МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кваліфікаційна наукова
праця на правах рукопису

ЖУРАВЧАК ДАНИЇЛ ЮРІЙОВИЧ

УДК 004.056.53

ДИСЕРТАЦІЯ

УДОСКОНАЛЕННЯ МЕТОДІВ ВИЯВЛЕННЯ ПРОГРАМ-ВИМАГАЧІВ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ

125 Кібербезпека

12 “Інформаційні технологій”

Подається на здобуття наукового ступеня доктора філософії
Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело _____ /Журавчак Даниїл Юрійович/

Науковий керівник: Дудикевич Валерій Богданович, д.т.н, професор

Львів – 2024

АНОТАЦІЯ

Журавчак Д.Ю. **Удосконалення методів виявлення програм-вимагачів в режимі реального часу.** – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 125 “Кібербезпека”. –Національний університет “Львівська політехніка” України, Львів, 2024.

Зміст анотації. Ця дисертаційна робота присвячена розв'язанню актуального науково-практичного завдання покращення методів виявлення програм-вимагачів в режимі реального часу. Це охоплює розробку нових стратегій та інструментів, які здатні виявляти та нейтралізувати програми-вимагачі негайно після їх активації на операційних системах у комп’ютерних мережах. Це значно підвищить швидкість реагування на інциденти кібербезпеки, зменшить потенційні матеріальні та репутаційні втрати від атак програм-вимагачів та покращить загальну стійкість інформаційних систем приватних та державних структур. Крім того, ця робота також спрямована на покращення розуміння природи програм-вимагачів та розробку ефективних стратегій їх протидії.

Метою даної дисертаційної роботи є підвищення ефективності захисту комп’ютерних мереж та операційних систем від вірусів-вимагачів за допомогою моделей машинного навчання, які працюють у режимі виявлення в реальному часі, чи близькому до нього. Під час дисертаційного дослідження було виявлено, що традиційні методи виявлення та захисту комп’ютерних мереж не відповідають потребам сьогодення. Тому було запропоновано та розроблено модулі виявлення та моніторингу на базі фільтру eBPF, результат роботи яких у форматі журналів подій нормалізувався та готувався для застосуваннях у математичних моделях машинного навчання.

eBPF, як сучасне та потужне технологічне рішення, може надати необхідні інструменти для розробки методів, особливо у контексті систем реагування на інциденти безпеки в режимі реального часу. Таким чином, ця дисертація прагне

збагатити розуміння та практичні навички у застосуванні eBPF для боротьби з програмами-вимагачами, із особливим наголосом на реальному часі виявлення та реагування.

У цій роботі представлено різні методології використання eBPF для удосконалення виявлення програм-вимагачів в режимі реального часу. У світі, де кіберзлочинність неперервно еволюціонує, а методи атак стають все складнішими, захист комп'ютерних мереж стає більш важливим та складним завданням.

Об'єктом дослідження є програми-вимагачі та системи захисту від них в режимі реального часу. Це включає аналіз поведінки таких програм, їх вплив на системи, в яких вони працюють, і технології захисту, що використовуються для виявлення та нейтралізації цих загроз.

Предмет дослідження: Методи та засоби виявлення програм-вимагачів на основі eBPF та машинного навчання в режимі реального часу. Зокрема, розглядаються аспекти, пов'язані з розробкою та оптимізацією нових алгоритмів та моделей для більш ефективного виявлення загроз типу вірусів-вимагачів.

Викладені в роботі висновки та пропозиції мають і теоретичне і прикладне значення. Наведені у дисертації положення можуть бути використані у: науково-дослідницькій діяльності – для розвитку дослідження об’єкту та покращення розробленої системи; практичній діяльності – для оптимізації захисту об’єкту для якого буде розроблено програмне забезпечення, яке базується на викладеній в дисертації методології.

Перший розділ "**Аналіз проблеми виявлення та протидії програмам-вимагачам"** присвячений всебічному аналізу вірусів-вимагачів. У ньому розглянуто класифікацію, історію розвитку, методи розповсюдження та вплив на інформаційні системи цих шкідливих програм. Детально досліджено структуру та функціональні особливості систем SIEM (Security Information and Event Management) та EDR (Endpoint Detection and Response), а також основні моделі та методи для виявлення та нейтралізації загроз, пов'язаних з програмами-вимагачами.

Включено аналітичний огляд сучасного стану кіберзлочинності з акцентом на вплив вірусів-вимагачів, із особливою увагою до тенденцій та потенційних загроз, які вони становлять. У цьому розділі також обґрунтовується вибір напряму дослідження та постановка завдань дисертації.

Другий розділ **"Використання eBPF для виявлення програм-вимагачів",** здійснює глибокий аналіз технології eBPF (Extended Berkeley Packet Filter) та її можливостей у контексті боротьби з програмами-вимагачами. У цьому розділі розглянуто архітектуру eBPF, її ключові можливості та потенціал для ефективного виявлення та нейтралізації загроз. Особлива увага приділена інтеграції алгоритмів машинного навчання з eBPF для підвищення ефективності виявлення зловмисних дій. Розділ завершується формулюванням комплексного підходу до використання eBPF для виявлення програм-вимагачів в режимі реального часу, включаючи детальний розгляд методології розробки та імплементації модулів на основі eBPF для моніторингу системних викликів, файлової та мережевої активності. Окремий підрозділ присвячений використанню eBPF у середовищі Windows, де описано підходи до інтеграції eBPF з операційною системою Windows для моніторингу та аналізу загроз у режимі реального часу.

Третій розділ **"Створення інтегрованого методу аналізу вірусів-вимагачів на базі моделей машинного навчання",** присвячений розробці та впровадженню ефективних методів аналізу даних для виявлення програм-вимагачів, зосереджуючись на використанні моделей машинного навчання. У цьому розділі детально розглянуто архітектуру інтегрованої системи виявлення вірусів-вимагачів, включаючи всі етапи від збору даних до їх обробки та аналізу.

Спочатку розглянуто процес формування наборів даних для тренування моделей машинного навчання, зокрема методи збору, нормалізації та анотації даних, які є критично важливими для точного виявлення загроз. Особлива увага приділена вибору відповідних моделей машинного навчання, таких як дерева

рішень, методи опорних векторів, нейронні мережі та глибоке навчання, кожна з яких має свої переваги та недоліки у контексті виявлення програм-вимагачів.

Розділ також описує розробку модельного конвеєра, включаючи етапи попередньої обробки даних, вибір та налаштування моделей, тренування та тестування. Значна увага приділена метрикам оцінки ефективності моделей, таким як точність, повнота, F1-міра та інші, які використовуються для оцінки якості моделей у виявленні кіберзагроз.

Особливий акцент зроблено на інтеграції моделей машинного навчання з технологією eBPF для підвищення ефективності та швидкості виявлення програм-вимагачів у реальному часі. Це включає використання eBPF для збору системних даних, які потім використовуються для тренування та тестування моделей машинного навчання. Розділ завершується аналізом результатів та обговоренням потенційних шляхів подальшого вдосконалення інтегрованої системи, зокрема, оптимізацією алгоритмів та розширенням функціональних можливостей.

Таким чином, третій розділ забезпечує всебічне розуміння процесу створення та впровадження інтегрованого методу аналізу вірусів-вимагачів на базі моделей машинного навчання, що є ключовим елементом для підвищення ефективності захисту комп'ютерних систем від сучасних кіберзагроз.

Четвертий розділ **"Аналіз ефективності запропонованих рішень"**, присвячений детальному аналізу та оцінці ефективності розроблених методів виявлення та протидії програмам-вимагачам. У цьому розділі описано процес організації тестового середовища для проведення безпечних імітацій атак програм-вимагачів, що дозволяє здійснювати ретельний збір та аналіз даних у контрольованих умовах.

На початку розділу обговорюється структура та налаштування тестового середовища, яке включає різні типи програм-вимагачів та сценарії їхніх атак. Особлива увага приділяється методам збору даних під час тестових атак, зокрема використанню eBPF для моніторингу системних викликів, файлової активності та

мережевого трафіку. Це дозволяє отримати високоякісні дані для подальшого аналізу ефективності запропонованих методів.

Далі розглядаються результати експериментів, які демонструють ефективність моделей машинного навчання у виявленні програм-вимагачів. Аналізуються ключові метрики, такі як точність, повнота, F1-міра та Коефіцієнт Кореляції Метью (MCC), що дозволяють оцінити якість роботи моделей у різних сценаріях атак. Особлива увага приділяється порівнянню результатів запропонованих методів з традиційними підходами до виявлення вірусів-вимагачів, що дозволяє чітко визначити переваги та недоліки кожного підходу.

Розділ також містить аналіз споживання системних ресурсів розробленими методами, що є важливим аспектом для їхньої практичної реалізації у реальних системах. Оцінюється продуктивність моделей у різних умовах, включаючи навантаження на процесор, пам'ять та мережеві ресурси, що дозволяє забезпечити баланс між ефективністю виявлення та оптимальним використанням ресурсів.

На завершення розділу наведено рекомендації щодо впровадження розроблених методів у практичні комп'ютерні системи. Обговорюються можливості інтеграції запропонованих рішень з існуючими системами безпеки, такими як SIEM та EDR, що дозволить значно підвищити рівень захисту від програм-вимагачів. Окрім цього, пропонуються шляхи подальшого вдосконалення методів на основі результатів проведених експериментів та аналізу.

У **висновках** дисертаційної роботи викладено основні результати і рекомендації застосування створеної інтегрованої системи виявлення вірусів-вимагачів у режимі реального часу, а також представлено та охарактеризовано кількісні та якісні метрики ефективності запропонованого рішення.

У **додатках** до дисертації долучено програмні коди реалізації модулів eBPF та моделей машинного навчання, акти впровадження результатів дисертаційної роботи, а також список наукових праць і апробацій автора за темою дисертації.

Ключові слова: віруси-вимагачі, ebpf, siem, edr, моніторинг безпеки, кібербезпека, антивірус, машинне навчання, кіберзлочин, кіберінцидент, комп'ютерні мережі.

Список публікацій здобувача:

Наукові праці, в яких опубліковано наукові результати дисертації:

1. Zhuravchak, D. "СТВОРЕННЯ СИСТЕМИ ЗАПОБІГАННЯ ПОШИРЕННЯ ВІРУСІВ ВИМАГАЧІВ ЗА ДОПОМОГОЮ МОВИ ПРОГРАМУВАННЯ PYTHON ТА УТИЛІТИ AUDITD НА БАЗІ ОПЕРАЦІЙНОЇ СИСТЕМИ LINUX". Електронне фахове наукове видання "Кібербезпека: освіта, наука, техніка", вип. 4, вип. 12, Червень 2021, с. 108-16, doi:10.28925/2663-4023.2021.12.108116.

2. Zhuravchak Danyil, Opanovych Maksym, Dudykevych Valerii, Piskozub Andrian, (2022). Detection Method Of Credential Dumping Method Through Exploiting Vulnerable Windows Error Reporting Service In Windows Operating Systems. Сучасна спеціальна техніка, 2 (69), 38-52. https://doi.org/10.36486/mst2411–3816.2022.2(69).2

3. Zhuravchak, D. ., V. Dudykevych, і A. Tolkachova. "ДОСЛІДЖЕННЯ СТРУКТУРИ СИСТЕМИ ВИЯВЛЕННЯ ТА ПРОТИДІЇ АТАКАМ ВІРУСІВ-ВИМАГАЧІВ НА БАЗІ ENDPOINT DETECTION AND RESPONSE". Електронне фахове наукове видання "Кібербезпека: освіта, наука, техніка", вип. 3, вип. 19, Березень 2023, с. 69-82, doi:10.28925/2663-4023.2023.19.6982.

4. Zhuravchak, D., Tolkachova, A., Piskozub, A., Dudykevych, V., і Korshun, N. "Monitoring Ransomware with Berkeley Packet Filter." CEUR Workshop Proceedings, vol. 3550, Cybersecurity Providing in Information and Telecommunication Systems II 2023. Proceedings of the Cybersecurity Providing in Information and Telecommunication Systems II co-located with the International Conference on Problems of Infocommunications. Science and Technology (PICST 2023), Kyiv, Ukraine, October 26, 2023 (online), 2023, pp. 95–106.

5. Піскозуб, А. З., Журавчак, Д. Ю., і Толкачова, А. Ю. "Дослідження Вразливостей у Чатботах з Використанням Великих Мовних Моделей." Безпека Інформації, том 29, № 3, 2023, с. 111–117.

6. Журавчак, Д., П. Глущенко, М. Опанович, В. Дудикевич, і А. Піскозуб. “КОНЦЕПЦІЯ НУЛЬОВОЇ ДОВІРИ ДЛЯ ЗАХИСТУ ACTIVE DIRECTORY ДЛЯ ВИЯВЛЕННЯ ПРОГРАМ-ВИМАГАЧІВ”. Електронне фахове наукове видання “Кібербезпека: освіта, наука, техніка”, вип. 2, вип. 22, Грудень 2023, с. 179-90, doi:10.28925/2663-4023.2023.22.179190.

7. Журавчак, Даниїл, Едуард Кійко, і Валерій Дудикевич. "Використання EBPF для Ідентифікації Вірусів-Вимагачів, що Використовують DNS-Запити DGA." Information Technology and Security, vol. 11, no. 2 (21), 2023, pp. 166–174.

8. Журавчак, Д. Ю. "Моніторинг Вірусів-Вимагачів за Допомогою Розширеного Берклійського Пакетного Фільтра (eBPF) та Машинного Навчання." Наукоємні Технології, том 60, № 4, 2023, с. 352–363.

9. Zhuravchak D., Dudykevych V. Real-time ransomware detection by using eBPF and natural language processing and machine learning // Advanced information and communication technologies : proceedings of the 5th IEEE International conference (Lviv, Ukraine, November 21–25, 2023). – 2023. – С. 221–224.

Наукові праці, які засвідчують апробацію матеріалів дисертації:\

10. Zhuravchak, D., Ustyianovych, T., Dudykevych, V., Venny, B., і Ruda, K. "Ransomware Prevention System Design Based on File Symbolic Linking Honeypots." 2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Cracow, Poland, 2021, pp. 284-287. IEEE,

11. Журавчак Д.Ю. , Опанович М. Ю. Аналіз атак вірусів-шифрувальників на системи типу Active Directory за допомогою кореляції закономірностей в

кіберзлочинах та аномальної активності // “Технічні засоби захисту інформації”, семінар при вченій раді НАН України, Київ, Україна, 2022 р.

12. Журавчак Д.Ю, проф. Дудикевич В.Б. Аналіз методів Threat Hunting для проактивного виявлення програм-вимагачів // “Технічні засоби захисту інформації”, семніар при вченій раді НАН України, Київ, Україна, 2023 р.

13. Журавчак Д.Ю, проф. Дудикевич В.Б. Аналіз мережевого трафіку з використанням eBPF/XDP для ефективного виявлення програм-вимагачів // “Технічні засоби захисту інформації”, семніар при вченій раді НАН України, Київ, Україна, 2024 р.

14. Журавчак Д. Ю., Дудикевич В. Б., Опанович М. Ю., Піскозуб А. З. СТВОРЕННЯ СИСТЕМИ БЕЗПЕРЕРВНОГО РЕАГУВАННЯ НА ІНЦИДЕНТИ ІНФІКУВАННЯ ВІРУСАМИ-ВИМАГАЧАМИ В ACTIVE DIRECTORY // Збірник тез доповідей підготовлено за матеріалами Міжнародної наукової інтернет-конференції (випуск 70) 22-23 вересня 2022 р. на сайті www. konferenciaonline. org. ua. - 2022. - С. 25.

15. Журавчак, Даниїл, і Валерій Дудакевич. "Виклики Та Перспективи Впровадження Машинного Навчання Для Виявлення Програм-Вимагачів В Режимі Реального Часу." Захист Інформації І Безпека Інформаційних Систем: Матеріали IX Міжнародної Науково-Технічної Конференції, Львів, 25–26 травня 2023 р., 2023, с. 141–143.

SUMMARY

Zhuravchak D.Y. **IMPROVEMENT OF REAL TIME RANSOMWARE DETECTION METHODS.** – Qualifying scientific work on manuscript rights.

Dissertation for obtaining the scientific degree of Doctor of Philosophy in the specialty 125 "Cyber Security". - Lviv Polytechnic National University, Lviv, 2024.

Abstract of content. The thesis is devoted to solving the urgent scientific and practical task of improving the methods of detecting ransomware in real time. This includes the development of new strategies and tools that can detect and neutralize ransomware immediately after it is activated on operating systems in computer networks. This will significantly increase the speed of response to cybersecurity incidents, reduce potential material and reputational losses from ransomware attacks, and improve the overall resilience of information systems of private and public entities. In addition, this work is also aimed at improving the understanding of the nature of ransomware and developing effective strategies to counter it.

The purpose of this thesis is to improve the efficiency of protecting computer networks and operating systems from ransomware using machine learning models that operate in real-time or close to real-time detection. During the dissertation research, it was found that traditional methods of detecting and protecting computer networks do not meet the needs of today. Therefore, detection and monitoring modules based on the eBPF filter were proposed and developed, the result of which was normalized in the format of event logs and prepared for use in mathematical machine learning models.

eBPF, as a modern and powerful technological solution, can provide the necessary tools for method development, especially in the context of real-time security incident response systems. Thus, this thesis seeks to enrich the understanding and practical skills in applying eBPF to ransomware, with a particular focus on real-time detection and response.

This paper presents various methodologies for using eBPF to improve real-time ransomware detection. In a world where cybercrime is continuously evolving and attack

methods are becoming more sophisticated, protecting computer networks is becoming more important and challenging.

The object of research is ransomware and real-time protection systems against it. This includes analyzing the behavior of such programs, their impact on the systems they operate on, and the security technologies used to detect and neutralize these threats.

Subject of research: Methods and tools for detecting ransomware based on eBPF and machine learning in real time. Aspects related to the development and optimisation of new algorithms and models for more effective detection of ransomware threats are considered.

The conclusions and proposals presented in this work have both theoretical and applied significance. The provisions presented in the thesis can be used in: research activities - to develop the study of the object and improve the developed system; practical activities - to optimize the protection of the object for which the software based on the methodology presented in the thesis will be developed.

The first chapter, "Analysis of the Problem of Detecting and Countering Ransomware," is devoted to a comprehensive analysis of ransomware. It discusses the classification, history of development, distribution methods and impact on information systems of these malicious programs. The structure and functional features of SIEM (Security Information and Event Management) and EDR (Endpoint Detection and Response) systems, as well as the main models and methods for detecting and neutralising ransomware threats, are studied in detail. An analytical review of the current state of cybercrime is included, with a focus on the impact of ransomware, with particular attention to trends and potential threats posed by ransomware. This chapter also justifies the choice of the research area and sets out the objectives of the thesis.

The second chapter, "Using eBPF for Ransomware Detection", provides an in-depth analysis of the eBPF (Extended Berkeley Packet Filter) technology and its capabilities in the context of ransomware. This section discusses the eBPF architecture, its key features and potential for effective threat detection and neutralisation. Particular

attention is paid to the integration of machine learning algorithms with eBPF to improve the efficiency of malicious activity detection. The chapter concludes with a comprehensive approach to using eBPF for real-time ransomware detection, including a detailed discussion of the methodology for developing and implementing eBPF-based modules for monitoring system calls, file and network activity. A separate section is devoted to the use of eBPF in the Windows environment, describing approaches to integrating eBPF with the Windows operating system for real-time threat monitoring and analysis.

The third chapter, "Creating an Integrated Method for Analysing Ransomware Based on Machine Learning Models," is dedicated to the development and implementation of effective data analysis methods for detecting ransomware, focusing on the use of machine learning models. This chapter describes in detail the architecture of an integrated ransomware detection system, including all stages from data collection to processing and analysis.

First, the process of generating datasets for training machine learning models is discussed, including methods of data collection, normalisation, and annotation, which are critical for accurate threat detection. Particular attention is paid to the selection of appropriate machine learning models, such as decision trees, support vector machines, neural networks, and deep learning, each of which has its own advantages and disadvantages in the context of ransomware detection.

The chapter also describes the development of a model pipeline, including data pre-processing, model selection and tuning, training, and testing. Considerable attention is paid to model performance metrics, such as accuracy, completeness, F1-measure, and others, which are used to assess the quality of models in detecting cyber threats.

Emphasis is placed on integrating machine learning models with eBPF technology to improve the efficiency and speed of real-time ransomware detection. This includes using eBPF to collect system data, which is then used to train and test machine learning models. The chapter concludes with an analysis of the results and a discussion of potential

ways to further improve the integrated system, including algorithm optimisation and functional enhancements. Thus, the third chapter provides a comprehensive understanding of the process of creating and implementing an integrated method for analysing ransomware based on machine learning models, which is a key element for improving the effectiveness of protecting computer systems from modern cyber threats.

The fourth section, "Analysis of the effectiveness of the proposed solutions," is devoted to a detailed analysis and evaluation of the effectiveness of the developed methods for detecting and countering ransomware. This section describes the process of organizing a test environment for conducting secure simulations of ransomware attacks, which allows for thorough data collection and analysis under controlled conditions.

The chapter begins by discussing the structure and setup of a test environment that includes different types of ransomware and their attack scenarios. Special attention is paid to data collection methods during test attacks, including the use of eBPF to monitor system calls, file activity, and network traffic. This allows us to obtain high-quality data for further analysis of the effectiveness of the proposed methods.

Next, we discuss the results of experiments that demonstrate the effectiveness of machine learning models in detecting ransomware. Key metrics such as accuracy, completeness, F1-measure, and Matthew's Correlation Coefficient (MCC) are analyzed to assess the quality of the models' performance in various attack scenarios. Particular attention is paid to comparing the results of the proposed methods with traditional approaches to ransomware detection, which allows us to clearly identify the advantages and disadvantages of each approach.

The chapter also contains an analysis of the consumption of system resources by the developed methods, which is an important aspect for their practical implementation in real systems. The performance of the models is evaluated under various conditions, including CPU, memory, and network resources, which allows for a balance between detection efficiency and optimal resource utilization.

The section concludes with recommendations for implementing the developed methods in practical computer systems. We discuss the possibilities of integrating the proposed solutions with existing security systems, such as SIEM and EDR, which will significantly increase the level of protection against ransomware. In addition, ways to further improve the methods based on the results of the experiments and analysis are proposed.

The conclusions of the thesis outline the main results and recommendations for the application of the created integrated real-time ransomware detection system, as well as present and characterise the quantitative and qualitative performance metrics of the proposed solution.

The appendices to the thesis include the program codes for implementing the eBPF modules and machine learning models, acts of implementation of the results of the thesis, as well as a list of the author's scientific papers and tests on the topic of the thesis.

Keywords: ransomware, eBPF, SIEM, EDR, security monitoring, cybersecurity, antivirus, machine learning, computer networks.

LIST OF PUBLICATIONS OF THE ACQUIRER

Scientific works in which the main scientific results of the dissertation are published:

1. Zhuravchak, D. "CREATION OF A SYSTEM FOR PREVENTING THE SPREAD OF RANSOMWARE VIRUSES WITH THE ASSISTANCE OF THE PYTHON PROGRAMMING LANGUAGE AND AUDITD UTILITY BASED ON THE LINUX OPERATING SYSTEM". Electronic professional scientific publication "Cybersecurity: Education, Science, Technology", Vol. 4, Issue 12, June 2021, pp. 108-16, doi:10.28925/2663-4023.2021.12.108116.

2. Zhuravchak Danyil, Opanovych Maksym, Dudykevych Valerii, Piskozub Andrian, (2022). Detection Method Of Credential Dumping Method Through Exploiting

Vulnerable Windows Error Reporting Service In Windows Operating Systems. Modern Special Equipment, 2 (69), 38-52. https://doi.org/10.36486/mst2411–3816.2022.2(69).2

3. Zhuravchak, D., V. Dudykevych, and A. Tolkachova. "RESEARCH OF THE STRUCTURE OF THE SYSTEM FOR DETECTING AND COUNTERACTING RANSOMWARE ATTACKS BASED ON ENDPOINT DETECTION AND RESPONSE". Electronic professional scientific publication "Cybersecurity: Education, Science, Technology", vol. 3, issue 19, March 2023, pp. 69-82, doi:10.28925/2663-4023.2023.19.6982.

4. Zhuravchak, D., Tolkachova, A., Piskozub, A., Dudykevych, V., and Korshun, N. "Monitoring Ransomware with Berkeley Packet Filter." CEUR Workshop Proceedings, vol. 3550, Cybersecurity Providing in Information and Telecommunication Systems II 2023. Proceedings of the Cybersecurity Providing in Information and Telecommunication Systems II co-located with the International Conference on Problems of Infocommunications. Science and Technology (PICST 2023), Kyiv, Ukraine, October 26, 2023 (online), 2023, pp. 95-106.

5. Piskozub, A. Z., Zhuravchak, D. Y., and Tolkacheva, A. Y. "Research of Vulnerabilities in Chatbots Using Large Language Models." Information Security, vol. 29, no. 3, 2023, pp. 111-117.

6. Zhuravchak, D., P. Glushchenko, M. Opanovych, V. Dudykevych, and A. Piskozub. "ZERO TRUST CONCEPT FOR ACTIVE DIRECTORY PROTECTION FOR RANSOMWARE DETECTION". Electronic professional scientific publication "Cybersecurity: Education, Science, Technology", vol. 2, issue 22, December 2023, pp. 179-90, doi:10.28925/2663-4023.2023.22.179190.

7. Zhuravchak, Daniel, Eduard Kiko, and Valeriy Dudykevych. "Using EBPF for Identification of Ransomware that uses DNS queries DGA." Information Technology and Security, vol. 11, no. 2 (21), 2023, pp. 166-174.

8. Zhuravchak, D. Y. "Monitoring of Ransomware Viruses with the Help of Extended Berkeley Batch Filter (eBPF) and Machine Learning." Science-intensive Technologies, vol. 60, no. 4, 2023, pp. 352-363.

9. Zhuravchak D., Dudykevych V. Real-time ransomware detection by using eBPF and natural language processing and machine learning // Advanced information and communication technologies : proceedings of the 5th IEEE International conference (Lviv, Ukraine, November 21–25, 2023). – 2023. – С. 221–224.

Scientific works certifying the approval of the dissertation materials:

10. Zhuravchak, D., Ustyianovych, T., Dudykevych, V., Venny, B., and Ruda, K. "Ransomware Prevention System Design Based on File Symbolic Linking Honeypots." 2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Cracow, Poland, 2021, pp. 284-287. IEEE,

11. Zhuravchak D., Opanovych M. Analysis of ransomware attacks on Active Directory-type systems by correlating patterns in cybercrime and anomalous activity // "Technical means of information protection", seminar at the Academic Council of the National Academy of Sciences of Ukraine, Kyiv, Ukraine, 2022.

