається з декількох записів, кожен з яких складається з декількох полів. Поля можуть містити назву програми або веб-сайту, URL-адресу, ім'я користувача, пароль і, можливо, інші релевантні дані. Аналіз цієї структури дозволяє ідентифікувати та витягти паролі, пов'язані з певними обліковими записами або платформами.

```
Anubis Stealer v.2.0 Reload Russian Paradise

IP : XX.XX.XX.XX
Country Code : RU
Country :Russia
State Name : Krasnoyarskiy Kray
City :Krasnoyarsk
Timezone :Asia/Krasnoyarsk
ZIP : 660000
ISP : EDN SOVINTEL
Coordinates :XX.XXXX , XX.XXXX

Username : 8IIJwP1eKsoL
PCName : DESKTOP-VTTZIB1
UUID : 6a0ba276-d85a-b488-28b6-89ed85ec22ad
HWID : 88871F81EC6E4FAEA5AECF186CDCBFA9
OS : Windows 10
CPU : Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz
CPU Threads: 4
GPU : Microsoft Basic Display Adapter
RAM :2 GB
MAC :22:BD:20:52:41:53
Screen Resolution :1024x768
System Language : English (United States)
Layout Language : US
PC Time : 10/2/2020 6:04:42 PM
Browser Versions
Mozilla Version: 61.0
Chrome Version:77.0.3865.90
```

Рисунок 8 – Приклад вкраденої інформації про жертву з логів стилера Anubis

Розглянемо основні етапи роботи з сирими даними представлені на схемі нижче.

Рисунок 9 – Основні етапи роботи з сирими даними

Алгоритм.

Крок 1. Екстракція.

Першим кроком є добування даних з різних форматів файлів, таких як zip, rar, tgz, 7z архіви, а також з тіл EML-повідомлень і відповідних вкладень. Архіви рекурсивно розпаковуються в тимчасову директорію в файловій системі, а для кожного EML-повідомлення та його вкладень створюються віртуальні файли, які полегшують подальший аналіз.

Крок 2. Нормалізація.

Витягнуті дані можуть бути в різних форматах, включаючи HTML. Для забезпечення узгодженості та полегшення аналізу дані нормалізуються. Наприклад, якщо дані у форматі HTML, вони перетворюються на звичайний текст без HTML-розмітки для полегшення обробки та інтерпретації.

Крок 3. Попередня обробка.

Поділ нормалізованого тексту на послідовні блоки, де кожен блок являє собою запис у форматі «ключ—значення».

Крок 4. Обробка блоків.

Крок 4.1. Аналіз блоків.

Порівнюються записи у форматі «ключ—значення» в кожному блоці з попередньо визначеними регулярними виразами. Цей процес зіставлення допомагає витягти з даних певну інформацію, таку як імена користувачів, паролі, мітки часу або інші релевантні точки даних.

Крок 4.2. Ідентифікація блоків.

На основі вмісту кожного блоку визначається його тип: чи містить блок інформацію про обліковий запис або інформацію, про комп’ютер жертви.

Крок 4.3. Групування блоків.

У деяких випадках може існувати кілька блоків одного типу, наприклад, кілька блоків, що містять інформацію про скомпрометовану систему, тоді ці пов'язані блоки необхідно об’єднати в один консолідований блок.

Однак важливо зазначити, що одним з основних недоліків ідеї даного аналізатору є те, що він не надає чіткої інформації про походження факту компрометації, тобто аналіз зосереджується в першу чергу на вмісті самих викрадених даних, а не на тому, щоб отримати уявлення про конкретне ШПЗ, яке до цього причетне.

2.4 Інтерфейс фреймворку

Фреймворк надає гнучкий і розширюваний інтерфейс для інтеграції модулів у систему. Щоб забезпечити безперебійну інтеграцію, кожен модуль повинен декларувати свою конфігурацію, використовуючи стандартизований формат, визначений фреймворком. Конфігурація визначає необхідні параметри та налаштування, які потрібні модулю для належного функціонування.

Інтерфейс кожного модуля повинен мати наступний вигляд:

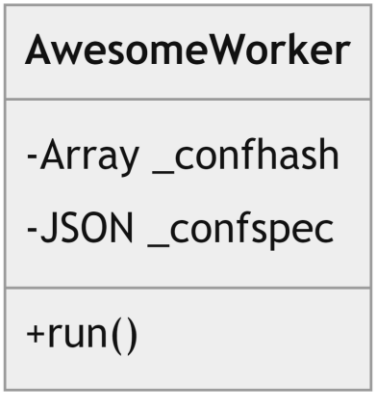

Рисунок 10 – Інтерфейс модуля

```
_confhash = ['query', 'limit']

_confspec = {
    'type': 'object',
    'properties': {
        'query': {'type': 'string'},
        'limit': {'type': 'number', 'minimum': 1, 'default': 10},
        'api_url': {'type': 'string'},
        'api_key': {'type': 'string'}
    },
    'required': ['api_key']
}
```

Рисунок 11 – Приклад конфігурації модуля

В наведеному вище прикладі поле _confhash – містить назви важливих параметрів модуля. На основі даного поля створюється унікальний ідентифікатор, який відрізняє завдання на основі цього модуля від інших завдань, що дозволяє уникнути створення дублікатів. Поле _confspec – задає всі можливі параметри для конфігурації специфічних налаштувань у форматі JSON [12], які перевіряються перед виконанням завдання.

Кожен модуль повинен реалізувати головний метод run(), який слугує точкою входу для виконання модуля в загальному потоці програми. Цей головний метод викликається і управляється фреймворком, що дозволяє йому організовувати виконання декількох модулів у скоординований спосіб. Дотримуючись такого підходу, фреймворк забезпечує чітку та послідовну структуру для розробки та виконання модулів.

