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: «Програмні механізми ідентифікації атак нульового дня ransomware для ОС Windows»

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

Тислицький Данііл Володимирович _____
(прізвище, ім'я, по батькові) (підпис)

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

Рецензент к.т.н., доцент, Лавренюк Алла Миколаївна _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

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

Тислицькому Даніілу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Програмні механізми ідентифікації атак нульового дня ransomware для ОС Windows»,
керівник роботи Гальчинський Леонід Юрійович, к.т.н., доцент,
(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 ___ червня 2023 р.

3. Вихідні дані до роботи: Програмний механізм ідентифікації I/O Completion Port, реалізований мовою Python.

4. Зміст роботи:

Вступ
Основні поняття та теоретичні відомості
Аналіз наявних зразків шпз, що використовують I/O Completion Ports на прикладі Sodinokibi / REvil
Розробка програмного механізму ідентифікації шкідливого програмного забезпечення, що використовує I/O Completion Port
Висновки

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

6. Дата видачі завдання 1 листопада 2022

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Отримання завдання на дипломну роботу та визначення тематики	01.11.2022	виконано
2	Формулювання теми дипломної роботи, визначення мети.	15.12.2022	виконано
3	Узгодження теми з дипломним керівником	17.12.2022	виконано
4	Постановка задач необхідних для реалізації дипломної роботи	20.12.2022	виконано
5	Робота над першим розділом: збір та обробка теоретичного матеріалу для досліджень	18.01.2023-05.03.2023	виконано
6	Робота над другим розділом: огляд та аналіз Ransomware Sodinokibi/REvil	05.03.2023-16.04.2023	виконано
7	Робота над другим розділом: реалізація перебору файлової системи за допомогою I/O Completion Port та APC засобами мови C++	17.04.2023-23.04.2023	виконано
8	Робота над другим розділом: порівняльна характеристика I/O Completion Port з альтернативними механізмами, тестування наявних методів детектування ШПЗ	24.04.2023-07.05.2023	виконано
9	Робота над третім розділом: реалізація програмного механізму ідентифікації роботи I/O Completion Port	08.05.2023-21.05.2023	виконано
10	Робота над третім розділом: тестування розробленого програмного механізму на різних виконуваних файлах	22.05.2023-28.05.2023	виконано
11	Оформлення дипломної роботи згідно методичних вказівок	29.05.2023-04.06.2023	виконано
12	Оформлення ілюстративних матеріалів	05.06.2023-11.06.2023	виконано
13	Передзахист дипломної роботи	12.06.2023	
14	Захист дипломної роботи		

Здобувач вищої освіти _____ Даніїл ТИСЛИЦЬКИЙ
(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

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

РЕФЕРАТ

Робота складається з 3 розділів, що складають 69 сторінки (без додатків) та містять 43 ілюстрації, 2 таблиці та 22 бібліографічні найменування.

Метою роботи є розробка програмного механізму ідентифікації ШПЗ, що використовують поточну модель асинхронних запитів вводу-виводу за допомогою I/O Completion Ports.

Методи дослідження: аналіз (аналіз наявних теоретичних джерел та механізмів детектування ШПЗ), порівняння (порівняльна характеристика I/O Completion Port з альтернативним механізмом), вимірювання (вимірювання часової характеристики, що використовується при порівнянні), абстрагування (реалізація елементу ШПЗ, що використовує I/O Completion Port без корисного навантаження), метод емпіричного рівня (виявлення переваг використання I/O Completion Port в порівнянні з альтернативами), гіпотетичний метод (висунення гіпотези про використання I/O Completion Port як індикатора ransomware), розробка програмного рішення (розробка програмного механізму ідентифікації використання I/O Completion Port).

В результаті роботи було отримано програмний сканер, що може з досить високою точністю ідентифікувати використання I/O Completion Port та детектувати його підозрілу поведінку.

Результати було представлено на XXI Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики (11-12 травня 2023 р., м. Київ, Україна) та Всеукраїнській науково-практичній конференції “Theoretical and Applied Cybersecurity (TACS-2023)” (26 травня 2023 р., м. Київ, Україна).

Ключові слова: асинхронний ввід/вивід, атака нульового дня, RaaS, I/O Completion Port.

ABSTRACT

The work consists of 3 chapters, comprising 69 pages (excluding appendices) and containing 43 illustrations, 2 tables and 22 references.

The purpose of the work is to develop a software mechanism for identifying the NICs that use the current model of asynchronous I/O requests using I/O Completion Ports.

Research methods: analysis (analysis of existing theoretical sources and mechanisms for detecting malware), comparison (comparative characteristic of I/O Completion Port with an alternative mechanism), measurement (measurement of the time characteristic used in the comparison), abstraction (implementation of an malware element that uses I/O Completion Port without payload), empirical level method (identifying the advantages of using I/O Completion Port in comparison with alternatives), hypothetical method (hypothesising the use of I/O Completion Port as a ransomware indicator), software solution development (developing a software mechanism for identifying the use of I/O Completion Port).

As a result of the work, a software scanner was developed that can identify the use of I/O Completion Port and detect its suspicious behaviour with a fairly high accuracy.

The results were presented at the XXI All-Ukrainian Scientific and Practical Conference of Students, Postgraduates and Young Scientists "Theoretical and Applied Problems of Physics, Mathematics and Informatics" (11-12 May 2023, Kyiv, Ukraine) and the All-Ukrainian Scientific and Practical Conference "Theoretical and Applied Cybersecurity (TACS-2023)" (26 May 2023, Kyiv, Ukraine).

Keywords: asynchronous I/O, zero-day attack, RaaS, I/O Completion Port.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційна система

ШПЗ – шкідливе програмне забезпечення

I/O – input/output

API – Application Programming Interface

AIO – Asynchronous input/output

APC – Asynchronous procedure calls

MSMQ – Microsoft Message Queuing

RaaS – Ransomware-as-a-Service

VSS – Volume Shadow Copy Service

URL – Uniform Resource Locator

ВСТУП

Станом на 2022 рік безумовним лідером серед операційних систем, що використовуються користувачами з усього світу для десктопних рішень є Windows. Так станом на січень 2023 року, у відповідності до статистики представленої організацією Statcounter, по всьому світу 74,14 % користувачів використовують Windows різних версій. Що ж стосується України, то дана ОС використовується 86,6 % користувачів [1]. З цього випливає, що розробка ШПЗ для даного сімейства ОС є найпривабливішою задачею для сучасних правопорушників.

Актуальність роботи полягає в тому, що сучасні види ШПЗ, зокрема ransomware, використовують надійні механізми захистів свого продукту для обходу антивірусних програмних засобів. Через це розробка механізмів захисту, що покладається на підозрілу поведінку, викликану використанням вбудованих API функцій ОС, є необхідною для зменшення ризику використання атак подібного типу.

Мета дослідження: розробка програмного механізму ідентифікації ШПЗ, що використовують поточну модель асинхронних запитів вводу-виводу за допомогою I/O Completion Ports.

Завдання дослідження:

1) Проаналізувати наявні зразки ШПЗ, що використовують I/O Completion Ports, в якості механізму виконання корисного навантаження.
2) Перевірити наявні рішення виявлення згаданих програм ШПЗ.
3) Проаналізувати принцип використання і роботи Windows API функцій, що використовують I/O Completion Ports на рівні ОС.
4) Розробити програмне рішення, що буде виявляти ШПЗ подібного типу та діагностувати підозріле застосування заданого механізму ОС Windows.

Об’єкт дослідження: ШПЗ, що використовує I/O Completion Ports, в якості механізму пришвидшення обробки об’єктів, наявних у файловій системі.

Предмет дослідження: методи і засоби ідентифікації ШПЗ.

Методи дослідження: аналіз (аналіз наявних теоретичних джерел та механізмів детектування ШПЗ), порівняння (порівняльна характеристика I/O Completion Port з альтернативним механізмом), вимірювання (вимірювання часової характеристики, що використовується при порівнянні), абстрагування (реалізація елементу ШПЗ, що використовує I/O Completion Port без корисного навантаження), метод емпіричного рівня (виявлення переваг використання I/O Completion Port в порівнянні з альтернативами), гіпотетичний метод (висунення гіпотези про використання I/O Completion Port як індикатора ransomware), розробка програмного рішення (розробка програмного механізму ідентифікації використання I/O Completion Port).

Апробація роботи: за заданою тематикою було розроблено 2 роботи за темами: «Використання механізмів асинхронного вводу/виводу при реалізації атак нульового дня ОС Windows» та «Ідентифікація I/O Completion Port як механізм підвищення захисту від атак нульового дня ОС Windows». Їх було представлено на XXI Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики (11-12 травня 2023 р., м. Київ, Україна) та Всеукраїнській науково-практичній конференції “Theoretical and Applied Cybersecurity (TACS-2023)” (26 травня 2023 р., м. Київ, Україна) відповідно, що й послугувало апробацією даної роботи.

1 ОСНОВНІ ПОНЯТТЯ ТА ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Об’єкти ядра ОС Windows

При розробці програмного забезпечення для ОС Windows розробники стикаються з роботою з об’єктами ядра Windows. Система створює та маніпулює декількома типами об’єктів ядра, такими як об’єкти маркерів доступу, об’єкти подій, об’єкти файлів, об’єкти відображення файлів, об’єкти I/O Completion port, об’єкти завдань, об’єкти поштового слота, об’єкти м’ютексу, об’єкти каналу, об’єкти процесу, об’єкти семафора, об’єкти потоку, об’єкти таймера очікування та об’єкти розділів реєстру [2]. Для того, щоб подивитися наявні об’єкти ядра, що використовуються в поточній системі можна скористатися утилітою WinObj [3], що розповсюджується компанією Microsoft для загального користування.

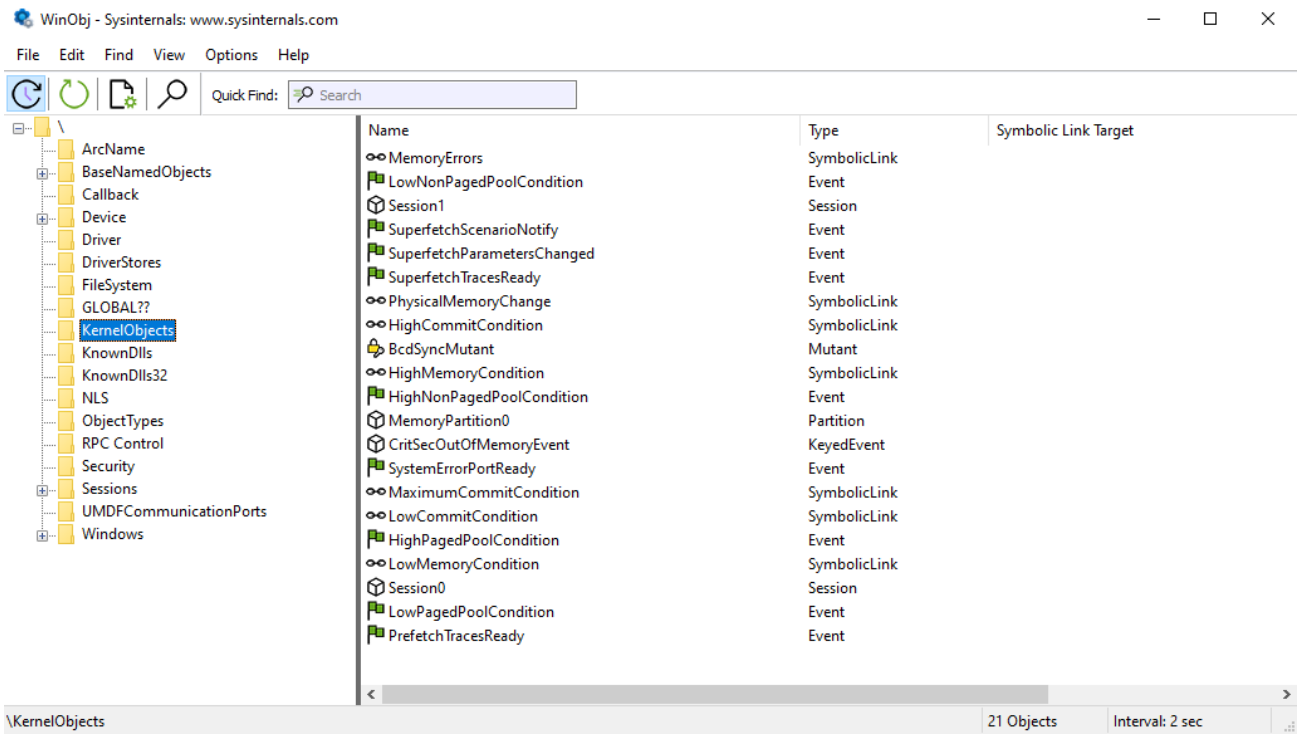


Рисунок 1.1 - Об'єкти ядра ОС Windows, отримані за допомогою утиліти WinObj

Ці об’єкти створюються шляхом виклику різних API функцій з іменами, які не обов’язково відповідають типу об’єктів, що використовуються на рівні ядра. Власне кажучи кожен об’єкт ядра — це просто блок пам’яті, виділений ядром і доступний лише для ядра, що представляє собою структуру даних, члени якої зберігають інформацію про об'єкт. Деякі члени (дескриптор безпеки, підрахунок використання тощо) однакові для всіх типів об’єктів, але більшість з них є специфічними для конкретного типу об’єкта.

Оскільки структури даних об’єктів ядра доступні лише для ядра, програма не може їх знайти у пам’яті та безпосередньо змінити їхній вміст. Корпорація Microsoft навмисно вводить це обмеження, щоб гарантувати, що структури об’єктів ядра підтримують узгоджений стан. Це обмеження також дозволяє Microsoft додавати, видаляти або змінювати членів у цих структурах, не порушуючи жодних програм.

Дескриптор (HANDLE) — це 32-розрядне значення в Windows x86 і 64-розрядне значення в x86-64. Саме ці значення й використовуються при виклику API функцій, для того, щоб ОС розуміла маніпуляції з яким об’єктом ядра є необхідними.

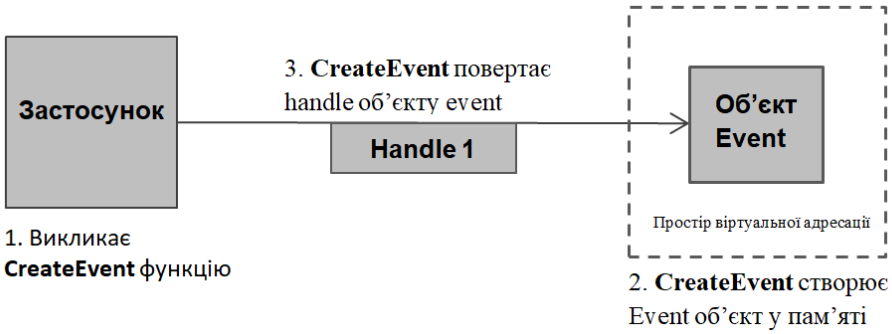


Рисунок 1.2 – Звернення до об’єкту ядра через Windows API функцію

Дескриптори об'єктів ядра залежать від процесу. Тобто процес повинен створити об'єкт, або відкрити існуючий об'єкт, щоб отримати дескриптор об'єкта ядра. Обмеження на кількість дескрипторів ядра для кожного процесу становить 2 ^ 24. Однак дескриптори зберігаються у пулі, що вивантажується, тому фактична

кількість дескрипторів, які ви можете створити, залежить від доступної пам'яті [4].

Контекст безпеки доступу до об’єктів ядра забезпечується за допомогою дескрипторів безпеки. Дескриптор безпеки описує, хто є власником об’єкта, яка група та користувачі можуть отримати доступ до об’єкта або використовувати його, а також якій групі та користувачам заборонено доступ до об’єкта.

Усі Windows API функції, що звертаються до об’єктів ядра, містять вказівник на структуру SECURITY_ATTRIBUTES. Хоча її й можна задати явно, але все ж таки на практиці в якості параметру вказівника на цю структуру частіш за все вказується Null, що означає надати контекст безпеки за замовчуванням.

1.2 Механізм асинхронного вводу/виводу (AIO)

Асинхронне введення/виведення (AIO) — це тип обробки вводу/виводу, який дозволяє виконувати кілька операцій введення/виводу одночасно, не блокуючи їх операційною системою. Це означає, що система може продовжувати обробку інших завдань, чекаючи завершення операцій введення/виведення. У традиційній обробці введення/виведення процес блокується, доки не буде завершена операція введення/виведення, що може призвести до зниження продуктивності, особливо при роботі з великими обсягами даних.

В AIO операції введення/виведення ініціюються асинхронно, і процес отримує сповіщення, коли операція завершена. Це дозволяє підвищити продуктивність, оскільки процес може продовжувати обробляти інші завдання, очікуючи завершення операцій введення/виведення.

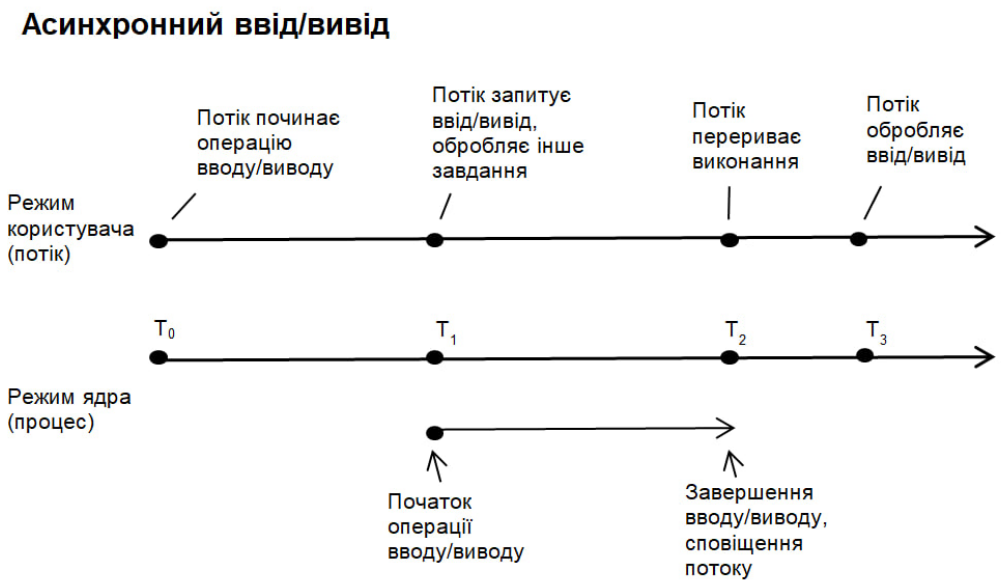


Рисунок 1.3 – Часова модель AIO

AIO можна реалізувати за допомогою різних методів, таких як асинхронні виклики процедур (APC), підпрограми завершення та порти завершення вводу/виводу. Він зазвичай використовується у високопродуктивних мережевих програмах, де потрібно швидко й ефективно передати великий обсяг даних.