12. Zhuravchak D.Y., Prof. Dudykevych V.B. Analysis of Threat Hunting methods for proactive detection of ransomware // "Technical means of information protection", seminar at the Academic Council of the National Academy of Sciences of Ukraine, Kyiv, Ukraine, 2023.

13. Analysis of network traffic using eBPF/XDP for effective detection of ransomware // "Technical means of information protection", Seminar at the Academic Council of the National Academy of Sciences of Ukraine, Kyiv, Ukraine, 2024.

14. Zhuravchak D., Dudykevych V., Opanovych M., Piskozub A. CREATION OF A CONTINUOUS REACTION SYSTEM FOR INCIDENTS OF INFECTION BY RANSOMWARE VIRUSES IN ACTIVE DIRECTORY // Collection of abstracts

prepared on the basis of the materials of the International Scientific Internet Conference (issue 70) on 22-23 September 2022 at www. konferenciaonline. org. ua. - 2022. - С. 25.

15. Zhuravchak, Danyil, and Valerii. "Challenges and Prospects for Implementing Machine Learning for Real-Time Ransomware Detection." Information Protection and Security of Information Systems: Proceedings of the IX International Scientific and Technical Conference, Lviv, 25-26 May 2023, 2023, pp. 141-143.

ЗМІСТ

ЗМІСТ 18
ВСТУП 22
РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМИ ВИЯВЛЕННЯ ТА ПРОТИДІЇ ПРОГРАМАМ-ВИМАГАЧАМ 33
1.1. Аналіз сучасного стану досліджень та публікацій 33
1.2. Віруси-вимагачі: Опис та класифікація, види 37
1.2.1. Історія вірусів-вимагачів, їх еволюція та поширення 46
1.2.2. Поширення вірусів-вимагачів 49
1.3. Основні моделі і методи виявлення і протидії програмам-вимагачам. Аналіз методів 55
1.3.1. Аналіз вірусів-вимагачів 64
1.4. Аналітичний огляд існуючих підходів до забезпечення безпеки комп'ютерних мереж від програм-вимагачів 66
1.5. Архітектура SIEM та EDR рішень 68
1.6. Аналіз сучасного стану кіберзлочинності в Україні та світі. Використання вірусів-вимагачів під час війни в Україні 69
1.6.1 Найбільші кримінальні групи сьогодення 69
1.6.2 Тенденції у світі 71
1.6.3. Брокери початкового доступу 75
1.7. Обґрунтування вибору напряму дослідження та постановка завдань 77
Висновки до 1 розділу 79
РОЗДІЛ 2 ВИКОРИСТАННЯ EBPF ДЛЯ ВИЯВЛЕННЯ ПРОГРАМ-ВИМАГАЧІВ 81
2.1. Вступ до eBPF: Огляд архітектури і можливостей 81
2.2. Методологія розробки модулів на базі eBPF 87
2.3 Застосування eBPF для відстеження системних викликів 94
2.3.1 Поєднання політик Seccomp з eBPF у єдиний модуль Seccomp-eBPF 97
2.3.2 Seccomp Notifier 97

2.4 Моніторинг файлів та криптографічної активності з використанням honeypot та eBPF 99
2.5 Аналіз мережевого трафіку з використанням eBPF 106
2.6 Моніторинг процесів з використанням eBPF 110
2.7 Використання eBPF для моніторингу показників продуктивності 113
2.7.1 Загальна ідея показників продуктивності 113
2.8 Доступ до даних ядра за допомогою eBPF 116
2.9 Імплементація роботи модулів eBPF на платформі Windows 119
Висновки до 2 розділу 121
РОЗДІЛ 3 СТВОРЕННЯ ІНТЕГРОВАНОГО МЕТОДУ АНАЛІЗУ ВІРУСІВ-ВИМАГАЧІВ НА БАЗІ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ 123
3.1 Архітектура інтегрованої системи виявлення вірусів-вимагачів 124
3.2 Розроблення модельного конвеєра 125
3.2.1 Формування наборів даних 132
3.3 Вибір моделей машинного навчання для аналізу даних 135
3.3.1 Розробка моделі класифікації програм-вимагачів за допомогою дерев рішень та ансамблів випадкового лісу 141
3.3.2 Метод опорних векторів, як інструмент для розділення “безпечних” та “небезпечних” програм на основі гіперплощини, що максимізує відстань між класами 147
3.3.3 Розроблення моделі виявлення вірусів-вимагачів з використанням глибоких нейронних мереж 154
3.4 Метрики оцінки машинного навчання 158
Висновки до 3 розділу 164
РОЗДІЛ 4 АНАЛІЗ ЕФЕКТИВНОСТІ ЗАПРОПОНОВАНИХ РІШЕНЬ 166
4.1 Організація тестового середовища для проведення атак вірусів-вимагачів з метою оцінки ефективності рішень 166
4.2 Проведення тестових атак і збір даних для аналізу 169
4.2.1 Безпечна імітація різних видів вірусів-вимагачів 170
4.3 Аналіз результатів дії моделей машинного навчання при різних експериментах 176
4.3.1 Точність, повнота та інші метрики якості 179

4.3.2 Споживана системними ресурсами продуктивність 185
4.4 Співставлення ефективності різних методів виявлення 186
4.5 Порівняльний аналіз з традиційними методами виявлення вірусів-вимагачів 189
Рекомендації щодо впровадження запропонованих методів у практичних комп'ютерних системах 190
Висновки до 4 розділу 192
ВИСНОВКИ 194
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ 196
ДОДАТОК А. Акти впровадження 214
ДОДАТОК Б. Фрагменти програмного коду, скриптів автоматизації та машинного навчання 218
ДОДАТОК В. Список публікацій здобувача за темою дисертації та відомості про апробацію результатів дисертації 231

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

1. AES – Advanced Encryption Standard
2. APT – Advanced Persistence Threat
3. BIOS – Basic Input-Output System
4. BPF – Берклівський фільтр пакетів
5. C&C – Command and Control
6. DGA – Domain Generation Algorithm
7. DNS – Domain Name Server
8. eBPF – Розширений Берклівський фільтр пакетів
9. ECDH – Elliptic-Curve Diffie-Hellman
10. EFS – Encrypted File System
11. VM – Virtual Machine
12. RDP – Remote Desktop Protocol
13. SMB – Server Message Block, протокол передачі повідомлень сервера
14. WMI – Windows Management Instrumentation, інструментарій управління Windows
15. EPS – Endpoint Protection System, система захисту кінцевих точок
16. EDR – Endpoint Detection and Response, система виявлення та реагування на кінцевих точках
17. SIEM – Security Information and Event Management, система управління інформаційною безпекою та подіями
18. IDS – Intrusion Detection System, система виявлення вторгнень
19. IPS – Intrusion Prevention System, система запобігання вторгненням
20. PCAP – Packet Capture, захоплення пакетів

ВСТУП

Програми-вимагачі стали однією з найбільших загроз для кібербезпеки, використовуючи широкий спектр методів уникнення виявлення. Атака вірусів-вимагачів може призвести до повної зупинки функціонування підприємства, об’єкту критичної інфраструктури, лікарні, або держустанови.

Це створює необхідність розробки нових та більш ефективних методів їх виявлення та нейтралізації.

Актуальність теми.

Тема "Удосконалення методів виявлення та протидії вірусам-вимагачам у режимі реального часу" є актуальною у зв'язку зі зростанням кількості кібератак вірусів-вимагачів у різних країнах світу. Ці віруси стали одними з найбільш поширених і небезпечних загроз для безпеки інформації та комп'ютерних систем протягом останніх двох десятиліть.

За останні кілька років спостерігається зростання кількості атак вірусів-вимагачів на різні організації, включаючи компанії, установи, медичні заклади, освітні установи та державні інституції. Наприклад, в 2020 році велика кількість лікарень, що працювали з пацієнтами з COVID-19, були атаковані вірусами-вимагачами, що призвело до тимчасової недоступності медичної допомоги.

У зв'язку з цим дослідження та розробка нових методів виявлення та протидії вірусам-вимагачам є надзвичайно важливим завданням. Це необхідно для забезпечення безпеки інформації та захисту комп'ютерних систем. Вдосконалення методів виявлення та протидії вірусам-вимагачам у режимі реального часу може допомогти зменшити ризик втрати даних та забезпечити швидку реакцію на потенційні загрози.

В останні роки кіберзлочинність, зокрема програми-вимагачі, стала значною загрозою для глобальної кібербезпеки, що вимагає негайної уваги від дослідників та професіоналів у цій сфері. У цьому контексті, ряд книг та досліджень було

присвячено аналізу загроз програм-вимагачів, розробці методів виявлення та протидії.

У книзі Метью Райана "Ransomware Revolution: The Rise of a Prodigious Cyber Threat" детально проаналізовано вплив програм-вимагачів на сучасний світ, об'єднуючи технічні, економічні та психологічні аспекти цієї загрози. З однієї точки зору, міжнародний характер кіберпростору ускладнює притягнення до відповідальності злочинців, що стоять за атаками вірусів-вимагачів. Нерідко кіберзлочинці діють з територій, де відсутні ефективні механізми правової взаємодії між державами, що значно ускладнює процес їх ідентифікації та екстрадиції. Економічні наслідки атак вірусів-вимагачів можуть бути руйнівними для бізнесу, особливо для малих та середніх підприємств, які не мають достатніх ресурсів для швидкого відновлення своїх систем. Виплати великих сум викупу, втрата даних та недовіра з боку клієнтів можуть призвести до значних фінансових втрат і навіть до банкрутства[1].

У книзі Аллана Ліска та Тімтоті Галло "Ransomware: Defending Against Digital Extortion" запропоновано всебічний погляд на програми-вимагачі, а також методи їх виявлення і протидії. Книга містить практичні поради щодо захисту цифрових активів і є незамінним ресурсом для фахівців у сфері кібербезпеки. Однак дана книга не покриває методів своєчасного виявлення, а методи покладаються на антивіруси вендорів, що не забезпечують високу ефективність проти атак типу нульового дня від APT угрупувань[2].

Книга, що написана групою авторів Абхіджит Моханта, Мунір Хахад, Кумарагуру Велмуруган "Preventing Ransomware: Understand, Prevent, and Remediate Ransomware Attacks" надає докладний огляд стратегій запобігання атакам програм-вимагачів. Автори діляться рекомендаціями щодо проактивної конфігурації безпекового програмного забезпечення для ефективного захисту, але не охоплюють методи реагування та виявлення на інциденти створенні вірусами-

вимагачами. Автори натомість виділяють необхідність використання рішень, які допоможуть проактивно полювати на загрози програм-вимагачів[3].

Росс Брюер у статті "Ransomware Attacks: Detection, Prevention and Cure" описує загрозу від програм-вимагачів та надає комплексний підхід до їх виявлення, запобігання та відновлення компютерної мережі після атак. Проте, дослідження не враховує останніх тенденцій у розвитку вірусів-вимагачів, а натомість описує загальновідомі методи, що обмежує ефективність запропонованих методів. Окрім цього є враження, що підходи, рекомендовані для відновлення системи після інциденту, не були достатньо протестовані у реальних умовах, так, як вони описуються лише теоретично, що ставить під сумнів їх практичну цінність даного дослідження [4].

Окрім книг є велика кількість наукових праць, які описують проблематику, виклики та можливості виявлення та протидії програмам-вимагачам.

Одне з таких досліджень, проведене Тайхманн, Ботіччіу та Серджі (2023), зосереджене на еволюції атак програм-вимагачів у світлі останніх кіберзагроз, з особливим акцентом на вплив геополітичних конфліктів на кіберклімат. Автори аналізують зв'язок між останніми подіями в Україні та діяльністю груп програм-вимагачів, що свідчить про зростаючу політизацію кіберпростору[5].

У статті Самарасекера (2022) розглядається вплив агресивного нападу Росії на Україну та його наслідки для кібербезпеки, включаючи спостереження за новими формами шкідливого програмного забезпечення, що використовуються проти українських систем [6].

Вітчизняні автори, такі як Фаріон-Мельник зосереджуються на ризиках, захисті та профілактичних заходах проти атак програм-вимагачів. Їхнє дослідження наголошує на швидкому зростанні цього виду кіберзлочинності та необхідності розвитку ефективних методів захисту, надаючи важливий внесок у розробку національних стратегій кібербезпеки. Їхні рекомендації можуть стати основою для покращення захисту інформаційних систем в Україні[7].

Адамов та Карлсон (2017) детально досліджують стан вірусів-вимагачів, їх тенденції та методики захисту. Автори аналізують різноманітні зразки шкідливих програм, щоб виявити протиріччя у їхньому розповсюдженні та ефективності протидії. Важливо відзначити, що дослідження показує значне поширення вірусів-вимагачів у регіонах, як-от Росія та Україна, а також акцентує на важливості англомовних користувачів у їх цільових атаках[8].

У даній дисертаційній роботі розглядається проблема своєчасного виявлення програм-вимагачів у режимі реального часу, або майже до реального часу, тобто з похибкою до 30 секунд. Однак необхідно визнати певні виклики та бар'єри, які вимагають уваги при формулюванні стратегій для забезпечення надійного виявлення шкідливих програм типу вірус-вимагач.

Перший виклик – еволюція та адаптація вірусів та вірусів-вимагачів. Програми-вимагачі швидко еволюціонують, адаптуючись до змін у технологіях і стратегіях захисту. Це означає, що механізми, які були ефективними вчора, можуть виявитися недієвими сьогодні. Наприклад, розробники вірусів постійно вдосконалюють методи уникнення виявлення антивірусними програмами, використовуючи поліморфізм і метаморфізм для зміни коду вірусу при кожному його виконанні, чи через певний короткий період часу.

Другий виклик полягає у тому, що більшість вірусів-вимагачів використовують атаки, відомі як атаки “нульового дня”. Атаки “нульового дня” експлуатують вразливості у програмному забезпеченні, про які ще не відомо виробникам цього ПЗ, тому вони є особливо ефективними і небезпечними. Програми-вимагачі, які мають у арсеналі такі атаки, можуть легко обійти існуючі механізми захисту. Розробка стратегій для захисту від таких атак вимагає неперервного моніторингу і аналізу нових загроз та швидкого оновлення захисних систем.

Третій виклик пов’язаний з великою кількість інсайдерів, а також зростаюча популярність брокерів первинного доступу (Initial Access Brokers). Інсайдери

можуть навмисно або ненавмисно стати джерелом безпекових інцидентів, включаючи витік конфіденційної інформації, шахрайство або впровадження шкідливого програмного забезпечення. У свою чергу брокери первинного доступу спеціалізуються на продажу несанкціонованого доступу до корпоративних мереж зловмисникам, включаючи розповсюджувачів програм-вимагачів.

Четвертий виклик полягає у тому, що сучасні операційні системи та комп'ютерні мережі щодня стають все складнішими. Зростання складності корпоративних мереж та широке використання хмарних технологій створює нові виклики для забезпечення мережевої безпеки. Віруси-вимагачі часто розповсюджуються через мережеві з'єднання, використовуючи слабкі місця у мережевій інфраструктурі. Ефективний захист вимагає комплексного підходу до мережевої безпеки, включаючи сегментацію мережі, шифрування трафіку та застосування сучасних систем виявлення та запобігання вторгненням.

Дана дисертація описує всі переваги та недоліки захисту від вірусів-вимагачів за допомогою технології eBPF, а також моделей машинного навчання.

Зв'язок роботи з науковими програмами, планами, темами.

Дисертаційні дослідження виконувались у відповідності до наукового напряму кафедри захисту інформації Національного університету "Львівська політехніка"

- Основні положення та результати дисертаційної роботи впроваджені у навчальний процес кафедри "Захист інформації" Національного університету "Львівська політехніка" при вивченні дисциплін: "Міжнародні стандарти з кібербезпеки" для студентів напрямку підготовки 125 "Кібербезпека", спеціалізації "Управління інформаційною безпекою".

"Дослідження систем технічного захисту інформації, каналів зв'язку та комп'ютерних мереж, фізичного захисту інформації та криптографії.", в межах кафедральної науково-дослідної роботи: "Розроблення та удосконалення методів і засобів захисту інформації для протидії несанкціонованому доступу в

інформаційно-комунікаційних мережах” (шифр ЗІ-7) (№ держреєстрації 0119U101690) (2019р.-2022р.);

А також в діяльності підприємств ТОВ “ЕПАМ СИСТЕМЗ” та ТзОВ “ВІП СТУДІЯ”.

Мета дисертації полягає у розробці та вдосконалені сучасних методів виявлення та моніторингу програм-вимагачів в режимі реального часу.

Завдання. Дисертаційна робота присвячена вирішенню актуального науково-практичного завдання підвищення ефективності виявлення кіберзлочинів, що викликані поширенням вірусів-вимагачів.

Для успішного досягнення мети даної роботи необхідно виконати наступні завдання:

1. Провести глибокий технічний аналіз існуючого спектру вірусів-вимагачів, визначити їх класифікацію та характерні атрибути та ознаки в даних подій безпеки.
2. Оцінити наявні методики виявлення та нейтралізації загроз від вірусів-вимагачів, виявити потенційні прогалини та обмеження.
3. Провести детальне дослідження можливостей технології eBPF у контексті розробки ефективних засобів моніторингу та кіберзахисту.
4. Вдосконалення існуючих алгоритмів та підходів до виявлення та класифікації програм-вимагачів, з урахуванням специфіки сучасних кіберзагроз.
5. Розробити моделі виявлення програм-вимагачів на основі машинного навчання, яка базується на принципах машинного навчання, щоб аналізувати об'єми великих даних та ефективно виявляти потенційні загрози.
6. Провести експериментальні дослідження для випробування та оцінки розроблених методів та моделей, з метою демонстрації їхньої ефективності у реальних умовах.

Об'єктом дослідження є програми-вимагачі та системи захисту від них у режимі реального часу. Це включає аналіз поведінки таких програм, їх вплив на

системи, в яких вони працюють, і технології захисту, що використовуються для виявлення та нейтралізації цих загроз.

Предмет дослідження: Методи та засоби виявлення програм-вимагачів на основі eBPF та машинного навчання в режимі реального часу. Зокрема, розглядаються аспекти, пов'язані з розробкою та оптимізацією нових алгоритмів та моделей для більш ефективного виявлення загроз типу вірусів-вимагачів.

Методи дослідження. В процесі досліджень використано методи математичної статистики, збору та обробки даних, методи машинного навчання, методи нейронних мереж та глибокого навчання, метод опорних векторів, методи дерев рішень, та методи оцінки якостей математичних моделей.

Науковою новизною роботи є розробка нових методів виявлення та протидії вірусам-вимагачам у режимі реального часу на основі сучасних технологій машинного навчання, а також інтеграція різних методів захисту для підвищення ефективності захисту комп'ютерних систем від цих шкідливих програм.

Наукова новизна полягає у тому, що:

1. Вперше розроблено модель інтегрованої системи збору даних для виявлення вірусів-вимагачів, що об'єднує застосування eBPF для моніторингу системних викликів, файлової та криптографічної активності, аналізу мережевого трафіку та процесів. Ця система забезпечує унікальний набір даних (features), які використовуються для ефективного ідентифікування потенційних загроз в режимі реального часу.

2. Вперше запропоновано комплексну модель класифікації вірусів-вимагачів з використанням ансамблю дерев рішень та випадкового лісу, що дозволяє з високою точністю розрізняти "безпечні" та "небезпечні" програми на основі аналізу складних поведінкових шаблонів та криптографічної активності.

3. Вперше запропоновано методологію застосування глибоких нейронних мереж для ідентифікації складних шаблонів у даних зібраних модулями eBPF, що

представляють поведінку вірусів-вимагачів, забезпечуючи новий рівень точності виявлення невідомих або еволюціонованих загроз.

4. Отримали подальший розвиток методи виявлення кіберзагроз за допомогою аналізу мережевого трафіку з використанням eBPF, що значно підвищує швидкість та точність ідентифікації потенційних атак вірусів-вимагачів у порівнянні з традиційними підходами.

5. Вдосконалено метод симуляції кібератак за допомогою моделі емуляцій дій шахрая для тестування та оцінки ефективності розроблених моделей, одночасно, включаючи запуск вірусів-вимагачів у контрольованому лабораторному середовищі. Це дозволило детально аналізувати реакцію моделей на різноманітні сценарії атак та оптимізувати їх для максимальної ефективності.

6. Отримали подальший розвиток методики порівняльного аналізу та оцінки ефективності математичних апаратів виявлення та протидії програмам-вимагачам, за допомогою метрики MCC (коефіцієнту кореляції Метью), що виявився ефективнішим для оцінювання моделей, які працюють з незбалансованими даними, характерними для сценаріїв кіберзагроз типу вірусів-вимагачів.

Практична цінність полягає у можливості використання розроблених методів та інструментів для ефективного виявлення та протидії вірусам-вимагачам у режимі реального часу, що дозволить підвищити рівень захисту комп'ютерних систем від цих загроз. Результати дослідження можуть бути корисними для фахівців з інформаційної безпеки, розробників антивірусного програмного забезпечення, а також для широкого кола користувачів комп'ютерних систем.

1. Використання розробленого комплексного методу аналізу даних на рівні ядра та ідентифікації програм-вимагачів із використанням фільтру eBPF, що значно підвищує швидкість та точність виявлення кіберзагроз. Впровадження цього методу у системи кібербезпеки дозволяє оперативно відслідковувати та реагувати

на підозрілі зміни в системних викликах, файловій активності та мережевому трафіку, сприяючи своєчасному виявленню атак.

2. Використання розробленої моделі класифікації на основі ансамблю дерев рішень та випадкового лісу продемонструвало підвищену точність у виявленні шкідливих програм, досягаючи в середньому точності вище 95% і F1-метрики 97.7%, що є значним внеском у розвиток інструментів кібербезпеки.

3. Застосування методології глибоких нейронних мереж для аналізу складних шаблонів даних демонструє передовий підхід до виявлення новітніх кіберзагроз. Розроблені моделі забезпечили точність ідентифікації на рівні 97.8%, прецизійність 96.9% та F1-метрику 97.7%, значно покращуючи аналітичні можливості комп'ютерних систем і забезпечуючи надійне виявлення невідомих раніше загроз та вірусів-вимагачів, що еволюціонують.

4. Проведені експерименти у контрольованому лабораторному середовищі підтвердили високу точність розроблених моделей, де модель на базі глибоких нейронних мереж продемонструвала точність до 97.8% і Коефіцієнт Кореляції Меттью (MCC) 0.95, що вказує на високий рівень адекватності та надійності виявлення кіберзагроз.

Наукові та практичні результати виконаних досліджень використані у навчальному процесі кафедри захисту інформації Національного університету “Львівська політехніка”, зокрема для студентів спеціальності 125 “Кібербезпека” в курсі лекцій з дисципліни “Міжнародні стандарти з кібербезпеки”.

Основні результати дисертаційної роботи використано і впроваджено з метою покращення захищеності комп’ютерної мережі та систем в компанії ТОВ “ЕПАМ СИСТЕМЗ”, реагування на інциденти кібербезпеки, компанією ТзОВ “ВІП СТУДІЯ” що підтверджено актами впровадження.

Особистий внесок. Основні наукові результати дисертаційної роботи отримано автором самостійно. У працях, опублікованих у співавторстві, внесок Журавчака Д.Ю. є вирішальним, зокрема авторові належать (нумерація згідно

Додатку В): У статті [1], було представлено систему запобігання поширенню вірусів за допомогою Python та утиліти auditD. В [2], досліджено методи виявлення методу зливу облікових даних через вразливі сервіси Windows. Робота [3] описує аналіз систем виявлення та протидії атакам вірусів-вимагачів на базі EDR. У статті [4], виконано аналіз вразливостей у чатботах з використанням великих мовних моделей. В [5], представлено аналіз застосування концепції нульової довіри для захисту Active Directory. Стаття [6] розкриває методику використання eBPF для ідентифікації вірусів-вимагачів, що використовують DNS-запити DGA. В [7], описано моніторинг вірусів-вимагачів через розширений Берклійський пакетний фільтр та машинне навчання. Результати [8] демонструють систему на базі програмних-приманок для виявлення вірусів-вимагачів. В роботах [9] і [10] аналізуються атаки на системи типу Active Directory та методи Threat Hunting для проактивного виявлення вірусів. У [11], розглядається аналіз мережевого трафіку з використанням eBPF/XDP. Нарешті, в [12] вивчено систему безперервного реагування на інциденти інфікування вірусами-вимагачами в Active Directory, а в [13, 14] подано аналіз впровадження машинного навчання для виявлення вірусів-вимагачів у реальному часі.

Апробація результатів. Основні результати дисертаційного дослідження апробовано на міжнародних наукових та науково-практичних конференціях, наукових школах та консорціумах, семінарах: Proceedings of the Cybersecurity Providing in Information and Telecommunication Systems II co-located with the International Conference on Problems of Infocommunications. Science and Technology (PICST 2023), Kyiv, Ukraine, October 26, 2023 (online); 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Cracow, Poland, 2021; Міжнародна наукова інтернет-конференція 22-23 вересня 2022 р. на сайті www. konferenciaonline. org. ua; IX Міжнародна Науково-Технічна Конференція, Львів, 25–26 травня 2023 р.; Міжвідомчому міжрегіональному семінар Наукової Ради НАН України "Технічні

засоби захисту інформації" (2022, 2023, 2024 років, Київ, Україна); Наукові семінари кафедри захисту інформації, Національного університету "Львівська Політехніка" (2020-2024 рр.).

Публікації. Основні результати дослідження викладено у чотирнадцяти наукових публікаціях, а саме: у восьми статтях (із них сім – у фахових наукових виданнях України та одна – у періодичних виданнях зарубіжної держави) і шести тезах та виступів на науково-практичних заходах.

Структура та обсяг дисертації. Дисертаційна робота викладена на 233 сторінках та складається з анотації, змісту, переліку скорочень, вступу, чотирьох основних розділів, в яких міститься 41 рисунків та 19 таблиць, списку використаних джерел з 210 найменувань, а також 4 додатки. За структурою, мовою та стилем викладення дисертація відповідає вимогам МОН України. Робота написана грамотною українською мовою з використанням сучасної наукової термінології, а стиль викладення матеріалу є послідовним та логічним.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМИ ВИЯВЛЕННЯ ТА ПРОТИДІЇ ПРОГРАМАМ-ВИМАГАЧАМ

Цей розділ присвячений детальному аналізу програм-вимагачів, їхніх оперативних механізмів, а також розмаїття векторів та методик, що часто задіюються під час здійснення подібних кібернападів. Окрім цього розглядаються різні групи програм-вимагачів і висвітлюються найпоширеніші вектори атак, які використовувалися в реальних сценаріях.