Фреймворк заохочує модульність і незалежність між модулями. Кожен модуль розроблений так, щоб працювати як автономна одиниця з власним набором залежностей. Такий принцип проектування дозволяє розробляти і тестувати модулі ізольовано, що робить їх більш придатними для повторного використання і полегшує їх підтримку.

Вимоги до програмного забезпечення для фреймворку:

- Python – версія 3.9 або пізніше

- Рекомендована операційна система – Linux

Необхідні залежності для фреймворку встановлюються за допомогою менеджера пакетів pip. Перед використанням користувач повинен налаштувати змінні оточення.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ФРЕЙМВОРКУ

3.1 Загальна структура додатку

Реалізація фреймворку ґрунтується на концепції робітників, які є екземплярами, що зображують запущені завдання. Кожен робітник є екземпляром класу Worker і відповідає конкретному завданню, визначеному в класі Task. Завдання конфігуруються на основі специфіки конкретного ШПЗ, включаючи необхідний грабер для роботи з певним протоколом і пов'язані з ним облікові дані. Завдання також мають інтервал, який вказує на те, як часто вони повинні виконуватися.

Клас Task зберігає важливу інформацію про кожне завдання, таку як час останнього запуску, час останнього завершення, останній результат і кількість послідовних помилок. Якщо кількість помилок перевищує заданий ліміт, завдання автоматично вимикається.

Рисунок 12 – Діаграма класу Task

Сирі дані, представлені класом RawData, зберігаються в локальній файловій системі в оригінальному форматі і супроводжуються відповідним записом у базі даних. Для кожного елементу необроблених даних обчислюється хеш MD5, який слугує унікальним ідентифікатором даних, що дозволяє ефективно зберігати, знаходити і порівнювати елементи даних.

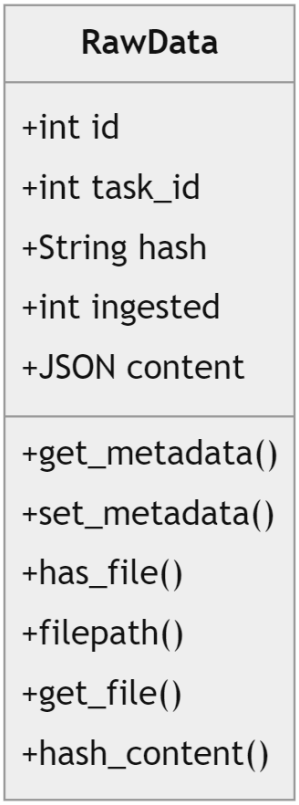

Рисунок 13 – Діаграма класу RawData

Одним з основних завдань аналізатора є робота з різними типами файлових об'єктів, кожен з яких представляє певне джерело даних. Ці файлові об'єкти забезпечують стандартизований спосіб ідентифікації та обробки різних типів даних, що зустрічаються під час аналізу.

Наявні типи файлових об'єктів:

- FileObject – описує базовий інтерфейс для інших об’єктів, безпосередньо не використовується.

- PersistentFileObject – представляє оригінальний вихідний файл у незмінному форматі.
- VirtualObjectFile – розроблений для обробки повідомлень електронної пошти. Коли вихідні дані містять повідомлення електронної пошти, аналізатор витягує тіло листа і будь-які вкладення, пов'язані з ним.
- TemporaryFileObject – використовується, коли вихідні дані містять архіви, такі як ZIP, RAR або GZ файли, для тимчасового зберігання розпакованих файлі.

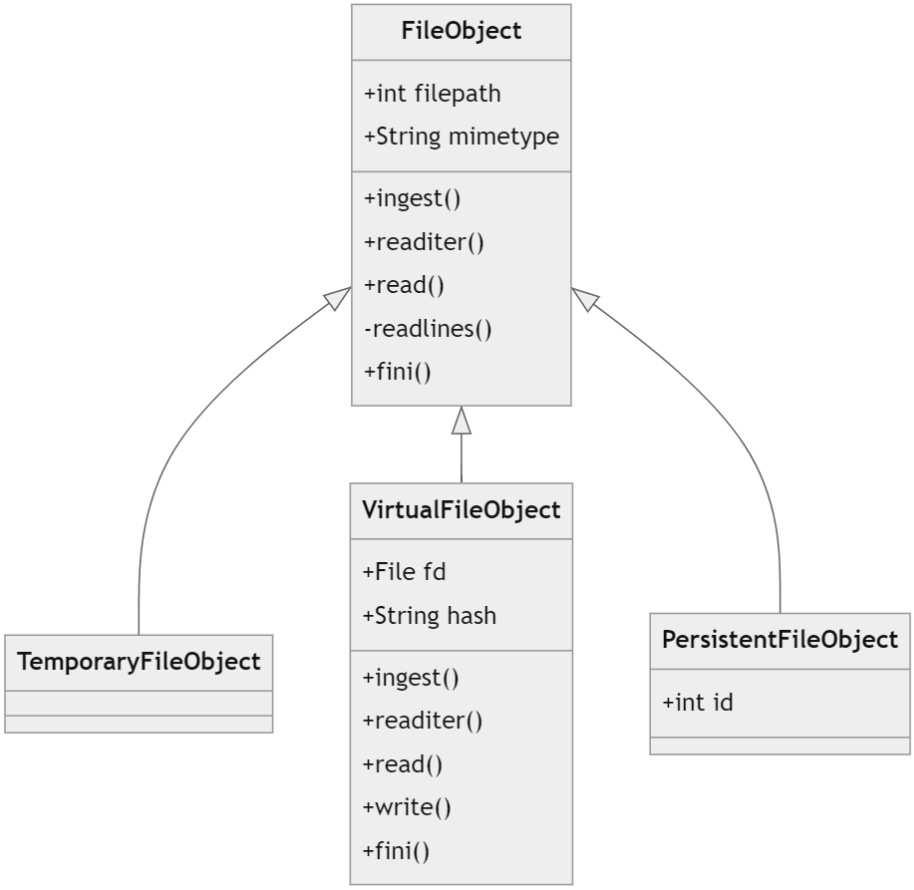

Рисунок 14 – Діаграма класів для файлових об’єктів

3.2 Модуль добування конфігурацій шкідливого програмного забезпечення

Модуль, представлений класом HatchingTriageApi, призначений для вилучення конфігурацій зразків ШПЗ з Hatching Triage API і генерації відповідних завдань для граберів. Розглянемо його ключові функціональні можливості.