AIO підтримується в різних операційних системах, включаючи Windows і Linux, і доступний у різних мовах програмування. Він вимагає ретельного програмування, оскільки під час використання AIO легко ввести помилки через його асинхронний характер. Розглянемо базову реалізацію асинхронного вводу/виводу для об’єктів ОС Windows з використанням Windows API.

При відкритті об’єкту файлу за допомогою функції CreateFile слід вказати атрибут FILE_FLAG_OVERLAPPED в якості параметру dwFlagsAndAttributes. При цьому вказівник на структуру OVERLAPPED буде передаватися для викликів у функції ReadFile та WriteFile. В іншому випадку файл відкривається за замовчуванням в синхронному режимі.

Структура OVERLAPPED містить інформацію необхідну для реалізації AIO та виглядає наступним чином [5]:

```
typedef struct _OVERLAPPED {
```

```
  ULONG_PTR Internal;
  ULONG_PTR InternalHigh;
  union {
    struct {
      DWORD Offset;
      DWORD OffsetHigh;
    } DUMMYSTRUCTNAME;
    PVOID Pointer;
  } DUMMYUNIONNAME;
  HANDLE    hEvent;
} OVERLAPPED, *LPOVERLAPPED;
```

Після відкриття файлового об'єкта для асинхронного введення-виводу структура OVERLAPPED повинна бути правильно створена, ініціалізована та передана в кожен виклик таких функцій, як ReadFile та WriteFile. При цьому при роботі зі структурою OVERLAPPED слід дотриматися наступних вимог, наведених в документації:

- Не слід очищати та змінювати структуру OVERLAPPED або буфер даних, доки не будуть завершені всі асинхронні операції вводу-виводу з файловим об'єктом.
- Якщо вказівник на структуру оголошується як локальна змінна, то забороняється виходити з локальної функції до тих пір, поки не будуть завершені всі асинхронні операції вводу/виводу.
- Якщо локальна функція завершується передчасно, структура OVERLAPPED виходить за межі зони видимості і стає недоступною для будь-яких функцій ReadFile або WriteFile , з якими вона стикається за межами цієї функції [6].

1.3 I/O Completion Port

I/O Completion Ports (порти завершення вводу-виводу) – це високопродуктивний механізм для керування асинхронними операціями введення-виведення в ОС Windows. Вони вперше були представлені в Windows NT і з тих пір стали важливою частиною Windows API. Основне призначення портів завершення вводу-виводу — забезпечити масштабований механізм для керування операціями вводу-виводу, особливо в багатопоточних середовищах, де багато операцій вводу-виводу потрібно обробляти одночасно.

Порт завершення вводу-виводу — це, по суті, черга повідомлень про завершення вводу-виводу, які генеруються після завершення операції введення-виведення. Операційна система використовує порт завершення вводу-виводу для сповіщення програми про завершення операції введення-виведення. Це дозволяє програмі продовжувати виконувати інший код, поки операція вводу/виводу очікує на виконання, що значно покращує загальну продуктивність і масштабованість програми.

Щоб використовувати порти завершення вводу-виводу, програма створює порт завершення вводу-виводу та пов’язує з ним один або кілька дескрипторів вводу-виводу. Це значення вказує на максимальну кількість потоків, пов’язаних із портом, які мають працювати в певний момент часу. Ідеальним є таке значення, коли в системі наявний лише один активний потік для кожного ядра процесора в системі, але це значення може відрізнятися.

ОС Windows використовує отримане значення для реалізації механізму контролю, пов’язаних з портом вводу-виводу. Якщо кількість активних потоків, пов’язаних із портом, дорівнює значенню паралельності потоку, який очікує на порт завершення, то йому не буде дозволено працювати. Замість цього активний

потік завершить обробку свого поточного запиту, після чого він перевірить, чи інший пакет очікує на порту. Якщо так, потік просто захоплює пакет і йде його обробляти. Коли це відбувається, перемикання контексту не відбувається, і ядра процесору використовуються майже на повну потужність [7].

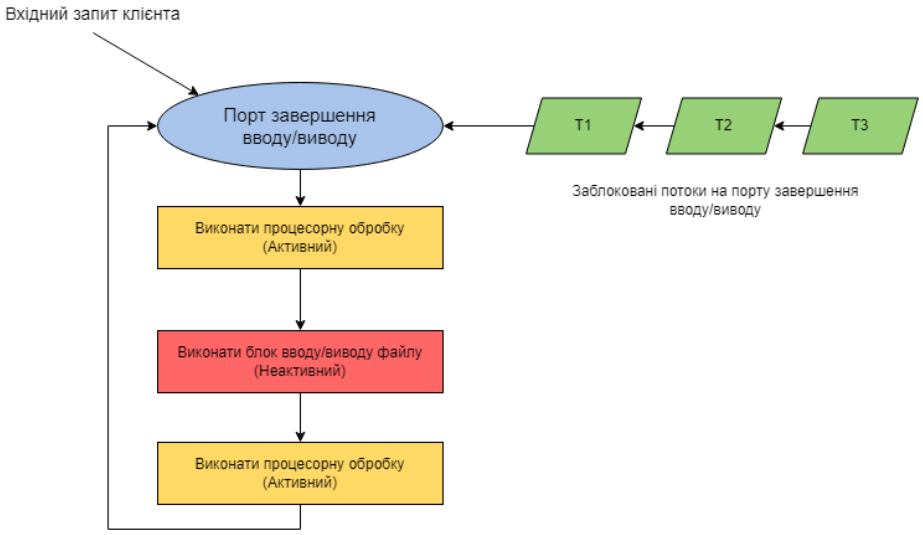


Рисунок 1.4 – Блок-схема застосування I/O Completion port

Для створення та асоціації порту вводу-виводу та відкритих дескрипторів файлів використовується API функція CreateIoCompletionPort. Тут слід зазначити, що дескриптори відкритих файлів у даному випадку виступають своєрідною абстракцією, що позначає будь-які системні об’єкти, що можуть підтримувати асинхронних ввід-вивід.

```
HANDLE WINAPI CreateIoCompletionPort(
  _In_     HANDLE    FileHandle,
  _In_opt_ HANDLE    ExistingCompletionPort,
  _In_     ULONG_PTR CompletionKey,
  _In_     DWORD     NumberOfConcurrentThreads
);
```

Щоб виконувати введення-виведення на дескриптори пов’язані з портом, використовуються API функції ReadFile та WriteFile. Таким чином, операція введення-виведення ставиться в чергу на порт завершення.

Потік очікує асинхронного завершення в черзі не шляхом очікування події, а шляхом виклику GetQueuedCompletionStatus із зазначенням порту завершення. Після завершення функція повертає ключ, який було вказано, коли дескриптор (той, чия операція завершена) спочатку був доданий до порту за допомогою CreateIOCompletionPort.

Цей ключ може вказувати ідентифікатор фактичного дескриптора завершеної операції та іншу інформацію, пов’язану з операцією вводу-виводу. Зауважте, що потік Windows, який ініціював читання або запис, не обов’язково буде потоком, який отримає сповіщення про завершення; будь-який потік, що очікує, може отримати сповіщення про завершення. Таким чином, приймаючий потік може ідентифікувати дескриптор завершеної операції з ключа завершення.

При цьому додається наступне зауваження: не слід використовувати блокування за допомогою таких механізмів як mutex, CRITICAL_SECTION тощо, коли викликається GetQueuedCompletionStatus, оскільки потік, який знімає блокування, ймовірно, не є тим потоком, який його отримав [8].

1.4 Альтернативні механізми AIO

Існує кілька альтернативних механізмів крім I/O Completion Ports в Windows для обробки AIO:

Overlapped I/O, що дозволяє програмам надсилати кілька асинхронних запитів введення-виведення до драйверу пристрою та отримувати результати пізніше, коли введення-виведення завершиться. Він схожий на I/O Completion Ports, але не забезпечує переваг масштабованості.

Пул потоків (thread pool) — це набір потоків, створених і керованих операційною системою. Програми можуть надсилати робочі елементи до пулу потоків, і система призначатиме їх потоку для виконання. Пули потоків можна використовувати для обробки асинхронних операцій введення-виведення, але вони не такі ефективні, як I/O Completion Ports.

Асинхронні виклики процедур (APC) — це механізм для виконання функції в контексті потоку. Програми можуть використовувати APC для реалізації асинхронного вводу-виводу, ставлячи в чергу функцію для виконання після завершення операції введення-виведення.

Асинхронний обмін повідомленнями забезпечує такий механізм, при якому програми можуть використовувати методи асинхронного обміну повідомленнями для обміну даними між процесами або потоками. Одним із популярних асинхронних механізмів обміну повідомленнями є черга повідомлень Windows (MSMQ), яка забезпечує надійну інфраструктуру обміну повідомленнями для програм Windows.

Кожен із цих механізмів має свої сильні та слабкі сторони, і вибір того, який механізм використовувати, залежить від конкретних вимог програми. Проте порти завершення введення-виведення зазвичай вважаються найбільш ефективним і масштабованим механізмом для обробки асинхронних операцій вводу-виводу в Windows.

1.5 RaaS (Ransomware As A Service)

В сучасному світі програми-вимагачі, або ransomware є однією з найпоширеніших тактик монетизації атаки. Навіть враховуючи шалений прогрес у методах захисту, що використовуються в різних компаніях для зменшення ймовірності бути атакованими, атаки, що керуються людиною є найбільш комплексними і досконалими. Раніше для реалізації такого вектору необхідно було володіти певними навичками і технічним підґрунтям для реалізації атаки, але з розвитком комерційної діяльності, зокрема використання RaaS, ця уявна лінія, що відділяла досконалих хакерів потроху почала зникати.

Програма-вимагач як послуга (RaaS) є бізнес-моделлю, що дозволяє злочинним підприємствам надавати інструменти для атак програм-вимагачів через онлайн-сервіс. Схоже на інші моделі "як послуга", такі як "програмне забезпечення як послуга" (SaaS) або "платформа як послуга" (PaaS), клієнти RaaS орендують послуги програм-вимагачів замість того, щоб володіти ними, як у традиційній моделі розповсюдження програмного забезпечення [9].

Робота заданого механізму полягає в декількох типах оплати за реалізовану послугу: щомісячний платіж, процент від прибутку або єдиноразова купівля своєрідної ліцензії на використання програмного продукту. Зазвичай платежі проходять з використанням криптовалют, наприклад, Bitcoin. Після успішної атаки вже можна переходити до реалізації атаки, яка зазвичай виконується за допомогою інших типів атак, що доставляють корисне навантаження на цільову систему.

Як відомо, розповсюдження таких продуктів відбувається через darknet. Також постачальники RaaS мають цілодобову підтримку, що дозволяє вирішувати питання пов’язані з проблемами запуску наданого ШПЗ. Крім того

постачальник може мати певний форум, де клієнти можуть обмінюватися власними ідеями, щодо використання продукту.

Наразі багато постачальників послуг RaaS стали більш детально підходити до можливостей своїх клієнтів. Так, наприклад, ще деякий час тому достатньо було лише мати гроші, щоб мати змогу оплатити ШПЗ, що призводило до неабиякого ризику бути виявленими у випадку невмілого користування програмою. Отже, тепер покупцю необхідно мати певні, хоча б базові навички, у галузі комп'ютерних технологій, щоб заволодіти необхідним зразком. Також, слід зазначити, що існує своєрідний моральний кодекс, де прописуються відповідні галузі, які забороняється атакувати за допомогою ШПЗ придбаного за RaaS. До таких галузей відносяться ті, що безпосередньо впливають на життя та здоров'я людей, наприклад, критична інфраструктура держав або медичні установи.

В результаті можемо отримати наступну партнерську модель RaaS при забезпеченні атаки за допомогою програм-вимагачів:

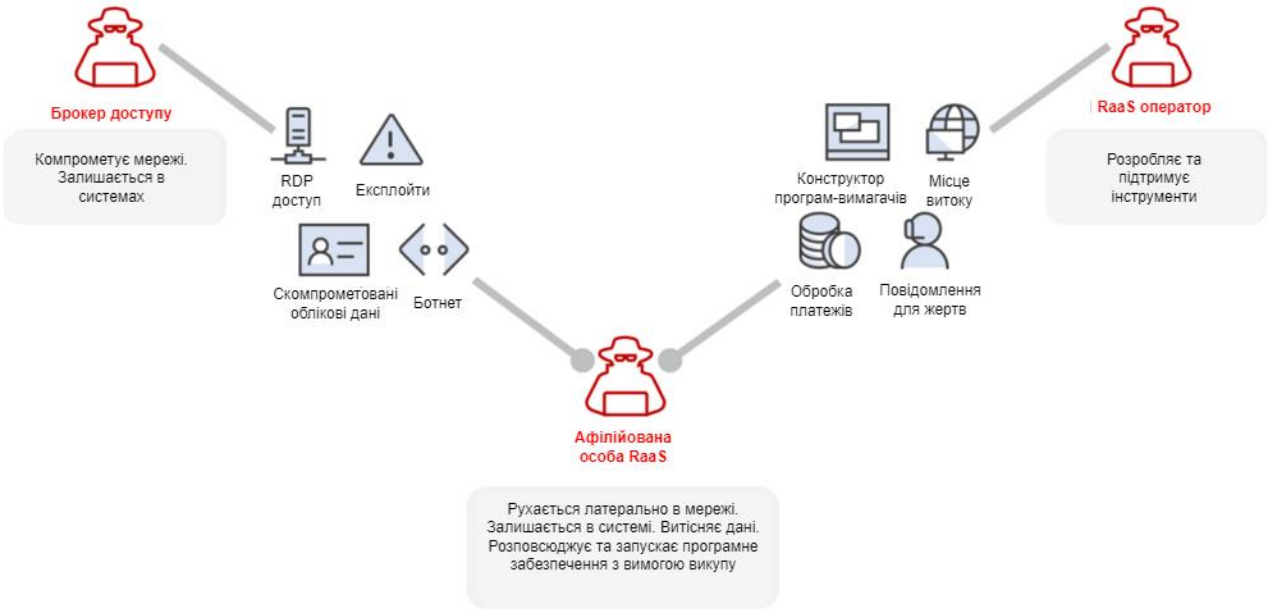


Рисунок 1.5 – Забезпечення атаки програмами-вимагачами за допомогою RaaS [10]

Атаки, що використовують RaaS все більш набирають обертів в останні роки. Ось декілька відомих прикладів:

- GandCrab: ця атака була однією з перших, які використовували RaaS модель. GandCrab був доступний для оренди на підставі місячної підписки та відсотку від викупу. Він був відповідальний за нанесення значної шкоди більш ніж 1,5 мільярда доларів США.
- REvil: це найбільш відома атака RaaS, яка отримала популярність у 2020 році. REvil була доступна на базі партнерської програми, яка дозволяла розповсюджувати атаки в обмін на комісійні.
- DarkSide - це група програм-вимагачів, яка продає RaaS. Атаку Colonial Pipeline у 2021 році було приписано саме DarkSide.

Висновки до розділу 1

У даному розділі було розглянуто поняття об’єктів ядра ОС Windows, а також основні теоретичні засади механізму асинхронного вводу/виводу. При цьому основна увага була зконцентрована на портах завершення вводу/виводу (I/O Completion Ports), їх принципу роботи та реалізації. Для подальшої порівняльної характеристики було визначено альтернативні механізми, що можуть використовуватися для розв’язання аналогічних задач. Також проаналізовано принцип роботи програм-вимагачів, що розповсюджуються за принципом RaaS.

2 АНАЛІЗ НАЯВНИХ ЗРАЗКІВ ШПЗ, ЩО ВИКОРИСТОВУЮТЬ I/O COMPLETION PORTS НА ПРИКЛАДІ SODINOKIBI / REVIL

2.1 Загальні відомості та історія Sodinokibi / REvil

Sodinokibi, також відома як REvil, — це група програм-вимагачів, що вперше з'явилася в квітні 2019 року. Група націлена в основному на великі корпорації та державні організації та відома своїм використанням складних методів злому, зокрема експлойтів нульового дня, для отримання несанкціонованого доступу до цільової системи. Також дане ШПЗ розповсюджується в якості RaaS.

Після того, як вимагач отримає доступ до цільової системи, він зашифрує всі дані жертви та вимагатиме викуп в обмін на ключ розшифровки. Група відома тим, що використовує низку тактик для посилення тиску на жертв, щоб вони заплатили, включно з погрозами оприлюднити конфіденційні дані, якщо викуп не буде сплачено.

Sodinokibi швидко здобув популярність завдяки своїм складним методам, таким як здатність уникати виявлення та обходити антивірусне програмне забезпечення. Він також був примітний використанням афілійованих мереж, де окремі афілійовані особи могли заробити частину викупу за кожне успішне зараження.

Групу, що стоїть за Sodinokibi, пов'язують із кількома резонансними атаками, включно з нападом на 42 мільйони доларів проти юридичної фірми Grubman Shire Meiselas & Sacks, яка представляє деякі з найбільших імен у сфері розваг.