У першому розділі детально аналізується проблематика виявлення та протидії програмам-вимагачам. Він містить систематичний огляд основних моделей і методів, що зараз використовуються для боротьби з цими шкідливими програмами, зокрема обговорюються їхні переваги та недоліки. Проводиться аналітичний огляд існуючих підходів до забезпечення безпеки комп'ютерних мереж та розглядається архітектура Security information and event management та endpoint protection and response рішень.

У розділі також здійснюється аналіз сучасного стану кіберзлочинності в Україні та світі, а це надає більш глибоке розуміння масштабу проблеми. Основуючись на цьому аналізі, здійснюється обґрунтування вибору напряму дослідження та постановка завдань. Перший розділ завершується розробкою висновків, що зроблені на основі проведеного дослідження.

1.1. Аналіз сучасного стану досліджень та публікацій

Хоч перший вірус-вимагач був створений 1984 року [9] і попри те, що методів виявлення та протидії є велика кількість, віруси-вимагачі вважаються найбільшою

загрозою кібербезпеки сьогодення. Дослідження методів виявлення та протидії вірусам вимагачам включає як традиційні методи на основі сигнатури та поведінки, так і нові підходи, що використовуються для аналізу програм, системи керування подіями безпеки інформації (SIEM) та коригування мережевого трафіку. На основі огляду літератури можна виділити наступні переваги та недоліки існуючих методів:

Стаття "Design and Implementation of an Intrusion Detection System by Using Extended BPF in the Linux Kernel" авторства Wang, S.-Y. та Chang, J.-C., присвячена розробці та впровадженню системи виявлення вторгнень з використанням розширеного Berkeley Packet Filter (eBPF) в ядрі Linux [10]. Це дослідження дозволяє створювати більш надійні та продуктивні системи виявлення вторгнень, порівняно з традиційними підходами, які базуються на просторі користувача та спираються на відстеження трафіку передбачуваних зразків. Однак дана стаття має декілька можливостей до покращення, а саме : масштабованість системи на основі eBPF може бути обмеженою у великих середовищах на базі операційної системи Windows і створення модулів, що будуть кросплатформні закриють потребу.

Стаття "Creating Complex Network Services with eBPF: Experience and Lessons Learned" авторства S. Miano, та інших [11]. Стаття розглядає створення складних мережевих сервісів з використанням технології eBPF (розширений Berkeley Packet Filter). Автори діляться своїм досвідом та уроками, отриманими під час розробки складних мережевих сервісів на основі eBPF. І також вказують, що створення складних мережевих функцій з використанням eBPF виявилося складним через обмежені можливості технології, що позначається на гнучкості їх застосування у реальних мережевих середовищах. Однак в 2018 році компанія Meta створила додаток з відкритим кодом Katran для аналізу мережевого трафіку[12].

Стаття "Demystifying the Performance of XDP BPF" авторства Hohlfeld, O.; Krude, J.; Reelfs, J.H.; Ruth, J.; Wehrle, K., досліджує продуктивність eXpress Data Path (XDP) для аналізу данних мережі наданий розширеним Berkeley Packet Filter (eBPF), фокусуючись на факторах, які впливають на продуктивність XDP [13].

Стаття "A Protocol-Independent Container Network Observability Analysis System Based on eBPF" авторства Liu, C. Та інших, досліджує розробку системи вивчення і аналізу мережевої доступності контейнерів на основі eBPF (розширений Berkeley Packet Filter), що працює незалежно від мережевих протоколів [14].

Стаття "Detection of Denial of Service Attack in Cloud-Based Kubernetes Using eBPF" авторства A. Sadik, та інших., досліджує застосування розширеного Berkeley Packet Filter (eBPF) для виявлення атак на відмову обслуговування (DoS) у хмарних системах на основі Kubernetes [15]. Стаття "Introducing SmartNICs in Server-Based Data Plane Processing: The DDoS Mitigation Use Case" авторства S. Milano та інших., розглядає впровадження інтелектуальних мережевих адаптерів (SmartNICs) в обробку серверів у випадку захисту від атак DDoS (розподілених атак типу "заперечення обслуговування") [16]. Обидві статті підкреслюють необхідність інноваційних підходів у протидії кіберзагрозам, хоча вони також наголошують на потребі додаткового дослідження для покращення масштабованості та інтеграції з іншими рішеннями безпеки.

Стаття "A Framework for eBPF-Based Network Functions in an Era of Microservices" авторства Sebastiano Miano, розглядає новий фреймворк для реалізації мережевих функцій на основі eBPF (розширений BPF) в контексті мікросервісів, полегшуючи розробку та впровадження програмних мережевих функцій на основі eBPF [17].

Підвищення методів виявлення та протидії вірусам вимагачам у режимі реального часу є важливою проблемою в галузі кібербезпеки. Використання eBPF може надати значні переваги та допомогти долати певні недоліки у дослідженні цього питання.

Враховуючи згадані статті, по темі eBPF можна виділити такі переваги досліджень:

1. Висока швидкість обробки: eBPF дозволяє набагато швидше обробляти мережевий трафік та повноцінно відстежувати активність у режимі реального часу

порівняно з більш традиційними аналогами, що працюють на рівні простору користувача.

2. Вища точність виявлення: eBPF дозволяє розробляти гнучкі та адаптивні системи виявлення, що допомагає більш точно виявляти несанкціоновану активність на ранніх стадіях атаки.

3. Гнучкість: eBPF дозволяє інтегрувати способи виявлення та протидії вірусам безпосередньо в ядро операційної системи, що дає змогу більш глибокого аналізу мережевого трафіку та швидкого застосування до новітніх типів атак.

4. BPF сприяє автоматизації процесів забезпечення безпеки, уможливлюючи негайне виявлення та реагування на загрози з використанням інтегрованих рішень у режимі реального часу.

Недоліки досліджень:

1. Складність розробки: Використання eBPF для розробки систем виявлення та протидії вірусам може бути складним процесом, який вимагає глибоких знань щодо eBPF та мережевої безпеки. Для забезпечення успішного впровадження необхідні тісна співпраця між командами розробників у галузі кібербезпеки.

2. Обмеження стосовно апаратного забезпечення: Ефективне впровадження eBPF може залежати від наявності сучасного апаратного забезпечення, в тому числі мережевих адаптерів, які ще можуть бути дорогими або складними в придбанні та розгортанні.

3. Недостатньо досліджень в спеціалізованих і специфічних контекстах: В контексті дослідження більше використання eBPF є порівняно новим напрямком досліджень, що може вимагати ще більше дослідницьких робіт для реалізації та оцінки його ефективності в різних контекстах та середовищах.

На основі цих переваг та недоліків, можна зробити висновок, що eBPF потенційно має значний прикладний потенціал у виявленні та протидії вірусам-вимагачам у режимі реального часу. Однак, з метою отримання найкращих результатів для різноманітних сценаріїв та середовищ, від розробників у галузі

кібербезпеки потрібно зробити спільні зусилля для розробки та дослідження ефективних технік і рішень на основі eBPF.

1.2. Віруси-вимагачі: Опис та класифікація, види

У сучасному світі, де інформаційні технології стали неодмінним аспектом повсякденного життя, захист інформації є ключовим завданням для організацій у всіх сферах діяльності. Одним з найбільш зловмисних і загрозливих видів кібератак є віруси-вимагачі, відомі також як рансомвар [18].

Віруси-вимагачі - це шкідливе програмне забезпечення, що шифрує дані користувача і вимагає від нього відшкодування за розшифрування. Вони можуть призвести до значних фінансових втрат, перерви в роботі та пошкодження репутації, особливо в бізнес-середовищі. Віруси-вимагачі можуть бути великою проблемою не тільки для окремих користувачів, але і для великих організацій. З огляду на стратегії розповсюдження, віруси-вимагачі часто поширюються серед користувачів, які можуть бути розподілені на ті, що поширюються через спам-кампанії, шахрайські веб-сайти, інфіковані додатки, або експлуатацію вразливостей в системі [19]. Залежно від їх впливу на інфіковану систему, їх можна поділити на шифрувальні віруси-вимагачі, які шифрують дані користувача, і блокувальні, які блокують доступ до системи в цілому. Існують також багато інших підтипів вірусів-вимагачів, включаючи досить нові форми, такі як віруси-вимагачі для мобільних пристроїв, віруси-вимагачі "інсайдери", які інфікують і шифрують дані вже всередині мережі, і так звані віруси-вимагачі "для деструктивної дії", які насправді не надають можливості відновити дані, навіть якщо відшкодування було сплачено [20].

По-перше, найчастіше зараження вірусом-вимагачем відбувається, коли користувач відвідує веб-сайт із порушеннями безпеки. Популярний метод, який

використовують зловмисники, - надсилання фішингового електронного листа, що містить посилання на веб-сайт, на якому розміщено код програми-вимагача. Ця форма атаки, також відома як "соціальна інженерія" [21], робить все можливе, щоб виглядати легітимно.

По-друге, код програми-вимагача розроблений таким чином, щоб вражати системи через одну з багатьох відомих вразливостей програмного забезпечення або операційної системи. Додаткові форми зараження вірусом-вимагача спеціально орієнтовані на користувачів з вищими рівнями дозволів, наприклад, адміністраторів, для впровадження шкідливого коду.

Після того, як код було доставлено та запущено в системі, можуть статися дві речі. Програма-шифрувальник заблокує користувачам доступ до системи. Криптографічні програми-вимагачі шифрують дані за допомогою складних математичних ключів шифрування. Системи, що постраждали від атаки вірусу-вимагача, можуть зазнати значних пошкоджень, або ж можуть бути атаковані певні типи файлів чи систем - наприклад, бази даних SQL чи файли Microsoft Office.

Види. Вимагачі є видом шкідливого ПЗ і вони успадковують багато технік від інших шкідливих програм. Більшість технік пояснюються в контексті з Windows, тому були згадані деякі API Windows. Існують наступні типи рансомвару [22]:

- Scareware та підроблене безпекове програмне забезпечення
- ScreenLocker
- Браузерний рансомвар
- Крипто-рансомвар
- Рансомвар, що цілиться на інфраструктуру
- Boot рансомвар

Для кожного типу рансомвару розглядаються такі пункти:

- Техніки, що використовуються з родиною рансомвару
- Деякі популярні рансомвари у родині

- Вказівки щодо аналізу такого рансомвару для аналітиків шкідливого програмного забезпечення

ScreenLocker Ransomware. Цей рансомвар не шифрує файли на машині жертви. Він блокує весь екран і не дозволяє жертві робити що-небудь інше, поки він не заплатить викуп [23]. Ось список деяких популярних рансомварів ScreenLocker:

1. Reveton
2. Urausy
3. Kovter
4. Tobfy
5. Weelsof
6. BlueScreen
7. Koktrom
8. Winlock
9. LockScreen

Boot Ransomware. Існує кілька типів шкідливих програм, які можуть працювати під час завантаження системи. Іноді їх називають буткітами. Boot-вимагач - це буткіт, який видає попереджувальне повідомлення ще до того, як буде здійснено вхід в операційну систему як користувач [24].

Завантажувальні програми захоплюють контроль ще до того, як операційна система повністю завантажиться. Як наслідок, у жертви залишається не так багато варіантів. Вони також ускладнюють роботу дослідників. Petya - це вірус-вимагач, яка заражає головний завантажувальний запис (MBR) [25].

Перш ніж перейти до зараження завантажувального файлу, давайте коротко розглянемо процес завантаження Windows. Для того, щоб комп'ютер успішно завантажився, всі ці компоненти повинні працювати належним чином: BIOS, операційна система та апаратне забезпечення. Якщо один з цих компонентів вийде з ладу, це, швидше за все, призведе до збою послідовності завантаження. Коли процесор увімкнено, BIOS завантажується з BIOS ROM. BIOS ініціює POST (самоперевірку при увімкненні), яка перевіряє, чи правильно працюють такі пристрої, як клавіатура, оперативна пам'ять і диски. BIOS шукає завантажувальний пристрій. MBR - це початок першого розділу диска, або можна сказати, що він присутній у секторі 0 фізичного жорсткого диска. MBR зчитується в пам'ять і виконується, і починається з коду, який називається завантажувачем BootStrap. MBR має таблицю, яка називається таблицею розділів (pt), яка зберігає інформацію про розділ. У таблиці розділів є лише один активний розділ, який називається завантажувальним. Перший сектор активного розділу називається завантажувальним сектором або Volume Boot Record (VBR). VBR є однією з найважливіших структур і може містити розмір блоку, розмір розділу, MF тощо. Головна файлова таблиця (MFT) - це таблиця, яка містить відомості про файли, їх розмір, мітку часу (коли вони були створені або змінені), доступ до файлів (дозволи на читання/запис) і так далі. MFT присутня у файловій системі NTFS у Windows. Відомо, що Petya шифрує MFT [26]. Таким чином, Windows не буде знати

місцезнаходження файлів. Навіть якщо окремі файли не зашифровані, їх неможливо відновити, оскільки MFT, яка є базою знань про файли, зашифрована.

Завантажувач BootStrap завантажує сектор в пам'ять і передає йому керування. VBR знаходить і завантажує код завантажувача. У Windows XP завантажувачем є NTLDR, а у Windows 7 використовується база даних конфігурації завантаження (BCD). У Windows XP NTLDR знаходить список операційних систем для завантаження. Він знаходиться у файлі boot.ini. NTLDR завантажує реєстр і пристрої, необхідні під час завантаження. У Windows 7 замість NTLDR використовується BOOTMGR, а список ОС для завантаження міститься у базі даних конфігурації завантаження (BCD). Після цього етапу winload.exe завантажує реєстр і пристрої у Windows 7. Потім управління передається NTOSKRNL.exe, який завантажує драйвери і служби, необхідні системі:

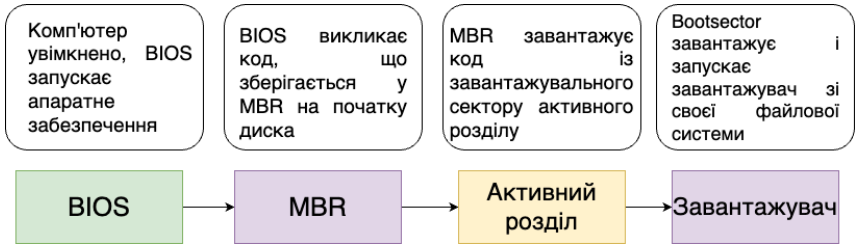


Рис. 1.1 Процес завантаження Windows

Crypto Ransomware. Віруси-вимагачі дійсно є одним з найбільш шкідливих видів шкідливого програмного забезпечення сьогодні. Вони шифрують файли на комп'ютері користувача, фактично блокуючи доступ до його власних даних. Потім програма-вимагач вимагає викуп в обмін на ключ до розшифрування, щоб відновити доступ до файлів [27]. Однією з можливих причин збільшення кількості криптовалютних програм-вимагачів може бути відносна легкість їхньої розробки порівняно з іншими формами шкідливих програм. Замість того, щоб писати складний код для проникнення в систему або викрадення даних, програми-вимагачі просто сканують каталоги користувача на наявність відповідних файлів - зазвичай

тих, які здаються особистими або важливими, - і шифрують їх. Багато різновидів вірусів-вимагачів не переймаються тим, щоб ховатися в системі або встановлювати компоненти руткітів, оскільки їм достатньо запуститися лише один раз, щоб зашифрувати всі файли. Деякі з них також мають механізми перевірки, чи не заражена система іншими вірусами-шифрувальниками, щоб уникнути дублювання. Існує багато різновидів програм-шифрувальників. Ось кілька найпомітніших прикладів: Locky, Cerber, CryptoLocker, Petya.

Принцип роботи. Віруси-вимагачі технічно роблять наступні дії:

Знаходить файли в локальній системі. На комп'ютері з Windows він може використовувати API FindFirstFile(), FindNextFile() для перебору каталогів файлів. Багато програм-вимагачів також шукають файли на спільних дисках.

Далі він перевіряє розширення файлу, яке потрібно зашифрувати. Більшість з них мають чітко визначений список розширень файлів, які потрібно зашифрувати. Навіть якщо він шифрує виконувані файли, він не повинен шифрувати жодні системні виконувані файли. Це гарантує, що не буде можливості відновити файли з резервної копії, видаливши резервну копію. Іноді це робиться за допомогою інструменту vssadmin [28]. Багато вірусів-вимагачів використовують команду vssadmin, що надається Windows для видалення тіньових копій. Тіньові копії - це резервні копії файлів і томів. Для керування тіньовими копіями використовується інструмент vssadmin (адміністрування vss). Абревіатура VSS - це скорочення від тіньової копії тома, яку також називають службою створення знімків томів. Нижче наведено знімок екрана інструмента vssadmin:

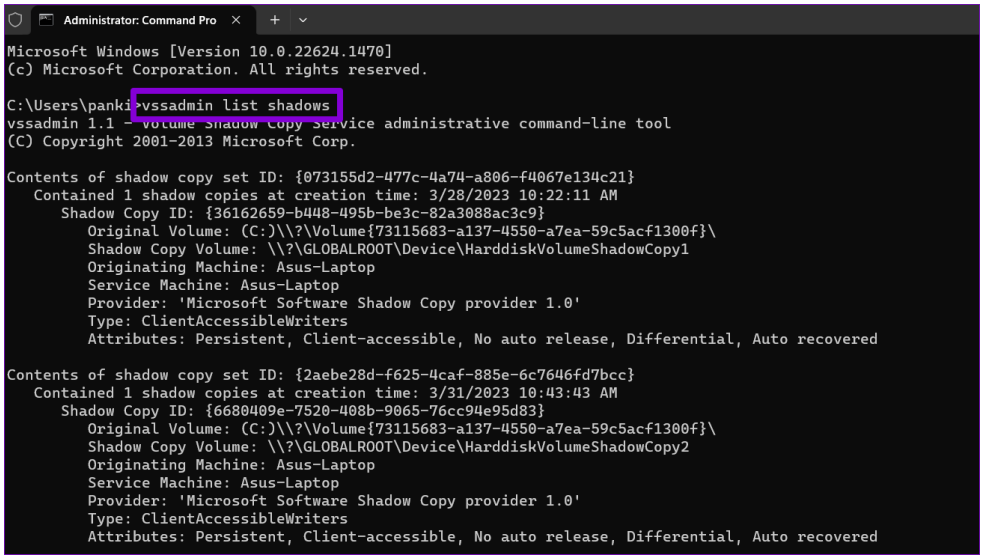


Рис. 1.2 vssadmin утиліта

Після шифрування файлів програма-вимагач залишає жертві текстовий файл. Зазвичай у ньому повідомляється, що файли в системі були зашифровані, і щоб їх розшифрувати, потрібно заплатити викуп. У вимозі про викуп міститься інструкція про те, як заплатити викуп. Сьогодні багато криптографічних алгоритмів використовуються шкідливими програмами. Шкідливе програмне забезпечення може використовувати криптографію для наступних цілей:

- Заплутування власного коду, щоб антивірус не міг легко ідентифікувати справжній код
- Для зв'язку з власним сервером
- Шифрування файлів на комп'ютері-жертві

Криптографічна система може мати наступні компоненти:

- Відкритий текст
- Ключ шифрування
- Шифротекст, тобто зашифрований текст
- Алгоритм шифрування, який також називається шифром
- Алгоритм розшифрування

Існує два типи криптографічних алгоритмів залежно від типу використовуваного ключа:

- Симетричний
- Асиметричний

Пояснення алгоритму: відправник - це особа, яка надсилає дані після їхнього шифрування, а одержувач - це особа, яка розшифровує дані за допомогою ключа. У шифруванні з симетричним ключем відправник і одержувач використовують один і той самий ключ, який ще називають секретним ключем. Відправник використовує ключ для шифрування даних, тоді як одержувач використовує той самий ключ для розшифрування. Наступні алгоритми використовують симетричний ключ:

- RC4
- AES
- DES
- 3DES
- BlowFish

Рис. 1.3 Симетричне шифрування

Симетричний ключ простіше реалізувати, але він стикається з проблемою безпечного обміну ключами [29]. Відкритий або асиметричний ключ вирішує

проблему обміну ключами шляхом використання пари ключів: відкритого та закритого. Відкритий ключ може розповсюджуватися незахищеним способом, в той час як закритий ключ завжди зберігається у власника в таємниці. Будь-який з ключів може бути використаний для шифрування, а інший - для розшифрування.

Тут представлені найпопулярніші алгоритми асиметричних шифрів:

- RSA
- Diffie-Hellman
- ECC
- DSA

Рис. 1.4 Асиметричне шифрування

Криптографічні програми-вимагачі починалися з простої криптографії з симетричним ключем. Але незабаром дослідники змогли легко розшифрувати ці ключі. Тому вони почали використовувати асиметричне шифрування. Програми-вимагачі нинішнього покоління почали розумно використовувати як симетричні, так і асиметричні ключі. Інструменти та концепції шкідливих програм залишаються тими ж самими.

1.2.1. Історія вірусів-вимагачів, їх еволюція та поширення

Оскільки віруси дуже відрізняються за тактикою, методами і процедурами і навіть за способами отримання початкового доступу, переміщення мережею і шифрування файлів, - необхідно розглянути багато типів програм-вимагачів, які еволюціонували з часом. На цій часовій шкалі [30] показано багато важливих моментів в історії програм-вимагачів, багато з яких розглядаються в цьому розділі дисертації:

- Перший випадок інфікування вірусом-вимагачем відбувся 1989 році AIDS-троян, також відомий як PC Cyborg [31]. Створений доктором Джозефом Поппом та розповсюджений серед 20000 учасників конференції Всесвітньої організації охорони здоров'я зі СНІДу. Випущено на 5¼ дискетах. Вимагав $189 викупу.
- GPCoder 2004/2005 Повідомлення, що відображалося на домашньому екрані користувача, спрямовувало його до файлу .txt, на робочому столі. Файл містив детальну інформацію про те, як сплатити викуп і розблокувати уражені файли. Вимагається викуп у розмірі $200 [32]. Очікувалося, що викуп буде сплачено через Western Union.
- Archiveus Trojan 2006. Атака переважно на Windows. Шифрував каталог MyDocuments. Перший вірус-вимагач, що використовує шифрування RSA. Після шифрування файлів, він відображав повідомлення з вимогою виконати певні дії (наприклад, купити певний продукт) для отримання ключа шифрування та розблокування файлів. Хоча Archiveus був досить простим у своєму виконанні, він став прототипом для майбутніх вірусів-вимагачів, включаючи такі відомі зразки, як CryptoLocker та WannaCry [32].
- Locker (або також відомий як FBI MoneyPak) був видом вірусу-вимагача, що почав своє широке поширення в 2009 році. Основною мішенню цього

типу вірусу стали мобільні пристрої. Locker використовувався для блокування доступу до пристрою користувача, вимагаючи оплату "штрафу" через сервіс FBI MoneyPak для розблокування [33]. Сповіщення про штраф часто містило інформацію, що нібито користувач порушив декілька законів, і для того, щоб уникнути кримінальної відповідальності, необхідно заплатити штраф. Примітні приклади цього типу вірусів включають WinLock та Reveton. WinLock блокував доступ до комп'ютера та вимагав від користувачів оплату за допомогою платіжних карток.

- CryptoLocker - це потужний вірус-вимагач, що з'явився в 2013 році. Він став відомим як перший вірус-вимагач, який вимагав від своїх жертв оплату в біткоїнах [33]. CryptoLocker шифрував файли на зараженому комп'ютері, використовуючи сильне шифрування RSA, а потім відображав повідомлення з вимогою оплатити "викуп" в біткоїнах для отримання ключа для розблокування файлів. Це було зроблено для того, щоб уникнути відслідковування платежів і приховати ідентичність зловмисників. Однією з причин, чому CryptoLocker був таким успішним, є те, що він змусив людей вперше дізнатися про криптовалюту і, в частині випадків, змусив їх придбати біткоїни. Цей вірус використовував різні методи поширення, включаючи спам-листи та ботнети. Цей вірус-вимагач призвів до значних фінансових втрат для багатьох організацій та індивідуальних користувачів і спонукав спільноту кібербезпеки до розробки нових методів захисту від таких атак.
- CryptoWall, що з'явився в 2014 році, став одним з найбільш руйнівних вірусів-вимагачів у світі. Він використовував вразливість в Java для зараження систем користувачів. Подібно до CryptoLocker, CryptoWall шифрував файли користувача та вимагав від жертви викуп у вигляді біткоїнів за ключ для розблокування. Однак CryptoWall істотно відрізнявся від CryptoLocker за своєю методологією поширення та інфраструктурою командування і контролю.

CryptoWall використовувався для атаки на майже 1,000 жертв, викликаючи оцінювані втрати в щонайменше $18 мільйонів [34].

- Locky став першим вірусом-вимагачем, що отримав широке поширення. З'явившись в 2016 році, він почав великомасштабну кампанію фішингу, відправляючи до 500,000 листів щодня. Locky шифрував файли користувача і вимагав від жертв відшкодування у вигляді біткоїнів за ключ розшифрування [35]. В 2016 році з'явилося декілька інших вірусів-вимагачів, що стали відомими:
    - Cerber: Вірус, що розмовляє з жертвами, використовуючи синтезатор мови, щоб озвучити вимоги про викуп.
    - Jigsaw: Знаменитий тим, що поступово видаляв файли з зараженої системи, поки викуп не було сплачено.
    - TeslaCrypt: Спеціалізувався на шифруванні файлів відеоігор.
    - SamSam: Цільово атакував великі організації, включаючи міські уряди.
    - Petya: Замість шифрування файлів, цей вірус шифрував мастер-завантажувач жорсткого диску, заблокувавши доступ до операційної системи.
- WannaCry, що з'явився в 2017 році, був одним з найбільш руйнівних вірусів-вимагачів у світі. Він атакував приблизно 200,000 комп'ютерів у 15 країнах. Відомо, що ця атака використовувала вразливість в Windows, відому як EternalBlue, яка була викрадена з Агентства національної безпеки США [36]. Представники США та Великої Британії заявили, що за атакою WannaCry стояла Північна Корея. Вірус NotPetya був варіантом вірусу Petya, який цільово атакував Україну, включаючи Національний банк України. Він став відомим своїм широким поширенням та великими збитками. Представники США оцінили збитки від вірусу-вимагача NotPetya більш ніж в $10 мільярдів. Особливість NotPetya полягає в тому, що, хоча він мав зовнішній вигляд вірусу-вимагача, його основною метою було не шифрування файлів для отримання викупу, а збитки великого масштабу.