Під час ініціалізації модуль визначає свої конфігураційні специфікації, які включають запит, ліміт, URL-адресу API та ключ API. Параметр query визначає критерії пошуку сімейств ШПЗ, а параметр limit – максимальну кількість зразків, які можна отримати. URL-адреса API та ключ API необхідні для доступу до API Hatching Triage.

```python
class HatchingTriageApi(Worker):

    _confhash = ['query', 'limit']

    _confspec = {
        'type': 'object',
        'properties': {
            'query': {'type': 'string'},
            'limit': {'type': 'number', 'minimum': 1, 'default': 10},
            'api_url': {'type': 'string'},
            'api_key': {'type': 'string'}
        },
        'required': ['api_key']
    }

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self.tracked_malware = ['agenttesla', 'snakekeylogger', 'matiex', 'hawkeye', 'hawkeye_reborn']

        self.query = self.task.config.get('query', ' OR '.join([f'family:{x}' for x in self.tracked_malware]))
        self.limit = self.task.config.get('limit', 200)
        self.api_url = self.task.config.get('api_url', 'https://api.tria.ge')
        self.api_key = self.task.config.get('api_key', None)

        self.max_workers = 50

        self.feed = HatchingFeed(self.api_url, self.api_key)
```

Рисунок 15 – Ініціалізація модуля для роботи з Hatching Triage API у конструкторі класу HatchingTriageApi

Під час виконання методу run() об'єкт класу HatchingTriageApi взаємодіє з API Hatching Triage і отримує зразки ШПЗ на основі запиту. Він також враховує

час останнього завершення пов'язаного завдання, щоб визначити часовий діапазон для пошуку змінених або нових зразків.

Для кожного знайденого зразка модуль перевіряє, чи були отримані пов’язані конфігурації ШПЗ. Якщо нічого не знайдено, зразок пропускається, оскільки він не містить потрібної інформації. Однак, якщо хоч одну конфігурацію було отримано, модуль створює завдання для граберів на основі кожної конфігурації.

Щоб полегшити створення завдань грабера, Worker використовує функцію define_task(), яка визначає назву грабера та конфігурацію завдання на основі конкретного зразка і його конфігурації. Потім завдання зберігається в базі даних за допомогою методу add_task().

```python
async def run(self):
    logger.debug('run hatching triage worker')

    lastcheckdelta = self.task.config.get('timedelta_days', 7)
    lastchecktime = datetime.fromtimestamp(
        self.task.lastfinishtime - timedelta(days=lastcheckdelta).total_seconds(),
        tz=timezone.utc
    )

    new_tasks_created = 0

    # Fetch samples from Hatching Triage API
    async for sample in self.feed.public_samples(self.query, lastchecktime, unique=True, limit=self.limit):

        # If no credentials found skip
        if not sample.extracted:
            continue

        # Create grabber task for each credential
        for config in sample.extracted:
            logger.debug(f'process sample {sample.id}, config: {config}')

            try:
                tname, tconfig = define_task(sample, config)
                tid = await self.core.add_task(tname, config=tconfig)

                if tid:
                    logger.info(f'a new task {tid} successfully created, config: {tconfig}')

                    new_tasks_created += 1
            except Exception as e:
                logger.exception(e)

    logger.info('finished with %d new tasks created', new_tasks_created)
```

Рисунок 16 – Метод run з класу HatchingTriageApi

```
def define_task(sample, config):
    protocol = config['protocol']

    if protocol == 'smtp' or protocol == 'imap':
        return ('EmailGrabber', {
                'sample': sample.to_db(),
                'host': config['host'],
                'username': config['username'],
                'password': config['password'],
                })

    raise WorkerNotFound(f'No relevant worker found: {config}')
```

Рисунок 17 – Допоміжна функція define_task

3.3 Модуль отримання необроблених даних з пошти

Модуль, представлений класом EmailGrabber, відповідає за отримання електронних листів і вилучення відповідної інформації з поштового сервера за допомогою протоколу IMAP.

Під час ініціалізації програма перевіряє і встановлює необхідні параметри конфігурації, включаючи хост, ім'я користувача і пароль для поштового сервера. Крім того, він дозволяє вказати необов'язкові параметри, такі як використання SSL, таймаут, критерії пошуку та максимальна кількість невдалих спроб.

Грабер визначає набір критеріїв, представлених змінною _criteria, які використовуються для ідентифікації листів, що містять потенційні облікові дані. Ці критерії включають пошук певних текстових шаблонів у тілі та темі листа. Також наявна підтримка користувацьких критеріїв, які можна вказати в конфігурації завдання.

Додатково використовується стороння бібліотека imap_tools, яка надає допоміжні методи для роботи з протоколом IMAP, а також врахує певні нюанси, які можуть виникнути на різних серверах, такі як проблеми з кодуванням в назвах папок чи листів.

```
def __init__(self, *args, **kwargs):
    super().__init__(*args, **kwargs)

    self._host = self.task.config['host']
    self._username = self.task.config['username']
    self._password = self.task.config['password']

    self._use_ssl = self.task.config.get('ssl', True)
    self._timeout = self.task.config.get('timeout', 30)

    self._max_unsuccessful_attempts = 24
    self._custom_criteria = self.task.config.get('lookup_criteria', None)

    self._criteria = OR(
        OR(text=['name=User', 'name=Password']),
        AND(
            OR(body=["Username", "USR", "User"]),
            OR(body=["Password", "PSWD", "Pass"]),
        ),
        OR(body=['Snake Keylogger'], subject=["Snake Keylogger"]),
    )

    if self._custom_criteria:
        self._criteria = OR(self._criteria, self._custom_criteria)

    self._last_check = self.task.config.get('last_check', {})

    self._imap = ImapModule()
```