Незважаючи на репутацію групи, хакери, що стоять за Sodinokibi, зіткнулися з серйозними проблемами в останні роки, включаючи арешт ряду осіб, пов’язаних із шкідливим програмним забезпеченням, і збої в їхній інфраструктурі. Незважаючи на це, група продовжує становити серйозну загрозу для організацій і окремих осіб, тому людям важливо залишатися пильними та вживати заходів для захисту своїх систем і даних.

2.2 Жертви, що були атаковані за допомогою Sodinokibi / REvil та рішення прийнятті для протидії

Як було зазначено раніше, Sodinokibi / REvil почала свою активність ще у 2019 році і за цей час встигла стати однією із найпопулярніших програм-вимагачів. Відомі факти багатьох компаній-жертв, що отримали велику шкоду від зараження цим ШПЗ. Серед них можна виокремити такі як: JBS USA, Acer, Kaseya, Travelex та ін. Розглянемо декілька з цих інцидентів більш детальніше.

Наприклад, Acer є одним з найбільших виробників комп’ютерної техніки у світі. У березні 2021 року компанія повідомила про атаку Sodinokibi / REvil, яка призвела до вимоги викупу в розмірі 50 мільйонів доларів США. При цьому в якості доказів зловмисники опублікували фрагменти, що містили у собі фінансову звітність, банківські повідомлення та операції та інші файли. При цьому Acer почала процедуру реагування на інцидент, повідомивши про це компетентні органи та ініціювавши внутрішнє розслідування. Тут слід зазначити, що при атаці на корпорацію таких масштабів розшифрування файлів не дає змоги повернутися до звичайного режиму роботи, оскільки побічні наслідки вимагають теж часу, що призводить до погіршення якості надаваних послуг [11].

Іншим прикладом можна вважати атаку на одну з найбільших компаній світу з виробництва м’яса JBS USA. Атака програмою-вимагачем сталася рано вранці 31 травня 2021 року, внаслідок чого JBS відключила свою мережу, щоб запобігти поширенню атаки. Внаслідок цих дій декілька підприємств у США, Канаді, Австралії припинили свою діяльність терміном близьким до 24 годин, оскільки були відрізані від мережі. IT-фахівці одразу заявили, що всі резервні копії та файли не постраждали, однак інші джерела вказували на те, що було зашифровано декілька сетів даних, що призвело до довгого відновлення доступу до мережі [12].

Оскільки атаки набрали великих обертів, то для розслідування і пошук кримінального угрупування було задіяно і державні інституції. Зокрема були створені відповідні комісії з розслідувань у країнах ЄС та США.

У травні 2021 року Франція, Німеччина, Румунія, Європол та Євроюст підсилили свої заходи для боротьби з програмою-вимагачем, створивши спільну слідчу групу. Крім того, Bitdefender, співпрацюючи з правоохоронними органами, розмістив на веб-сайті No More Ransom інструмент, який може допомогти жертвам Sodinokibi/REvil відновити свої файли та відновити свої комп'ютерні системи після атак, що сталися до липня 2021 року. В жовтні того ж року на польському кордоні було заарештовано громадянина України, якого підозрюють у скоєнні атаки на Kaseya, за яку Sodinokibi/REvil вимагала викуп близько 70 мільйонів євро і яка торкнулася близько 1500 підприємств. Також у 2021 році влада Південної Кореї заарештувала трьох осіб, причетних до програм-вимагачів GandCrab і Sodinokibi/REvil, а у листопаді того ж року влада Кувейту заарештувала ще одну філію GandGrab, загалом заарештовано сім підозрюваних, пов'язаних з двома сімействами програм-вимагачів, яких підозрюють у нападі на близько 7 000 жертв [13].

Натомість, державний департамент США пропонує винагороду в розмірі до 10.000.000 доларів США за інформацію, що дозволяє ідентифікувати або

визначити місцезнаходження будь-якої особи (осіб), які займають ключову керівну посаду в транснаціональній організованій злочинній групі, яка є варіантом програми-вимагача Sodinokibi. Крім того, пропонується винагорода в розмірі до 5.000.000 доларів США за інформацію, що веде до арешту та/або засудження в будь-якій країні будь-якої особи, яка змовилася або намагається брати участь в інциденті з програмою-вимагачем Sodinokibi [14].

Рисунок 2.1 – Плакат FBI про розшук зловмисників, що причетні до REvil

2.3 Принцип роботи Sodinokibi / REvil

Sodinokibi розповсюджується різними способами, такими як: фішингові атаки[1], спам і завантаження у вже скомпрометовану систему. Програма-вимагач встановлюється безпосередньо хакером, який отримав доступ до незахищеного порту RDP, використав фішинг електронної пошти для віддаленого підключення до мережі через комп'ютер співробітника або використав шкідливі вкладення, завантаження, експлойти програмних виправлень або вразливості, щоб отримати доступ до мережі.

Розглянемо приклад розповсюдження Sodinokibi ransomware у Китаї за допомогою спам атаки, що використовували листи, що начебто були надіслані від великої транспортної компанії DHL.

Рисунок 2.2 – Спам лист DHL, надісланий китайським користувачам [15]

[1] Фішингова атака – це тип атаки соціальної інженерії, під час якої зловмисник надсилає повідомлення, як правило, електронну пошту чи текстове повідомлення, яке, здається, надійшло з авторитетного чи надійного джерела, наприклад банку чи платформи соціальних мереж, намагаючись змусити одержувача розкрити конфіденційну інформацію або натиснути посилання, яке встановлює зловмисне програмне забезпечення.

Як можна побачити на Рисунку 2.1 лист є дуже схожим на те, що може прислати компанія при власній розсилці, але при цьому ще містить вкладений .zip архів, що після розархівування містить в собі лише один файл DHL.doc.exe.

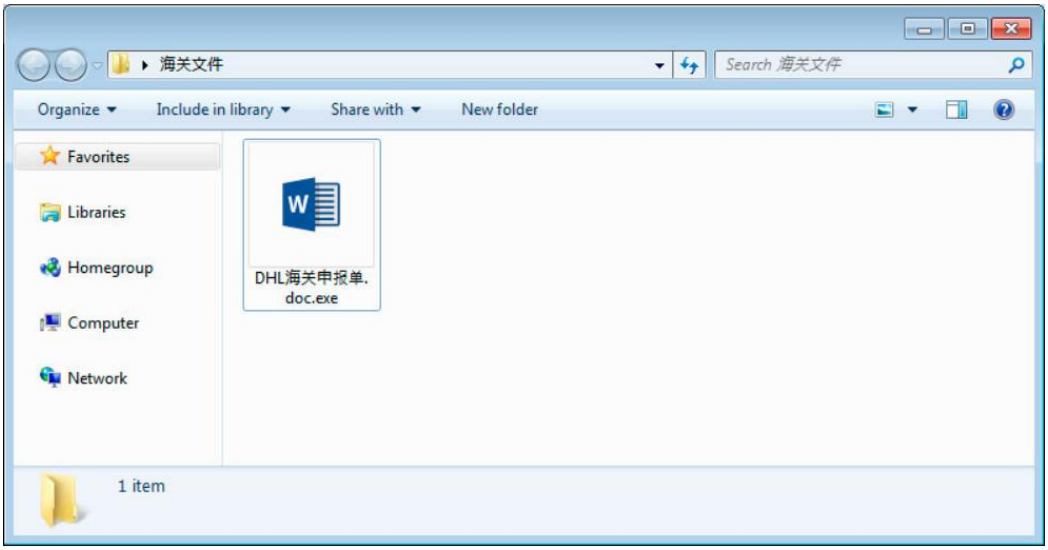


Рисунок 2.3 – Файл DHL.doc.exe, що починає запуск ШПЗ [15]

Відомим є факт, що у більшості користувачів ОС Windows залишаються налаштування за замовчуванням. Таким чином, розширення файлів далеко не завжди відображаються у жертви, що забезпечує те, що видно лише DHL.doc та логотип текстового редактору Microsoft Word, що майже не викликає підозри у звичайного користувача.

Після запуску ШПЗ для початку деактивуються певні служби і процеси перед тим як почати шифрування файлів. Зазвичай ці процеси пов’язані з резервним копіюванням файлів, службами антивірусних засобів захисту. Після зупинки згаданих раніше процесів видаляються вже наявні резервні копії створені за допомогою служби Windows VSS[2].

Для ОС Windows версії XP використовується команда для вбудованої командної оболонки cmd.exe:

[2] Служба VSS — це вбудована технологія резервного копіювання та відновлення в операційних системах Windows. Це дозволяє користувачам створювати знімки або копії даних, що зберігаються на диску в певний момент часу, у той час як диск продовжує використовуватися для інших операцій.

```
cmd.exe /c vssadmin.exe Delete Shadows /All /Quiet & bcdedit /set {default}
recoveryenabled No & bcdedit /set {default} bootstatuspolicy
ignoreallfailures
```

В ОС Windows більш нових версій вже використовується командна оболонка PowerShell з наступною командою:

```
Get-WmiObject Win32_Shadowcopy | ForEach-Object {$_.Delete();}
```

Далі програма переходить вже до виконання корисного навантаження, а саме перебору файлової системи та власне шифрування інформації користувачів. Вимагач за замовчуванням має шифрувати усі файли системи, але це може бути і не так. Якщо вказати необхідні параметри командного рядка, то можна відмінити шифрування певних файлів, розширень, мережевих директорій, або навіть задати прямий шлях до директорії, яку необхідно зашифрувати. На практиці найбільш популярним же є шифрування усіх файлів з урахуванням певних виключень. В наступному списку як раз можна побачити «білий» список каталогів (fld), файлів (fls) та розширень (ext):

```
"wht": {
    "fld": [
      "msocache", "intel", "$recycle.bin", "google", "perflogs",
      "system volume information", "windows", "mozilla", "appdata",
      "tor browser", "$windows.~ws", "application data", "$windows.~bt",
      "boot", "windows.old"
    ],
    "fls": [
      "bootsect.bak", "autorun.inf", "iconcache.db", "thumbs.db", "ntuser.ini",
      "boot.ini", "bootfont.bin", "ntuser.dat", "ntuser.dat.log", "ntldr",
      "desktop.ini"
    ],
    "ext": [
      "com", "ani", "scr", "drv", "hta", "rom", "bin", "msc", "ps1", "diagpkg",
      "shs", "adv", "msu", "cpl", "prf", "bat", "idx", "mpa", "cmd", "msi",
      "mod", "ocx", "icns", "ics", "spl", "386", "lock", "sys", "rtp", "wpx",
      "diagcab", "theme", "deskthemepack", "msp", "cab", "ldf", "nomedia", "icl",
      "lnk", "cur", "dll", "nls", "themepack", "msstyles", "hlp", "key", "ico",
      "exe", "diagcfg"
    ]
  }
```

Тут слід зазначити, що ШПЗ також має виключення щодо регіонів виконання корисного навантаження. Для цього програма перевіряє ідентифікатор клавіатури за допомогою функції GetKeyboardLayoutList(). При цьому перевіряється лише молодший байт ідентифікатора, котрий при значенні у

інтервалі від \x18 до \x44 повертає значення true, тобто даний розклад клавіатури користувача знаходиться у «білому списку». Таким чином, повний список мов клавіатур, що входять до «білого списку» REvil наведено у таблиці 2.1.

Таблиця 2.1 – Мови клавіатури, що знаходяться у «білому списку» REvil

Мова клавіатури	**Ідентифікатор**	**Мова клавіатури**	**Ідентифікатор**
1	**2**	**3**	**4**
Албанська	0x0000041c	Перська (стандартна)	0x00050429
Вірменський схід	0x0000042b	Румунська (Legacy)	0x00000418
Вірменська фонетика	0x0002042b	Румунська (програмісти)	0x00020418
Вірменська друкарська машинка	0x0003042b	Румунська (стандартна)	0x00010418
Вірменська захід	0x0001042b	Російська	0x00000419
Азербайджанська (стандартна)	0x0001042c	Російська – Мнемотехніка	0x00020419
Азербайджанська кирилиця	0x0000082c	Російська (друкарська машинка)	0x00010419
Азербайджанська лат	0x0000042c	Самі Розширена Фінляндія-Швеція	0x0002083b
Білоруська	0x00000423	Самі Розширена Норвегія	0x0001043b
Боснійська (кирилиця)	0x0000201a	Сербська (кирилиця)	0x00000c1a
Центральнокурдська	0x00000429	Сербська (латиниця)	0x0000081a
Хорватська	0x0000041a	Сетсвана	0x00000432
Деванагарі-ІНСКРИПТ	0x00000439	Словацька	0x0000041b
Естонська	0x00000425	Словацька (QWERTY)	0x0001041b
Фарерська	0x00000438	Словенська	0x00000424
Фінська з саамами	0x0001083b	Лужицька розширена	0x0001042e
Грузинська	0x00000437	Сорбська стандартна	0x0002042e

Кінець таблиці 2.1

1	2	3	4
Грузинська (ергономічна)	0x00020437	Лужицька стандартна (спадщина)	0x0000042e
Грузинська (QWERTY)	0x00010437	Шведська	0x0000041d
Школи Міністерства освіти і науки Грузії	0x00030437	Шведська з саамами	0x0000083b
Грузинська (старі алфавіти)	0x00040437	Таджицька	0x00000428
Хінді традиційна	0x00010439	Татарський	0x00010444
Казахська	0x0000043f	Татарська (Спадок)	0x00000444
Киргизька кирилиця	0x00000440	Тайський Кедмані	0x0000041e
Латиська (стандарт)	0x00020426	Thai Kedmanee (без ShiftLock)	0x0002041e
Латиська (Legacy)	0x00010426	Тайська паттачоте	0x0001041e
Литовська	0x00010427	Тайський Pattachote (без ShiftLock)	0x0003041e
Литовська IBM	0x00000427	Турецька Ф	0x0001041f
Литовська стандартна	0x00020427	Турецька QoETO.exe	0x0000041f
Македонія (КЮРМ)	0x0000042f	Туркменська	0x00000442
Македонія (КЮРМ) - Стандарт	0x0001042f	Українська	0x00000422
Мальтійська 47-Кей	0x0000043a	Українська (розширена)	0x00020422
Мальтійська 48-ма	0x0001043a	Урду	0x00000420
Норвезька з саамами	0x0000043b	Узбецька кирилиця	0x00000843
Перська	0x00000429	В'єтнамська	0x0000042a

Таким чином, можна прийти до узагальненої схеми потоку використання Sodinikibi.

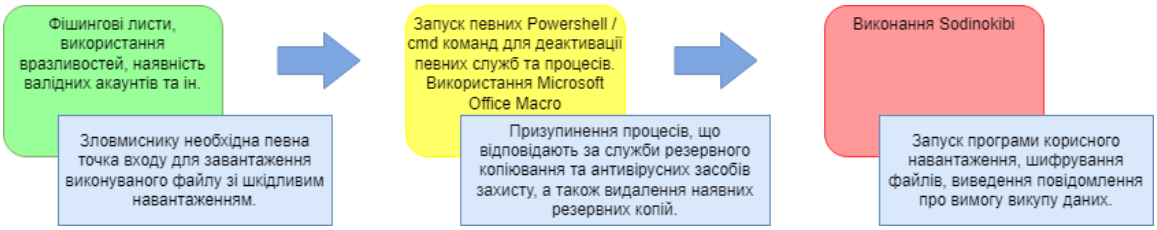


Рисунок 2.4 – Спрощений потік виконання Sodinokibi

Згідно статичного аналізу наявного зразка, що був виконаний Ніколя Гійо [16] робота з перерахунку файлової системи відбувається у декілька етапів: перерахунок локальних файлів та файлів мережевих ресурсів. В даній роботі більш детально будемо розглядати вразливість виключно локальної файлової системи.

```
mw_wrap_rc4_decrypt(&encrypted_strings_blob, 909, 5, 14, &output_string);// \\?\A:\
v5 = 0;
mw_maybe_memcpy(RootPathName, &output_string);
while ( RootPathName[4] <= 'Z' )              // Enumerate all drives
{
  if ( GetDriveTypeW(RootPathName) - 2 <= 2 ) // If Drive exists
  {
    mw_enum_path_files(RootPathName, file_enumerator);
    DriveLetter = RootPathName[4];
    if ( DriveLetter >= 'a' && DriveLetter <= 'z' )
      RootPathName[4] = DriveLetter & 0xFFDF;
  }
  ++RootPathName[4];
  RootPathName[7] = 0;
}
```

Рисунок 2.5 – Приклад декомпільованого коду, що використовується при перерахуванні файлів

В прикладі «Рисунку 2.3» можемо побачити, що в циклі while виконується перевірка на наявність дисків від «A:\» до «Z:\». За допомогою функції GetDriveTypeW() отримуємо інформацію про наявність локального диску (значення, що повертаються усі, окрім 5 – DRIVE_CDROM та 6 – DRIVERAMDISK), після чого викликається функція mw_enum_path_files(). Її робота розпочинається з того, що перевіряється чи знаходиться директорія у «білому» списку, якщо ж це не так, то за допомогою API функцій FindFirstFile() та FindNextFile() виконується перелік файлів. Кожен з файлів, що знаходиться в «білому» списку просто ігнорується, в іншому випадку його ім’я записується у

файл {EXT}-readme.txt, де EXT відповідає новому розширенню зашифрованого файлу, що генерується випадковим чином і містить інструкції щодо викупу.