- DarkSide є досить новим вірусом-вимагачем, який вперше з'явився в 2020 році і отримав значну увагу в 2021 році після атаки на Colonial Pipeline, одну з найбільших нафтопровідних компаній у Сполучених Штатах [37]. Атака спричинила тимчасове припинення операцій компанії, змусивши її зупинити всі потоки трубопроводу на шість днів. Це призвело до серйозних проблем з постачанням палива на східному узбережжі США. Colonial Pipeline вирішила сплатити викуп у розмірі 4.4 мільйонів доларів у біткоїнах, щоб відновити доступ до своїх систем.
- SamSam вперше з'явився в 2016 році, і він відрізнявся від самого початку. Його не доставляли за допомогою фішингу. Натомість SamSam експлуатував уразливості в JBOSS та шукав відкриті сервери Remote Desktop Protocol (RDP), щоб запустити атаки на паролі з перебором для отримання доступу. На відміну від сучасних груп вірусів-вимагачів, SamSam не встановлював вірус-вимагач на одну машину. Замість цього він використовував різні інструменти та експлуатації для поширення по мережі жертви, як тільки він мав доступ до одного хоста, і встановлював вірус-вимагач на якомога більше машин. Протягом кількох років SamSam вдарив по кількох видатних цілях, найбільше помітно - по медичному центру Hollywood Presbyterian у Лос-Анджелесі та місту Атланта. Атака вірусу-вимагача на Атланту на кілька тижнів вивела з ладу міські послуги та коштувала близько 17 мільйонів доларів на відновлення. За час свого багаторічного існування вважається, що SamSam зібрав майже 6 мільйонів доларів викупу [38].

1.2.2. Поширення вірусів-вимагачів

Шкідливе програмне забезпечення може використовувати такі техніки розповсюдження:

- Спам та фішинг

- Інфіковані веб-сайти
- Leteral Movements

Вищезазначені методи не обмежуються лише вірусами-вимагачами, але також поширюються на інші види шкідливого програмного забезпечення.

Атаки через пошту (фішинг та спам) [39]. Електронні листи можуть використовуватися як засоби атаки. Додатки та URL-адреси часто використовуються для доставки шкідливого програмного забезпечення. Електронні листи, пов'язані зі шкідливими намірами, називаються спамом, фішингом і так далі. Спам - це небажані електронні листи, які часто надсилаються великій кількості електронних адрес [39]. Спам може бути навмисним або ненавмисним. Більшість спаму призначена для рекламних цілей. Іноді продукт, який рекламується в спамі, є підробленим. Спам можна розподілити на декілька типів залежно від змісту листа:

- Фішинг
- Спір-фішинг
- Атака “водопій”
- Китобойство
- Фішинг-клонування

Фішинг - це вид спаму. Він також включає в себе надсилання шкідливих посилань або вкладень жертві за допомогою соціальної інженерії. Однією з цілей фішингу є крадіжка паролів. Фішингові листи, які включають крадіжку паролів, спонукають жертву вводити свої облікові дані на підробленому сайті. Зміст повідомлення у фішинговому спамі може обдурити жертву, і змусити до введення своїх облікових даних. У повідомленні йде попередження жертві, що якщо вона не увійде на сайт, то її обліковий запис буде заблоковано. Можуть бути й інші типи повідомлень, які можуть бути спокусливими, наприклад, виграш призів і так далі [40]. Професіонали в галузі безпеки класифікували фішинг на основі змісту електронного листа та жертви.

Спір-фішинг - це атака фішингу, при якій цілюється окрема особа, організація або група. Мета нападника може вар'юватися від крадіжки конфіденційних даних до фінансового шахрайства [41].

Атака “водопій” - це добре спланована цільова атака. Зловмисник збирає інформацію про жертву. Хакер виявляє вразливості на веб-сайті, проводячи тестування на проникнення. Потім він використовує помилку на веб-сайті і компрометує його. Наступного разу, коли жертва відвідує сайт, існує шанс зламу [42].

Китобойство, також відоме як атака “шахрайство з CEO”, - це атака фішингу, призначена для пастки старших керівників організації [42]. До керівників входять генеральні директори та віце-президенти, які володіють конфіденційною фінансовою та іншою бізнес-інформацією. Метою атаки може бути фінансовий збиток або отримання конкурентної інформації.

Фішинг-клонування - це інший вид фішингу. Його також називають обманним фішингом [43]. У цьому виді фішингу нападник копіює легітимний лист, який був раніше надісланий жертві. Електронні листи з посиланнями або вкладеннями є типовими для цього виду фішингу. Зміст листа залишається таким самим, окрім вкладення або посилання, яке замінюється на шкідливе. Лист надсилається жертві зі спуфінгової електронної адреси. Спуфінгова електронна адреса дуже схожа на реальну електронну адресу. Наприклад, dave@abcd.com можна змінити на deva@abcd.com. Жертва, ймовірно, не зверне увагу на електронну адресу і подумає, що це прийшло від відомого відправника, і в результаті може натиснути на шкідливе посилання.

Існує багато інших форм фішингу, і люди використовували різні термінології для його опису. Однак будь-яка атака фішингу має потенціал перенести шкідливе ПЗ і, отже, віруси-вимагачі.

Нижче наведено приклад фішингового листа, який стверджує, що в ньому міститься інформація про голосову пошту жертви:

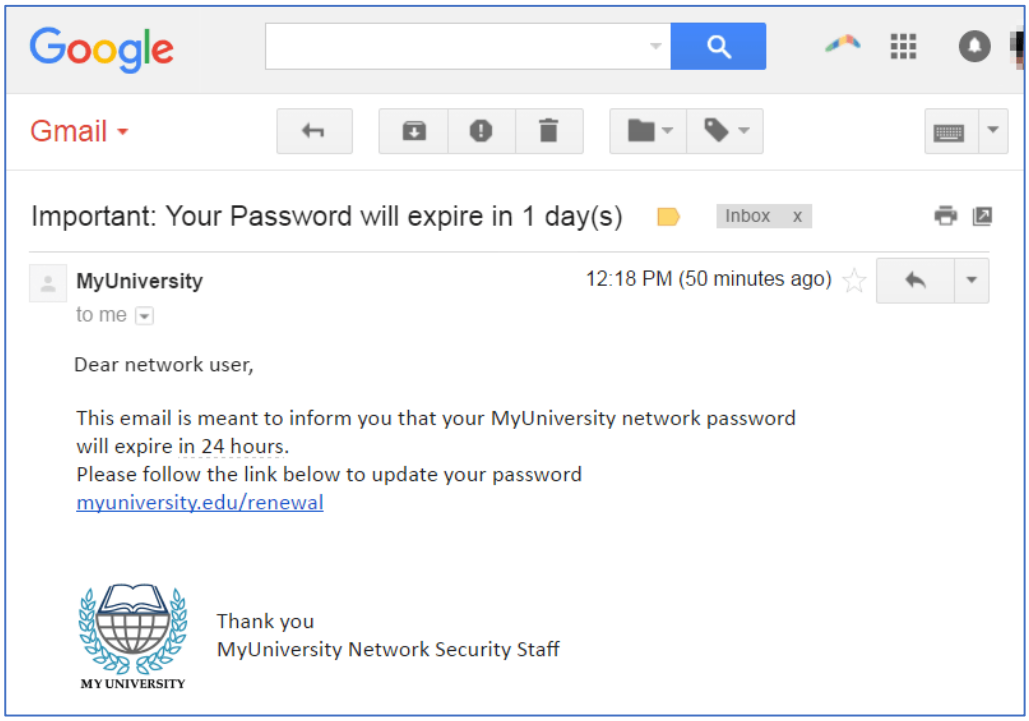


Рис. 1.5 Приклад фішинг листа

Лист на попередньому знімку екрану має вкладення. Відомо, що вкладення VBScript (скрипт Visual Basic) у цьому листі завантажує рансомвар Locky. Після того як вірус-вимагач попав у мережу за допомогою фішингу йому необхідно поширитися у внутрішній мережі на всі інші критичні станції.

Leteral Movements (Бічний рух). Поширення вірусу-вимагача або будь-якого іншого шкідливого ПЗ з одного комп'ютера на інший в тій же мережі називається боковим рухом. Боковий рух - це найновіший метод рансомвару [44]. Якщо в організації заражено один комп'ютер, він може поширити інфекцію на інші комп'ютери в організації. Деякі з останніх атак використовували наступні техніки для бокового поширення в мережі:

- Експлуатація слабких паролів, які використовуються в системах тієї ж мережі або паролів, які використовуються в системах за замовчуванням
- Експлуатація вразливостей в різних системах, які використовуються в мережі, таких як Server Message Block (SMB) [45]

- Неправильне використання адміністративних інструментів Windows, таких як Remote Desktop, PsExec та WMI [46]
- Інфекція файлів
- Автоматичне відтворення

Рис. 1.6. Загальні етапи Leteral Movement

Щоб поширитися через мережу, шкідливому ПЗ потрібно виявити комп'ютери в мережі. Потім вірус може спробувати отримати доступ до інших комп'ютерів, експлуатуючи вразливості в службах або комп'ютерах, що присутні в мережі. Іноді він може використовувати прості техніки, такі як використання типових паролів, які використовуються пристроями та службами в мережі. Шкідливе ПЗ намагається перерахувати пристрої та служби в мережі. Active Directory містить інформацію про користувачів, сервери та інші ресурси, такі як принтери, сканери та спільні папки з файлами в мережі. Адміністратори можуть контролювати мережу за допомогою Active Directory [47]. Якщо шкідливе ПЗ може отримати доступ до Active Directory в мережі Windows, то воно може виявити інші комп'ютери в мережі. Вірус Samas, з яким стикалися в середині 2017 року, - це одне таке шкідливе ПЗ, яке виявляє комп'ютери в мережі за допомогою Active Directory,

а потім пробує стандартні імена користувачів та паролі на цих комп'ютерах для поширення на решту комп'ютерів [48].

Після виявлення комп'ютерів у мережі, шкідливе ПЗ намагається поширитися на них. Одним із способів поширення є неправильне використання адміністративних інструментів, а інший - експлуатація вразливостей в програмному забезпеченні, встановленому на цих системах. PsExec та Windows Management Instrumentation (WMI) - це адміністративні інструменти, які можна використовувати для виконання команд на віддалених комп'ютерах. PsExec - це інструмент, який за замовчуванням недоступний на Windows. NotPetya має копію виконуваного файлу PsExec, щоб він міг використовувати його за потреби. WMI є частиною Windows. Для виконання команди на віддаленій машині обидва PsExec та WMI потребують облікові дані (ім'я користувача та пароль) для віддаленої машини. NotPetya містить з собою інший інструмент, який називається Mimikatz. Mimikatz - це хакерський інструмент, який може отримувати паролі з віртуальної пам'яті процесів на системі [49]. Після отримання пароля NotPetya може скопіювати себе на віддалену машину, а потім виконати копію за допомогою PsExec/WMI.

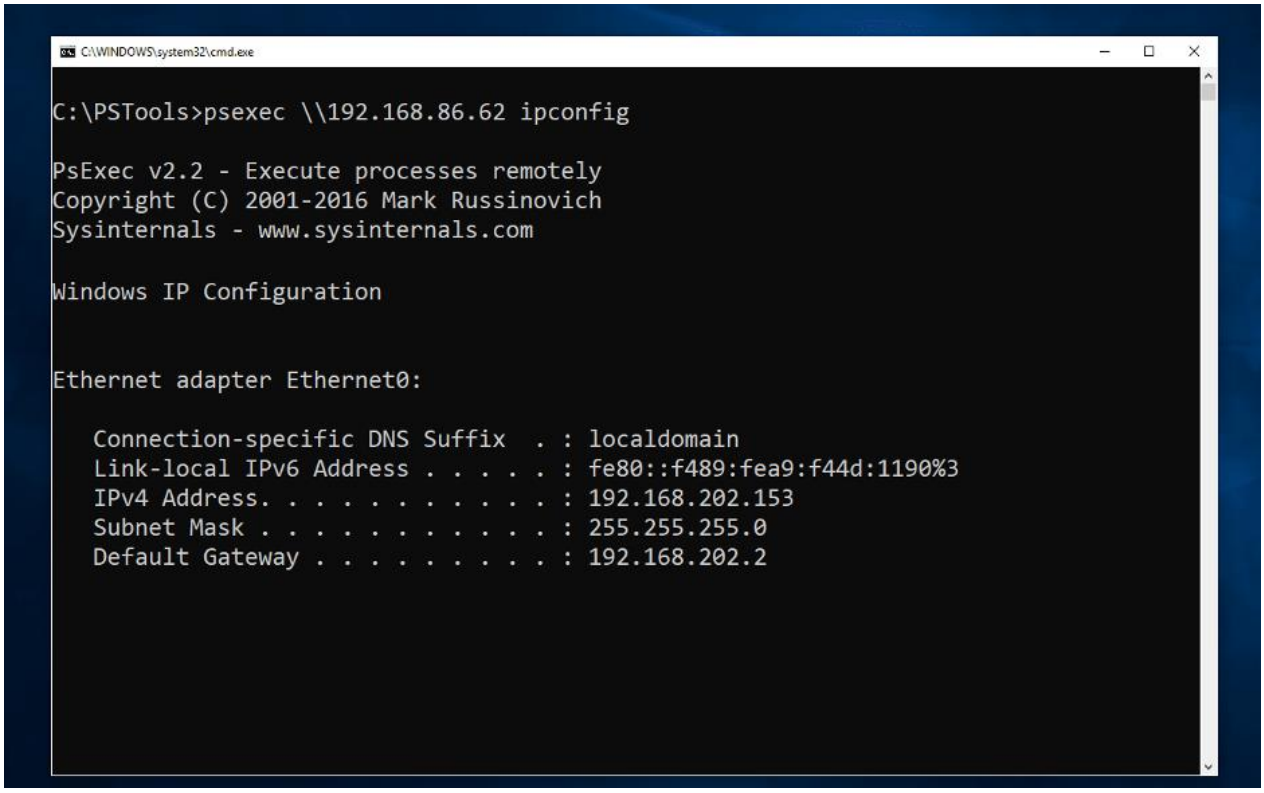


Рис. 1.7. PsExec утиліта

1.3. Основні моделі і методи виявлення і протидії програмам-вимагачам. Аналіз методів

Основними моделями і методами виявлення і протидії програмам-вимагачам в сучасній комп'ютерній безпеці є статичний аналіз, динамічний аналіз та проактивні методи виявлення.

Статичний аналіз. Зосереджується на вивченні програмного коду вірусів без їх виконання. Цей підхід включає аналіз хешів, рядків або використання машинного навчання для класифікації шкідливого коду. Проте, він може виявитися неефективним проти вірусів, які використовують техніки обфускації коду. Статичний аналіз полягає у визначенні характеристик файлу, таких як його тип та інші особливості, які можна ідентифікувати без його запуску. [50].

Дослідники антивірусів збирають багато варіантів однієї й тієї ж сім'ї шкідливого ПЗ, виявляють спільні статичні властивості в них і створюють цифровий підпис. Статичні властивості, які використовуються у підписі, можуть включати хеш певних областей файлу, властивості, розмір тощо. Але оскільки штами часто варіюються статично, антивірусні продукти повинні регулярно оновлювати свої підписи.

Динамічний аналіз - це метод, що включає виконання вірусів в контрольованому середовищі, наприклад, в сендбоксі, та спостереження за їхньою поведінкою, такою як використання ресурсів системи, мережева активність та зміни в системі. На відміну від статичного аналізу, динамічний аналіз може виявити віруси, які використовують обфускацію коду, але він є більш ресурсомістким і потребує більше часу [51].

Динамічний аналіз є одним з найважливіших етапів аналізу шкідливого ПЗ. Окрім цього можна назвати динамічний аналіз - аналізом поведінки, оскільки даний тип аналізу описує, що робить шкідливий код, іншими словами, поведінку файлу .exe, або зміни, які відбулися в системі після запуску даного файлу.

Кожен з цих методів має свої переваги і недоліки, і жоден з них не може гарантувати 100% захист від вірусів-вимагачів. З цих причин, для вирішення проблеми виявлення і протидії вірусам-вимагачам було вирішено використовувати технологію eBPF, що дозволяє відстежувати системні виклики на рівні ядра операційної системи, тим самим надаючи глибокий аналіз в діяльність процесів в системі.

Таблиця 1.1.

Порівняння видів вірусного аналізу

Вид аналізу	**Переваги**	**Недоліки**
Статичний аналіз	1. Швидкість: Статичний аналіз може бути виконаний	1. Обфускація: Віруси, що використовують обфускацію або

	швидко, оскільки він не вимагає виконання вірусу. 2. Безпека: Не викликає загрози, оскільки програма не запускається.	поліморфізм, можуть уникнути виявлення. 2. Відсутність контексту: Статичний аналіз не дає інформації про те, як програма буде виконуватися в реальному середовищі.
Динамічний аналіз	1. Детальний аналіз: Дозволяє отримати більше інформації про поведінку програми, включаючи використання ресурсів, мережеву активність, та зміни в системі. 2. Ефективність проти обфускації: Може виявити віруси, які використовують техніки обфускації коду.	1. Витрати часу: Динамічний аналіз зазвичай вимагає більше часу, ніж статичний. 2. Можлива загроза: Незважаючи на те, що аналіз відбувається в контрольованих умовах, існує небезпека, що вірус зможе проникнути поза ці обмеження.
Проактивні методи виявлення	1. Прогнозування нових загроз: Може виявити невідомі загрози, на основі характеристик відомих вірусів. 2. Адаптивність: Моделі машинного навчання можуть адаптуватися до змін в атаках.	1. Помилкове виявлення: Проактивні методи можуть іноді помилково виявляти легітимні програми як шкідливі. 2. Потреба в навчанні: Моделі машинного навчання потребують великого набору даних для навчання, що може бути викликом.

Методи протидії	1. Захист в реальному часі: Методи протидії можуть блокувати шкідливу діяльність в реальному часі, попереджаючи потенційну шкоду. 2. Ізоляція впливу: Техніки, такі як віртуалізація та контейнеризація, можуть ізолювати вплив програми-вимагача на систему.	1. Відсутність 100% захисту: Немає гарантії, що методи протидії завжди зможуть відбити атаку. 2. Ризик вразливостей: Безпека заснована на довірі до використовуваних рішень, які також можуть мати вразливості.

Ми можемо шукати такі зміни в системі Windows, щоб ідентифікувати шкідливе ПЗ:

- Зміни файлової системи
- Зміни реєстру
- Мережеве з'єднання
- Зміни процесів

Зміни файлів, реєстру і мережі є зрозумілими. Зміни процесів можуть включати зміни в створенні нового процесу та потоків. Зміни процесів також включають зміни в віртуальній пам'яті процесу.

Існують інструменти для моніторингу згаданих раніше змін. Набір віртуальних машин для аналізу шкідливого ПЗ повинен мати принаймні один інструмент, який належить до наступних категорій:

- Інструмент для моніторингу файлів: filemon, procmon
- Інструмент для моніторингу реєстру: regmon, procmon
- Інструмент для автозавантаження: autoruns

- Інструмент для моніторингу мережі: Wireshark, fakenet, Microsoft Network Monitor
- API-логер: StraceNT, пісочниці, Sysnalyzer API logger
- Інструмент для інспекції процесів: Process explorer, process hacker
- Пісочниці можна розглядати як еквівалент комбінації цих інструментів.

***Моніторинг файлів та регістрів*.** Зміни файлів та реєстру - одна з важливих подій, які використовуються для ідентифікації шкідливого програмного забезпечення. Microsoft Sysinternals надає для цього regmon та filemon [52]. Sysinternal пропонує procmon, який може охопити моніторинг реєстру та файлів.

Зображення 1.8 показує активність читання (ReadFile), закриття (CloseFile) та запису файлу (WriteFile) процесом vssvc.exe. vssvc.exe створює файл C:\Users\amohanta\Favorites\Microsoft Websites\Microsoft Store.url.readme_txt, потім записує до нього і після запису закриває його.

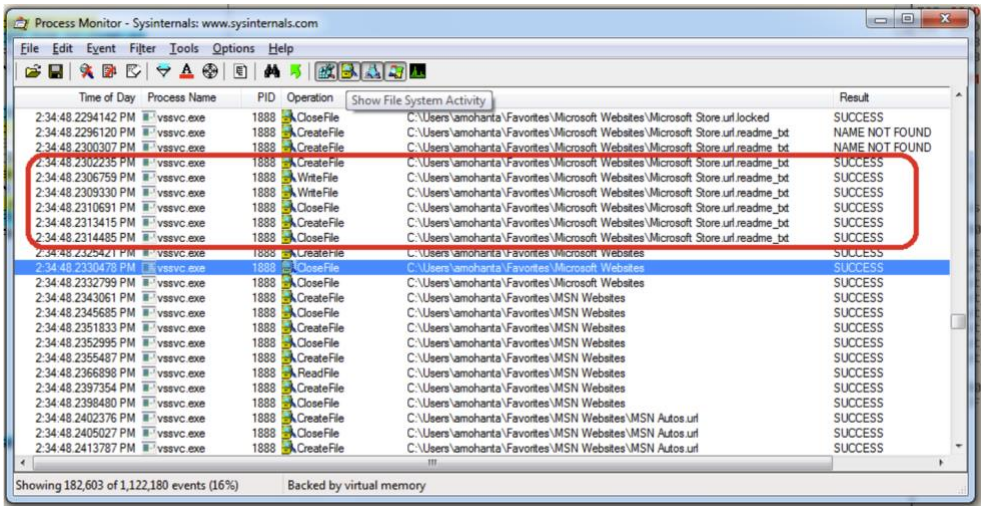


Рис. 1.8. Procmon демонструє файлову активність vssvc.exe процесу

Procmon також може моніторити діяльність процесів, таку як початок та вихід потоку, мережеві з'єднання та багато іншого. Важливо відфільтрувати активності,

оскільки багато системних процесів постійно вносять зміни до файлів та реєстру [52].

Microsoft Sysinternal Autoruns є корисним засобом для виявлення всіх процесів, які автоматично активуються під час старту Windows.

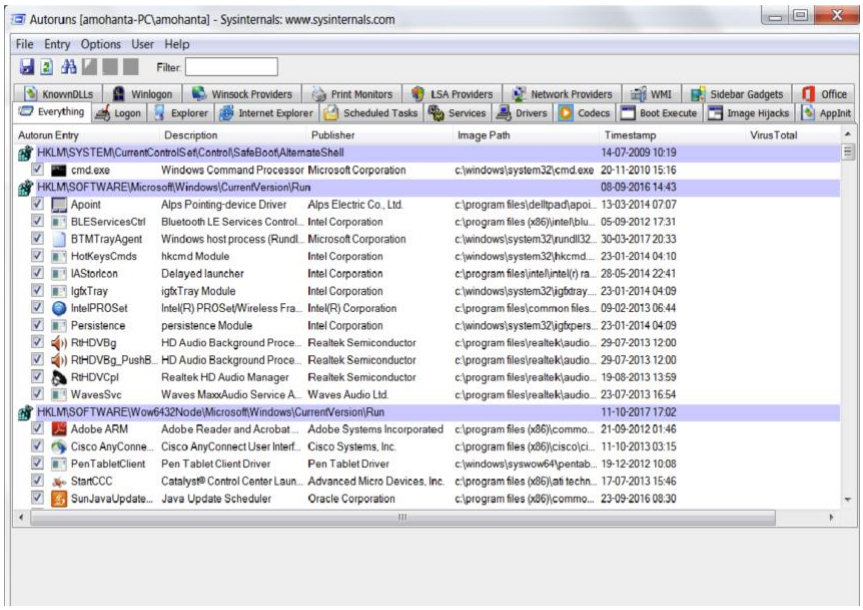
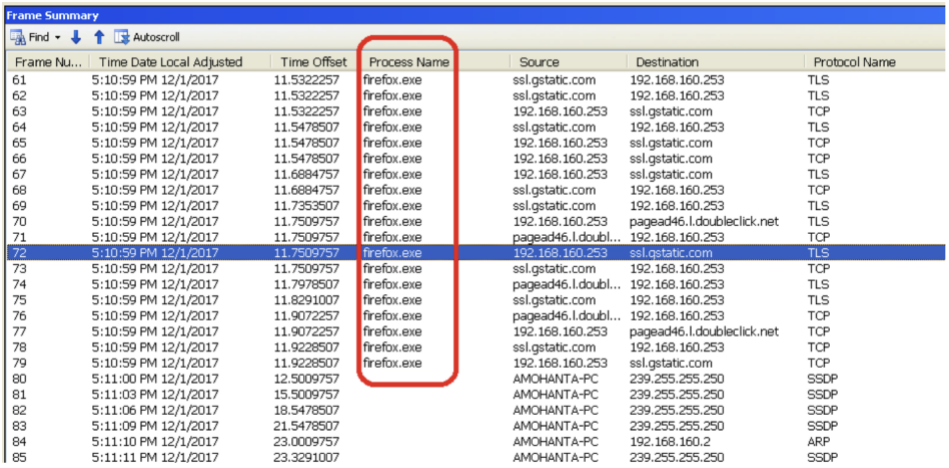


Рис. 1.9. Autoruns утиліта

Цей інструмент відображає багато інформації. Він включає записи, пов'язані з запуском, запланованими завданнями та службами, які можуть активувати шкідливі програми при запуску Windows. Цей інструмент також можна використовувати для усунення неполадок та проведення судової експертизи, щоб ідентифікувати небажане програмне забезпечення, яке може запускатися без відома користувача.