Рисунок 18 – Конструктор класу EmailGrabber

Метод run() встановлює з'єднання з поштовим сервером за допомогою протоколу IMAP і облікових даних заданих в конфігурації завдання. Перша спроба з’єднання намагається використати SSL-з'єднання, у випадку помилки або тайм-ауту, SSL для завдання вимикається.

```
try:
    self._imap.open(self._host, use_ssl=self._use_ssl, ssl_context=create_default_context(), timeout=self._timeout)
except SSLError:
    logger.debug("ssl disabled")

    # Failed to establish SSL session
    # Disable SSL in config
    self._use_ssl = self.task.config['ssl'] = False
    self._imap.open(self._host, use_ssl=self._use_ssl, timeout=self._timeout)

logger.debug(f'imap login: user: %s, password: %s', self._username, self._password)
self._imap.login(self._username, self._password)
```

Рисунок 19 – Підключення до поштового серверу

Основне тіло методу run() рекурсивно перевіряє кожну папку, отримує її статус та порівнює його з останнім перевіреним значенням UID (унікального ідентифікатора) для цієї папки. Використовуючи визначені критерії та діапазон дат, грабер шукає листи, які відповідають критеріям, і витягує необхідну

інформацію. Потім витягнуті дані зберігаються зі змістовним ім'ям файлу для подальшого аналізу. Останній перевірений UID повідомлення для кожної папки зберігається в базі даних, щоб уникнути повторного вилучення.

```
for folder in self._imap.folders():
    try:
        self._imap.select(folder['name'])
    except MailboxFolderSelectError:
        continue

    try:
        folder_status = self._imap.get_folder_status(folder['name'])
    except MailboxFolderStatusError:
        continue

    last_uid = self._last_check.get(folder['name'], 1)
    next_uid = folder_status['UIDNEXT']

    # No updates required
    # Folder is up to date
    if next_uid == last_uid:
        continue

    for (s_uid, e_uid) in pairwise(range_with_end(last_uid, next_uid, 1000)):
        try:
            msgs = self._imap.fetch(
                AND(
                    self._criteria,
                    'UID %d:%d' % (s_uid, e_uid),
                    date_gte=date.today() - relativedelta(years=1)
                ),
                mark_seen=False,
                bulk=True
            )

            for msg in msgs:
                msg_filename = " - ".join([msg.subject, msg.from_, msg.date_str]).replace('/', '_') + '.eml'
                await self.save_extracted(msg_filename, msg.obj.as_bytes())
                total_extracted += 1

        finally:
            self._last_check[folder['name']] = e_uid
```

Рисунок 20 – Процес рекурсивного пошуку листів за критерієм

3.4 Модуль аналізу необроблених даних

Перед тим, як перейти до більш докладного опису, зазначимо, що обробка корисного навантаження, що міститься в листах, архівах та інших файлах, відбувається на підставі значення їх MIME-типу, за яким визначається відповідна дія синтаксичного аналізу.

Програмна реалізація модулю аналізу необроблених даних включає в себе декілька ключових етапів:

1. Планування завдань екстракції.

У функції schedule_loop створюється набір extraction_tasks для відстеження поточних завдань екстракції. Функція schedule_extraction призначена для планування підпрограми екстракції, яка створює завдання і додає його до набору завдань екстракції. Цей механізм гарантує, що декілька завдань можуть бути заплановані одночасно.

2. Аналіз текстових даних.:

Якщо MIME-тип починається з "text/" [13, 14], це означає, що корисне навантаження містить текстові дані. У цьому випадку виконується функція run_parsers, яка виконує синтаксичний аналіз текстових даних.

3. Аналіз повідомлень з електронної пошти.

Для корисного навантаження з MIME-типами, що починаються з "message/" [15], які зазвичай представляють собою повідомлення електронної пошти, підпрограма extract_email планується на виконання за допомогою функції schedule_extraction. Ця підпрограма відповідає за видобування та аналіз вмісту листа, що дає змогу надалі обробляти дані, пов'язані з електронною поштою.

```
async def extract_email(self, node):
    payload = node.value

    msg = message_from_bytes(await payload.read())
    msgname = mesg_filename(msg)

    msg_node = await node.open((msgname,))

    if msg.html:
        await msg_node.update(
            await VirtualFileObject.ainit(msgname, byts=bytes(msg.html, encoding='utf-8'), mimetype='text/html')
        )
    elif msg.text:
        await msg_node.update(
            await VirtualFileObject.ainit(msgname, byts=bytes(msg.text, encoding='utf-8'), mimetype='text/plain')
        )

    if msg_node.value:
        await self.add_data(msg_node)

    for att in msg.attachments:
        if att.size == 0:
            continue
        att_node = await node.open((att.filename,))

        await att_node.update(await VirtualFileObject.ainit(att.filename, byts=att.payload, mimetype=att))
        await self.add_data(att_node)
```

Рисунок 21 – Процес створення віртуальних файлів для повідомлень та вкладень

4. Аналіз архіву:

Коли MIME-тип відповідає певним типам архівних файлів, наприклад, "application/zip", "application/rar", "application/x-7z-compressed" або "application/gzip" [16], код планує виконання відповідних процедур розпакування (extract_zip, extract_rar, extract_archive). Ці підпрограми виконують видобування файлів з архіву і подальший аналіз видобутого вмісту.

```
async def extract_zip(self, node: Node):
    return await self.extract_arc(zipfile.ZipFile, node)

async def extract_rar(self, node: Node):
    return await self.extract_arc(rarfile.RarFile, node)

async def extract_arc(self, cls, node: Node):
    tmpdir = tempfile.TemporaryDirectory()
    self.tempdirs.append(tmpdir)

    payload = node.value

    with cls(payload.filepath) as arc:
        for entry in arc.infolist():
            if entry.is_dir() or entry.file_size == 0:
                continue

            filen = entry.filename

            if re.search(dirblacklist, filen) or pathsuffix(filen) not in suffixwhitelist or re.search(fileblacklist, filen):
                continue

            filep = await executor(arc.extract, entry, tmpdir.name)

            filenode = await node.open(normpath(filen))
            await filenode.update(await TemporaryFileObject.ainit(filep))
            await self.add_data(filenode)

    if not node.kids:
        await node.remove()

async def extract_archive(self, node: Node):
    tmpdir = tempfile.TemporaryDirectory()
    self.tempdirs.append(tmpdir)

    await executor(patoolib.extract_archive, node.value.filepath, verbosity=-1, interactive=False, outdir=tmpdir.name)
    logger.debug('file: %s extracted to %s', node.value.filepath, tmpdir)

    await self.create_tree(node, tmpdir.name)
```

Рисунок 22 – Процес розпакування архіву

5. Робота з невідомими MIME-типами:

У випадках, коли MIME-тип не відповідає жодному з попередньо визначених сценаріїв, відповідний вузол видаляється з черги, запобігаючи непотрібній обробці.

Основна робота аналізатора виконується з MIME-типом, що починається з 'text/'. Якщо корисне навантаження містить HTML-вміст, він перетворюється на звичайний текст за допомогою бібліотеки 'html2text'.

Після перетворення у звичайний текст синтаксичний аналізатор переходить до етапу попередньої обробки. Текст розбивається на послідовні блоки, кожен з яких представляє собою запис у форматі «ключ—значення» за допомогою регулярного виразу (рис. 24). Даний крок дозволяє структурувати дані, що полегшує їх аналіз і вилучення потрібної інформації.

```python
async def preprocess(self, genrlines):
    results = []
    block = []
    continious_block = False

    async for line in genrlines:
        line = line.rstrip('\x00\r\n')

        match = re.match(_pattern, line, re.I)
        if not match:
            continious_block = False
            continue

        if not continious_block:
            if block:
                results.append(block)
                block = []

        block.append((match.group('key'), match.group('value')))
        continious_block = True

    if block:
        results.append(block)

    return results
```

Рисунок 23 – Попередня обробка тексту

```python
_pattern = r'^\W*(?P<key>\w[\w_\- ]*?\w)\s*[:=]\s*(?P<value>.*?)\s*$'
```

Рисунок 24 – Регулярний вираз для пошуку рядків у форматі «ключ—значення»

Після того, як текст розбито на блоки, застосовується кілька процесорів для аналізу та виявлення різних аспектів даних. Першим процесором є BlockAnalyzer (рис. 25), який зіставляє пари ключ:значення в кожному блоці за допомогою попередньо визначених шаблонів у вигляді регулярних виразів (рис. 26). Це дозволяє витягувати конкретну інформацію, таку як імена користувачів, паролі або інші релевантні дані.

```
class BlockAnalyzer(BlockProcessor):
    def process(self, blocks):
        taggedblocks = []

        for block in blocks:
            tagged = []
            for fikey, fival in block:
                tags = []
                for key, props in _patterns:
                    if re.match(props['pattern'], fikey, re.I):
                        tags.append(key)

                if not tags:
                    continue

                tagged.append((fikey, fival, tags))
            if not tagged:
                continue
            taggedblocks.append(tagged)
        return taggedblocks
```

Рисунок 25 – Процес аналізу блока

```
('acc:url', {'pattern': r'(host|url|link|hostname|site|site_url|server|сайт)$'}),
('acc:username', {'pattern': r'(username|user name|login|user|usr|e-mail|email|логин)$'}),
('acc:password', {'pattern': r'(password|pass|pswd|pwd|пароль)$'}),
('acc:application', {'pattern': r'(application|app|web browser|browser|browser name|software|soft|found from)$'})

('bot:time', {'pattern': r'(local time|time|date|date and time|log date)$'}),
('bot:timezone', {'pattern': r'(timezone)$'}),
('bot:username', {'pattern': r'(user name|username|user)$'}),
('bot:compname', {'pattern': r"(computer's name|computer name|computer|compname|pc name|"
                             r"pcname|pc|personal computer|computername|machine name)$"}),
('bot:os', {'pattern': r'(osfullname|windows|os version|operation system)$'}),
('bot:cpu', {'pattern': r'(cpu|processor|cpu info)$'}),
('bot:ram', {'pattern': r'ram$'}),
('bot:ip', {'pattern': r'(ip|ip address|computer ip)$'}),
('bot:id', {'pattern': r'(bot_id|machineid|hwid|hardware id)$'}),
('bot:geo', {'pattern': r'(country|country name|location)'})
```

Рисунок 26 – Шаблони для пошуку корисної інформації

Наступним процесором є BlockIdentifier (рис. 27), який визначає тип блоку на основі його вмісту. Він визначає, чи містить блок інформацію про обліковий запис або інформацію, пов'язану зі скомпрометованою системою. Ця класифікація допомагає розрізняти різні типи даних і дозволяє проводити цілеспрямований аналіз.

```
class BlockIndentifier(BlockProcessor):

    def _identify(self, block):
        counter = {}

        for fikey, fival, tags in block:
            for tag in tags:
                typ, nam = tag.split(':', 1)
                counter[typ] = counter.get(typ, 0) + 1

        if counter:
            maxkey = max(counter, key=counter.get)
            isuniq = list(counter.values()).count(counter[maxkey])

            if isuniq:
                return Block(maxkey, block)
        return Block('unknown', block)

    def process(self, blocks):
        return [self._identify(block) for block in blocks]
```

Рисунок 27 – Процес ідентифікації типу блока

Крім того, процесор BlockMerger застосовується, коли є кілька однотипних блоків, зокрема блоків, пов'язаних зі скомпрометованою системою. Процесор об'єднує ці блоки разом, поєднуючи відповідну інформацію і надаючи консолідоване уявлення про жертву.

```
class BlockMerger(BlockProcessor):
    def process(self, blocks):
        mergedbot = []

        for block in list(blocks):
            if block.type == 'bot':
                mergedbot.extend(block.blocktup)
                blocks.remove(block)

        if mergedbot:
            blocks.append(Block('bot', mergedbot))

        return blocks
```

Рисунок 28 – Процес групування пов'язаних блоків

Після обробки та аналізу даних використовується метод обходу дерева для групування блоків облікових записів з відповідними блоками про систему.