Зрозуміло, що виконання цього коду у межах всієї файлової системи, що складає гігабайти або навіть терабайти інформації, потребує великого проміжку часу, тому слід використовувати можливості обчислювальної системи на максимум. Тут на допомогу зловмисникам як раз і приходить використання найефективнішого але й найменш відомого механізму асинхронної роботи I/O Completion Port. В нашому випадку він використовується наступним чином:

1. Створення I/O Completion Port

```
IOCompletionPortHandle = CreateIoCompletionPort(INVALID_HANDLE_VALUE, 0, 0, NumberOfConcurrentThread);
IOCP_info->CompletionPortHandle = IOCompletionPortHandle;
if ( !IOCompletionPortHandle )
{
  mw_wrap_HeapDestroy(IOCP_info->HeapHandle);
  return 0;
}
if ( mw_create_thread_pool(IOCP_info, encryption_routine) )
  return 1;
```

Рисунок 2.6 – Створення I/O Completion Port

2. Створення пулу потоків

```
while ( 1 )
{
  ThreadHandle = CreateThread(0, 0, encryption_routine, IOCP_info, 0, 0);
  if ( !ThreadHandle )
    break;
  ++IOCP_info->nb_threads;
  wrap_CloseHandle(ThreadHandle);
  if ( ++v2 >= 2 * mw_get_cpu_nb() )
    return 1;
}
```

Рисунок 2.7 – Створення пулу потоків

3. Усі потоки очікують події GetQueuedCompletionStatus()
4. Коли файл знайдений за допомогою mw_enum_path_files() він додається до I/O Completion Port

```
if ( mw_add_file_to_CompletionPort(IOCP_info, processing_info->FileHandle, 0) )
{
  processing_info->next_processing_step = 1;
  if ( mw_wrap_PostQueueCompletionStatus(IOCP_info, 0, 0, processing_info) )
  {
    LODWORD(result) = 1;
    goto LABEL_9;
```

Рисунок 2.8 – Додавання файлів до I/O Completion Port

5. Пакет завершення публікується, щоб функція PostQueuedCompletionStatus() повідомила потік про те, що файл потрібно зашифрувати.

Далі відбувається безпосередньо шифрування файлів, механізм якого власне є настільки складним і комплексним, що детектування атаки на попередніх етапах є найбільш пріоритетним завданням.

Шифрування відбувається у декілька етапів. По-перше, зчитується блок файлу, що складає розміром 1 МБ, потім цей блок шифрується згідно певного алгоритму шифрування. Ці кроки повторюються до тих пір, поки весь вміст файлу не буде зашифровано. Далі до кінця файлу додаються метадані файлу і до початкового файлу додається розширення {EXT}, що згадувалося раніше. Тут слід зазначити, що не завжди шифрується вміст всього файлу, оскільки робота з блоками всього по 1 МБ є досить кропіткою та займає багато часу. Таким чином, можуть шифруватися блоки з певним інтервалом, значення якого вказується зловмисником у конфігураційному файлі, який іде разом з програмою-вимагачем.

Дослідники виявили, що в якості механізму шифрування використовується суміш алгоритмів Salsa-20 та AES. Це вдалося виявити через те, що алгоритми є загальновідомими і відповідно мають відому послідовність байтів у своїй реалізації, що дозволяють однозначно їх ідентифікувати за допомогою YARA правил, або відомого плагіну FindCrypt [17].

У разі успішного виконання шифрування остаточним етапом є заміна картинки фону робочого стола, що вказує на ім’я файлу з подальшими інструкціями. У цьому файлі також наявна URL для ресурсу, що повинен

розшифровувати вміст файлів. Адреса також генерується випадково для кожного випадку зараження системи за допомогою наступного патерну:

```
https://<c2 domain>/<URL_sub1>/<URL_sub2>/<random_resource_nma>.<ext>
```

Тут C2 сервер є доменом, який обирається з певного набору доменних імен, що контролюються зловмисниками. URL_sub1 – значення, що обирається тільки з наступного списку ["wp-content", "static", "content", "include", "uploads", "news", "data", "admin"]. URL_sub2 – також значення, що вибирається випадковим чином зі списку ["images", "pictures", "image", "temp", "tmp", "graphic", "assets", "pics". », «game»]. В якості ext обирається одне з наступних розширень зображень: ["jpg", "png", "gif"]. Таким чином приклади URL-адрес, що використовуються наведено на рисунку 2.9.

```
https://cymru.futbol/wp-content/assets/rjozgsac.gif
https://chorusconsulting.net/static/images/okhmjbkeggsrchqqwv.jpg
https://stagefxinc.com/uploads/pictures/audhents.png
https://kartuindonesia.com/data/temp/shen.jpg
https://craftingalegacy.com/content/pics/pqucnayd.png
https://cleanroomequipment.ie/admin/game/fhskeydbns.gif
```

Рисунок 2.9 – Приклад URL-адрес до C2 серверу, що використовується REvil

При відкритті цієї URL-адреси користувачу необхідно ввести відповідний ключ у форматі Base64, що також наявний у вищезгаданому файлі з інформацією про викуп. Після цього відкривається головна сторінка, що повідомляє жертву про вартість викупу у біткойнах, час який залишився на виконання викупу, а також вартість, яка логічно зросте після завершення наданого часу. Приклад цієї сторінки можна побачити на рисунку 2.10.

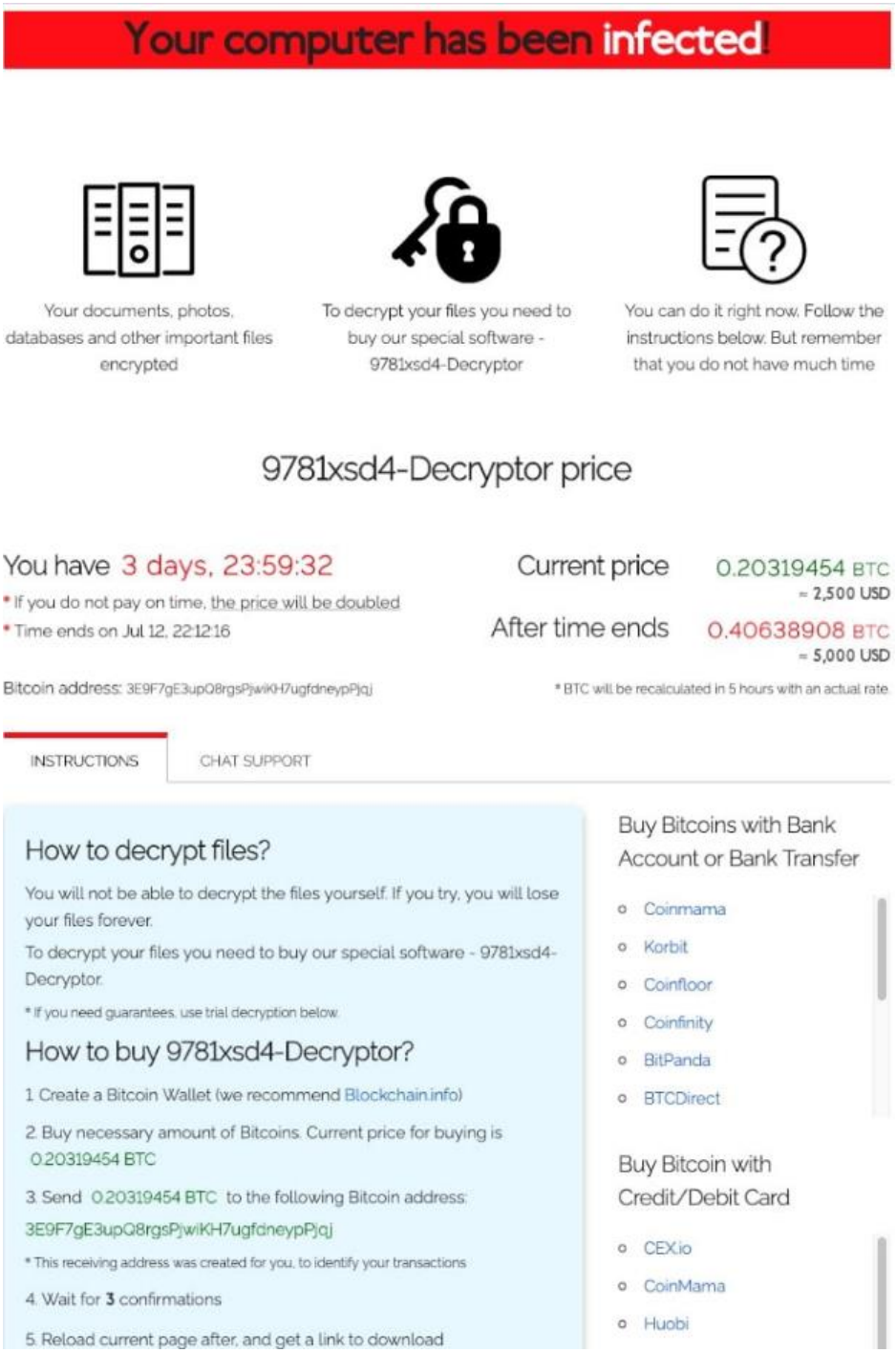


Рисунок 2.10 – Приклад web-сторінки з інформацією про викуп [18]

Також тут можна побачити інформацію про інструкції, що необхідно зробити користувачу для реалізації викупу, а також доступ до чату з підтримкою. При цьому зловмисники надають можливість переконатися, що файли дійсно можуть бути розшифрованими. Для цього у користувача наявна можливість розшифрувати один файл з власної системи, що є зашифрованою картинкою в

одному з форматів: jpj, png, gif [18]. Для цього використовується інструмент показаний на рисунку 2.11.

Рисунок 2.11 – Форма для розшифрування файлів [18]

2.4 Ефективність використання I/O Completion Port в порівнянні з альтернативними механізмами

Як зазначалося у розділі 1 даної роботи, I/O Completion Port є не єдиним механізмом, що може використовуватися при обробці програм, що використовують асинхронний ввід/вивід. Проте на прикладі Sodinokibi/REvil гарно видно, що за певної причини хакери вирішили використовувати саме цей механізм, незважаючи на його не високу популярність. Саме тому й виникає необхідність у порівнянні інших альтернативних рішень.

В результаті альтернативи було вирішено обрати Asynchronous Procedure Calls (APC). APC це використання функції, яка виконується асинхронно в контексті певного потоку. Коли APC стоїть у черзі потоку, система видає програмне переривання. Наступного разу, коли потік буде заплановано, він запустить функцію APC. В даному випадку будемо використовувати класичний вид реалізації APC, хоча існують і певні покращення часу роботи, наприклад за допомогою використання threading pool, але вони є трудомісткими для програмістів та пришвидшують роботу невідповідно до вкладених зусиль.

Для порівняльного аналізу були реалізовані дві програми мовою C++, що за змістом дуже схожі до кроків, описаних у принципі роботи програми-вимагача Sodinokibi/REvil, за єдиним виключенням, що створений екземпляр не містить жодного корисного навантаження, що може пошкодити ОС Windows, або якось зіпсувати файли користувачів. Перша програма використовує I/O Completion Port, а друга Asynchronous Procedure Calls (APC). Вихідні коди програм можна подивитися у «Додатку А» та «Додатку Б» відповідно.

Суть роботи програм полягає в тому, що за допомогою механізмів асинхронного вводу/виводу виконуються операції рекурсивного обходу файлів за вказаною директорією, перевірка чи знаходиться файл у білому списку, запис імен знайдених файлів в окремий файл та власне читання файлу, що буде імітувати процес навантаження. При цьому будемо засікати час, що потрібен на виконання програми і фіксувати його для різної кількості файлів заданих у початковій директорії.

2.4.1 Принцип роботи програми, що використовує I/O Completion Port

Робота програми починається зі зчитування імен файлів, що знаходяться в білому списку. Ця процедура виконується за допомогою функції read_lines(), що зчитує по рядку з наданого файлу та записує їх ім’я в вектор, що власне і повертається на виході.

```
25  std::vector<std::wstring> read_lines(const std::wstring& filename) {
26      std::vector<std::wstring> lines;
27
28      std::wifstream file(filename);
29      if (!file) {
30          // handle file opening error
31          std::wcerr << "Error: failed to open file " << filename << std::endl;
32          return lines;
33      }
34
35      std::wstring line;
36      while (std::getline(file, line)) {
37          lines.push_back(line);
38      }
39
40      return lines;
41  }
```

Рисунок 2.12 – Функція зчитування білого списку файлів у вектор

В нашому випадку файл білого списку має назву whitelist_files.txt та має наступний зміст як на рисунку 2.13.

whitelist_files.txt

```
1  bootsect.bak
2  autorun.inf
3  iconcache.db
4  thumbs.db
5  ntuser.ini
6  boot.ini
7  bootfont.bin
8  ntuser.dat
9  ntuser.dat.log
10 ntldr
11 desktop.ini
```

Рисунок 2.13 – Білий список файлів

Далі за допомогою функції CreateIoCompletionPort() власне створюється об’єкт I/O Completion Port, який буде використовуватися в подальшому. З цього

самого моменту також починається відлік часу роботи нашої програми. Після виконується створення потоків, що будуть опрацьовувати необхідні нам операції. В даному випадку кількість потоків буде визначатися кількістю ядер центрального процесору встановленого на фізичній системі. Для тестового середовища, що використовувалося для порівняння, кількість потоків дорівнює 8.

```
153     HANDLE iocp = CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, 0, 0);
154     DWORD start = GetTickCount64();
155     if (iocp == NULL) {
156         std::cerr << "Error: CreateIoCompletionPort failed with error code " << GetLastError() << std::endl;
157         return 1;
158     }
159     const int num_threads = std::thread::hardware_concurrency();
160     std::vector<std::thread> threads;
161     for (int i = 0; i < num_threads; i++) {
162         threads.emplace_back(thread_proc, iocp);
163     }
```

Рисунок 2.14 – Створення об’єкту I/O Completion Port

Як можна побачити на рисунку 2.14, робота потоків асоціюється з функцією thread_proc(), що у нескінченному циклі намагається отримати інформацію з черги I/O Completion Port за допомогою функції GetQueuedCompletionStatus() та обробити отримані дані зі структури overlapped. При цьому завершення потоків відбувається у момент коли структура overlapped переходить у значення Null, що свідчить про закінчення черги на виконання операцій вводу/виводу.

```
void thread_proc(HANDLE iocp) {
    DWORD bytes_transferred = 0;
    ULONG_PTR completion_key = 0;
    LPOVERLAPPED overlapped = NULL;

    while (true) {
        std::unique_lock<std::mutex> lock(data_mutex);
        BOOL result = GetQueuedCompletionStatus(
            iocp,
            &bytes_transferred,
            &completion_key,
            &overlapped,
            INFINITE);

        if (!result) {
            std::cerr << "Error: GetQueuedCompletionStatus failed with error code " << GetLastError() << std::endl;
            continue;
            //lock.unlock();
        }
        else {
            //lock.unlock();
            if (overlapped == NULL) {
                std::cout << "Thread " << std::this_thread::get_id() << " is exiting" << std::endl;
                //lock.unlock();
                break;
            }

            // обробка завершення I/O

            // отримати інформацію про файл
            OverlappedPlus* overlapped_plus = (OverlappedPlus*)overlapped;
            FileInfo* file_info = overlapped_plus->data;

            std::wcout << L"Processing file " << file_info->filename << L"..." << std::endl;

            // обробка файлу
            process_file(file_info->filename);
            delete file_info;
            delete overlapped_plus;
            lock.unlock();
        }
    }
}
```

Рисунок 2.15 – Функція thread_proc, що обробляється потоками

На рисунку 2.15 видно, що отримання даних про файл виконується за допомогою структури OverlappedPlus. Ця структура містить інформацію про структуру Overlapped та структуру FileInfo, де зберігаються необхідні дані про файл.

```
struct FileInfo {
    std::wstring filename;
    DWORD size;
};

struct OverlappedPlus {
    OVERLAPPED overlapped;
    FileInfo* data;
};
```

Рисунок 2.16 – Структури FileInfo та OverlappedPlus

Після отримання імені файлу, виконується функція process_file(), що отримує ім’я файлу та записує його у необхідний нам файл, в даному випадку example.txt.

```
43  void process_file(const std::wstring& filename) {
44      std::wofstream file;
45      file.open("example.txt", std::ios_base::app); // open file in append mode
46      if (file.is_open()) {
47          file << filename << L"\n"; // write wstring to file
48          file.close();
49          std::wcout << L"Successfully wrote to file." << std::endl;
50      }
51      else {
52          std::wcerr << L"Failed to open file." << std::endl;
53      }
54      std::wcout << filename << std::endl;
55  }
```

Рисунок 2.17 – Функція обробки файлів process_file()

Для того, щоб виконати рекурсивний обхід заданої директорії та асоціювати I/O Completion Port з необхідним HANDL-ом файлу, використовується функція scan_directory().