Більшість експертів, які працюють з мережами, знають про інструмент Wireshark [53]. Wireshark доступний як на Linux, так і на Windows. Microsoft надає інструмент Microsoft Network Monitor, який має можливість знімати пакети. Однією з додаткових переваг є те, що він показує нам, який процес створює мережеві

комунікації. Можна легко асоціювати мережеве з'єднання з процесом, що є додатковою перевагою порівняно з іншими інструментами моніторингу мережі:


Frame Summary

Find · Autoscroll

Frame Nu...	Time Date Local Adjusted	Time Offset	Process Name	Source	Destination	Protocol Name
61	5:10:59 PM 12/1/2017	11.5322257	firefox.exe	ssl.gstatic.com	192.168.160.253	TLS
62	5:10:59 PM 12/1/2017	11.5322257	firefox.exe	ssl.gstatic.com	192.168.160.253	TLS
63	5:10:59 PM 12/1/2017	11.5322257	firefox.exe	192.168.160.253	ssl.gstatic.com	TCP
64	5:10:59 PM 12/1/2017	11.5478507	firefox.exe	ssl.gstatic.com	192.168.160.253	TLS
65	5:10:59 PM 12/1/2017	11.5478507	firefox.exe	192.168.160.253	ssl.gstatic.com	TCP
66	5:10:59 PM 12/1/2017	11.5478507	firefox.exe	192.168.160.253	ssl.gstatic.com	TCP
67	5:10:59 PM 12/1/2017	11.6884757	firefox.exe	192.168.160.253	ssl.gstatic.com	TLS
68	5:10:59 PM 12/1/2017	11.6884757	firefox.exe	ssl.gstatic.com	192.168.160.253	TCP
69	5:10:59 PM 12/1/2017	11.7353507	firefox.exe	ssl.gstatic.com	192.168.160.253	TLS
70	5:10:59 PM 12/1/2017	11.7509757	firefox.exe	192.168.160.253	pagead46.l.doubleclick.net	TLS
71	5:10:59 PM 12/1/2017	11.7509757	firefox.exe	pagead46.l.doubl...	192.168.160.253	TCP
72	5:10:59 PM 12/1/2017	11.7509757	firefox.exe	192.168.160.253	ssl.gstatic.com	TLS
73	5:10:59 PM 12/1/2017	11.7509757	firefox.exe	ssl.gstatic.com	192.168.160.253	TCP
74	5:10:59 PM 12/1/2017	11.7978507	firefox.exe	pagead46.l.doubl...	192.168.160.253	TLS
75	5:10:59 PM 12/1/2017	11.8291007	firefox.exe	ssl.gstatic.com	192.168.160.253	TLS
76	5:10:59 PM 12/1/2017	11.9072257	firefox.exe	pagead46.l.doubl...	192.168.160.253	TCP
77	5:10:59 PM 12/1/2017	11.9072257	firefox.exe	192.168.160.253	pagead46.l.doubleclick.net	TCP
78	5:10:59 PM 12/1/2017	11.9228507	firefox.exe	ssl.gstatic.com	192.168.160.253	TCP
79	5:10:59 PM 12/1/2017	11.9228507	firefox.exe	192.168.160.253	ssl.gstatic.com	TCP
80	5:11:00 PM 12/1/2017	12.5009757		AMOHANTA-PC	239.255.255.250	SSDP
81	5:11:03 PM 12/1/2017	15.5009757		AMOHANTA-PC	239.255.255.250	SSDP
82	5:11:06 PM 12/1/2017	18.5478507		AMOHANTA-PC	239.255.255.250	SSDP
83	5:11:09 PM 12/1/2017	21.5478507		AMOHANTA-PC	239.255.255.250	SSDP
84	5:11:10 PM 12/1/2017	23.0009757		AMOHANTA-PC	192.168.160.2	ARP
85	5:11:11 PM 12/1/2017	23.3291007		AMOHANTA-PC	239.255.255.250	SSDP

Рис. 1.10. Моніторинг мережі

Це спрощує роботу аналітика, дозволяючи визначити, який процес створює з'єднання. При виявленні нехарактерних мережевих з'єднань від системних процесів Windows, таких як explorer.exe та winlogon.exe, може виникати підозра щодо введення шкідливого коду в процес.

Інструменти для ведення журналу API показують послідовність, в якій виконуваний файл викликає API. Запис для конкретного API може включати параметри, передані API, і значення, повернуті ним. StraceNT та apimonitor - це деякі інструменти, які можна використовувати для ведення журналу API [54].

Інструмент має зрозумілий для користувача інтерфейс та розширений функціонал, що сприяє аналітиці. Фіксація подій API є критичною для інтерпретації діяльності, що пов'язана зі змінами в системних процесах програмного забезпечення.

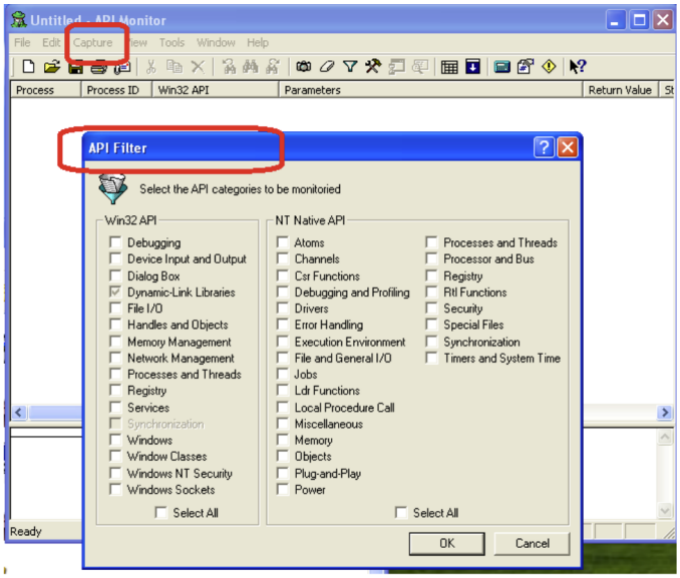


Рис. 1.11. Вікно API Filter

Можна налаштувати інструмент, перейшовши до меню “Capture” (Захоплення). Буде отримана можливість змінити значення API Filter (Фільтр API). Потім можна вибрати API, які необхідно вести в журнал, наприклад Registry (Реєстр). Можна почати вести журнал знову, перейшовши до меню “Capture” (Захоплення) та натиснувши на Capture API Events (Захоплення подій API):

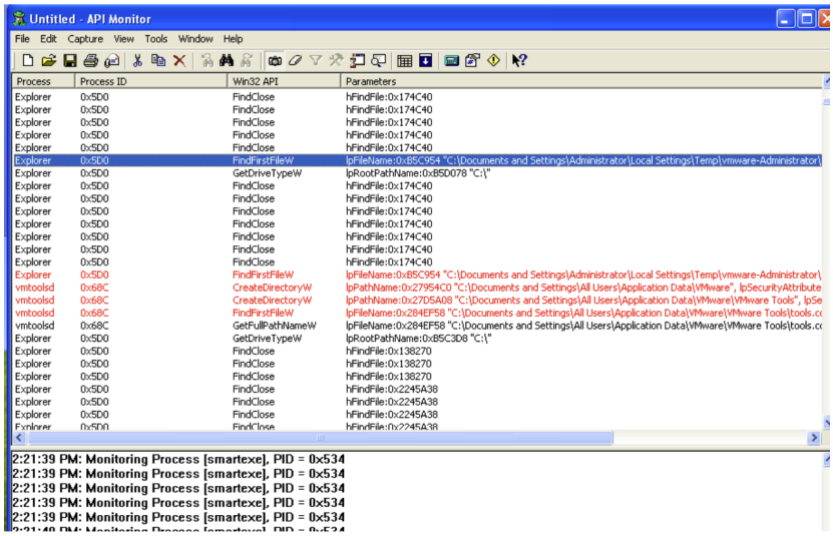


Рис. 1.12 Apimonitor логи

Аналітики з шкідливого ПЗ можуть використовувати журнали API, щоб знайти внутрішню роботу шкідливого програмного забезпечення.

Інспекція процесів є потужним методом [55]. У віртуальній пам'яті процесу можна перевірити присутні рядки. При аналізі рядків у файлі на диску, зазвичай, не виявляється нічого змістовного (статичний аналіз за допомогою інструменту рядків), оскільки шкідливе ПЗ залишається прихованим. Під час виконання запакованого шкідливого ПЗ створюється процес, у якому шкідливе ПЗ розпаковує необхідний для виконання та проведення шкідливої діяльності код у віртуальній пам'яті.

Отже, можна стверджувати, що фактичні рядки, які стосуються шкідливого ПЗ, спостерігаються у віртуальній пам'яті.

Process Explorer і Process Hacker використовується для перегляду рядків у віртуальній пам'яті процесу [55].

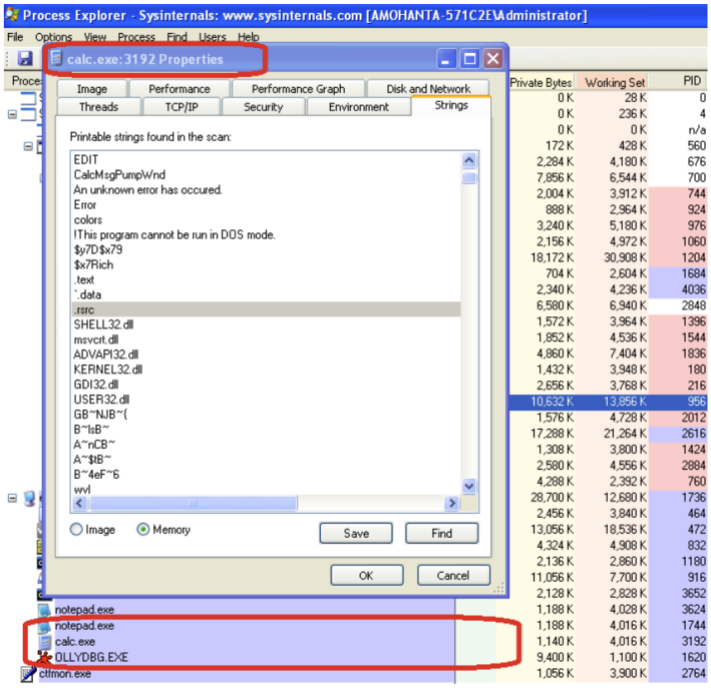


Рис. 1.13 Рядки у віртуальній пам'яті для calc.exe процесі

Вище наведене зображення Рис. 1.13. показує знімок процесу калькулятора Windows, calc.exe, в інструменті Process Explorer. Вікно програми відкривається при натисканні на запис процесу в Process Explorer. Далі можна перейти до вкладки Strings та натиснути на кнопку Memory, щоб переглянути рядки, збережені у

віртуальній пам’яті процесу. Натискання на кнопку Image дозволяє відобразити рядки, що знаходяться у файлі calc.exe на диску, подібно до того, як це робить інструмент strings, що використовується для статичного аналізу. Process Hacker є аналогічним інструментом, але надає додаткові деталі. [55].

1.3.1. Аналіз вірусів-вимагачів

Від шкідливого програмного забезпечення, яке призначено для шифрування файлів на машині, очікується така поведінка:

- Зміна файлів: велика активність модифікацій файлів, та активність пов'язана з багатьма розширеннями файлів.
- Зміни в реєстрі: це може бути подібно до іншого шкідливого програмного забезпечення.
- Мережева комунікація: Шкідливе програмне забезпечення намагається підключитися до свого сервера.
- Пам'ять процесу: Можна бачити повідомлення про вимогу викупу в пам'яті шкідливого програмного забезпечення.

Часто можна спостерігати список розширень файлів, які шкідливе програмне забезпечення планує зашифрувати. Також видно команди, що використовуються шкідливим програмним забезпеченням. Шкідливе ПЗ іноді змінює розширення файлів на власне розширення, наприклад, вірус Locky змінює розширення зашифрованих файлів на .locky [56]:

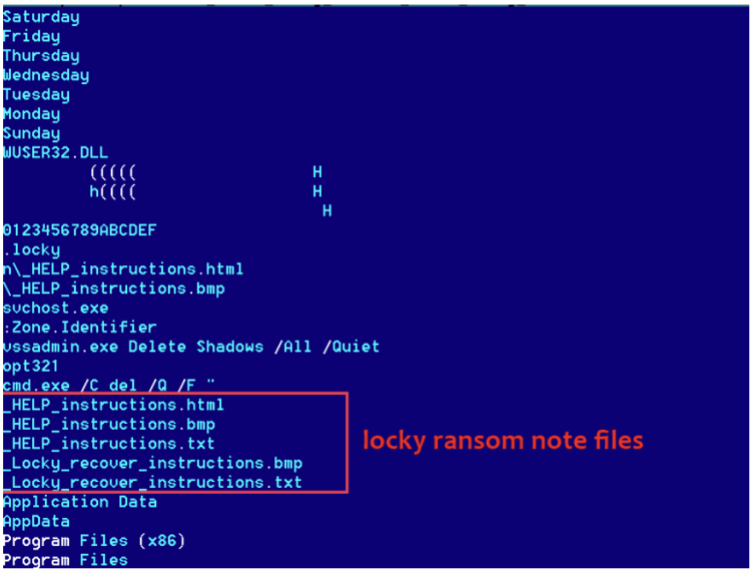


Рис. 1.14 Пам’ять Locky віруса-вимагача

У пам'яті шкідливого програмного забезпечення Locky видно інструкції з розширенням .locky, що достатньо для ідентифікації шкідливого програмного забезпечення як Locky. Окрім цього є присутніми рядки _HELP_instructions.html, _HELP_instructions.bmp та _HELP_instructions.bmp на скріншотах. Ці файли створюються Locky на машині жертви. Ці файли містять вимоги викупу, які містять повідомлення про викуп і говорять про те, як жертва повинна заплатити викуп.

Загальні методи виявлення та захисту. Жертва зобов'язана заплатити викуп. Загальні методи виявлення та захисту описувалися у контексті необхідного програмного забезпечення та пристроїв для захисту окремих осіб та організацій. Однак, ці засоби самі по собі не здатні захистити організацію без наявності відповідного процесу захисту. Необхідно, щоб адміністратори та керівництво переконались у дотриманні працівниками всіх необхідних процесів захисту. Нижче наведено перелік рекомендацій для адміністраторів:

- Операційна система повинна регулярно оновлюватися.
- Адміністратори повинні слідкувати за бюлетенями безпеки та відповідно оновлювати патчі.
- Адміністратори повинні переконатися, що працівники організації не використовують недозволене програмне забезпечення.

- Антивірусні підписи завжди повинні бути оновлені.
- Правила IPS, IDS та брандмауерів також слід регулярно оновлювати.
- Будь-які види послуг, якими користуються клієнти, такі як ваші веб-сайти, повинні бути належним чином протестовані на вразливості, щоб не допустити розміщення експлойтів.
- Слід регулярно створювати резервні копії даних.
- Адміністратори повинні належним чином встановити контроль доступу, тобто визначити, хто має право доступу до певних даних.
- Співробітники повинні бути навчені протистояти всім видам атак

Іноді для дуже малих організацій та стартапів купівля комерційного програмного забезпечення може бути економічно невигідною, але, врешті-решт, їм також потрібно захищати себе. Тому адміністраторів слід навчити налаштовувати програмне забезпечення з відкритим вихідним кодом, таке як cuckoo і snort, і вони повинні навчитися розгортати правила, доступні у відкритому доступі [57].

1.4. Аналітичний огляд існуючих підходів до забезпечення безпеки комп'ютерних мереж від програм-вимагачів

Наукова спільнота поступово відмовляється від традиційної мережево-орієнтованої безпеки на користь підходу, що заснований на аналізі поведінки процесів. Це дозволяє здійснювати контроль та ухвалення рішень, які опираються не на аналіз заголовків пакетів, а на детальне розуміння динаміки процесів. Якщо розглядати реальність, то важко знайти приклади, коли останнім часом було виявлено складні атаки, засновані на перехопленні пакетів.

Рис. 1.15 Традиційний моніторинг мережі та виявлення загроз. Моніторинг кібербезпеки

Рис. 1.16 Традиційний моніторинг мережі та виявлення загроз

Безпека комп'ютерних мереж є фундаментальною проблемою, яка вимагає постійної уваги і оновлення, оскільки загрози для кібербезпеки постійно еволюціонують. Виявлення і протидія програмам-вимагачам потребують спеціалізованих інструментів та підходів. Сучасні методи включають застосування мережевого моніторингу, систем виявлення вторгнень (IDS), систем запобігання вторгненням (IPS), систем захисту кінцевих точок (EPS) та інших технологій [58].

Однак, не всі ці системи є достатньо ефективними проти вірусів-вимагачів. Наприклад, традиційні антивірусні програми зазвичай залежать від визначення

загрози на основі сигнатур, що означає, що вони можуть виявляти тільки відомі варіанти вірусів-вимагачів. Проте, з урахуванням швидкого розвитку і постійного вдосконалення вірусів-вимагачів, цей підхід може бути недостатнім.

Системи виявлення вторгнень та системи запобігання вторгненням працюють на мережевому рівні і виявляють аномальну поведінку, що може вказувати на атаку [59]. Однак, ці системи також можуть бути недостатньо ефективні проти вірусів-вимагачів, особливо в тих випадках, коли віруси-вимагачі використовують техніки маскування для уникнення виявлення.

Виклик полягає в розробці більш передових підходів до виявлення та протидії вірусам-вимагачам, які зможуть адаптуватися до нових загроз і використовувати більш витончені методи виявлення. Однією з ключових сфер, що розвиваються в цьому напрямку є використання eBPF (Extended Berkeley Packet Filter), який дозволяє створювати більш гнучкі і динамічні правила виявлення загроз на основі поведінки.

Далі в цьому розділі буде розглянуто архітектуру SIEM та EDR рішень, аналіз сучасного стану кіберзлочинності в Україні та світі, обґрунтуємо вибір напряму дослідження та поставимо завдання дослідження.

1.5. Архітектура SIEM та EDR рішень

Системи управління подіями та інформацією безпеки (SIEM) та системи виявлення та відповіді на кінцеві точки (EDR) є ключовими інструментами в сучасному кібербезпеці [60]. Однак, їхня ефективність проти вірусів-вимагачів часто може бути обмеженою через використання традиційних методів виявлення загроз на основі сигнатур та аномалій. Архітектура SIEM базується на централізованому зборі та аналізі подій безпеки з різних джерел у мережі. SIEM-системи здатні агрегувати та корелювати великі обсяги даних, що допомагає

виявляти складні загрози та розробляти відповідь на них [60]. Однак, SIEM-системи можуть бути менш ефективними проти нових та невідомих вірусів-вимагачів, які не відповідають відомим сигнатурам загроз.

EDR-системи, з іншого боку, зосереджуються на захисті кінцевих точок - пристроїв користувачів, таких як комп'ютери та сервери. Вони використовують більш розширений набір інструментів для виявлення загроз, включаючи моніторинг поведінки, аналіз пам'яті та мережевий аналіз [61]. Проте, незважаючи на це, EDR-системи також можуть бути обмежені в своїй здатності виявляти та блокувати віруси-вимагачі, особливо ті, які використовують техніки обходу антивірусного захисту та виявлення. Слід зазначити, що архітектура SIEM та EDR не є взаємовиключними, і в ідеалі обидві системи повинні бути використані разом для найбільш повного захисту. Однак, з огляду на постійні та еволюючі виклики у сфері кібербезпеки, виникає необхідність у розробці нових стратегій та впровадженні технологій, таких як eBPF, що здатні доповнити та покращити застосування SIEM та EDR систем.

1.6. Аналіз сучасного стану кіберзлочинності в Україні та світі. Використання вірусів-вимагачів під час війни в Україні

1.6.1 Найбільші кримінальні групи сьогодення

Цей список постійно змінюється через різні фактори, станом на вересень 2023 року, деякі з ключових груп, активних у сфері, включали [62]:

1. REvil (Sodinokibi): Одна з найбільш активних груп, що використовують шкідливе ПЗ [63].

2. Conti: Ця група прославилася своїми складними атаками та високими вимогами щодо викупу [64].

3. DarkSide: Ця група здобула погану славу після запуску атаки на Colonial Pipeline, що спричинила значні перебої у постачанні палива в США у травні 2021 року [65].

4. LockBit 2.0: штам шкідливого ПЗ як сервісу (RaaS), який слідує за трендом подвійного вимагання викупу, тобто вони шифрують файли жертв і загрожують опублікувати вкрадені дані [66].

5. Avaddon: Ця група була однією з найбільш агресивних груп, вимагаючи викупу і загрожуючи витиком даних, якщо їх вимоги не були виконані [67].

6. Ryuk: Незважаючи на те, що ця група вже давно в сфері шкідливого ПЗ, група Ryuk продовжує становити значну загрозу, особливо для сектору охорони здоров'я [68].

7. DoppelPaymer: Відома високими вимогами до викупу і нападами на великі організації [69].

Ці групи створюють серйозні загрози через свою складність і масштаб своєї діяльності. Варто також зазначити, що атаки шкідливого ПЗ мають циклічний характер і часто надходять хвилями, тому періоди відносної тиші часто можуть бути відслідковані шквалом атак.

1.6.2 Тенденції у світі

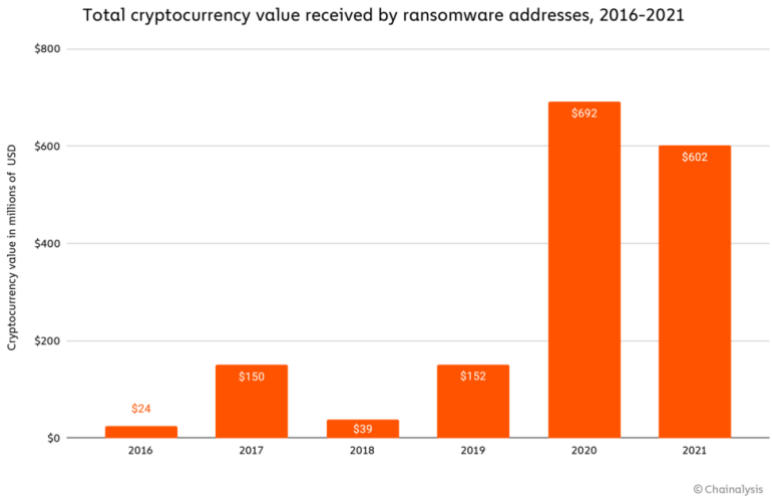


Рис. 1.17. Компанія Chainalaysis відображає загальну кількість криптовалюти отриману через віруси-вимагачі

На рисунку 1.17 зображено, що станом на сьогодні компанією Chainalaysis було виявлено у 2021 році платежів від програм-вимагачів на суму трохи більше $602 млн [70]. Однак, як і минулого року підраховано, що ця цифра занижена, і що реальна загальна сума за 2021 рік, ймовірно, буде набагато вищою. Насправді, незважаючи на ці цифри, окремі свідчення, а також той факт, що доходи від програм-вимагачів у першій половині 2021 року перевищили доходи за першу половину 2020 року, що 2021 рік зрештою виявиться ще більшим роком для програм-вимагачів.

Найактуальніші дані за перше півріччя 2023 свідчать про те, що активність зловмисників на шляху до побиття попередніх рекордів, оскільки зростає кількість як великих, так і малих платежів.

Згідно зі звітом компанії Chainalysis, що займається аналізом блокчейн-технологій, програми-вимагачі - єдина категорія криптовалютних злочинів, яка цього року демонструє зростання, тоді як всі інші, включаючи зломи, шахрайство, шкідливе програмне забезпечення, продаж матеріалів для зловживань, шахрайські магазини та доходи від даркнет-ринку, демонструють різке падіння [70].

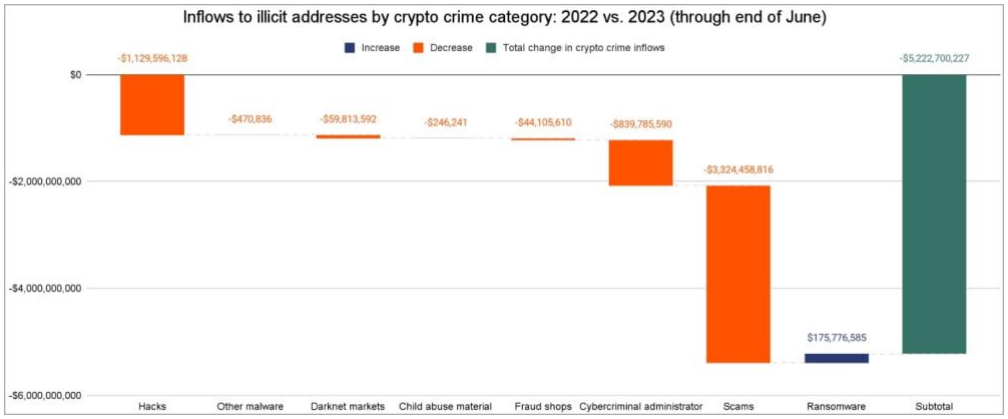


Рис. 1.18. Тенденції отримання незаконних доходів – порівняння 2022 та 2023 років (Chainanlysis)

Програми-вимагачі - це єдина форма криптовалютних злочинів, яка зростає у 2023 році, - йдеться у звіті Chainalysis [71]. Фактично, зловмисники, які використовують програми-вимагачі, досягли другого за величиною показника в історії, вимагаючи щонайменше 449,1 мільйона доларів США до червня.

Як показано на наведеному нижче графіку Chainalysis, сукупний річний дохід від програм-вимагачів за 2023 рік досяг 90% від загального показника за 2022 рік у першій половині року.

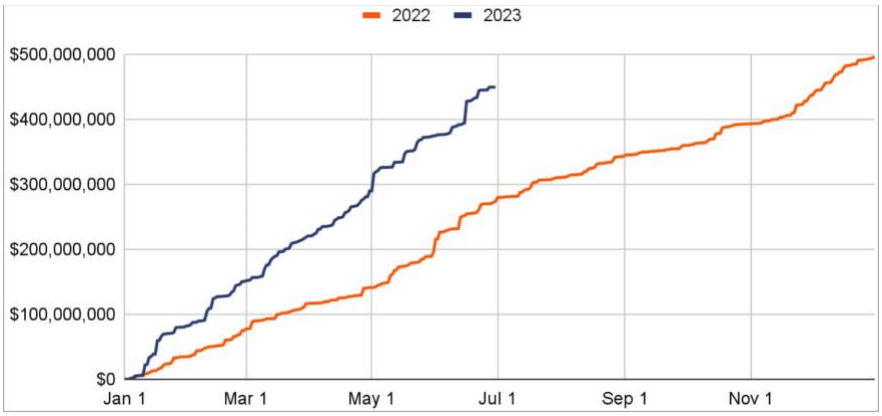


Рис. 1.19. Кумулятивне отримання доходів від програм-вимагачів – порівняння 2022 та 2023 років (Chainanlysis)

Порівняльний аналіз даних 2022 року з 2023 роком. Якщо темпи зростання доходів збережуться на цьому рівні, то у 2023 році зловмисники отримають від

жертв близько 900 мільйонів доларів США, що трохи нижче рекордного показника 2021 року - 940 мільйонів доларів США. Аналітики вважають, що рушійною силою такого різкого зростання доходів є так зване "полювання на велику дичину", оскільки кіберзлочинці повернулися до полювання на великі організації, у яких можна вимагати великі суми грошей [71]. Це відображено на графіку розподілу виплат викупів, який за перше півріччя 2023 року демонструє безпрецедентне зростання правої частини, що відповідає великим платежам. BlackBasta, LockBit, ALPHV/Blackcat і Clop лідирують як основні отримувачі великих платежів, причому середній розмір платежу Clop становить $1,7 млн, а медіана - $1,9 млн.