Крок гарантує, що відповідна інформація про обліковий запис пов'язана з відповідними даними про скомпрометовану систему, забезпечуючи комплексне уявлення про взаємозв'язки між різними компонентами.

На основі згрупованих блоків формуються записи про всі скомпрометовані облікові дані, які згодом зберігаються в базу даних.

У наведеному нижче графі представлено EML-файл отриманий зі скомпрометованої електронної пошти, яку використовувало ШПЗ для ексфільтрації логів з жертв.

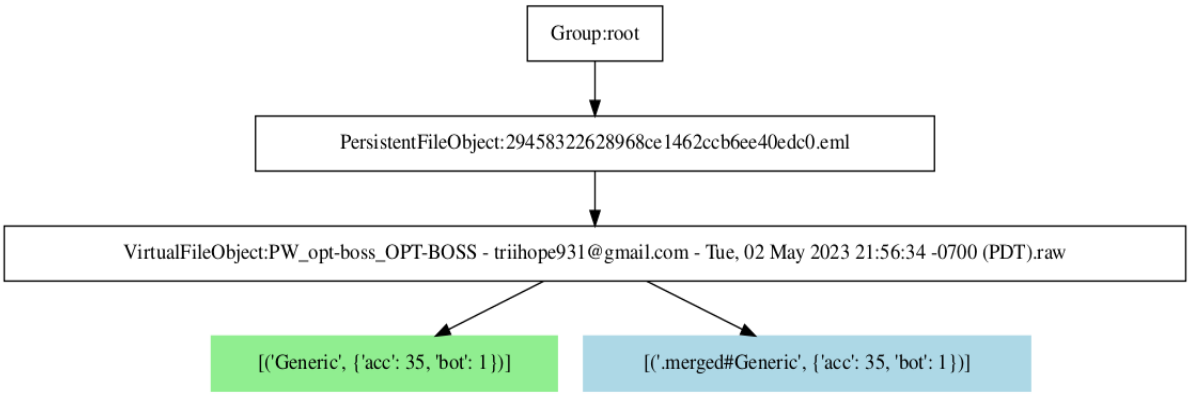

Рисунок 29 – Граф аналізу EML-файла

Оригінальний файл повідомлення(викачаного з електронної пошти за допомогою протоколу IMAP) представлений об’єктом класу PersistentFileObject. Зі змісту повідомлення було створено об’єкт класу VirtualFileObject. Аналізатор знайшов 35 вкрадених облікових даних та інформацію про комп’ютер жертви, які згодом були об’єднані в 35 окремих звітів про скомпрометовані облікові дані.

4 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ

4.1 Використовувані джерела даних

На етапі експериментального тестування за допомогою модуля вилучення конфігурацій ШПЗ, який в якості джерела даних використовує онлайн-пісочницю Hatching Triage, було отримано та проаналізовано 201 унікальну конфігурацію ШПЗ.

Період тестування:

Початок: 2023-05-18T00:00:00

Кінець: 2023-05-25T00:00:00

Нижче наведено розподіл за протоколами, які використовувались під час ексфільтрації різними ШПЗ з отриманих конфігурацій.

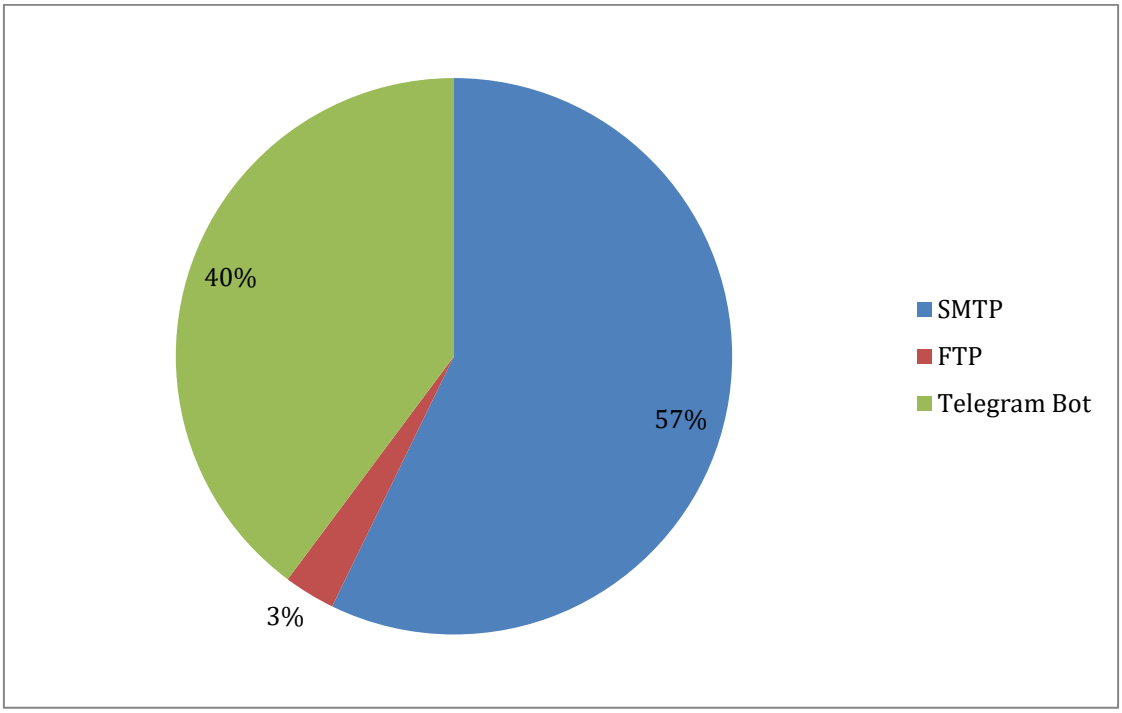

Рисунок 30 – Засоби ексфільтрації

Даний розподіл можна пояснити кількома факторами:

- Протокол SMTP є найпопулярнішим варіантом серед авторів ШПЗ через його широке використання та доступність скомпрометованих облікових даних. ШПЗ може використовувати раніше скомпрометовані облікові дані електронної пошти для надсилання викрадених даних, усуваючи необхідність у створенні власних облікових записів електронної пошти, що спрощує процес ексфільтрації.
- З іншого боку, ексфільтрація через FTP вимагає від авторів ШПЗ розгортання і підтримки власних FTP-серверів, що вимагає більше зусиль і ресурсів у порівнянні з SMTP.
- Створення Telegram-бота зазвичай займає близько 5 хвилин, що робить його зручним вибором для ШПЗ, яким потрібен простий і швидкий метод ексфільтрації. Проте важливо зазначити, що використання Telegram-бота може призвести до розкриття критичної інформації про власника бота.

4.2 Результати обробки даних

Загалом було завантажено 6100 логів стилерів, з яких вилучено 1492 облікових даних. Кількість унікальних облікових даних була відносно невеликою і складалася з 341 унікального запису. Пояснюється це тим, що багато логів містили повторювану інформацію, оскільки ШПЗ часто надсилає логи через регулярні проміжки часу.

Крім того, було помічено, що значна частина викрадених облікових даних складалася з фіктивних даних, згенерованих в онлайн-пісочницях, які імітують реальне середовище і генерують сфабриковані облікові дані для імітації активності користувачів.

Проте, фреймворк успішно виявив чисельні факти компрометації даних, пов'язаних з такими популярними сервісами, як Facebook, Google, Amazon, Discord, Twitter і Reddit.

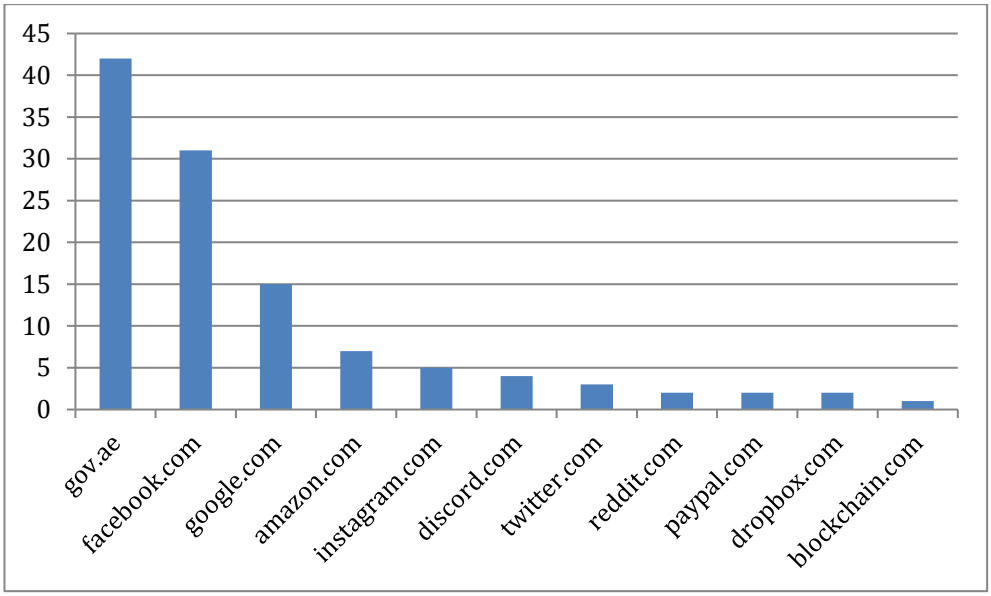

Рисунок 31 – Домени скомпрометованих облікових даних

Більшість скомпрометованих облікових даних належала доменам в Об'єднаних Арабських Еміратах, включаючи домени ".gov.ae", які відповідають урядовим структурам. Це спостереження свідчить про наявність шкідливої кампанії з кібершпіонажу, спеціально спрямованої проти уряду ОАЕ, що не є рідкісним явищем у сфері кібербезпеки.