В даному випадку рекурсивний обхід буде виконуватися за допомогою recursive_directory_iterator, що наявний у namespace std::filesystem. Тут необхідно зробити зауваження, що згаданий namespace може бути використаний лише зі стандартом C++ 17 та більш новими. Далі виконується перевірка на знаходження файлу у білому списку й якщо ні, то починається асоціація файлу з об’єктом I/O Completion Port. При цьому створюються структури FileInfo та OverlappedPlus. HANDLE файлу отримується за допомогою WinAPI функції CreateFileW(). Додавання запиту вводу/виводу до I/O Completion Port виконується також за допомогою функції CreateIoCompletionPort, де в якості першого аргументу передається HANDLE файлу, а другого – HANDLE об’єкту I/O Completion Port.

```cpp
void scan_directory(const std::wstring& directory_name, HANDLE iocp, const std::vector<std::wstring>& whitelist) {
    for (const auto& entry : recursive_directory_iterator(directory_name)) {
        if (entry.is_regular_file()) {
            // перевірка файлу проти білого списку
            std::wstring filename = entry.path().filename();
            bool is_whitelisted = std::find(whitelist.begin(), whitelist.end(), filename) != whitelist.end();

            if (!is_whitelisted) {
                // створити структуру FileInfo для передачі інформації про файл
                FileInfo* file_info = new FileInfo();
                file_info->filename = entry.path();
                file_info->size = entry.file_size();

                // створити структуру OverlappedPlus для передачі FileInfo в I/O completion port
                OverlappedPlus* overlapped_plus = new OverlappedPlus();
                ZeroMemory(&overlapped_plus->overlapped, sizeof(OVERLAPPED));
                overlapped_plus->data = file_info;

                // додати I/O запит до I/O completion port
                HANDLE file_handle = CreateFileW(
                    file_info->filename.c_str(),
                    GENERIC_READ,
                    FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE,
                    NULL,
                    OPEN_EXISTING,
                    FILE_FLAG_OVERLAPPED,
                    NULL);

                if (file_handle == INVALID_HANDLE_VALUE) {
                    std::cerr << "Error: CreateFile failed with error code " << GetLastError() << std::endl;
                    delete file_info;

                    delete overlapped_plus;
                    continue;
                }

                if (!CreateIoCompletionPort(file_handle, iocp, (ULONG_PTR)file_handle, 0)) {
                    std::cerr << "Error: CreateIoCompletionPort failed with error code " << GetLastError() << std::endl;
                }
                char buffer[1024];
                ReadFile(file_handle, buffer, sizeof(buffer), NULL, &overlapped_plus->overlapped);
                //std::wcout << filename << std::endl;
                CloseHandle(file_handle);
            }
        }
    }
}
```

Рисунок 2.18 – Функція рекурсивного обходу заданої директорії та асоціювати I/O Completion Port з необхідним HANDL-ом файлу scan_directory()

Після виходу з функції thread_proc() потоки необхідно коректно завершити та зафіксувати час закінчення обробки файлів для розрахунку часу виконання, що буде використовуватися при порівнянні.

```
    // закінчення роботи потоків
    for (int i = 0; i < num_threads; i++) {
        PostQueuedCompletionStatus(iocp, 0, NULL, NULL);
    }
    for (auto& thread : threads) {
        thread.join();
    }

    CloseHandle(iocp);
    DWORD end = GetTickCount64();

    DWORD duration = end - start;
    std::cout << "Execution time " << duration << " miliseconds" << std::endl;

    system("pause");
    return 0;
}
```

Рисунок 2.19 – Закінчення роботи потоків та фіксація часу

Робота програми представлена у вигляді консольного застосунку, що буде показувати процес обробки файлів у заданій директорії, ілюструвати завершення роботи потоків, вказуючи їх ідентифікатори id, а також виводити час необхідний на виконання програми.

```
D:\1500_files\Labs_matlab\Labs_matlab\laba6\Variant4_l4.mat
Processing file D:\1500_files\Labs_matlab\Labs_matlab\laba6\Variant5_l4.mat...
Successfully wrote to file.
D:\1500_files\Labs_matlab\Labs_matlab\laba6\Variant5_l4.mat
Processing file D:\1500_files\Labs_matlab\Labs_matlab\laba6\Variant6_l4.mat...
Successfully wrote to file.
D:\1500_files\Labs_matlab\Labs_matlab\laba6\Variant6_l4.mat
Processing file D:\1500_files\Labs_matlab\Labs_matlab\laba6\Variant7_l4.mat...
Successfully wrote to file.
D:\1500_files\Labs_matlab\Labs_matlab\laba6\Variant7_l4.mat
Processing file D:\1500_files\Labs_matlab\Labs_matlab\mccExcludedFiles.log...
Successfully wrote to file.
D:\1500_files\Labs_matlab\Labs_matlab\mccExcludedFiles.log
Processing file D:\1500_files\Labs_matlab\Labs_matlab\readme.txt...
Successfully wrote to file.
D:\1500_files\Labs_matlab\Labs_matlab\readme.txt
Processing file D:\1500_files\Labs_matlab\Labs_matlab\variants.numbers...
Successfully wrote to file.
D:\1500_files\Labs_matlab\Labs_matlab\variants.numbers
Processing file D:\1500_files\Labs_matlab\Labs_matlab\Error: GetQueuedCompletionStatus failed with error code 38
Thread 17176 is exiting
Thread 16488 is exiting
Thread 5700 is exiting
Thread 6496 is exiting
Thread 1380 is exiting
Thread 4100 is exiting
Thread 17156 is exiting
Thread 6620 is exiting
Execution time 547 miliseconds
Для продолжения нажмите любую клавишу . . .
```

Рисунок 2.20 – Результат виконання програми

На виході отримуємо файл example.txt, що відповідає змісту як показано на рисунку 2.21.

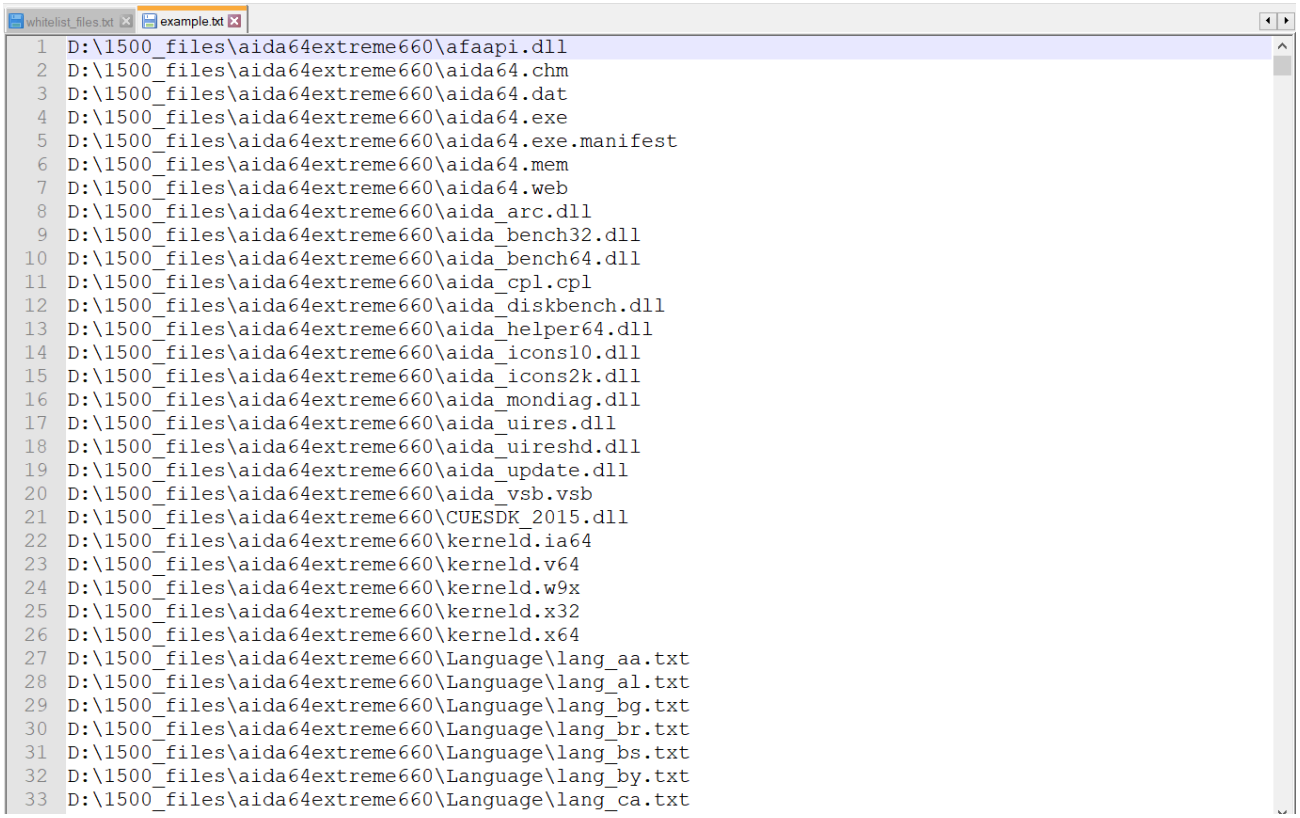


Рисунок 2.21 – Вміст файлу example.txt, отриманий при використанні I/O Completion Port

2.4.2 Принцип роботи програми, що використовує Asynchronous Procedure Calls

Аналогічно до попередньої програми, що використовує I/O Completion Port, робота починається зі зчитування імен файлів, що знаходяться у білому списку. У даному випадку цим файлом є whitelist_files.txt. Зчитування відбувається за допомогою функції read_lines(), що цілком ідентична попередньому пункту.

Далі викликається функція scan_directory (рисунок 2.22), що власне і містить головну відмінність. На вхід до неї подається ім’я директорії з якої слід проводити рекурсивний обхід файлів, а також вектор, що містить файли білого списку. При цьому після виконання перевірки на наявність відповідного імені

файлу у векторі відбувається додавання APC до потоку виконання, що й розглянемо більш детально.

```
45  void scan_directory(const std::wstring& directory_name, const std::vector<std::wstring>& whitelist) {
46      for (const auto& entry : recursive_directory_iterator(directory_name)) {
47          if (entry.is_regular_file()) {
48              // перевірка файлу проти білого списку
49              std::wstring filename = entry.path().filename();
50              bool is_whitelisted = std::find(whitelist.begin(), whitelist.end(), filename) != whitelist.end();
51
52              if (!is_whitelisted) {
53                  // додати APC до потоку
54                  QueueUserAPC((PAPCFUNC)process_file, GetCurrentThread(), (ULONG_PTR)filename.c_str());
55              }
56          }
57      }
58  }
```

Рисунок 2.22 – Функція рекурсивного обходу заданої директорії та додавання APC до потоку виконання

Для асоціації об’єкту APC з чергою APC для певного потоку виконання використовується Windows API функція QueueUserAPC(), що має в загальному випадку наступний вигляд [19]:

```
DWORD QueueUserAPC(
  [in] PAPCFUNC  pfnAPC,
  [in] HANDLE    hThread,
  [in] ULONG_PTR dwData
);
```

При цьому в якості параметрів отримуються pfnAPC (вказівник PAPCFUNC на функцію APC, що знаходиться в області бачення програми та викликається при отриманні повідомлення очікування від вказаного потоку), hThread (об’єкт типу HANDLE, що вказує на відповідний потік виконання, при цьому слід зазначити, що потік повинен мати права THREAD_SET_CONTEXT для успішного виконання) та dwData (аргументи APC функції, що передаються у вигляді ULONG_PTR вказівника). В якості dwData в нашому випадку передається вказівник на ім’я знайденого файлу для запису.

PAPCFUNC callback функція APC при цьому має вигляд як на рисунку 2.23.

```cpp
void CALLBACK process_file(ULONG_PTR param) {
    std::wstring filename = std::to_wstring(param);
    std::wofstream file;
    file.open("example.txt", std::ios_base::app); // open file in append mode
    if (file.is_open()) {
        file << filename << L"\n"; // write wstring to file
        file.close();
        std::wcout << L"Successfully wrote to file." << std::endl;
    }
    else {
        std::wcerr << L"Failed to open file." << std::endl;
    }
    std::wcout << filename << std::endl;
}
```

Рисунок 2.23 – PAPCFUNC callback функція APC

Вивід програми при цьому залишається незмінним у порівнянні з попередньою програмою, що використовує I/O Completion Port та представляє собою текстовий файл example.txt, що містить перелік знайдених файлів (рисунок 2.24).

example.txt

```
D:\1500_files\Labs_matlab\Labs_matlab\!!README.txt
D:\1500_files\Labs_matlab\Labs_matlab\.DS_Store
D:\1500_files\Labs_matlab\Labs_matlab\.git\COMMIT_EDITMSG
D:\1500_files\Labs_matlab\Labs_matlab\.git\config
D:\1500_files\Labs_matlab\Labs_matlab\.git\description
D:\1500_files\Labs_matlab\Labs_matlab\.git\HEAD
D:\1500_files\Labs_matlab\Labs_matlab\.git\hooks\applypatch-msg.sample
D:\1500_files\Labs_matlab\Labs_matlab\.git\hooks\commit-msg.sample
D:\1500_files\Labs_matlab\Labs_matlab\.git\hooks\post-commit.sample
D:\1500_files\Labs_matlab\Labs_matlab\.git\hooks\post-receive.sample
D:\1500_files\Labs_matlab\Labs_matlab\.git\hooks\post-update.sample
D:\1500_files\Labs_matlab\Labs_matlab\.git\hooks\pre-applypatch.sample
D:\1500_files\Labs_matlab\Labs_matlab\.git\hooks\pre-commit.sample
D:\1500_files\Labs_matlab\Labs_matlab\.git\hooks\pre-push.sample
D:\1500_files\Labs_matlab\Labs_matlab\.git\hooks\pre-rebase.sample
D:\1500_files\Labs_matlab\Labs_matlab\.git\hooks\prepare-commit-msg.sample
D:\1500_files\Labs_matlab\Labs_matlab\.git\hooks\update.sample
D:\1500_files\Labs_matlab\Labs_matlab\.git\index
D:\1500_files\Labs_matlab\Labs_matlab\.git\info\exclude
D:\1500_files\Labs_matlab\Labs_matlab\.git\logs\HEAD
D:\1500_files\Labs_matlab\Labs_matlab\.git\logs\refs\heads\master
D:\1500_files\Labs_matlab\Labs_matlab\.git\objects\03\9214e23186c2b186b1294aff93c55a84e159c1
D:\1500_files\Labs_matlab\Labs_matlab\.git\objects\06\bb669238a10c7a2ed41b66b763d5b4e4d314df
D:\1500_files\Labs_matlab\Labs_matlab\.git\objects\08\1ecd3b2428e64e271eee5d4772f5d64c666fe4
D:\1500_files\Labs_matlab\Labs_matlab\.git\objects\0b\877694d183bd5898b731d8234b061ebf7f78e3
D:\1500_files\Labs_matlab\Labs_matlab\.git\objects\0c\79f8cb214697ac68797f6b0d9ecabd2df2994a
D:\1500_files\Labs_matlab\Labs_matlab\.git\objects\0d\8150e04bf22767a176b7ea0a856871438b229d
D:\1500_files\Labs_matlab\Labs_matlab\.git\objects\0e\9823660736fcf79bd475260098680863894012
D:\1500_files\Labs_matlab\Labs_matlab\.git\objects\0f\2afc1c0ac7d0af69cdeda33602e686b2631bc8
D:\1500_files\Labs_matlab\Labs_matlab\.git\objects\10\22c2919a458e17a23c17acf0978463ae6a02d6
D:\1500_files\Labs_matlab\Labs_matlab\.git\objects\11\6457077dcd74303c3a24f2130fea24961aa561
D:\1500_files\Labs_matlab\Labs_matlab\.git\objects\13\df05923b6b0b87e4fa44df19f2ed24a4418787
D:\1500_files\Labs_matlab\Labs_matlab\.git\objects\14\5c41cdd5f6d7f60ff998af36b837cc14afd41e
```

Рисунок 2.24 – Вміст файлу example.txt, отриманий при використанні APC

2.4.3 Порівняльна характеристика використання I/O Completion Port та APC

За результатами часових замірів виконання програм на директоріях, що містять задану кількість файлів було отримано наступні значення, що подано в табличному вигляді (Таблиця 2.2).

Таблиця 2.2 – Залежність кількості файлів від часу при використанні I/O Completion Port та Asynchronous Procedure Calls

Кількість файлів, шт	IOCP Час, мс	APC Час, мс
100	156	625
500	219	3062
1000	391	6547
1500	547	9797
3000	969	13373
6000	1609	26748

Як можемо побачити (рисунок 2.25) використання Asynchronous Procedure Calls (APC) є вкрай не ефективним при значному обсязі файлів. Натомість час виконання програми, що використовує I/O Completion Port, у декілька разів менше за наведену альтернативу. Також він є не пропорційним коефіцієнту збільшення файлів. Так, наприклад, коефіцієнт збільшення часу при збільшенні файлів від 1500 до 3000 складає близько 1.77, натомість при збільшенні від 3000 до 6000 коефіцієнт складає вже близько 1.66. Це свідчить про те, що зі збільшенням кількості файлів, які оброблюються за допомогою I/O Completion Port його ефективність не зменшується, а може навіть трохи збільшуватися. Така поведінка не є властивою для жодного альтернативного методу асинхронного вводу/виводу, що наявні на даний час[20].

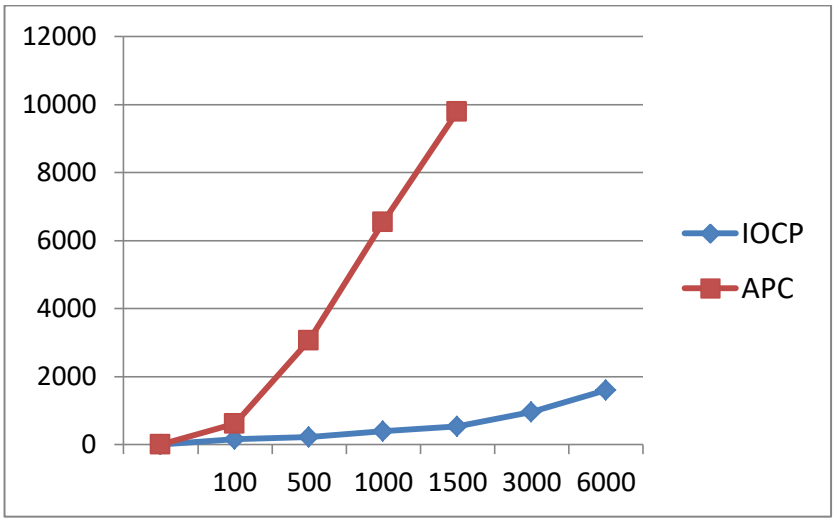


Рисунок 2.25 – Графік залежності часу від кількості файлів

2.5 Перевірка детектування підозрілої роботи I/O Completion Port наявними методами протидії ШПЗ

Для виконання перевірки наявних механізмів перевірки файлів на вміст шкідливого програмного забезпечення було обрано такі механізми як: VirusTotal, Cuckoo Sandbox, CrowdStrike. Розглянемо кожен з цих засобів трохи детальніше.