Clop відповідальний за дві масові хвилі атак, які використовували дві вразливості нульового дня в інструментах для передачі файлів: GoAnywhere від Fortra в першому кварталі року та MOVEit Transfer від Progress в другому.

Фактично, кампанія Clop's GoAnywhere, яка включала 129 атак, зробила березень 2023 року рекордним місяцем, як повідомляла тоді NCC Group. Хвиля атак MOVEit вже є більш масштабною і налічує 267 жертв, причому щотижня на сайті вимагачів Clop з'являється все більше інформації. Тенденція до зростання в першому півріччі 2023 року спостерігається і на іншому кінці спектру: платежі за викуп відносно невеликі, призначені для широкого кола жертв, які включають окремих осіб та невеликі підприємства, таким як Dharma, Phobos і STOP/DJVU [72].

Таблиця 1.2.

Кількість платежів

Назва штаму віруса-вимагача	**Середній платіж за 2023 рік**	**Медіанний розмір платежу за 2023 рік**
Dharma	$265	$275
Phobos	$1719	$300
Stop/djvu	$619	$563
BlackBasta	$	$147106

ALPHV/Blackcat	$1 504 479	$305 585
CIop	$1730486	$1946335

Екосистема кіберзлочинності продовжує стрімко розвиватися. Одним із визначальних факторів ландшафту програм-вимагачів у 2022-23 роках є триваюча війна в Україні, яка має особливий вплив з огляду на високу концентрацію кіберзлочинних угруповань з Росії та її сусідів. У перші місяці після вторгнення змінився масштаб та напрям програм-вимагачів та інших кіберзлочинів, що свідчить про зміну пріоритетів та обмежень для суб'єктів, що діють у цьому просторі. Хоча конфлікт не призвів до повномасштабної кібервійни, яку прогнозували деякі коментатори, кібератаки проти України були постійними та значними як до, так і під час вторгнення, а також були активною частиною власних оборонних операцій України.

Одним із можливих побічних ефектів війни є переорієнтація зусиль кіберзлочинців з Росії та близького зарубіжжя у двох напрямках. По-перше, існує припущення, що зусилля деяких злочинців були переорієнтовано з прибуткових кіберзлочинів, таких як атаки з вимогами викупу, на участь у воєнних операціях. Крім того, незважаючи на те, що кібератаки з вимогами викупу, як правило, відбуваються у воєнний час, в Україні не припиняються.

Крім того, незважаючи на те, що діяльність програм-вимагачів з Росії та її близького зарубіжжя триває, ці кіберзлочинці, як видається, також розширюють свої цілі, приділяючи більше уваги Глобальному Півдню, в тому числі переорієнтовуючись на цілі в Азії та Латинській Америці і відходячи від об'єктів критичної інфраструктури та інших чутливих об'єктів в країнах НАТО [73, 74]. Ця переорієнтація від об'єктів критичної інфраструктури та інших чутливих об'єктів в країнах НАТО може бути викликана бажанням уникнути інцидентів, які могли б посилити напруженість у відносинах між Росією та країнами-членами НАТО.

Довгострокові наслідки вторгнення для екосистеми кібербезпеки залишаються невизначеними. Кілька невирішених питань включають те, як цей конфлікт впливає на безпечні гавані для кіберзлочинців і як буде виглядати екосистема кіберзлочинності в міру розвитку конфлікту між Україною і Росією. Більше того, в міру того, як конфлікт розгортається, додаткові звіти розмивають наше розуміння загальної спрямованості програм-вимагачів. Як зазначалося раніше, образ ситуації досі залишається незавершеним.

1.6.3. Брокери початкового доступу

Згідно з оцінками Recorded Future, у 2020 році було вчинено 65 000 атак вірусів-вимагачів [75]. Це просто надто багато жертв для того, щоб навіть масивна мережа учасників та їх афілійованих структур могла отримати доступ, вкрасти файли і розгорнути інфраструктуру вірусів-вимагачів. Значущість полягає у скануванні Інтернету з метою ідентифікації вразливих систем. Деякі брокери початкового доступу(IAB/s) спеціалізуються на використанні чужих облікових даних, коли зловмисник намагається увійти за допомогою поширених комбінацій імені користувача/пароля, використовуючи метод грубої сили в швидкому порядку, тоді як інші фокусуються на повторному використанні облікових даних, коли зловмисник знаходить комбінації імені користувача/пароля на підпільних ринках і намагається використати їх на цілі. Роль IAB в атаці рансомваре - отримати і утримувати початкову точку опори. Вони потім продають доступ до акторів рансомваре за середньою ціною $5,400. Багато IAB спеціалізуються на експлуатації інших вразливих систем, таких як:

- Pulse Secure VPN [76]
- Citrix [77]
- Fortinet VPN [78]

- SonicWall Secure Mobile Access [79]
- Palo Alto VPN [80]
- F5 VPN [81]

Для груп вірусів-вимагачів відмивання грошей є складним. Актори вимагачів перейшли від проведення кількох простих транзакцій, які приховують їхні гроші, до того, як очистити мільйони доларів зібраних викупів. Коли в червні 2021 року було затримано відмивачів грошей від групи рансомваре Clop, було повідомлено, що вони успішно відмили більше ніж 500 мільйонів доларів зібраних викупів [82].

Оператори вірусів-вимагачів перекидають більшість коштів, викрадених у своїх жертв, до основних обмінників, обмінників високого ризику. Декілька операторів RaaS роблять акцент на тому, що їх платіжний портал інтегровано з міксуючими сервісами як функцію для приваблення афілійованих талантів. 73% всіх коштів, контрольованих акторами рансомваре, були відправлені всього на 83 депозитні адреси до червня 2021 року.

У 2020 році Chainalysis Inc. виявила 100 брокерів OTC, які спеціалізуються на перекиданні грошей для кіберзлочинців [83]. Брокери OTC - це особи або компанії, які володіють великими сумами криптовалюти. Коли трейдер хоче обміняти криптовалюту на інший вид криптовалюти або фіатну валюту анонімно, він може домовитися про встановлену ціну з OTC, який потім здійснює транзакцію.

Брокери експлойтів. Дослідники давно відмітили, що зловмисники, пов'язані з програмами-вимагачами, активно купують доступ до експлойтів. Ця практика дійсно стала відомою у випадку атаки віруса-вимагача Kaseya REvil [84]. Під час цієї атаки REvil, або один з його партнерів, використав раніше невідому вразливість (зазвичай називають її вразливістю "нульового дня") проти програмного забезпечення Virtual System Administrator (VSA) від Kaseya. Kaseya VSA - це програмне забезпечення для віддаленого управління, яке часто використовують постачальники управління сервісами (MSP) для віддаленого адміністрування та захисту своїх клієнтів, особливо менших клієнтів з обмеженим IT або безпековим

персоналом [84]. В цьому випадку мережу Kaseya ніколи не було скомпрометовано - афіліат REvil використав вразливість для експлуатації MSP, які використовували інструмент VSA від Kaseya. Навіть тоді, афіліат не шифрував мережі MSP, а використовував свій доступ для розгортання рансомваре до клієнтів MSP.

На момент написання цього, атака на Kaseya VSA є найбільш відомим використанням експлойту кіберзлочинною групою рансомваре. Але групи рансомваре регулярно поєднують експлойти в рамках своєї стратегії атаки. Зазвичай вони цілять на відомі вразливості для експлуатації, а не на вразливості нульового дня. Відомі експлойти все ще працюють, оскільки групи рансомваре і IAB розраховують на повільний цикл патчів, який утримують багато організацій.

У книзі " Ось так мені розповідають про кінець світу" журналістка Ніколь Перлрот детально описує ріст ринку експлойтів і конкуренцію між акторами держав-націй по отриманню вразливостей нульового дня та їхньої експлуатації [85]. Через великі суми грошей, які групи рансомваре заробили за останні кілька років, особливо з появою RaaS, вони здатні конкурувати з багатьма акторами держав-націй по отриманню експлойтів.

1.7. Обґрунтування вибору напряму дослідження та постановка завдань

Вибір напрямку дослідження для даної дисертації зумовлений кількома ключовими факторами:

1. **Актуальність проблеми**: Програми-вимагачі є однією з найбільш значущих загроз для кібербезпеки, що тільки посилюється з часом.

2. **Потреба в реальному часі**: Сучасні методи виявлення часто можуть бути реактивними, тобто вони забезпечують захист тільки після того, як атака вже відбулася.

3. **Технічна можливість**: Платформа eBPF надає засоби для ефективного моніторингу системи на низькому рівні, що може дозволити більш точне і швидке виявлення програм-вимагачів.

4. **Локальний та глобальний контекст**: В Україні та за її межами зростає активність кіберзлочинності, в тому числі використання вірусів-вимагачів у рамках кібервійни.

5. **Інтеграція з існуючими системами**: eBPF може бути інтегрований з існуючими SIEM та EDR рішеннями, що робить цей підхід практично можливим.

Постановка завдань

На основі обґрунтування вибору напрямку дослідження, можна сформулювати наступні завдання:

1. **Аналіз сучасного стану методів виявлення програм-вимагачів**: Вивчити існуючі методики та інструменти, їх переваги та недоліки.

2. **Дослідження можливостей eBPF**: Вивчити архітектуру та можливості eBPF у контексті кібербезпеки та виявлення програм-вимагачів.

3. **Розробка модулів на базі eBPF для виявлення програм-вимагачів**: Створити прототипи модулів для моніторингу системних викликів, файлів, мережевого трафіку та ін.

4. **Тестування та оцінка ефективності розроблених модулів**: Виконати практичні тести для оцінки швидкодії, точності та інших метрик.

5. **Інтеграція з існуючими системами безпеки**: Дослідити можливість інтеграції розроблених модулів з SIEM та EDR системами.

6. **Аналіз та оцінка захищеності**: Провести комплексну оцінку ефективності пропонованого рішення в реальних умовах.

7. **Дослідження ефективності та переваг розробленої методології**: Провести аналіз результатів та порівняти їх з альтернативними методами виявлення програм-вимагачів.

Виконання цих завдань дозволить сформулювати обґрунтовані висновки щодо ефективності використання eBPF для виявлення програм-вимагачів в реальному часі, а також надати практичні рекомендації для їх застосування в сучасних комп'ютерних системах.

Висновки до 1 розділу

Перший розділ дисертації зосереджується на виявленні та протидії програмам-вимагачам. На основі детального аналізу та дослідження, можна сформулювати наступні висновки:

1. Комплексний історичний огляд вірусів-вимагачів виявив еволюційний характер цих загроз і підкреслив швидкість з якою зловмисники адаптують свої методики до змін у кіберзахисті. Це показало, що необхідно постійно оновлювати методи виявлення та відповіді на загрози, а також розробляти проактивні механізми для визначення потенційно небезпечної поведінки програм-вимагачів до того, як вони заподіють шкоду.

2. Розгляд функціональності існуючих систем SIEM та EDR продемонстрував, що вони важливі для забезпечення базового рівня кіберзахисту, але також мають значні обмеження, зокрема у виявленні складних атак і швидкості відповіді на інциденти. Було встановлено, що існує великий простір для поліпшення в цій сфері, зокрема через впровадження модульних систем, здатних до самонавчання і швидкої адаптації до нових видів загроз.

3. Аналіз тенденцій в кіберзлочинності виявив, що брокери первинного доступу і атаки "нульового дня" стали значним викликом для організацій усіх масштабів. Це підсилює важливість розвитку захисних систем, які можуть швидко виявляти і нейтралізувати такі атаки, і підтверджує потребу у більш глибокому розумінні шляхів проникнення та методів поширення вірусів-вимагачів.

4. На основі аналізу сучасних методів виявлення та нейтралізації вірусів-вимагачів встановлено, що традиційні підходи не є достатньо ефективними проти нових, більш складних форм атак, таких як поліморфні та метаморфні шкідливі програми, що вимагає розробки нових стратегій та інструментів для їхнього виявлення. Детально розглянуто архітектуру та функціональні можливості систем SIEM та EDR, а також визначено їхні обмеження при виявленні складних загроз. Аналіз поточного стану кіберзлочинності підкреслив необхідність розвитку адаптивних систем кібербезпеки, здатних оперативно реагувати на швидкі зміни у тактиці кібератак.

5. Ретельний огляд впливу геополітичних подій на кіберзагрози продемонстрував, що кіберпростір стає все більш політизованим, базуючись на чому виникає потреба у гнучких і стратегічно зорієнтованих заходах кіберзахисту. Особливо це стосується розробки стратегій, які беруть до уваги можливість державного спонсорства кібератак і потребу у міжнародному співробітництві для захисту від таких атак.

РОЗДІЛ 2
ВИКОРИСТАННЯ EBPF ДЛЯ ВИЯВЛЕННЯ ПРОГРАМ-ВИМАГАЧІВ

2.1. Вступ до eBPF: Огляд архітектури і можливостей

Berkeley Packet Filter представляє собою компоненту ядра Linux, яка надає можливість виконувати користувацький код на вбудованій віртуальній машині в ядрі. Ця система поділяється на дві основні частини: традиційний BPF (cBPF) і удосконалений BPF (eBPF, або просто BPF). Старіша версія cBPF була обмежена лише аналізом пакетів, в той час як сучасний eBPF пропонує значно ширші можливості, включаючи зміну пакетів, адаптацію аргументів системних викликів, модифікацію програм у просторі користувача та інше [87].

Початок історії Berkeley Packet Filter (BPF) відбувся у 1992 році, коли він був введений як ефективний метод для перехоплення мережевих пакетів з метою моніторингу користувачем. Цей механізм був вперше задокументований Стівеном МакКенном та Ваном Джейкобсом з Національної лабораторії Лоуренса Берклі. Вони описали систему програмування BPF, використовуючи унікальний набір 32-бітових інструкцій, що має схожість із мовою асемблера. Нижче наведено приклад коду, який був безпосередньо взятий з їхньої публікації [88]:

```
ldh    [12]
  jeq    #ETHERTYPE IP, L1, L2
  L1:    ret    #TRUE
  L2:    ret    #0
```

Лістинг 2.1 Приклад коду

Цей фрагмент коду відфільтровує пакети, які не є пакетами Інтернет-протоколу. На вхід цього фільтра подається Ethernet-пакет, і перша команда (ldh)

завантажує 2-байтне значення, починаючи з байта 12 цього пакета. У наступній інструкції (jeq) це значення порівнюється зі значенням, яке представляє IP-пакет. Якщо вони збігаються, виконання переходить до інструкції з міткою L1, і пакет приймається, повертаючи ненульове значення (позначене тут як #TRUE). Якщо значення не збігається, пакет не є IP-пакетом і відхиляється з поверненням 0.

BPF став ефективним способом зміни поведінки ядра завдяки наступним основним можливостям:

- Він працює як окрема віртуальна машина, спеціально оптимізована для взаємодії з регістровими процесорами.

- BPF-програми використовують буфери для кожного окремого додатку, щоб уникнути потреби у копіюванні пакетної інформації для обробки.

- Щоб переконатися у безпечності та відсутності можливості потрапляння в нескінченні цикли, BPF-програми піддаються верифікації перед завантаженням у простір ядра.

eBPF, який використовує 64-бітні регістри, здатний відповідати на різноманітні події в ядрі і дозволяє інтегрувати користувацькі функції, не обтяжуючи при цьому операційну систему. Ось типовий процес використання eBPF:

1. Розробка скрипта, який взаємодіє з даними, отриманими в результаті подій, пов'язаних із системними викликами.

2. Компіляція цього скрипта за допомогою загальноприйнятих компіляторів, як-от GCC чи LLVM.

3. Завантаження скомпільованої програми в ядро системи після її успішної перевірки з боку верифікатора eBPF.

4. Прив'язка завантаженої програми BPF до потрібного мережевого інтерфейсу, файлу або процесу для її виконання та обробки в цьому контексті.

На рисунку є програма, що працює у користувацькому просторі і включає в себе програму eBPF для видимості на рівні процесу в ядрі Linux.

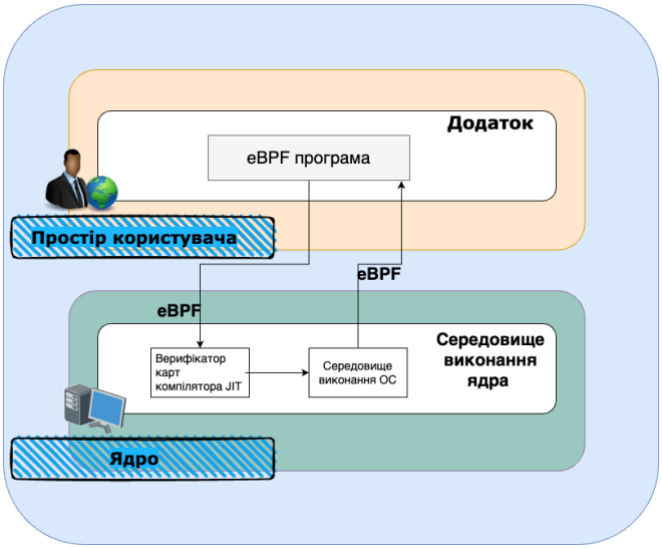


Рис. 2.1. Огляд архітектури eBPF

Зазвичай eBPF програми, які написані у вигляді байт-коду, розробляються за допомогою вищих мов програмування, таких як Python чи Golang, використовуючи компілятор, сумісний з байт-кодом eBPF. Після цього, програма eBPF встановлюється в ядро Linux, де вона проходить перевірку за допомогою верифікаційного механізму eBPF, що дозволяє запобігти потенційним помилкам. Надалі програма компілюється та інтегрується з певною подією в ядрі, зазвичай це відноситься до моніторингу системних викликів. При кожній такій події, програма активується, проводить моніторинг та аналіз, а потім відправляє зібрані дані назад до користувацького додатку у просторі користувача. Для зв'язку між eBPF програмою та користувацьким додатком використовуються так звані "Мапи eBPF" [89].

eBPF Мапи. Коли пакет ефективно оброблено та зібрано всі потрібні дані у ядерному просторі, наступним кроком є передача цієї інформації до простору користувача. І саме в цей момент використовуються мапи eBPF.

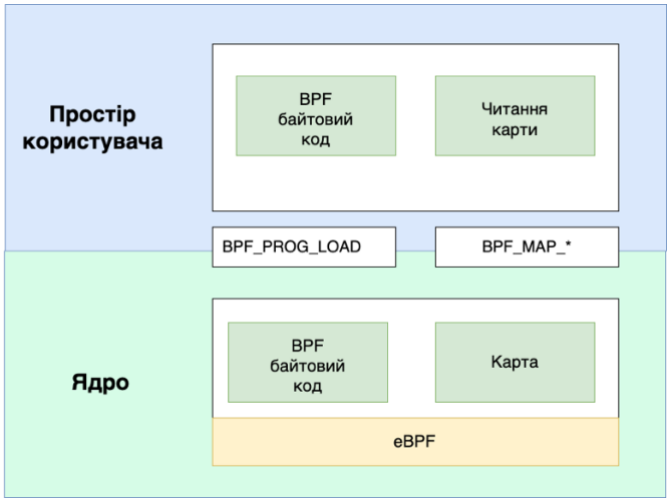


Рис. 2.2. Розташування мап eBPF

Мапи eBPF є типами структур даних, які можна використовувати з обох сторін: з простору користувача та в рамках BPF-програм у просторі ядра. Вони функціонують на принципі ключ-значення і можуть бути організовані як хеш-таблиці, масиви чи стеки. Розмір мапи eBPF має бути заздалегідь визначений під час розробки програми BPF відповідно до її потреб, при цьому він не може перевищувати ліміт у 4 КБ. Більшість типів мап підтримують такі базові операції як пошук, оновлення та видалення, а з простору користувача доступна ще одна операція - визначення наступного ключа. Загалом, наявні можливості мап eBPF достатні для виконання більшості стандартних завдань [90].

Моніторинг за допомогою застарілих інструментів ядра, диска та мережі. Класичне перехоплення пакетів має недоліки, особливо в хмарних середовищах. По-перше, цей процес є дорогим. Він вимагає пошуку за шаблоном для кожного пакета, що споживає обчислювальні ресурси та може бути неефективним у виявленні загроз. По-друге, значна частина сучасного трафіку шифрується. Let's Encrypt та інші сервіси зробили шифрування більш доступним та розповсюдженим, що ускладнює аналіз пакетів [91]. Традиційний метод захоплення пакетів стає менш придатним для хмарних та великомасштабних середовищ. Натомість, eBPF надає гнучкий, масштабований та ефективний підхід до моніторингу мережі та забезпечення безпеки.

Для розслідувань інцидентів використовують дискову експертизу. Під час розслідувань збирають побітові копії дисків для вилучення артефактів. Але артефакти можуть бути випадковими та не завжди можна точно визначити, які дані будуть корисні. Інформація, яка зберігається у пам'яті, може бути втрачена, якщо її не записати на диск. Експертиза пам'яті зосереджена на нових класах атак, але існуючі техніки захисту ускладнюють аналіз. eBPF, у свою чергу, забезпечує можливість моніторингу подій ядра та їх реєстрації з урахуванням атрибутів контейнера, створюючи таким чином гнучкий і ефективний фреймворк для забезпечення безпеки контейнерів.

Хмарний підхід. eBPF дозволяє проводити моніторинг подій безпеки в режимі реального часу, передаючи ці дані до простору користувача для аналітичної обробки, розслідування інцидентів або формування стратегій безпеки. Також, eBPF програми підтримують Kubernetes через API моніторингових програм, які використовують метадані ідентифікації для визначення зв'язку між подіями в ядрі та Kubernetes контейнерами [92]. Це забезпечує детальний моніторинг життєвого циклу процесів та мережевих з'єднань у хмарних середовищах, при цьому мінімізуючи використання пам'яті.

Віртуальна машина у ядрі. Под (Pod) - це набір процесів Linux, які виконуються в контексті простору імен ядра. Виконуючи системні виклики та запити до ядра операційної системи, де розташований eBPF, pod взаємодіє з ним. Наприклад, системний виклик execve() використовується для запуску нового процесу. При запуску curl з оболонки bash відбувається копіювання процесу bash, який викликає execve() для запуску дочірнього процесу curl. З допомогою eBPF можна перехопити будь-яку подію ядра, запустити відповідну програму, отримати потрібні значення та передати їх назад у простір користувача. Для ілюстрації, можна створити програму eBPF, яка активізується після завершення системного виклику execve(); витягує такі метадані, як ім'я двійкового файлу curl, PID, UID,

аргументи процесу та можливості процесу в Linux; після чого передає ці дані до простору користувача, як це представлено на рисунку:

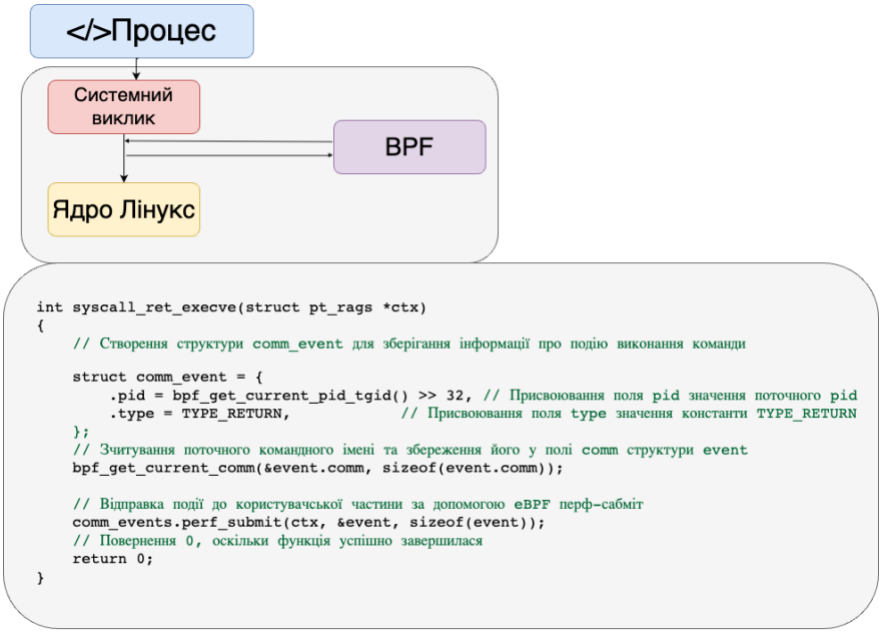


Рис. 2.3. Програма eBPF, підключена до системного виклику execve()

eBPF надає можливість перехоплення будь-якої події на рівні ядра, виконання власного коду з допомогою програмованої логіки. Можна розглядати це як функціональність віртуальної машини в ядрі з універсальним набором 64-бітних регістрів та програмами eBPF, прив'язаними до кодових шляхів ядра. Програми на eBPF можуть динамічно завантажуватися в ядро та видалятися з нього. Якщо така програма прив'язана до конкретної події, вона буде активована при виникненні цієї події, незалежно від того, що цю подію спричинило. eBPF надає велику перевагу над оновленням ядра або перезавантаженням системи для доступу до нових функцій. Це робить інструменти спостереження та безпеки, які базуються на eBPF, більш ефективними та миттєвими, оскільки вони забезпечують видимість всього, що відбувається на системі. У контейнеризованих середовищах це також включає видимість процесів в контейнерах та на хост-машинах.

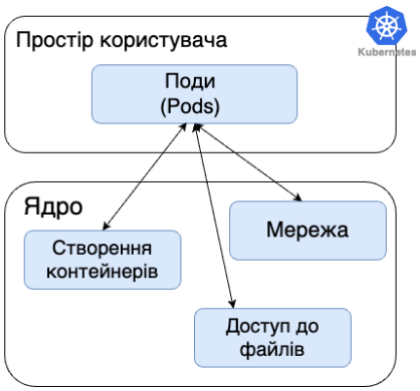


Рис. 2.4. eBPF програма має видимість на рівні ядра, що відбувається у всіх аплікаціях, що запущені на ноді Kubernetes

Внутрішня діяльність контейнеризованих середовищ допомагає виявляти віруси-вимагачі, не вимагаючи модифікації існуючого коду або конфігурації. Крім того, модулі на основі eBPF автоматично моніторять всі аплікації та підсистеми на хост-машині з моменту їх завантаження [92].