	id	task_id	data_id	
	Filter	Filter	Filter	gov.ae
1	256	11	129726	{"url": "https://echannels.moi.gov.ae/echannels/web/client/default.html", "username": "[redacted]
2	257	11	129726	{"url": "https://echannels.moi.gov.ae/echannels/web/client/default.html", "username": "[redacted]", "passwo
3	258	11	129726	{"url": "https://echannels.moi.gov.ae/echannels/web/client/default.html", "username": "[redacted]
4	260	11	129726	{"url": "https://eservices.moccae.gov.ae/Account/ForgetPassword.aspx", "username": "[redacted]", "password": "
5	261	11	129726	{"url": "https://eservices.moccae.gov.ae/Account/UserLogin.aspx", "username": "[redacted]", "password": "
6	263	11	129726	{"url": "https://eservices.moccae.gov.ae/account/userlogin.aspx", "username": "[redacted]", "password": "@
7	266	11	129726	{"url": "https://echannels.moi.gov.ae/echannels/web/client/default.html", "username": "[redacted]
8	267	11	129726	{"url": "https://es.adpolice.gov.ae/TrafficServices/Registration/Register.aspx", "username": "[redacted]", "pass
9	268	11	129726	{"url": "https://echannels.moi.gov.ae/echannels/web/client/default.html", "username": "[redacted]
10	269	11	129726	{"url": "https://echannels.moi.gov.ae/echannels/web/client/default.html", "username": "[redacted]
11	270	11	129726	{"url": "https://eservices.tax.gov.ae/ar-ae/login", "username": "[redacted]", "password": "Alshama
12	272	11	129726	{"url": "https://eservices.mohre.gov.ae/enetwasal/arabic/login.aspx", "username": "[redacted]", "password": "S123
13	273	11	129726	{"url": "https://eservices.mohre.gov.ae/enetwasal/arabic/login.aspx", "username": "[redacted]", "password": "
14	274	11	129726	{"url": "https://eservices.tax.gov.ae/ar-ae/login", "username": "[redacted]", "password": "
15	279	11	129726	{"url": "https://id.salik.gov.ae/manage/forgot", "username": "[redacted]", "password": "M6887726m", "unix_timesta
16	283	11	129726	{"url": "https://es.adpolice.gov.ae/TrafficServices/Registration/Login.aspx", "username": "[redacted]
17	286	11	129726	{"url": "https://eservices.tax.gov.ae/ar-ae/login", "username": "[redacted]", "password": "Bailtal
18	287	11	129726	{"url": "https://es.adpolice.gov.ae/TrafficServices/Registration/login.aspx", "username": "[redacted]", "pass

1 - 18 of 51

Рисунок 32 – Приклад скомпрометованих даних, отриманих фреймворком

ВИСНОВКИ

Розроблений фреймворк показав обнадійливі результати у сфері пошуку витоку та крадіжки облікових даних. Завдяки інтеграції різних модулів і універсального аналізатора фреймворк успішно витягував цінну інформацію з необроблених даних, виявляв закономірності у викрадених логах і групував релевантні дані для подальшого аналізу.

Однією з помітних переваг фреймворку є його здатність працювати з широким спектром форматів файлів. Він підтримує вилучення з архівів zip, rar, tgz і 7z, а також аналіз електронних повідомлень і вкладень. Хоча в поточній реалізації не вистачає граберів для FTP і Telegram, їх потенційне включення ще більше посилить можливості фреймворку і розширить сферу його застосування.

Варто зазначити, що ця система не позбавлена обмежень. Втрата інформації про конкретний стилер, відповідальний за крадіжку даних, є помітним недоліком. Крім того, залежність фреймворку від схожості структури логів може бути застосовна не у всіх випадках, що може вплинути на його ефективність проти більш складних і різноманітних варіантів шкідливого програмного забезпечення. Тим не менш, ці обмеження можуть слугувати можливостями для подальшого розвитку та вдосконалення.

Для порівняння, під час експериментального тестування фреймворк показав непогані результати, успішно вилучивши значну кількість логів і облікових даних викрадачів. Однак варто зазначити, що значна частина вилучених облікових даних дублювалася або містила фіктивні дані, згенеровані в онлайн-пісочницях, що свідчить про необхідність подальшого доопрацювання та фільтрації механізмів для підвищення релевантності отриманих даних.

Можливості для подальшого розвитку і застосування системи полягають у посиленні її інтеграції з існуючими платформами розвідки загроз, що дозволить більш безперешкодно обмінюватися інформацією і виявляти загрози

в режимі реального часу.

Загалом, розроблений фреймворк демонструє свій потенціал як цінний інструмент у сфері розвідки загроз та аналізу шкідливого програмного забезпечення. Хоча існують області для вдосконалення і подальшого розвитку, його здатність витягувати, обробляти і аналізувати вкрадені дані позиціонує його як цінний актив у боротьбі з кіберзагрозами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Що таке програма-стилер, якої шкоди вона може завдати комп’ютеру? [Електронний ресурс] – Режим доступу до ресурсу : https://www.cip.gov.ua/ua/faqs/sho-take-programa-stiler-yakoyi-shkodi-vona-mozhe-zavdati-komp-yuteru
2. Data Exfiltration [Електронний ресурс] – Режим доступу до ресурсу : https://www.fortinet.com/resources/cyberglossary/data-exfiltration
3. Telegram Bot API [Електронний ресурс] – Режим доступу до ресурсу : https://core.telegram.org/bots/api
4. VirusTotal [Електронний ресурс] – Режим доступу до ресурсу: https://www.virustotal.com/
5. Hatching Triage [Електронний ресурс] – Режим доступу до ресурсу: https://tria.ge/
6. Cuckoo Sandbox [Електронний ресурс] – Режим доступу до ресурсу: https://cuckoosandbox.org/
7. CAPE Sandbox [Електронний ресурс] – Режим доступу до ресурсу : https://capesandbox.com/analysis/
8. SQLite [Електронний ресурс] – Режим доступу до ресурсу: https://www.sqlite.org/index.html
9. Hatching Triage API Github [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/hatching/triage
10. Hatching Triage Search [Електронний ресурс] – Режим доступу до ресурсу: https://tria.ge/s
11. Hatching Triage Sample [Електронний ресурс] – Режим доступу до ресурсу: https://tria.ge/230526-pe5pbafd66
12. JSON Schema Draft-07 [Електронний ресурс] – Режим доступу до ресурсу: http://json-schema.org/draft-07/schema

13. IETF RFC 2646: The Text/Plain Format Parameter [Електронний ресурс] – Режим доступу до ресурсу: https://www.ietf.org/rfc/rfc2646.txt
14. IETF RFC 2854: The 'text/html' Media Type [Електронний ресурс] – Режим доступу до ресурсу: https://www.ietf.org/rfc/rfc2854.txt
15. IETF RFC 2822: Internet Message Format [Електронний ресурс] – Режим доступу до ресурсу: https://www.ietf.org/rfc/rfc0822.txt
16. ISO/IEC 21320-1:2015 - Document Container File [Електронний ресурс] – Режим доступу до ресурсу: https://www.iso.org/standard/60101.html