VirusTotal був заснований у 2004 році, як безкоштовний сервіс, який аналізує файли та URL-адреси на наявність шкідливого програмного забезпечення. Популярність саме цього засобу досягається через співпрацю з антивірусними компаніями, дослідниками та кінцевими користувачами. На сьогоднішній день спільнота VirusTotal налічує понад 500 000 зареєстрованих користувачів, серед яких є компанії Fortune 500, уряди та провідні охоронні компанії [21].

Cuckoo Sandbox — це автоматизована система аналізу зловмисного програмного забезпечення з відкритим кодом. Вона забезпечує безпечне

середовище для виконання підозрілих файлів і спостереження за їх поведінкою, дозволяючи аналітикам зрозуміти природу та вплив потенційних загроз. Cuckoo Sandbox використовує низку методів, таких як динамічний і статичний аналіз, щоб ідентифікувати та аналізувати зразки зловмисного програмного забезпечення. Він збирає та аналізує дані про мережевий трафік, системні виклики та дії з файлами, надаючи докладні звіти та інформацію про аналізоване шкідливе програмне забезпечення. Завдяки модульній та настроюваній архітектурі Cuckoo Sandbox є цінним інструментом для дослідників зловмисного програмного забезпечення, служб реагування на інциденти та спеціалістів із безпеки, які намагаються виявити та подолати кіберзагрози [22].

CrowdStrike — провідна компанія з кібербезпеки, яка пропонує широкий спектр хмарних рішень для захисту організацій від складних кіберзагроз. Їхня платформа поєднує передові технології, аналіз загроз та експертні послуги для забезпечення проактивних та ефективних заходів безпеки. Флагманський продукт CrowdStrike Falcon, забезпечує захист кінцевих точок наступного покоління, використовуючи машинне навчання та аналіз поведінки для виявлення та запобігання зловмисному програмному забезпеченню, програмам-вимагачам та іншим кібератакам. Платформа також пропонує можливості видимості та реагування в реальному часі, що дозволяє командам безпеки швидко розслідувати та усувати інциденти. Канали аналізу загроз CrowdStrike та служби проактивного пошуку дозволяють організаціям випереджати нові загрози та підвищувати загальну безпеку [23].

При аналізі використовувалася програма, що була розглянута раніше та вихідний код якої наведено у Додатку А.

Як було зазначено раніше, I/O Completion Port є досить специфічним механізмом ОС Windows, який взагалі використовується досить рідко. Тим паче, зловмисники почали використовувати його для пришвидшення роботи з файлами.

Поки відомо не багато прикладів його використання, але ж ніхто не застрахований від цього в майбутньому.

Сучасні методи аналізу шкідливого програмного забезпечення, що доступні для кожного з користувачів безкоштовно, не дають такої можливості детектування підозрілої поведінки файлів, що використовують I/O Completion Port. Для цього було перевірено такі системи, як VirusTotal (рисунок 2.26) та Cuckoo Sandbox (рисунок 2.27). В якості зразка було взято все той самий файл, що використовує I/O Completion Port (Додаток А). В результаті 1 з 69 антивірусних засобів детектував цей файл як підозрілий у VirusTotal (що може бути і звичайним false positive), а Cuckoo Sandbox показав оцінку 0.6 з 10, що відповідає нормі.

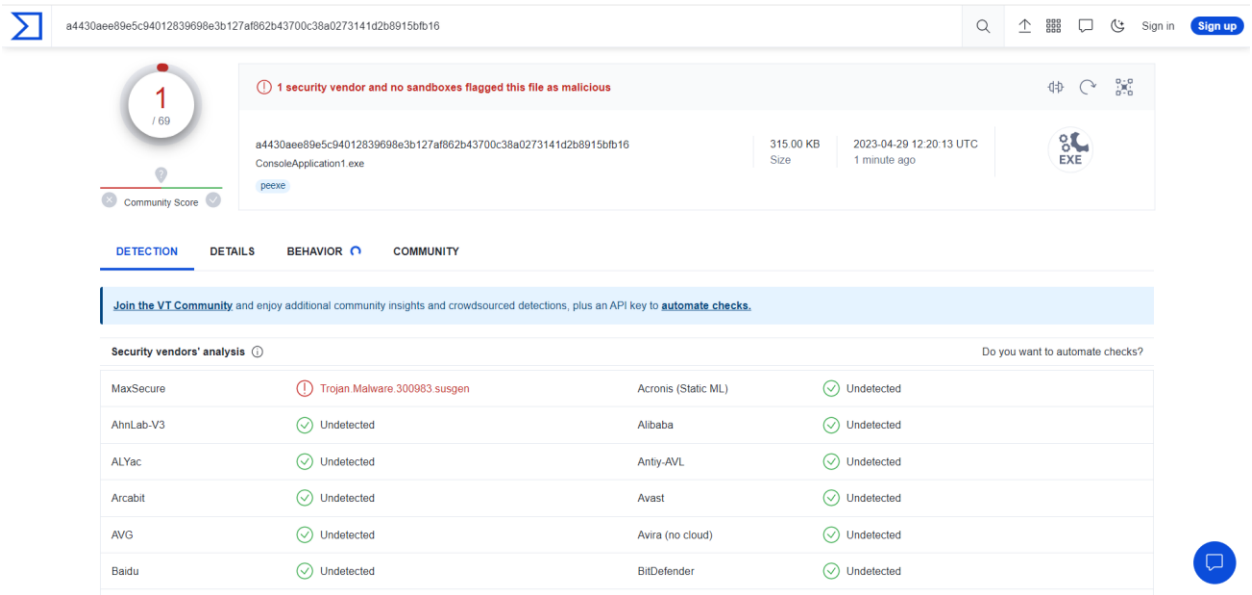


Рисунок 2.26 – Аналіз .exe файлу за допомогою VirusTotal

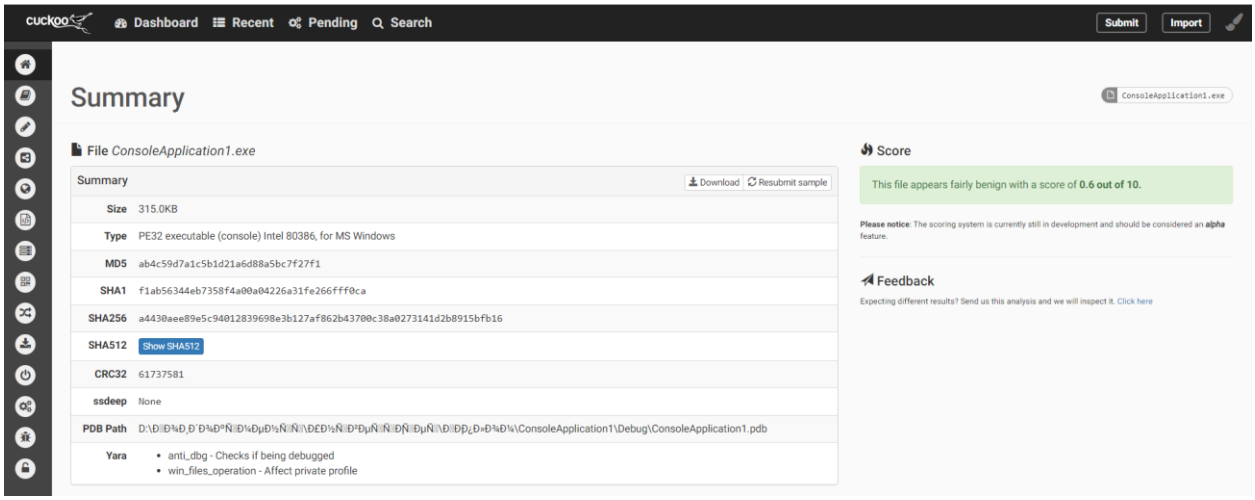


Рисунок 2.27 – Аналіз .exe файлу за допомогою Cuckoo Sandbox

Натомість велике корпоративне рішення CrowdStrike виявило програму як шкідливу і поставила оцінку (threat score) 55 зі 100 (рисунок 2.28). При цьому функції пов’язанні з використанням I/O Completion Port були занесені в секцію функцій роботи з файлами, що дає лише 1 бал з 55 отриманих. Таким чином, детектування I/O Completion Port є пріоритетною задачею, що буде розглядатися далі.

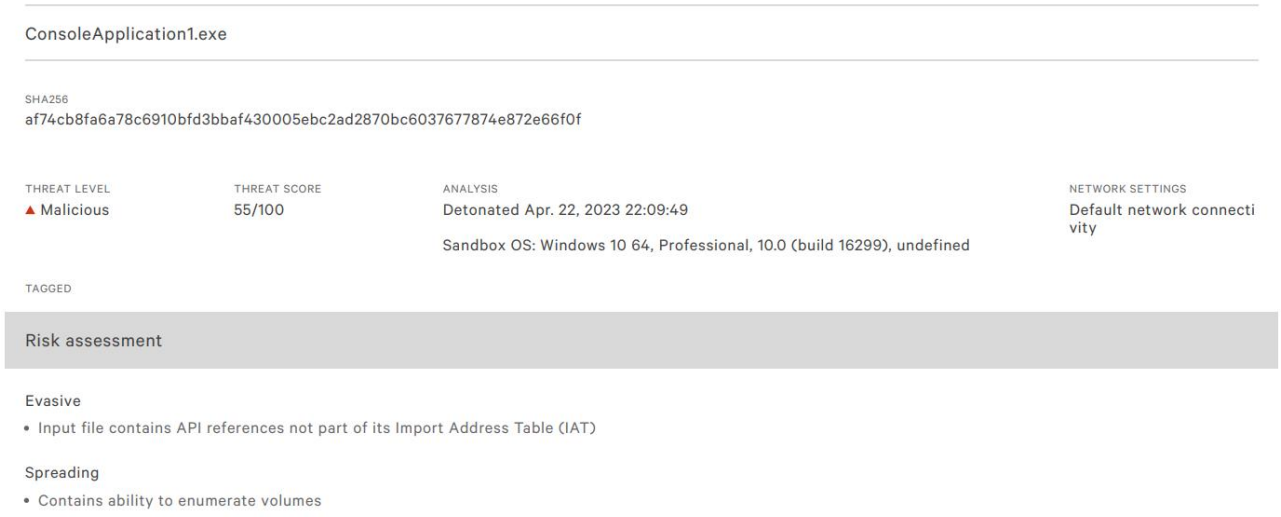


Рисунок 2.28 – Аналіз .exe файлу за допомогою CrowdStrike Falcon

Висновки до розділу 2

У другому розділі даної роботи було детально розібрано історію, принцип функціонування, розповсюдження та механізму шифрування одного з найбільш відомих RaaS Sodinokibi / REvil. Було виявлено, що при виконанні корисного навантаження даний ransomware використовує I/O Completion Port. Мовою C++ було реалізовано програми, що повторюють механізм за змістом схожий до проаналізованого ШПЗ, але без виконання шифрування файлів. При цьому одна програма використовувала I/O Completion Port, а друга – альтернативний механізм APC.

В результаті експерименту було проаналізовано мету вибору саме I/O Completion Port для реалізації необхідних задач, після чого було проведено аналіз розробленого програмного коду за допомогою трьох наявних методів детектування шкідливого програмного забезпечення (VirusTotal, Cuckoo Sandbox та CrowdStrike Falcon). Два засоби, що є доступними для кожного з користувачів та безкоштовними показали недостатню якість детектування підозрілості програми, що використовує I/O Completion Port.

3 РОЗРОБКА ПРОГРАМНОГО МЕХАНІЗМУ ІДЕНТИФІКАЦІЇ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ЩО ВИКОРИСТОВУЄ I/O COMPLETION PORT

3.1 Методи, що були покладені в основу розробки програмного механізму детектування шкідливого програмного забезпечення, що використовує I/O Completion Port.

При реалізації програмного механізму детектування виконуваних файлів, що використовують I/O Completion Port було покладено загальну ідею антивірусних засобів, що заснована на виконанні основних двох типів аналізу: статичного аналізу виконуваного файлу та динамічного аналізу.

Статичний аналіз виконуваного файлу (exe) передбачає перевірку його структури, характеристик і вмісту без його виконання. Це фундаментальна техніка, яка використовується для аналізу шкідливих програм, розробки програмного забезпечення та аудиту безпеки.

Під час статичного аналізу аналізуються різні аспекти файлу exe, наприклад інформація заголовка, таблиці імпорту та експорту, розділи ресурсів, розділи коду та вбудовані дані. Досліджуючи ці компоненти, аналітики можуть отримати уявлення про поведінку файлу, потенційні вразливості та можливі зловмисні наміри.

Натомість динамічний аналіз виконуваного файлу передбачає, що exe-файл запускається в пісочниці або на віртуальній машині, де його дії відстежуються та аналізуються. Це дозволяє аналітикам спостерігати за поведінкою файлу під час

виконання, включаючи системні виклики, мережевий трафік, файлові операції, зміни реєстру та взаємодію з іншими процесами.

Сканер реалізовано засобами мови python з використанням необхідних бібліотек, таких як: os, time, watchdog, requests, strings, json, subprocess, threading, signal та frida. Включаючи особливість використання однієї з бібліотек, додатково було реалізовано код мовою javascript.

Алгоритм сканеру, що було реалізовано, подано у вигляді activity діаграми у Додатку В.

3.1.1 Опис роботи головної програми реалізованого сканеру

Запуск реалізованого сканеру починається з виконання файлу monitor.py (Додаток Г). Суть роботи полягає в тому, що за допомогою функцій бібліотеки watchdog watchdog.observers Observer та watchdog.events FileSystemEventHandler відстежується вміст директорії Test_directory у фоновому режимі та будь-яка зміна складу директорії передається у програму. При цьому виконується перевірка розширення файлу, що передається та подію (event), яку було перехоплено. У разі детектування виконуваного файлу (розширення файлу exe) та події типу створення (create), файл передається до подальшого аналізу. Тут зазначимо, що може використовуватися будь-яка директорія. В реальному житті кращим варіантом буде моніторинг директорії завантажень.

Для реалізації вищезазначенної поведінки, було створено класс MyEventHandler як об’єкт наслідування класу FileSystemEventHandler, що містить

метод on_any_event(). Саме в цьому методі виконується моніторинг заданої дерикторії та перевірка типу файлу. Реалізація класу зображена на рисунку 3.1.

```
class MyEventHandler(FileSystemEventHandler):
    def on_any_event(self, event):
        # Event handling logic goes here
        ev_type = event.event_type
        ev_path = event.src_path
        _, file_extension = os.path.splitext(ev_path)
        print(f"Event: {ev_type} | Path: {ev_path}")
        if ev_type == "created" and file_extension == ".exe":
            print("Executable file added")
            result = static.static_analysis(ev_path)
            if result != "Exists":
                dynamic.dynamic_analysis(ev_path)
```

Рисунок 3.1 – Реалізація класу MyEventHandler

Далі програма передає шлях до виконуваного файлу спочатку на статичний, а потім і динамічний аналіз.

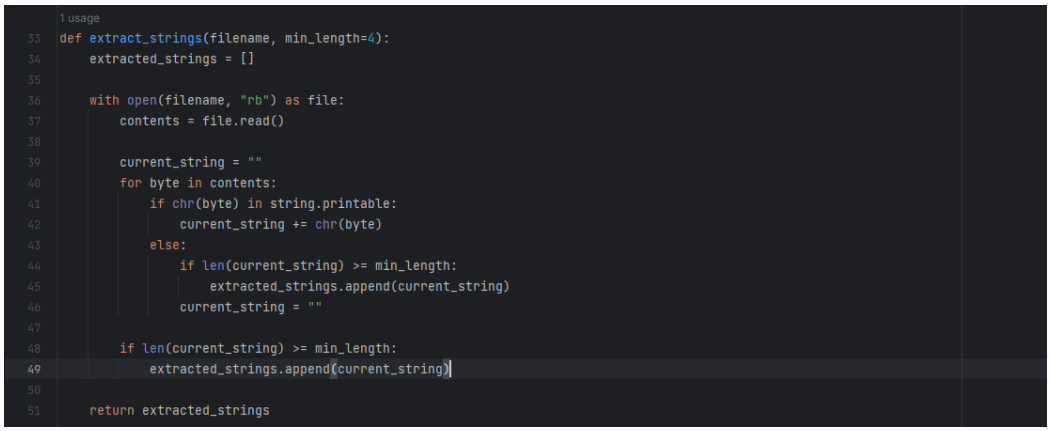
3.1.2 Статичний аналіз

Статичний аналіз виконується за допомогою виклику файлу static.py (Додаток Д). Суть статичного аналізу полягає у виконанні альтернативі команди командного рядка unix-подібних систем – strings, а також у передачі заданого файлу на аналіз VirusTotal, за допомогою VirusTotal API, ключ до якого можна отримати кожному після реєстрації.

При застосуванні до exe-файлу команда "strings" сканує файл і виводить усі знайдені розпізнані рядки символів. Це може включати звичайні текстові рядки, такі як повідомлення про помилки, дані конфігурації або назви функцій, вбудовані у файл.

Як зазначалося раніше, існує три головні функції для роботи з I/O Completion Port: GetQueuedCompletionStatus(), CreateIoCompletionPort(), PostQueuedCompletionStatus(), саме їх пошук і було реалізовано у списку рядків, що були знайдені у виконуваному файлі.