2.2. Методологія розробки модулів на базі eBPF

Проект розробки BPF модулів складається з двох типів вихідних файлів. Один з яких є файлом вихідного коду BPF-програми, що виконується у просторі ядра. Інший - це файл вихідного коду програми користувача, який використовується для завантаження програми BPF до ядра, вивантаження програми BPF з ядра, взаємодії зі станом ядра (наприклад, bpf_loader.c на рисунку нижче) [92]. Наразі програми BPF можуть бути розроблені лише на мові C, а точніше в обмеженому синтаксисі C, і єдиним, хто може ідеально скомпілювати вихідний код є компілятор clang. clang - це зовнішній компілятор для C, C++, Objective-C та інших мов програмування, що використовує в якості внутрішнього інтерфейсу LLVM [93].

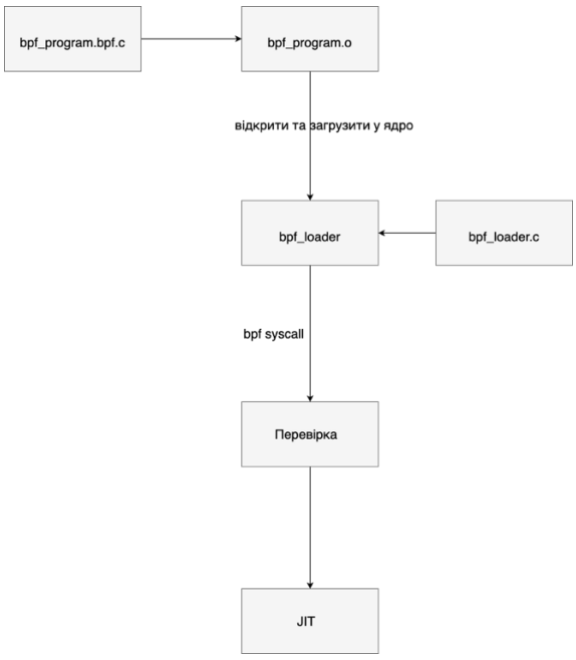


Рис. 2.5. Процес компіляції та завантаження програми BPF у простір ядра

Цільовий файл BPF (bpf_program.o) є ELF-файлом, вміст якого може бути прочитаний за допомогою інструменту командного рядка readelf. Інструмент llvm-objdump також надає інформацію про файл [94].

```
llvm-objdump -d bpf_program.o
```

Лістинг 2.2

Вміст, який виводить llvm-objdump, насправді є байтовим кодом BPF. Програма BPF завантажується в ядро як байт-код. Коли програма BPF завантажується в ядро, віртуальна машина BPF перевіряє байт-код BPF і запускає JIT-компіляцію для компіляції байт-коду в машинний код. Програми простору користувача, що використовуються для завантаження та вивантаження програм BPF, можуть бути розроблені на декількох мовах, як C, так і Python, Go, Rust тощо.

Програми BPF. Ядро Linux швидко розвивається і типи та структури даних в ядрі постійно змінюються. Поля одного і того ж типу структури в різних версіях

ядра можуть бути переставлені, перейменовані або видалені, замінені на зовсім інші поля тощо. Для BPF-програм, яким не потрібно дивитися на внутрішні структури даних ядра, проблем з перенесенням може не виникнути. Однак для тих BPF-програм, яким потрібно покладатися на певні поля у структурі даних ядра, важливо враховувати проблеми, які можуть виникнути у BPF-програмі через зміни у внутрішній структурі даних різних версій ядра. Для вирішення проблеми переносимості BPF у ядрі було впроваджено дві нові технології - BTF (BPF Type Format) та CO-RE (Compile Once - Run Everywhere) [95]. BTF надає структурну інформацію, щоб уникнути залежності від Clang та заголовків ядра, а CO-RE робить скомпільований байт-код BPF переносимим, уникаючи необхідності перекомпіляції LLVM.

BPF-програми, створені за допомогою цих нових методів, працюють на різних версіях ядра linux без необхідності перекомпілювати їх для конкретного ядра на цільовій машині. Також не потрібно встановлювати сотні мегабайт LLVM, Clang і залежностей заголовків ядра на цільову машину, як це було раніше.

Написання першої програми на базі eBPF. Візьмемо для прикладу BPF-програму рівня hello world та її завантажувач простору користувача, щоб побачити спосіб реалізації BPF-програми на основі структури, запропонованої libbpf-bootstrap. libbpf відноситься до tools/lib/bpf у кодовій базі ядра linux, яка є бібліотекою C [96]. Bpftool є допоміжною програмою bpf, що використовується у libbpf-bootstrap для генерації xx.skel.h. base_program.bpf.c - це вихідний код програми bpf, який компілюється у байт-код BPF ELF-файл base_program.bpf.o за допомогою команди clang -target=bpf. libbpf-bootstrap не використовує завантажувач простору користувача для безпосереднього завантаження base_program.bpf.o, натомість генерує файл base_program.skel.h на основі base_program.bpf.o за допомогою команди bpftool gen. Згенерований файл base_program.skel.h містить байт-код BPF-програми та функції для завантаження і вивантаження відповідної BPF-програми, які можуть бути викликані безпосередньо

з програми стану користувача. У програмі стану користувача BPF, base_program.c, достатньо включити файл base_program.skel.h, завантажити та підключити програму BPF до відповідної точки у шарі ядра відповідно до набору.

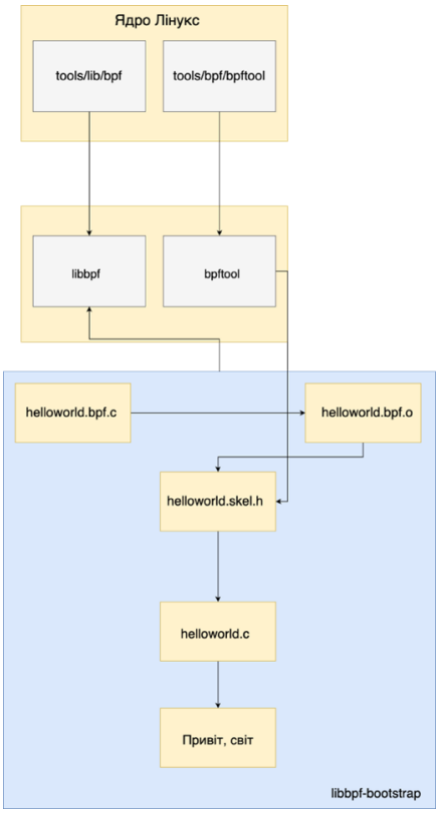


Рис. 2.6. Схема програми EBPF за допомогою libbpf-bootstrap

Приклад розробки eBPF-додатку на основі libbpf-bootstrap. Примітка: Експериментальне середовище - ubuntu 20.04 (версія ядра: 5.4.0-109-generic) [97].

1. Перш за все, потрібно встановити clang, компілятор для BPF програм. Рекомендується встановлювати clang 10 і вище, ось приклад команди:

```
apt-get install clang-10.
```

Лістинг 2.3

2. Завантаження libbpf-bootstrap. Це простий фреймворк для розробки.

```
git clone https://github.com/libbpf/libbpf-bootstrap.git
```

Лістинг 2.4

3. Ініціалізація та оновлення залежностей libbpf-bootstrap. libbpf-bootstrap має залежності libbpf, bpftool, налаштовані у проекті як підмодуль git:

```
cat .gitmodules
```

Лістинг 2.5

Отже, перед застосуванням проекту libbpf-bootstrap для розробки BPF-додатку потрібно ініціалізувати git-підмодулі та оновити їх до останньої версії.

```
git submodule update --init --recursive
```

Лістинг 2.6

4. За допомогою фреймворку libbpf-bootstrap легко додати новий BPF-додаток. Для цього переходять до каталогу libbpf-bootstrap/examples/c, де створюють два вихідні файли на мові C: base_program.bpf.c і base_program.c (Додаток Б).

Логіка програми bpf у base_program.bpf.c проста: вставити bpf_prog у точку виклику execve (встановлену макросом SEC), так що кожного разу, коли виконується виклик execve, bpf_prog буде викликатися назад. Логіка bpf_prog також дуже проста: вона виводить рядок журналу налагодження ядра. Компіляція нової програми base_program також неважка, головним чином тому, що проект libbpf_bootstrap має дуже розширений Makefile, нам просто потрібно додати запис base_program після змінної APP у Makefile.

```
// libbpf_bootstrap/examples/c/Makefile
```

```
APPS = base_program minimal minimal_legacy bootstrap uprobe kprobe fentry
```

Лістинг 2.7

Після цього виконується компіляція за допомогою команди make для програми base_program.

```
# Виконує компіляцію BPF програми
BPF:
  @echo "Компілюємо BPF код..."
  @$(CC) -o .output/base_program.bpf.o src/base_program.bpf.c

# Генерує заголовочний файл з каркасом
GEN-SKEL:
  @echo "Генеруємо скелетний заголовочний файл..."
  @bpf-gen-skel .output/base_program.bpf.o > .output/base_program.skel.h

# Компілює об'єктний файл програми
CC:
  @echo "Компілюємо програму..."
  @$(CC) -c src/base_program.c -o .output/base_program.o

# Створює виконуваний файл програми
BINARY:
  @echo "Створюємо виконуваний файл..."
  @$(CC) .output/base_program.o -o base_program

# Ціль за замовчуванням для Make
all: BPF GEN-SKEL CC BINARY
  @echo "Збірка завершена."
```

Лістинг 2.8

Програму base_program необхідно запустити з привілеями root:

```
# Запуск програми з адміністративними правами
sudo ./base_program
# Завантаження об'єкта BPF програми з буфера
libbpf: loading object 'base_program_bpf' from buffer
# Знаходження програми BPF у секції tracepoint для syscalls/sys_enter_execve
libbpf: sec 'tracepoint/syscalls/sys_enter_execve': found program 'bpf_prog'
# Обробка релокацій для секції tracepoint
```

```
libbpf: elf: section(9) '.reltracepoint/syscalls/sys_enter_execve', size 16
# Відображення глобальних даних в пам'яті
libbpf: map '.rodata.str1.1' (global data): at sec_idx 3, offset 0, flags 480.
# Ідентифікація мапи 0 як ".rodata.str1.1"
libbpf: map 0 is ".rodata.str1.1"
# Відображення глобальних даних для base_program в пам'яті
libbpf: map 'base_program.rodata' (global data): at sec_idx 4, offset 0, flags 480.
# Ідентифікація мапи 1 як "base_program.rodata"
libbpf: map 1 is "base_program.rodata"
# Збір релокацій для секції викликів системи
libbpf: sec '.reltracepoint/syscalls/sys_enter_execve': collecting relocation for section(2) 'tracepoint/syscalls/sys_enter_execve'
# Релокація вказівника на дані в програмі BPF
libbpf: sec '.reltracepoint/syscalls/sys_enter_execve': relo #0: insn #9 against '.rodata'
# Прив'язка даних до програми BPF
libbpf: prog 'bpf_prog': found data map 1 (base_program.rodata, sec 4, off 0) for insn 9
# Створення мапи '.rodata.str1.1'
libbpf: map '.rodata.str1.1': created successfully, fd=4
# Створення мапи 'base_program.rodata'
libbpf: map 'base_program.rodata': created successfully, fd=5
# Повідомлення про успішний запуск
Successfully started! Please run `sudo cat /sys/kernel/debug/tracing/trace_pipe` to see output of the BPF programs.
```

Лістинг 2.9

Наступну команду необхідно запустити в іншому терміналі, щоб переглянути вивід програми bpf (коли відбувається системний виклик execve).

```
# Виконання команди для перегляду виводу BPF програм з системного буфера слідчих повідомлень
sudo cat /sys/kernel/debug/tracing/trace_pipe


# Вивід відслідковування подій
# Процес git з PID 325411 на процесорі [002] викликав bpf_prog
git-325411  [002] .... 4769772.705141: 0: invoke bpf_prog: Hello, World!


# Процес sudo з PID 325745 на процесорі [005] викликав bpf_prog
sudo-325745  [005] .... 4772321.191798: 0: invoke bpf_prog: Hello, World!
```

```
# Анонімний процес з PID 325746 на процесорі [000] викликав bpf_prog
<...>-325746  [000] .... 4772322.798046: 0: invoke bpf_prog: Hello, World!
```

Лістинг 2.10

2.3 Застосування eBPF для відстеження системних викликів

Фільтрування системних викликів є важливим засобом захисту операційної системи від потенційно небезпечних користувацьких програм. Проте, традиційні підходи до цього механізму мають свої недоліки. В сучасних контейнерних технологіях широко застосовується фільтрація системних викликів у Linux, яка, на жаль, не завжди відповідає стандартам політик безпеки. Впровадження eBPF дозволяє формувати більш складні та адаптивні безпекові політики для Seccomp, враховуючи динамічні умови та змінні. Поєднання Seccomp та eBPF може стати ефективним інструментом для посилення безпеки на рівні ядра у системах [98].

Дослідження показують, що використання eBPF для фільтрації може значно покращити існуючі політики безпеки, наприклад, знижуючи потенційні ризики на ранній стадії виконання до 55% у випадку часової спеціалізації, зменшуючи реальні уразливості та збільшуючи ефективність роботи фільтрів [99].

Системи запускають безліч ненадійних додатків на довіреному ядрі операційної системи, що взаємодіють з ним за допомогою системних викликів. Фільтрування цих викликів є популярним механізмом безпеки системних викликів, який обмежує їх виклик за попередньо визначеними політиками безпеки. Ранні методи фільтрації системних викликів, використовують довірених агентів користувацького простору для реалізації політики безпеки системних викликів. Однак, агенти користувацького простору несуть значні накладні витрати на перемикання контексту для фільтрації системних викликів. Оскільки кожен системний виклик повинен підключатися у користувацький простір для перевірки

політики, а потім перемикатися на простір користувача для перевірки політики і вертатися назад.

Seccomp - це механізм в ядрі Linux, що надає можливість обмежувати доступ програми до системних викликів. В рамках фільтрації системних викликів Seccomp є важливим інструментом, оскільки він дозволяє визначити, які системні виклики дозволені для виконання від імені конкретної програми, тим самим забезпечуючи додатковий рівень безпеки. Наприклад, кожен Android-додаток обмежується за допомогою Seccomp; systemd використовує Seccomp для ізоляції процесів користувача [100,101]; проект Google's Sandboxed API [102] використовує Seccomp для ізоляції бібліотек C/C++; Технології легкої віртуалізації, такі як Google gVisor [103], Amazon Firecraker [104,105], Rkt [106], та Kubernetes [107] використовують Seccomp. Основним обмеженням Seccomp є недостатня програмованість для формулювання розширених політик безпеки. Від початкового режиму, який блокував усі системні виклики, окрім read(), write(), _exit() та sigreturn(), Seccomp тепер (починаючи з Linux v3.5) дозволяє писати користувацькі політики безпеки класичною мовою BPF (cBPF) [108] у режимі фільтра.

На жаль, можливість програмування cBPF надто обмежена - політики безпеки в Seccomp здебільшого обмежуються статичними списками дозволів. Це пов'язано насамперед з тим, що cBPF не надає механізму зберігання станів, а отже, фільтри cBPF мають бути без станів. Крім того, cBPF не надає інтерфейсу для виклику будь-яких інших утиліт ядра або інших BPF-програм. Як наслідок, багато бажаних та/або необхідних функцій фільтрації системних викликів не можуть бути реалізовані безпосередньо на основі Seccomp-cBPF, натомість вимагають значних модифікацій ядра (основна перешкода для розгортання). Визнаючи потребу у більш виразних політиках, Seccomp нещодавно додав нову функцію, відому як Notifier [108], для підтримки старої ідеї агентів простору користувача, яка, на жаль поділяє їх обмеження.

Слід зазначити, що відкриття інтерфейсу eBPF у Seccomp у початкових версіях Linux не вирішує проблем, через:

1. Відсутність основних функцій, наприклад, серіалізації.

2. Непридатність існуючих утиліт, таких як сховище завдань, для Seccomp, оскільки вони спрямовані лише на привілейовані процеси.

3. Небезпеку при використанні Seccomp для існуючих функцій, наприклад, поточна функція доступу до пам'яті користувача не вирішує проблеми TOCTTOU.

З цією метою було створено новий тип програми Seccomp-eBPF, який можна легко програмувати, щоб користувачі могли створювати розширені політики безпеки системних викликів у програмах-фільтрах eBPF [107]. Зокрема є функціонал розкриття, модифікації і створення нових допоміжних функцій eBPF для безпечного керування станом фільтра, ядром доступу і станом користувача, а також використовується синхронізація стану ядра та користувача, а також для використання примітивів синхронізації. Важливо, що система інтегрується з існуючими механізмами привілеїв і можливостей ядра, що дозволяє непривілейованим користувачам безпечно встановлювати розширені фільтри та запобігати ескалації привілеїв. Безпека Seccomp-eBPF еквівалентна двом існуючим компонентам ядра: Seccomp та eBPF. У модулі моніторингу системних викликів реалізовано багато функцій, необхідних для реальних випадків використання, таких як контрольна точка/відновлення в користувацькому просторі, сплячий фільтр та конфігурацію розгортання, щоб зробити Seccomp-eBPF привілейованою функцією. Окрім цього було також модифіковано існуюче середовище виконання контейнерів для підтримки фільтрації системних викликів на основі Seccomp-eBPF. Seccomp-eBPF використовується для реалізації різних нових функцій, включаючи обмеження кількості та швидкості системних викликів, захист цілісності потоку та серіалізацію.

2.3.1 Поєднання політик Seccomp з eBPF у єдиний модуль Seccomp-eBPF

Наразі Seccomp покладається на класичну мову BPF, яка дозволяє користувачам виражати фільтри системних викликів у вигляді програм [109, 110]. Він має дуже простий набір інструкцій на основі регістрів, що робить програми-фільтри легкими для перевірки. Через обмежені можливості програмування, фільтри cBPF у Seccomp здебільшого реалізують список дозволів: фільтр дозволяє системний виклик лише у тому випадку, якщо вказаний ідентифікатор системного виклику. Іноді cBPF-фільтр додатково перевіряє аргументи примітивних типів даних і запобігає системному виклику, якщо перевірка аргументів виявиться невдалою. Фільтри cBPF не володіють станами - результат дії фільтра Seccomp-cBPF залежить тільки від певного ідентифікатора системного виклику та значень аргументів (у списку дозволених), так як cBPF не пропонує жодного інструменту для контролю стану. Як уже було вказано на початку розділу розмір eBPF-фільтра обмежено 4096 інструкціями, тому складні політики безпеки слід реалізовувати за допомогою ланцюжком з декількох фільтрів. Усі встановлені фільтри у ланцюжку виконуються для кожного системного виклику, і повертається дія з найвищим пріоритетом. Однак така поведінка ланцюжка пов'язана з підвищенням продуктивності, головним чином через непрямі переходи [110]. Seccomp вирішив перетворити фільтри cBPF у код eBPF внутрішньо, щоб скористатися перевагами широко оптимізованих інструментальних засобів eBPF, які створюють код, що працює в 4 рази швидше на x86-64 порівняно з прямим використанням BPF [110].

2.3.2 Seccomp Notifier

Обмежена виразність BPF ускладнює реалізацію складних політик безпеки. Тому Seccomp нещодавно включив підтримку агента користувацького простору (який називається Notifier [111]) на додаток до фільтрів cBPF. Подібно до ранніх

фреймворків інтерпозиції системних викликів [111, 112, 113] він передає рішення довіреному користувацькому агенту. Зокрема, коли Seccomp перехоплює системний виклик, він блокує завдання, що його викликає, і перенаправляє контекст системного виклику (наприклад, ідентифікатор виклику PID та значення аргументів) до агента. Основним недоліком Seccomp Notifier є значні затримки через додаткові контекстні перемикання, які виникають при перемиканні до простору користувача туди і назад. Крім того, для перевірки вмісту аргументів системного виклику, які є вказівниками на простір користувача, Seccomp Notifier повинен використовувати ptrace для доступу до пам'яті відстежуваного процесу. На додаток до впливу на продуктивність, така перевірка підлягає умовам перегонів від часу перевірки до часу використання (TOCTTOU), за яких потік у програмі, що контролюється, може змінювати вміст пам'яті (а отже і значення аргументів) після того, як перевірка завершена Seccomp Notifier. Нарешті, необхідність запуску агента простору користувача у довіреному домені ускладнює його використання у деяких середовищах розгортання, наприклад, для бездемонових (non daemon) контейнерів [113]. З усіх цих причин Seccomp Notifier не підходить для складних системних політик фільтрації.

Для відстеження системних викликів за допомогою eBPF важливо зосередитися на ключових атрибутах виклику, щоб забезпечити докладний аналіз активності. Ось поля, які варто логувати:

Таблиця. 2.1.

Модель модуля

Назва поля	**Опис**
TIMESTAMP	Час, коли було здійснено системний виклик
PID	Ідентифікатор процесу, який здійснив виклик
TID	Ідентифікатор потоку в рамках процесу
UID	Ідентифікатор користувача, під яким запущено процес
SYSCALL	Назва системного виклику

ARGUMENTS	Аргументи системного виклику
RETURN_VALUE	Значення, повернуте системним викликом
STATUS	Статус виконання (наприклад, SUCCESS, FAILED, BLOCKED)
ERROR_MESSAGE	Повідомлення про помилку, якщо така є
SOURCE_FILE	Файл, з якого було здійснено виклик
LINE_NUMBER	Номер рядка в файлі, де було здійснено виклик.

Приклад виводу:

```
TIMESTAMP PID TID UID SYSCALL ARGUMENTS RETURN_VALUE STATUS ERROR_MESSAGE SOURCE_FILE LINE_NUMBER COMMENT
2023-10-26 13:15:25 3562 3563 1001 open /etc/passwd, O_RDWR -1 FAILED Permission denied app.c 123 Trying to access passwd file
2023-10-26 13:16:01 3588 3589 0 write fd=3, buf=0x7ff... 42 SUCCESS - daemon_process.c 45 Writing to log
```

Лістинг 2.11

Зібрані дані під час атак вірусів-вимагачів будуть використовуватися, як інформаційний продукт моделей машинного навчання для виявлення програм-вимагачів.

2.4 Моніторинг файлів та криптографічної активності з використанням хоніпотів та eBPF

Функціональність eBPF не обмежується лише відстеженням. З його допомогою можна швидко реагувати на підозрілу активність, блокуючи потенційно шкідливі процеси або атаки на ранніх етапах. Це забезпечує додатковий рівень

захисту для систем, знижуючи ризик пошкодження або втрати важливих даних. eBPF дозволяє створювати хуки (програми), які можуть бути асоційовані з різними подіями в системі, включаючи операції з файлами. Наприклад, можна стежити за системними викликами, які здійснюють операції над файлами, такі як open, read, write та delete. У цьому підрозділі буде описано eBPF-програму, завданням якої є моніторинг файлової системи та виявлення діяльності програми-вимагача. Пропонується розробити не просто модуль моніторингу файлової системи, а систему файлових приманок на базі ebpf. Використання хоніпотів - систем приманок, призначених для заманювання та виявлення зловмисників, може бути цінним інструментом для виявлення та аналізу атак з використанням програм-вимагачів [114]. Програма eBPF може перехоплювати низькорівневі події файлової системи і запускати оповіщення або реагування при виявленні підозрілих дій. Поєднуючи хоніпотів з моніторингом файлової системи, організації можуть створити багаторівневий захист, здатний виявляти і реагувати на атаки зловмисників в режимі реального часу.

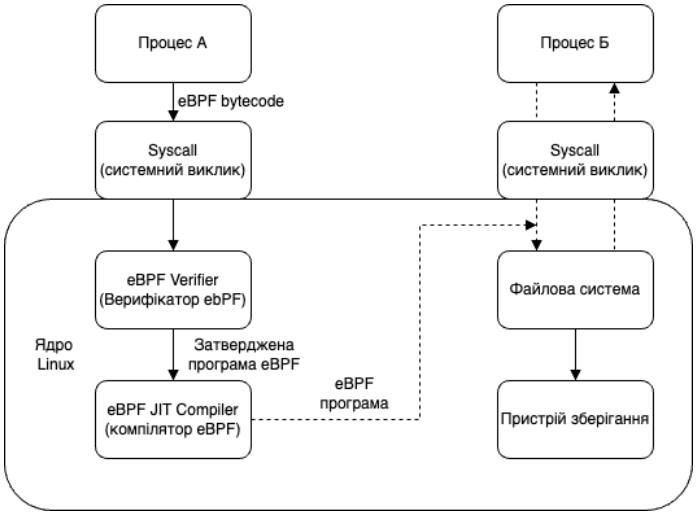


Рис. 2.7. Концептуальна архітектура файлового хоніпоту

Однією з ключових переваг хоніпоту у поєднанні з сучасними технологіями, такими як eBPF, є складність їх виявлення. Пасивні, непрямі та неінтерактивні

методи виявлення дозволяють шифрувальникам уникати будь-якого мережевого трафіку або системних викликів, які зазвичай змушують вірусні програми-вимагачі вважати, що їх аналізують або виявляють. Ця особливість робить вкрай малоймовірним, що програми-вимагачі зможуть з високою ймовірністю виявити присутність хоніпоту, не залишаючи при цьому жодних слідів своєї діяльності. Крім того, запропонований підхід може бути доопрацьований, оптимізований і розширений для конкретних середовищ і випадків використання [114]. Наприклад, Python-обгортка для eBPF може бути модифікована для підтримки додаткових функцій, таких як машинне навчання, статистичні або поведінкові алгоритми для більш просунутого виявлення загроз і кореляції з іншою інформацією про безпеку, яка також може бути отримана за допомогою підсистеми eBPF.

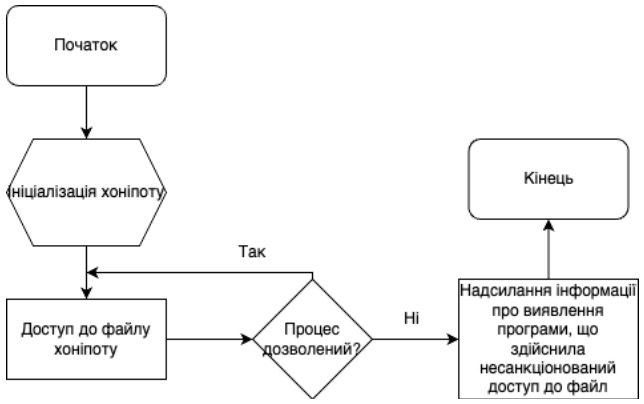


Рис. 2.8. Блок-схема роботи модулю на базі хоніпоту

Вибір відповідних зразків програм-вимагачі для тестування систем виявлення програм-вимагачі, заснованих на методах на базі хоніпотів є вирішальним для збільшення ефективності та точності виявлення системи [115]. З цією метою в дослідженні було визначено ряд відомих і широко використовуваних сімейств програм-вимагачів і відібрано зразки з кожного сімейства для тестування. Однак запустити ці зразки програм-вимагачів у контрольованому середовищі для тестування не завжди було просто. Багато сучасних штамів вірусів-вимагачі мають вбудовані функції "виявлення пісочниці", які дозволяють їм виявляти, що вони

працюють у віртуальному або емульованому середовищі, і відповідно змінювати свою поведінку. Це ускладнює точне моделювання реального сценарію атаки в лабораторних умовах, а також оцінку ефективності хоніпотів у виявленні та запобіганні таким атакам. Незважаючи на ці труднощі, дослідження змогло успішно протестувати мережу хоніпотів проти низки зразків програм-вимагачів і продемонструвати її ефективність у виявленні та запобіганні таким атакам. Це підкреслює важливість ретельного тестування та оцінки систем виявлення програм-вимагачів на основі мережі хоніпотів, а також необхідність постійних досліджень і розробок для того, щоб випереджати ландшафт загроз, який постійно змінюється. Для тестування було обрано кілька останніх зразків програм-вимагачів: CryptoLock, AIRad та DIMAQS.