Команда “strings” властива для unix-подібних систем. Для того, щоб реалізувати відповідний функціонал засобами мови програмування Python, було написано функцію extract_strings(), що на вхід приймає шлях до виконуваного файлу, а на виході повертає список рядків знайдених у виконуваному файлі. Детальна реалізація функції зображена на рисунку 3.2.



```
1 usage
def extract_strings(filename, min_length=4):
    extracted_strings = []

    with open(filename, "rb") as file:
        contents = file.read()

        current_string = ""
        for byte in contents:
            if chr(byte) in string.printable:
                current_string += chr(byte)
            else:
                if len(current_string) >= min_length:
                    extracted_strings.append(current_string)
                current_string = ""

        if len(current_string) >= min_length:
            extracted_strings.append(current_string)

    return extracted_strings
```

Рисунок 3.2 – Функція extract_strings()

Сканування VirusTotal відбувається за допомогою надсилання запиту до VirusTotal API, що містить вміст виконуваного файлу, в результаті повертається відповідь серверу, з якої можна отримати необхідну інформацію про заданий зразок виконуваного файлу. В нашому випадку зберігається така інформація як Analysis ID (ID аналізу), Permalink (посилання на репорт, створений VirusTotal), ScanDate (дата завантаження цього зразка на VirusTotal), Positives (кількість антивірусних засобів, що вважають програму зловмисною), Total (загальна кількість антивірусних засобів, що використовувалася при скануванні).

Для цього було реалізовано 2 функції: analyze_file() та get_report(). Як можна побачити з назв, одна з них відправляє відповідний файл до VirusTotal для

виконання подальшого аналізу, а інша отримує готову інформацію про результат сканування. Їх реалізації можна подивитися на рисунках 3.3 та 3.4 відповідно.

```
1 usage
def analyze_file(api_key, file_path):
    url = 'https://www.virustotal.com/vtapi/v2/file/scan'

    params = {
        'apikey': api_key
    }

    try:
        with open(file_path, 'rb') as file:
            files = {'file': file}
            response = requests.post(url, files=files, params=params)

            if response.status_code == 200:
                result = response.json()
                return result
            else:
                print(f'Error: {response.status_code} - {response.text}')
                return None
    except IOError as e:
        print(f'Error reading the file: {e}')
        return None
```

Рисунок 3.3 – Функція analyze_file()

```
1 usage
def get_report(api_key, resource):
    url = f'https://www.virustotal.com/vtapi/v2/file/report'

    params = {
        'apikey': api_key,
        'resource': resource
    }

    try:
        response = requests.get(url, params=params)

        if response.status_code == 200:
            result = response.json()
            return result
        else:
            print(f'Error: {response.status_code} - {response.text}')
            return None
    except requests.RequestException as e:
        print(f'Error requesting the report: {e}')
        return None
```

Рисунок 3.4 – Функція get_report()

Результати статичного аналізу після виконання заносяться у базу даних, що реалізована у вигляді json файлу.

3.1.3 Динамічний аналіз

Динамічний аналіз виконується за допомогу виклику dynamic.py (Додаток Е). Він спирається на перехоплення виклику функцій у реальному часу, а саме створення та закінчення роботи з I/O Completion Port. В якості механізму, що використовується для перехоплення виклику функцій було обрано Frida.

Frida — це потужний інструмент динамічного аналізу, який широко використовується для аналізу поведінки програмних. Це дозволяє дослідникам, фахівцям із безпеки та розробникам отримати глибоке розуміння внутрішньої роботи програми, впроваджуючи код JavaScript або Python у запущений процес.

Однією з головних переваг використання Frida для динамічного аналізу є її здатність взаємодіяти з програмою під час виконання. За допомогою Frida аналітики можуть перехоплювати виклики функцій, змінювати параметри функцій і навіть змінювати поведінку програми на льоту. Цей рівень гнучкості дозволяє поглиблено досліджувати та маніпулювати виконанням програми, сприяючи виявленню вразливостей, розумінню складної поведінки програмного забезпечення та зворотній інженерії власних алгоритмів.

Кросплатформна підтримка Frida ще більше підвищує її корисність. Його можна використовувати в різних операційних системах, включаючи Windows, macOS, Linux, iOS і Android. Ця універсальність робить Frida цінним

інструментом для аналізу застосунків на різних платформах, будь то нативні, веб чи гібридні [24].

Для запуску Frida були реалізовані наступні програмні коди frida_exec.py (допоміжна програма, що створена для запуску javascript коду Frida та моніторингу часу виконання динамічного аналізу Додаток Є) та frida_script.js (Додаток Ж).

Застосування Frida засновано на перехопленні першого виклику функції CreateIOCompletionPort() та PostQueuedCompletionStatus(). При перехопленні фіксується час, для отримання інформації про проміжок використання I/O Completion Port. Зібрані дані надсилаються у вигляді повідомлення до допоміжної програми для подальшої обробки (рисунок 3.5). При цьому аналіз виконується лише 20 секунд, оскільки більший час використання в даному випадку будемо вважати небезпечним. Для того, щоб зробити аналіз більш безпечним при детектуванні функції CreateIOCompletionPort() та відсутності детектування PostQueuedCompletionStatus() на інтервалі у 20 секунд процес, що виконує програму буде миттєво знищено.

```
// frida_script.js
var firstCall = true;
var firstCallTime = 0;
var firstCall2 = true;

Interceptor.attach(Module.findExportByName(null, 'CreateIoCompletionPort'), {
  onEnter: function (args) {
    if (firstCall) {
        firstCallTime = new Date().getTime();
        firstCall = false;
        send("[+] CreateIOCompletionPort detected");
    }
  }
});

Interceptor.attach(Module.findExportByName(null, 'PostQueuedCompletionStatus'), {
  onEnter: function (args) {
    if (firstCall2) {
        var currentTime = new Date().getTime();
        var elapsedTime = currentTime - firstCallTime;
        firstCall2 = false
        send("[+] PostQueuedCompletionStatus detected.\n[+] Time of use I/O Completion Port: " + elapsedTime);
    }
  }
});
```

Рисунок 3.5 – Функція get_report()

3.1.4 Зберігання результатів аналізу

Результати сканування зберігаються у json файлі signature_db.json. Це зроблено для того, щоб пришвидшити аналіз файлів. Так виконуванні файли, що були проаналізовані раніше зберігаються у базі, включаючи їх sha256 хеш значення та результати статичного та динамічного аналізу. Якщо sha256 хеш співпадає з наявним у базі, то результат аналізу буде видано користувачу миттєво й не буде необхідності у повторному скануванні, оскільки хеш однозначно ідентифікує файл.

3.2 Демонстрація програмної реалізації детектування використання I/O Completion Port

При додаванні виконуваного файлу до директорії Test_directory, сканер детектує таку поведінку (рис. 3.6).

```
Event: created | Path: C:\Users\Syncmaster\Desktop\Test_directory\ConsoleApplication1.exe
Executable file added
```

Рисунок 3.6 – Детектування додавання виконуваного файлу у директорію, що моніториться

Результат аналізу файлу, що не використовує I/O Completion Port буде містити порожній масив функцій отриманих, за допомогою strings, а також висновок про відсутність детектування при динамічному аналізі (рис. 3.7).

Рисунок 3.7 – Робота сканеру на файлі, що не використовує I/O Completion Port

Результат аналізу файлу, що використовує I/O Completion Port буде містити масив функцій, отриманих при статичному аналізі, сканування VirusTotal, а також результат динамічного аналізу. При цьому якщо I/O Completion Port використовується менше 20 секунд, то результат буде аналогічним до рисунку 3.8, в іншому ж випадку результат буде співпадати з рисунком 3.9.

Рисунок 3.8 – Робота сканеру на файлі, що використовує I/O Completion Port менше 20 секунд

Рисунок 3.9 – Робота сканеру на файлі, що використовує I/O Completion Port більше 20 секунд

Файл бази даних при цьому буде містити інформацію про кожне з цих сканувань (рис. 3.10).

Рисунок 3.10 – Файл бази даних signature_db.json

Висновки до розділу 3

На основі отриманих у попередніх роздіілах результатів, було виявлено, що існує дуже висока необхідність детектування використання I/O Completion Port. Тому було реалізовано програмний сканер, що використовує засоби статичного та динамічного аналізу використання I/O Completion Port. При цьому тестування написаного сканеру показало його високу ефективність при різних вхідних виконуваних файлах.

ВИСНОВКИ

Таким чином, при виконанні дипломної роботи було досліджено програмні механізми виявлення атак "нульового дня" ransomware для ОС Windows, з акцентом на використанні асинхронного вводу/виводу та специфічній реалізації порту завершення вводу/виводу (I/O Completion Port). Дослідження продемонструвало ефективність використання порту завершення вводу/виводу в реалізації механізму пошуку необхідних файлів у файловій системі скомпрометованого середовища, що передує шифруванню диска жертви, на прикладі Sodinokibi/REvil. Результати експериментів підтвердили, що порт завершення вводу/виводу пропонує найбільш сприятливі часові характеристики для асинхронних операцій вводу/виводу.

Крім того, в роботі досліджено можливості виявлення використання порту завершення вводу/виводу існуючими механізмами детектування шкідливого програмного забезпечення, такими як VirusTotal, Cuckoo Sandbox та провідним рішенням EDR/XDR - CrowdStrike Falcon, при виявленні використання порту завершення вводу/виводу. Було виявлено, що наявні на сьогоднішній день механізми не виявляють наявність порту завершення вводу/виводу в виконуваних файлах.

На основі отриманих результатів було розроблено програмний механізм з використанням засобів мови Python та інструментів статичного і динамічного аналізу для виявлення програм-вимагачів, які використовують порт завершення вводу/виводу. Розроблений програмний продукт пройшов успішне тестування на кількох екземплярах виконуваних файлів, у тому числі тих, що використовують порт завершення вводу/виводу різної тривалості або не використовують його взагалі.

Отримані результати підкреслюють важливість розгляду порту завершення вводу/виводу як потенційного індикатора атак нульового дня зловмисників. Розроблений програмний механізм надає практичне рішення для виявлення таких атак, сприяючи вдосконаленню заходів кібербезпеки для систем Windows. Подальші дослідження можуть покращити можливості виявлення та розширити сферу застосування програмного механізму, щоб охопити ширший спектр потенційних варіантів програм-вимагачів нульового дня.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. "StatCounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share" [Електронний ресурс] – Режим доступу: https://gs.statcounter.com/ (дата звернення: 05.02.2023)
2. Richter, J. Windows via C/C++, Fifth Edition. [Текст] / Richter, J., Nasarre, C. - Pearson Education, 2012. – 739 с.
3. Microsoft. (n.d.). WinObj [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/sysinternals/downloads/winobj (дата звернення: 06.02.2023)
4. Microsoft. (2021). "Kernel objects". [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/windows/win32/sysinfo/kernel-objects (дата звернення: 06.02.2023)
5. Microsoft Developer Network. (n.d.). Windows API: OVERLAPPED structure. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/windows/win32/api/minwinbase/ns-minwinbase-overlapped (дата звернення: 12.02.2023)
6. Microsoft Developer Network. (n.d.). Synchronous and Asynchronous I/O. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/windows/win32/fileio/synchronous-and-asynchronous-i-o (дата звернення: 12.02.2023)
7. Mark Russinovich Windows Internals, 7th Edition. [Текст] / Mark Russinovich, David A. Solomon, Alex Ionescu. - Microsoft Press, 2012. – 1120 с.
8. Hart, J. M. Windows System Programming. Publisher. [Текст] / Hart, J. M. - Publisher. 2005. – 647 с.
9. Cloudflare. (2021). Ransomware as a Service (RaaS). [Електронний ресурс] – Режим доступу: https://www.cloudflare.com/learning/security/ransomware/ransomware-as-a-service/ (дата звернення: 15.03.2023)

10. Microsoft. (2022, May 9). Ransomware-as-a-Service: Understanding the cybercrime gig economy and how to protect yourself. [Електронний ресурс] – Режим доступу: https://www.microsoft.com/en-us/security/blog/2022/05/09/ransomware-as-a-service-understanding-the-cybercrime-gig-economy-and-how-to-protect-yourself/ (дата звернення: 15.03.2023)
11. Cybereason. (2021, March 21). Revil Ransomware Gang Hit Acer with $50M Ransom Demand. [Електронний ресурс] – Режим доступу: https://www.cybereason.com/blog/sodinokibi/revil-ransomware-gang-hit-acer-with-50m-ransom-demand (дата звернення: 04.04.2023)
12. Bleeping Computer. (2021, June 3). FBI: REvil cybergang behind the JBS ransomware attack. [Електронний ресурс] – Режим доступу: https://www.bleepingcomputer.com/news/security/fbi-revil-cybergang-behind-the-jbs-ransomware-attack/ (дата звернення: 04.04.2023)
13. Europol. (2021, July 15). Five affiliates to Sodinokibi/REvil “unplugged”. [Електронний ресурс] – Режим доступу: https://www.europol.europa.eu/media-press/newsroom/news/five-affiliates-to-sodinokibirevil-unplugged (дата звернення: 04.04.2023)
14. U.S. Department of State. (2021, August 19). Sodinokibi Ransomware-as-a-Service (RaaS). Transnational Organized Crime Rewards Program. [Електронний ресурс] – Режим доступу: https://www.state.gov/transnational-organized-crime-rewards-program-2/sodinokibi-ransomware-as-a-service-raas/ (дата звернення: 04.04.2023)
15. BleepingComputer. (2021, September 6). REvil/Sodinokibi ransomware targets Chinese users with DHL spam. [Електронний ресурс] – Режим доступу: https://www.bleepingcomputer.com/news/security/revil-sodinokibi-ransomware-targets-chinese-users-with-dhl-spam/ (дата звернення: 19.02.2023)
16. Amossys. (2021, June 16). Sodinokibi malware analysis. [Електронний ресурс] – Режим доступу: https://www.amossys.fr/fr/ressources/blog-technique/sodinokibi-malware-analysis/ (дата звернення: 05.03.2023)

17. Polymorf. (2021). FindCrypt-yara. GitHub repository. [Електронний ресурс] – Режим доступу: https://github.com/polymorf/findcrypt-yara (дата звернення: 05.03.2023)
18. Secureworks. (2019, August 6). REVIL/Sodinokibi Ransomware. [Електронний ресурс] – Режим доступу: https://www.secureworks.com/research/revil-sodinokibi-ransomware (дата звернення: 05.03.2023)
19. Microsoft. (2021). QueueUserAPC function. Windows Dev Center. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-queueuserapc (дата звернення: 15.04.2023)
20. Використання механізмів асинхронного вводу/виводу при реалізації атак нульового дня ОС Windows/ Д. В. Тислицький, Л. Ю. Гальчинський // Теоретичні і прикладні проблеми фізики, математики та інформатики: матеріали XXI Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених / Д. В. Тислицький, Л. Ю. Гальчинський. – Київ, 2023. – (Видавництво «Політехніка»). – С. 313-317.
21. VirusTotal. About us. [Електронний ресурс] – Режим доступу: https://support.virustotal.com/hc/en-us/categories/360000160117-About-us (дата звернення: 19.04.2023)
22. CERT Estonia. Cuckoo Sandbox. [Електронний ресурс] – Режим доступу: https://cuckoo.cert.ee/ (дата звернення: 19.04.2023)
23. CrowdStrike. About us. [Електронний ресурс] – Режим доступу: https://www.crowdstrike.com/about-us/ (дата звернення 22.04.2023)
24. Frida. [Електронний ресурс] – Режим доступу: https://frida.re/ (дата звернення 05.05.2023)

Додаток А
Реалізація механізму перебору файлової системи за допомогою I/O Completion Port

```
#include <iostream>
#include <filesystem>
#include <vector>
#include <fstream>
#include <thread>
#include <Windows.h>
#include <string>
#include <mutex>

using namespace std::filesystem;

const std::wstring whitelist_filename = L"whitelist_files.txt"; // ім'я файлу білого списку
std::mutex data_mutex;

struct FileInfo {
    std::wstring filename;
    DWORD size;
};

struct OverlappedPlus {
    OVERLAPPED overlapped;
    FileInfo* data;
};

std::vector<std::wstring> read_lines(const std::wstring& filename) {
    std::vector<std::wstring> lines;

    std::wifstream file(filename);
    if (!file) {
        // handle file opening error
        std::wcerr << "Error: failed to open file " << filename << std::endl;
        return lines;
    }

    std::wstring line;
    while (std::getline(file, line)) {
        lines.push_back(line);
```

```
    }

    return lines;
}

void process_file(const std::wstring& filename) {
    std::wofstream file;
    file.open("example.txt", std::ios_base::app); // open file in append mode
    if (file.is_open()) {
        file << filename << L"\n"; // write wstring to file
        file.close();
        std::wcout << L"Successfully wrote to file." << std::endl;
    }
    else {
        std::wcerr << L"Failed to open file." << std::endl;
    }
    std::wcout << filename << std::endl;
}

void thread_proc(HANDLE iocp) {
    DWORD bytes_transferred = 0;
    ULONG_PTR completion_key = 0;
    LPOVERLAPPED overlapped = NULL;

    while (true) {
        std::unique_lock<std::mutex> lock(data_mutex);
        BOOL result = GetQueuedCompletionStatus(
            iocp,
            &bytes_transferred,
            &completion_key,
            &overlapped,
            INFINITE);

        if (!result) {
            std::cerr << "Error: GetQueuedCompletionStatus failed with error code
" << GetLastError() << std::endl;
            continue;
            //lock.unlock();
        }
        else {
            //lock.unlock();
            if (overlapped == NULL) {
```

```
                std::cout << "Thread " << std::this_thread::get_id() << " is
exiting" << std::endl;
                //lock.unlock();
                break;
            }

            // обробка завершення I/O

            // отримати інформацію про файл
            OverlappedPlus* overlapped_plus = (OverlappedPlus*)overlapped;
            FileInfo* file_info = overlapped_plus->data;

            std::wcout << L"Processing file " << file_info->filename << L"..." <<
std::endl;

            // обробка файлу
            process_file(file_info->filename);
            delete file_info;
            delete overlapped_plus;
            lock.unlock();
        }
    }
}

void scan_directory(const std::wstring& directory_name, HANDLE iocp, const
std::vector<std::wstring>& whitelist) {
    for (const auto& entry : recursive_directory_iterator(directory_name)) {
        if (entry.is_regular_file()) {
            // перевірка файлу проти білого списку
            std::wstring filename = entry.path().filename();
            bool is_whitelisted = std::find(whitelist.begin(), whitelist.end(),
filename) != whitelist.end();

            if (!is_whitelisted) {
                // створити структуру FileInfo для передачі інформації про файл
                FileInfo* file_info = new FileInfo();
                file_info->filename = entry.path();
                file_info->size = entry.file_size();

                // створити структуру OverlappedPlus для передачі FileInfo в I/O
completion port
                OverlappedPlus* overlapped_plus = new OverlappedPlus();
```

```
                ZeroMemory(&overlapped_plus->overlapped, sizeof(OVERLAPPED));
                overlapped_plus->data = file_info;

                // додати I/O запит до I/O completion port
                HANDLE file_handle = CreateFileW(
                    file_info->filename.c_str(),
                    GENERIC_READ,
                    FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE,
                    NULL,
                    OPEN_EXISTING,
                    FILE_FLAG_OVERLAPPED,
                    NULL);

                if (file_handle == INVALID_HANDLE_VALUE) {
                    std::cerr << "Error: CreateFile failed with error code " <<
GetLastError() << std::endl;
                    delete file_info;

                    delete overlapped_plus;
                    continue;
                }

                if (!CreateIoCompletionPort(file_handle, iocp,
(ULONG_PTR)file_handle, 0)) {
                    std::cerr << "Error: CreateIoCompletionPort failed with error
code " << GetLastError() << std::endl;
                }
                char buffer[1024];
                ReadFile(file_handle, buffer, sizeof(buffer), NULL,
&overlapped_plus->overlapped);
                //std::wcout << filename << std::endl;
                CloseHandle(file_handle);
            }
        }
    }
}

int main() {
    std::vector<std::wstring> whitelist = read_lines(whitelist_filename);

    // створення I/O completion port та потоків
    HANDLE iocp = CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, 0, 0);
```

```
    DWORD start = GetTickCount64();
    if (iocp == NULL) {
        std::cerr << "Error: CreateIoCompletionPort failed with error code " <<
GetLastError() << std::endl;
        return 1;
    }
    const int num_threads = std::thread::hardware_concurrency();
    std::vector<std::thread> threads;
    for (int i = 0; i < num_threads; i++) {
        threads.emplace_back(thread_proc, iocp);
    }

    // сканування директорії
    scan_directory(L"D:\\3000_files", iocp, whitelist);

    // закінчення роботи потоків
    for (int i = 0; i < num_threads; i++) {
        PostQueuedCompletionStatus(iocp, 0, NULL, NULL);
    }
    for (auto& thread : threads) {
        thread.join();
    }

    CloseHandle(iocp);
    DWORD end = GetTickCount64();

    DWORD duration = end - start;
    std::cout << "Execution time " << duration << " miliseconds" << std::endl;

    system("pause");
    return 0;
}
```

Додаток Б

Реалізація механізму перебору файлової системи за допомогою APC

```
#include <iostream>
#include <filesystem>
#include <vector>
#include <fstream>
#include <Windows.h>
#include <string>

using namespace std::filesystem;

const std::wstring whitelist_filename = L"whitelist_files.txt"; // ім'я файлу білого списку

std::vector<std::wstring> read_lines(const std::wstring& filename) {
    std::vector<std::wstring> lines;

    std::wifstream file(filename);
    if (!file) {
        // handle file opening error
        std::wcerr << "Error: failed to open file " << filename << std::endl;
        return lines;
    }

    std::wstring line;
    while (std::getline(file, line)) {
        lines.push_back(line);
    }

    return lines;
}

void CALLBACK process_file(ULONG_PTR param) {
    std::wstring filename = std::to_wstring(param);
    std::wofstream file;
    file.open("example.txt", std::ios_base::app); // open file in append mode
    if (file.is_open()) {
        file << filename << L"\n"; // write wstring to file
        file.close();
        std::wcout << L"Successfully wrote to file." << std::endl;
    }
    else {
```

```
        std::wcerr << L"Failed to open file." << std::endl;
    }
    std::wcout << filename << std::endl;
}

void scan_directory(const std::wstring& directory_name, const
std::vector<std::wstring>& whitelist) {
    for (const auto& entry : recursive_directory_iterator(directory_name)) {
        if (entry.is_regular_file()) {
            // перевірка файлу проти білого списку
            std::wstring filename = entry.path().filename();
            bool is_whitelisted = std::find(whitelist.begin(), whitelist.end(),
filename) != whitelist.end();

            if (!is_whitelisted) {
                // додати APC до потоку
                QueueUserAPC((PAPCFUNC)process_file, GetCurrentThread(),
(ULONG_PTR)filename.c_str());
            }
        }
    }
}

int main() {
    std::vector<std::wstring> whitelist = read_lines(whitelist_filename);

    DWORD start = GetTickCount64();

    // сканування директорії та виклик APC для кожного файлу
    scan_directory(L"D:\\3000_files", whitelist);

    SleepEx(INFINITE, TRUE);

    DWORD end = GetTickCount64();

    DWORD duration = end - start;
    std::cout << "Execution time " << duration << " miliseconds" << std::endl;

    return 0;
}
```

Додаток В

Алгоритм роботи програмного механізму ідентифікації використання I/O Completion Port

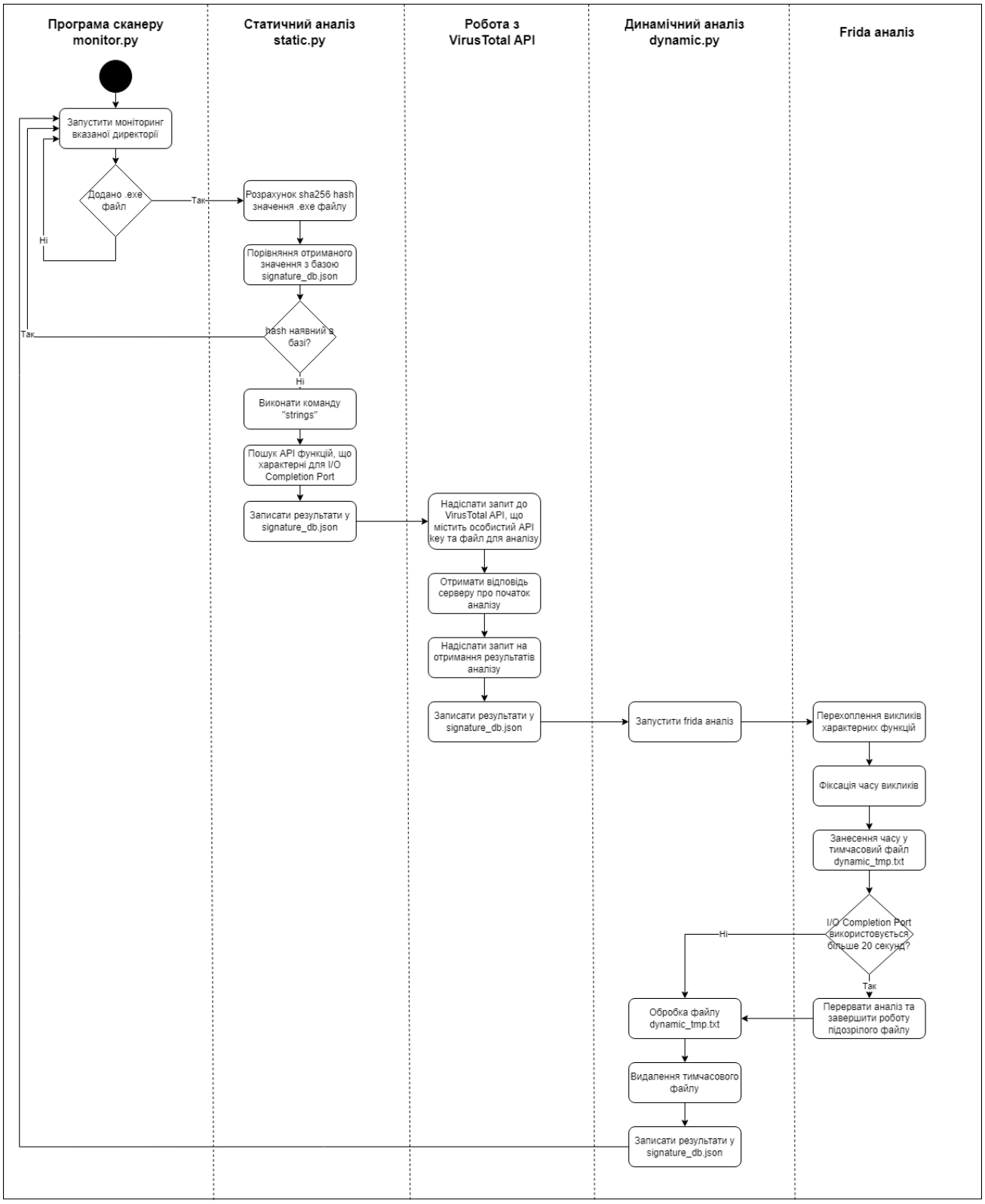


Рисунок В.1 Алгоритм роботи програмного механізму ідентифікації використання I/O Completion Port

Додаток Г

Програмна реалізація механізму ідентифікації використання I/O Completion Port у виконуваних файлах. Головний файл monitor.py.

```
import time
import os
import dynamic
import static
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler


class MyEventHandler(FileSystemEventHandler):
    def on_any_event(self, event):
        # Event handling logic goes here
        ev_type = event.event_type
        ev_path = event.src_path
        _, file_extension = os.path.splitext(ev_path)
        print(f"Event: {ev_type} | Path: {ev_path}")
        if ev_type == "created" and file_extension == ".exe":
            print("Executable file added")
            result = static.static_analysis(ev_path)
            if result != "Exists":
                dynamic.dynamic_analysis(ev_path)


# Directory path to monitor
directory = "C:\\Users\\Syncmaster\\Desktop\\Test_directory"

# Create an event handler instance
event_handler = MyEventHandler()

# Create an observer instance
observer = Observer()
observer.schedule(event_handler, directory, recursive=True)

# Start the observer
observer.start()

try:
    while True:
        time.sleep(1)  # Sleep to allow the observer to run in the background
```

```
except KeyboardInterrupt:
    observer.stop()

observer.join()
```

Додаток Д

Програмна реалізація механізму ідентифікації використання I/O Completion Port у виконуваних файлах. Статичний аналіз static.py.

```
import string
import requests
import hashlib
import json
import os


api_key = 'd0015704b7b584925d491d067b88b9afc8825e3fc75d90b985502177efd8821d'
json_path = "signature_db.json"


def read_from_json(file_path):
    with open(file_path, "r") as file:
        data = json.load(file)
        return data


def write_to_json(file_path, json_objects):
    with open(file_path, "w") as file:
        json.dump(json_objects, file)


def sha256_hash_calculation(file_path):
    sha256_hash = hashlib.sha256()

    with open(file_path, 'rb') as file:
        for chunk in iter(lambda: file.read(4096), b''):
            sha256_hash.update(chunk)

    return sha256_hash.hexdigest()


def extract_strings(filename, min_length=4):
    extracted_strings = []

    with open(filename, "rb") as file:
        contents = file.read()
```

```
        current_string = ""
        for byte in contents:
            if chr(byte) in string.printable:
                current_string += chr(byte)
            else:
                if len(current_string) >= min_length:
                    extracted_strings.append(current_string)
                current_string = ""

        if len(current_string) >= min_length:
            extracted_strings.append(current_string)

    return extracted_strings


def analyze_file(api_key, file_path):
    url = 'https://www.virustotal.com/vtapi/v2/file/scan'

    params = {
        'apikey': api_key
    }

    try:
        with open(file_path, 'rb') as file:
            files = {'file': file}
            response = requests.post(url, files=files, params=params)

            if response.status_code == 200:
                result = response.json()
                return result
            else:
                print(f'Error: {response.status_code} - {response.text}')
                return None
    except IOError as e:
        print(f'Error reading the file: {e}')
        return None


def get_report(api_key, resource):
    url = f'https://www.virustotal.com/vtapi/v2/file/report'

    params = {
```

```
        'apikey': api_key,
        'resource': resource
    }

    try:
        response = requests.get(url, params=params)

        if response.status_code == 200:
            result = response.json()
            return result
        else:
            print(f'Error: {response.status_code} - {response.text}')
            return None
    except requests.RequestException as e:
        print(f'Error requesting the report: {e}')
        return None


def static_analysis(file_path):
    exe_sha256_hash = sha256_hash_calculation(file_path)
    signatures = []

    if os.path.exists(json_path):
        signatures = read_from_json(json_path)
        for item in signatures:
            if item["SHA256 file hash"] == exe_sha256_hash:
                formatted_json = json.dumps(item, indent=4)
                print(formatted_json)
                return "Exists"

    strings = extract_strings(file_path)

    json_result = {"File": file_path, "SHA256 file hash": exe_sha256_hash, "Static
Analysis": {"Strings": []}, "VirusTotal": {"Analysis ID": "", "Permalink": "",
"Scan date": "", "Positives": "", "Total": ""}}

    for str in strings:
        if "CreateIoCompletionPort" in str and "CreateIoCompletionPort found" not
in json_result["Static Analysis"]["Strings"]:
            json_result["Static
Analysis"]["Strings"].append("CreateIoCompletionPort found")
        elif "GetQueuedCompletionStatus" in str and "GetQueuedCompletionStatus
```

```
found" not in json_result["Static Analysis"]["Strings"]:
            json_result["Static
Analysis"]["Strings"].append("GetQueuedCompletionStatus found")
        elif "PostQueuedCompletionStatus" in str and "PostQueuedCompletionStatus
found" not in json_result["Static Analysis"]["Strings"]:
            json_result["Static
Analysis"]["Strings"].append("PostQueuedCompletionStatus found")

    analysis_result = analyze_file(api_key, file_path)

    if analysis_result:
        json_result["VirusTotal"]["Analysis ID"] = analysis_result["scan_id"]
        json_result["VirusTotal"]["Permalink"] = analysis_result["permalink"]
    else:
        print('File analysis failed.')

    resource = analysis_result["scan_id"]
    report = get_report(api_key, resource)

    if report:
        json_result["VirusTotal"]["Scan date"] = report["scan_date"]
        json_result["VirusTotal"]["Positives"] = report["positives"]
        json_result["VirusTotal"]["Total"] = report["total"]
    else:
        print('Report retrieval failed.')

    signatures.append(json_result)
    write_to_json(json_path, signatures)
```

Додаток Е

Програмна реалізація механізму ідентифікації використання I/O Completion Port у виконуваних файлах. Динамічний аналіз dynamic.py.

```
import static
import subprocess
import os


def dynamic_analysis(ev_path):
    # Specify the command to run the Python script with command-line arguments
    command = ["python", "frida_exex.py", ev_path]

    # Execute the command
    subprocess.run(command, shell=True)

    new_lines = []

    if os.path.exists("dynamic_tmp.txt"):
        with open("dynamic_tmp.txt", "r") as f:
            lines = f.readlines()

        for line in lines:
            line = line.replace("\n", "")
            new_lines.append(line)

    json_db = static.read_from_json(static.json_path)
    exe_sha256_hash = static.sha256_hash_calculation(ev_path)

    if len(new_lines) != 0:
        if '[+] PostQueuedCompletionStatus detected.' not in new_lines:
            new_lines.append('[-] PostQueuedCompletionStatus not detected.')
            new_lines.append('[-] Time of usage I/O Completion Port more than 20
seconds')

        for item in json_db:
            if item["SHA256 file hash"] == exe_sha256_hash:
                item["Dynamic Analysis"] = new_lines

                formatted_json = static.json.dumps(item, indent=4)
                print(formatted_json)
    else:
```

```
        for item in json_db:
            if item["SHA256 file hash"] == exe_sha256_hash:
                item["Dynamic Analysis"] = "Can't identify I/O Completion Port
Usage"

                formatted_json = static.json.dumps(item, indent=4)
                print(formatted_json)
    static.write_to_json(static.json_path, json_db)

    if os.path.exists("dynamic_tmp.txt"):
        os.remove("dynamic_tmp.txt")
```

Додаток Є

Програмна реалізація механізму ідентифікації використання I/O Completion Port у виконуваних файлах. Запуск аналізу за допомогою Frida: frida_exec.py.

```
import frida
import sys
import time
import threading
import os
import signal


def on_message(message, data):
    with open('dynamic_tmp.txt', 'a') as file:  # Open file in append mode
        file.write(message['payload'] + '\n')  # Write the message payload to the
file


def analise_file(exe_path):
    pid = frida.spawn([exe_path])
    session = frida.attach(pid)

    with open('frida_script.js', 'r') as file:
        script_code = file.read()

    script = session.create_script(script_code)
    script.on('message', on_message)

    frida.resume(pid)

    script.load()

    def return_to_main():
        time.sleep(20)
        os.kill(pid, 9)
        os.kill(os.getpid(), signal.SIGINT)

    ctrl_d_thread = threading.Thread(target=return_to_main())
    ctrl_d_thread.start()

    sys.stdin.read()
```

```
# Specify the process name
file_path = sys.argv[1]
analise_file(file_path)
```

Додаток Ж

Програмна реалізація механізму ідентифікації використання I/O Completion Port у виконуваних файлах. Скріпт Frida: frida_script.js

```
// frida_script.js
var firstCall = true;
var firstCallTime = 0;
var firstCall2 = true;

Interceptor.attach(Module.findExportByName(null, 'CreateIoCompletionPort'), {
  onEnter: function (args) {
    if (firstCall) {
        firstCallTime = new Date().getTime();
        firstCall = false;
        send("[+] CreateIOCompletionPort detected");
    }
  }
});

Interceptor.attach(Module.findExportByName(null, 'PostQueuedCompletionStatus'), {
  onEnter: function (args) {
    if (firstCall2) {
        var currentTime = new Date().getTime();
        var elapsedTime = currentTime - firstCallTime;
        firstCall2 = false
        send("[+] PostQueuedCompletionStatus detected.\n[+] Time of use I/O
Completion Port: " + elapsedTime);
    }
  }
});
```