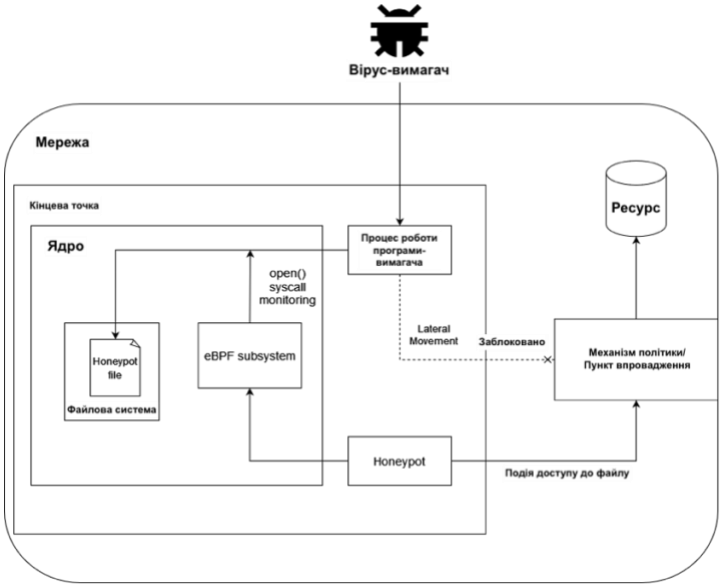


Рис. 2.9. eBPF ханіпот архітектура

BPF відстежує виклики системного виклику open(), перевіряє, чи збігається ім'я файлу, що відкривається, з ім'ям файлу хоніпотів, і якщо збігається - надсилає подію механізму політик. Це означає, що рівень виявлення не дорівнює 0%, але може бути вищим, якщо до цього файлу звертаються інші легітимні процеси або реальні користувачі. Тим не менш, існує список дозволених процесів, який можна

визначити, щоб виключити певні процеси, які можуть викликати подію безпеки, таким чином знижуючи рівень хибних спрацьовувань. Цей крок налаштування потребує попередньої інвентаризації, виконаної на хості, щоб виявити та виключити такі легітимні процеси. Виявлення наявності хоніпот на кінцевій точці може бути складним завданням для програм-вимагачів [116]. "Мишоловки" на основі eBPF також важко виявити з високою достовірністю, оскільки той самий набір інструментів, що використовується для цілей нашої "мишоловки", може бути використаний для моніторингу додатків/систем, спостереження та загального усунення несправностей системним адміністратором. Хоча можна перерахувати всі програми, завантажені в ядро, дуже малоймовірно, що програма зможе з високою впевненістю сказати, що конкретна програма eBPF є "пасткою", оскільки вона виглядає як сотні інших легальних програм та інструментів, які прослуховують виклики syscall, що виконуються або системою, або програмою, або адміністраторами. Наприклад, ось результат виконання команди bpftool, за допомогою якої можна переглянути список завантажених на даний момент eBPF-програм:

Таблиця 2.4.

Порівняння виводу bpftool для легальної системної програми та хоніпоту

Зонд, прикріплений системою	**Зонд, прикріплений до ханіпоту з медом**
{ "id": 33, "type": "cgroup_device", "tag": "03b4eaae2f14641a", "gpl_compatible": true, "loaded_at": 1676542285, "uid": 1000, "bytes_xlated": 296, "jited": true, "bytes_jited": 166, "bytes_memlock": 4096, "map_ids": [1] }	{ "id": 53, "type": "tracepoint", "name": "sys_enter_opena", "tag": "9081693d56ded011", "gpl_compatible": true, "loaded_at": 1676558452, "uid": 0, "bytes_xlated": 528, "jited": true, "bytes_jited": 315, "bytes_memlock": 4096, "map_ids": [6] }

Є згадка про системний виклик sys_enter_openat [117] (який є перехопленням точки трасування входу в системний виклик openat(), який, у свою чергу, викликається при виклику функції open() користувацькою програмою), але він виглядає як будь-яка інша програма, яка відстежує файли (наприклад, з метою ведення журналу). Тому сама наявність цієї інформації про програму недостатня для визначення, чи є вона бажаною метою.

В ході дослідження було проаналізовано поточний стан атак з використанням програм-вимагачів та їх вплив на організації, а також висвітлено обмеження традиційних заходів безпеки у виявленні та запобіганні цим атакам. Вивчаючи хоніпотів та їхні можливості у виявленні та запобіганні атакам з використанням програм-вимагачів, дослідження продемонструвало потенціал файлових eBPF хоніпотів як ефективного інструменту у виявленні цих атак, особливо в контексті архітектури нульової довіри.

Таблиця 2.5

Схема логування подій

Назва поля	**Опис**
TIMESTAMP	Час події. Важливо мати можливість відслідковувати події у хронологічному порядку
PID	Ідентифікатор процесу
PNAME	Ім'я процесу. Це може допомогти швидко розпізнати додаток або службу, яка викликала подію
UID	Ідентифікатор користувача, який ініціював процес

TYPE	Тип події (Open, Create, Delete, Encrypt, тощо)
FLAG	Статус події (наприклад: MIN - звичайна, MAJ - критична)
PATTERN	Послідовність дій, які можуть вказувати на підозрілу активність
TRESH_COUNT	Кількість подій за певний час, що може вказувати на аномальну активність
FILENAME	Шлях до файлу або криптографічної функції
RESULT	Результат дії (наприклад: SUCCESS, FAIL, ERROR_CODE)

Приклад виводу:

```
TIMESTAMP PID PNAME UID TYPE FLAG PATTERN TRESH_COUNT FILENAME RESULT
2023-10-26 12:45:21 26858 MyApp 1001 Enc MIN - ---E EVP_EncryptInit_ex SUCCESS
2023-10-26 12:46:15 28178 FileApp 1002 Crea MIN OCD- 60 /tmp/tmp8imczwuu/Asia/Magadan.aes FAIL...
```

Лістинг 2.12

Ці поля надають додаткову інформацію про події, сприяють кращому розумінню контексту та допомагають ефективніше реагувати на підозрілі або небезпечні дії. Ця програма відслідковує криптографічні операції та дії, пов'язані з файлами, в реальному часі, допомагаючи адміністраторам і аналітикам безпеки розуміти активність на системі.

2.5 Аналіз мережевого трафіку з використанням eBPF

Тип програми XDP

Один з небагатьох типів BPF-програм - XDP. Він виконується на ранній стадії обробки мережевого пакету. Щоб позначити програму BPF як XDP, користувач повинен додати прапорець BPF_PROG_TYPE_XDP при завантаженні в простір ядра. Тип програми XDP дозволяє виконувати різні дії з мережевим пакетом. Ось список найбільш поширених з них:

- XDP_PASS - передати мережевий пакет для подальшої обробки.
- XDP_DROP - скинути пакет і припинити його виконання у цей момент.
- XDP_ABORTED - припинити обробку пакета з виключенням точки трасування мережевого пакету.
- XDP_TX - відправити назад через той самий мережевий інтерфейс, з якого він прийшов.
- XDP_REDIRECT - перенаправити пакет на інший мережевий інтерфейс.

BPF-програми типу XDP надають корисні структури, такі як ethhdr для Ethernet-пакетів та iphdr для IP-пакетів, для розпакування мережевих пакетів. XDP-програму можна приєднати до мережевого інтерфейсу, за яким потрібно спостерігати, або одночасно до кількох. Ця можливість є ключовою для виявлення та протидії вірусів, адже вона забезпечує здатність відповідати на загрози на мережевому рівні, а не лише в ядрі операційної системи. Такий підхід до реагування на атаки дозволяє ідентифікувати їх на етапі ініціювання або розповсюдження, замість виявлення на стадії шифрування файлів або порушення доступу до них.

Отримати від'єднання BPF-програми від мережевого інтерфейсу можна за допомогою інструмента bpftool [118]. Однак, спроба приєднати кілька XDP-програм до одного мережевого інтерфейсу неможлива, оскільки до одного інтерфейсу можна приєднати лише одну програму. В даний час програми XDP є

найпоширенішим типом eBPF-програм. Важливо також зазначити наявність спеціальних типів програм, таких як "Програми-фільтри сокетів" та "Програми-опції сокетів", які дозволяють спостерігати за активністю в сокетах та мережі в цілому [119]. Вони дозволяють не тільки відслідковувати передані пакети, але й змінювати параметри конкретних сокетів чи з'єднань. За допомогою таких програм можна отримати більш детальний огляд мережевої активності та реагувати на інциденти, блокуючи трафік на рівні сокетів.

Вже існують опубліковані дослідження, що демонструють використання eBPF/XDP у сфері моніторингу мереж і кібербезпеки [120]. Наприклад, існує сценарій для зменшення наслідків DDoS-атак, в якому всю шкідливу активність, пов'язану з DDoS, має бути блоковано. У цьому контексті автори застосували eBPF/XDP для аналізу вхідного трафіку та проведення обчислень у користувацькому просторі, використовуючи евристичні алгоритми, які, хоча й не такі точні як повноцінні нейронні мережі, ефективно виконують свою роботу. На основі отриманих даних (наприклад, ідентифікованих шкідливих IP-адрес) створюються програми eBPF, які відмовляють у прийнятті трафіку з цих джерел. Що стосується лише спостережуваності у хмарному середовищі мікросервісів, було запропоновано фреймворк ViperProbe [120]. Цей інструмент розроблений для підсилення як мережевого, так і загальносистемного моніторингу, за допомогою використання різноманітних наявних типів BPF. Насправді, eBPF можна ефективно використовувати також для відстеження системних подій, таких як системні виклики, з метою збору статистики та запису всієї поведінки системи. ViperProbe демонструє обмеженість у витратах ресурсів, одночасно забезпечуючи ефективний аналіз показників eBPF та вивчення метрик Envoy. Результати показали, що метрики, зібрані за допомогою eBPF, виявилися значно більш вигідними для впровадження горизонтального автоматичного масштабування. Окрім цього варто згадати платформу Cilium, що набуває популярності [121], яка є опенсорсовим додатком для мережевого з'єднання між програмами та контейнерами, у яких вони

розгортаються за допомогою платформ управління контейнерами, таких як Docker та Kubernetes. В основі Cilium лежить нова технологія ядра Linux під назвою BPF, яка дозволяє динамічно вставляти потужні засоби контролю та управління безпекою логіки безпеки у саму систему Linux. Оскільки BPF працює всередині ядра Linux, політики безпеки Cilium можна застосовувати та оновлювати без будь-яких змін у коді програми або конфігурації контейнера.

Для прикладу, розглянемо простий сценарій: виявлення вірусів, що взаємодіють з певними відомими IP-адресами командних серверів. Для цього можна створити XDP/eBPF програму, яка буде блокувати весь трафік до і від цих IP-адрес:

```
SEC("filter")
int block_ransomware_traffic(struct __sk_buff *skb) {
   bpf_hdr_pointer_t hdr_pointer = {};
   struct ethhdr *eth = bpf_hdr_pointer_advance(&skb->data, &skb->data_end, &hdr_pointer, sizeof(*eth));
   if (!eth) {
      return XDP_DROP;      }
   // Перевіряємо, чи це IP пакет
   if (eth->h_proto != bpf_htons(ETH_P_IP)) {
      return XDP_PASS;     }
   struct iphdr *ip = bpf_hdr_pointer_advance(&skb->data, &skb->data_end, &hdr_pointer, sizeof(*ip));
   if (!ip) {
      return XDP_DROP;     }
   // Перевіряємо IP-адресу відправника
   if (ip->saddr == bpf_htonl(0xC0A80001)) {  // 192.168.0.1 як приклад
      return XDP_DROP;   }
```

```
return XDP_PASS;        }
```

Лістинг 2.13

Цей код використовує eBPF для перевірки кожного вхідного пакету і відкидає пакети, що приходять з IP-адреси 192.168.0.1.

Для модуля моніторингу мережевої активності важливо зосередитися на аспектах, що відображають спроби з'єднання, передачу даних та можливі вторгнення. Ось які поля можуть бути корисними:

Таблиця 2.6.

Схема логування мережевого трафіку

Назва поля	**Опис**
TIMESTAMP	Час події
SRC_IP	IP-адреса відправника
SRC_PORT	Порт відправника
DST_IP	IP-адреса отримувача
DST_PORT	Порт отримувача
PROTOCOL	Протокол (напр. TCP, UDP, ICMP).
PACKET_SIZE	Розмір пакета даних
FLAG	Флаги пакета (напр. SYN, ACK, FIN)
STATUS	Статус пакета (напр. SUCCESS, DROP, RETRY)
APP_LAYER_DATA	Інформація аплікаційного рівня (напр., HTTP request type, DNS query)
ALERT	Сигнали про підозрілу або аномальну активність

Приклад виводу:

```
TIMESTAMP SRC_IP SRC_PORT DST_IP DST_PORT PROTOCOL PACKET_SIZE FLAG STATUS APP_LAYER_DATA ALERT COMMENT
2023-10-26 12:45:21 192.168.1.5 54320 8.8.8.8 53 UDP 64 - SUCCESS DNS Query - Query to google DNS
```

```
2023-10-26 12:46:15  10.0.0.1  80  192.168.1.100  49582  TCP  512  SYN
DROP  HTTP GET  HIGH  Multiple SYN without ACK, possible SYN flood...
```

Лістинг 2.14

Ці поля надають детальний огляд мережевої активності, дозволяючи спеціалістам швидко визначити тип активності та реагувати на можливі загрози.

2.6 Моніторинг процесів з використанням eBPF

Моніторинг виконання процесів є одним з блоків безпеки під час виконання [122]. Моніторинг виконання процесів можна використовувати для виявлення процесів у виробничому (production) середовищі, які можуть бути неочікуваними, або виявлення шаблонів виконання, які ніколи не повинні з'являтися. Наприклад, веб-сервер у виробничому середовищі ніколи не повинен створювати оболонку. Аналогічно, може знадобитися інформація про те, що менеджер пакетів викликається для встановлення нових залежностей на хост. Перегляд аргументів виклику "curl" також може допомогти зрозуміти, які саме конфіденційні дані міг викрасти зловмисник [123]. Ще однією важливою частиною моніторингу виконання процесів є можливість збагачення контексту, який надається за допомогою сповіщень (незалежно від їх типу). Кеш процесів користувацького простору використовується для забезпечення дерева процесів. Під час аналізу дерева процесів виявляється джерело усіх процесів, що впливають на ініційовані ситуації, незалежно від батьківського процесу. Це є ще одним прикладом можливостей, які з’являються в результаті використання інструментів безпеки під час виконання: при вході до файлової системи proc можна спостерігати, що коли процес завершується, його дочірні процеси раптово переходять до процесу з ідентифікатором [124]. Це означає, що вони нещодавно відновлюють контекст дерева батьківських процесів, хоча вони могли бути важливою частиною контексту,

яка могла сказати вам, який сервіс чи процес було використано. Селектор process.ancestors у синтаксисі SecL дозволяє написати умову для одного з батьків процесу, незалежно від кількості проміжних процесів, які могли бути додані для спроби обійти виявлення [125]. Щоб спростити виявлення веб-оболонок, можна використовувати наступне правило:

```
exec path of file in ["/bin/bash", "/bin/sh", ...] && ancestor process path of file == "/bin/my_web_server"
```

Лістинг 2.15 Приклад правила моніторингу виконання процесу

Ще однією цікавою перевагою занурення в ядро глибше, ніж рівень системного виклику, є можливість працювати з фрагментами інформації, які зазвичай навіть не доступні у користувацькому просторі. Наприклад, можна отримати шар файлу у файловій системі overlayfs. Ця інформація має потужні наслідки для безпеки, оскільки вона може бути використана для визначення того, чи був файл, який збираються виконати, частиною базового образу контейнера, або чи був він змінений (чи просто створений) порівняно з його оригінальною версією в базовому образі. Виявлення такого складного варіанту використання можна описати одним простим правилом:

```
exec . file . in_upper_layer == true
```

Лістинг 2.16 Приклад правила верхнього рівня

Аналогічно, також можна збирати облікові дані процесу та доповнювати ними інші події. Це означає, що збирається повний набір ідентифікаторів користувачів і груп, а також можливості ядра і метадані виконуваних файлів. Це дозволяє писати додаткові правила для процесів з небезпечно широким доступом, таких як CAP_SYS_ADMIN, або просто виявляти виконання бінарних файлів з встановленими бітовими прапорами setuid або setgid.

Для модуля "Моніторинг процесів з використанням eBPF" важливо враховувати основні атрибути процесів, щоб забезпечити докладний і цілісний аналіз їхньої активності. Ось поля, які варто логувати:

Таблиця 2.7.

Схема логування модуля моніторингу процесів

Назва поля	**Опис**
TIMESTAMP	Час події
PID	Ідентифікатор процесу
PPID	Ідентифікатор батьківського процесу
UID	Ідентифікатор користувача, під яким запущено процес
COMMAND_NAME	Назва команди процесу
CMDLINE	Повна командна строка процесу
STATUS	Статус процесу (наприклад, RUNNING, SLEEPING, ZOMBIE).
CPU_USAGE	Використання CPU процесом
MEMORY_USAGE	Використання пам'яті процесом
START_TIME	Час запуску процесу
PRIORITY	Пріоритет процесу
NETWORK_ACTIVITY	Мережева активність, пов'язана з процесом
FILE_OPERATIONS	Операції з файлами, які виконує процес

Ці поля надають комплексний огляд активності процесів на системі та допомагають виявляти аномалії або підозрілі поведінки в процесах. Приклад виводу:

TIMESTAMP PID PPID UID COMMAND_NAME CMDLINE STATUS CPU_USAGE MEMORY_USAGE START_TIME PRIORITY NETWORK_ACTIVITY FILE_OPERATIONS ERROR_MESSAGE COMMENT

```
2023-10-26 13:18:25 3652 1 1001 nginx nginx -g daemon off; RUNNING 5% 80MB 2023-10-26 12:15 20 80Mbps /var/log/nginx/ - Web server process
2023-10-26 13:19:01 3678 3652 1001 nginx-worker - SLEEPING 2% 20MB 2023-10-26 12:16 19 20Mbps /var/www/html/ - Nginx worker thread...
```

Лістинг 2.17

2.7 Використання eBPF для моніторингу показників продуктивності

2.7.1 Загальна ідея показників продуктивності

Показники продуктивності є ключовими метриками, які дозволяють оцінювати роботу комп'ютерної системи або програми. Вони надають інформацію про те, як система використовує свої ресурси, такі як процесорний час, пам'ять, пропускна здатність мережі та швидкодія введення/виведення (I/O).

Вплив програм-вимагачів на показники продуктивності

Програми-вимагачі є шкідливими програмами, які шифрують дані користувача та вимагають викуп за їх розшифровку. Вони можуть впливати на ряд показників продуктивності:

- Використання CPU: Віруси-вимагачі можуть інтенсивно використовувати процесор під час шифрування файлів, що призводить до збільшення навантаження на CPU.
- Активність вводу/виводу: Шифрування та переміщення великої кількості файлів можуть призвести до істотного збільшення активності вводу/виводу, особливо якщо програма працює з файлами великого розміру.

- Використання пам'яті: Деякі віруси-вимагачі можуть використовувати значний об'єм оперативної пам'яті, що в свою чергу може вплинути на загальну продуктивність системи.

З урахуванням цього впливу програм-вимагачів на показники продуктивності може бути важливим для їхнього виявлення. Зокрема, спостереження за аномальними змінами цих показників може служити попередженням про потенційну присутність шкідливого програмного забезпечення в системі. eBPF, завдяки своїм механізмам моніторингу, може стати ефективним інструментом для відстеження таких змін і виявлення активності вірусів-вимагачів.

eBPF представляє собою потужний інструмент для моніторингу та аналізу системних подій на ядрі Linux. З його допомогою розробники можуть створювати програми, які взаємодіють безпосередньо з ядром, дозволяючи відстежувати різноманітні аспекти системної діяльності, включаючи показники продуктивності.

Можливості eBPF у моніторингу показників продуктивності:

1. **Низька накладна вартість:** В порівнянні з іншими інструментами моніторингу, eBPF має відносно невелику накладну вартість, що робить його ефективним для використання у високонавантажених системах.

2. **Гнучкість:** Програми eBPF можуть бути динамічно завантажені та вивантажені з ядра, дозволяючи налаштування моніторингу відповідно до потреб користувача.

Приклади інструментів та фреймворків на базі eBPF:

- **BCC (BPF Compiler Collection):** Набір утиліт та бібліотек для розробки, компіляції та запуску програм eBPF. Зокрема, включає інструменти для моніторингу продуктивності, такі як **biotop** (моніторинг I/O) та **cpudist** (дистрибуція використання CPU) [126].

- **BPFtrace:** Високорівнева мова для трейсингу на базі eBPF, яка дозволяє швидко писати скрипти для аналізу поведінки системи [127].

У контексті моніторингу програм-вимагачів eBPF може стати ключовим інструментом для виявлення аномальних патернів використання ресурсів, таких як велике навантаження на CPU або збільшення активності I/O, що можуть вказувати на активність шкідливого програмного забезпечення.

Для модуля "Використання eBPF для моніторингу показників продуктивності" рекомендується зосередитися на ключових показниках продуктивності системи, які впливають на загальну роботу і стабільність. eBPF дозволяє отримати докладний зріз даних на високому рівні. Ось поля, які було б корисно враховувати:

Таблиця 2.8.

Схема логування модуля моніторингу продуктивності

Назва поля	**Опис**
TIMESTAMP	Час події
CPU_USAGE	Загальне використання CPU в процентах
MEMORY_USAGE	Загальне використання оперативної пам'яті в процентах
DISK_IO	Швидкість введення/виведення диска
NETWORK_IO	Швидкість введення/виведення мережі
CACHE_HIT_RATE	Частота попадань в кеш
CONTEXT_SWITCHES	Кількість перемикань контексту на ядро за секунду
LOAD_AVERAGE	Середнє навантаження на систему за 1, 5 та 15 хвилин
THREAD_COUNT	Загальна кількість потоків, що виконуються
SYSTEM_CALLS_RATE	Кількість системних викликів за секунду

Приклад виводу:

```
TIMESTAMP CPU_USAGE MEMORY_USAGE DISK_IO NETWORK_IO LATENCY ERROR_RATE CACHE_HIT_RATE CONTEXT_SWITCHES LOAD_AVERAGE THREAD_COUNT SYSTEM_CALLS_RATE COMMENT
```

2023-10-26 14:25:05	50%	70%	120MB/s	500Mbps	2ms	0.5%	95%	1000	0.70,1.20,1.50	300	2500	Normal performance
2023-10-26 14:26:10	80%	90%	150MB/s	1Gbps	5ms	1%	90%	2000	1.00,1.30,1.60	500	3000	High traffic detected

Лістинг 2.18

Ці показники допоможуть операторам та інженерам відстежувати стан системи, виявляти проблеми продуктивності та реагувати на них відповідно.

2.8 Доступ до даних ядра за допомогою eBPF

Сьогодні BPF є популярним інструментом для розв’язання різних проблем, зокрема отримання інформації зі структур даних ядра. Існують різні способи, які використовують BPF для вивантаження інформації зі структур даних ядра в простір користувача.

Структурні дампери. Юнхонг Сон запропонував віртуальні файли, що дозволяють інтегрувати BPF-програми для створення файлів як у /proc для обраних даних [128]. Він створив нову віртуальну файлову систему для /sys/kernel/bpfdump, що відображає сталі дані під час монтажу та дозволяє ядру формувати підкаталоги, наприклад, для BPF карт.

Серед патчів — новий тип BPF-програми BPF_TRACE_DUMP, який виводить дані через функції bpf_seq_printf() та bpf_seq_write(). Програми стандартно завантажуються, з можливістю закріплення у /sys/kernel/bpfdump, як "myps" в папці /sys/kernel/bpfdump/task.

Алан Магуайр розробив метод налагодження з printk() для детального виводу даних [128]. Завдяки BTF (Binary Type Format), в ядрі тепер можна дістати більше інформації про структури. Його зміни дозволяють друкувати структури з назвами

полів, що корисно не лише для printk, але й для інструментів як Ftrace. Це демонструє два способи використання BPF для доступу до даних ядра поза його межами.

BPF Iterator. Існує декілька способів скидання даних ядра у простір користувача [129]. Найпопулярнішим з них є система /proc. Наприклад, 'cat /proc/net/tcp6' виводить всі tcp6-сокети в системі, а 'cat /proc/net/netlink' виводить всі netlink-сокети в системі. Однак формат виводу, як правило, фіксований, і якщо користувачі хочуть отримати більше інформації про ці сокети, їм доведеться вносити виправлення в ядро, що часто вимагає часу на публікацію та випуск. Те саме стосується популярних інструментів, таких як ss, де будь-яка додаткова інформація потребує виправлення ядра. Для вирішення цієї проблеми часто використовують інструмент drgn, який витягує дані з ядра без зміни ядра. Але головним недоліком drgn є продуктивність, оскільки він не може виконувати трасування вказівників всередині ядра. Крім того, drgn може видавати неправильні результати, якщо вказівник стає недійсним всередині ядра. Ітератор BPF вирішує вищезгадану проблему, надаючи гнучкість щодо того, що збирати при одноразовій зміні для певної структури даних у ядрі, і виконуючи все відстеження вказівника всередині ядра. Така гнучкість досягається за рахунок використання bpf програм. Коректність забезпечується реалізацією трасування вказівника всередині ядра з належним підрахунком посилань або захистом від блокування. У поточному стані ітератор змінює лише невелику частину структур даних у ядрі.

Ітератор BPF використовує файл ядра для передачі даних у простір користувача. Дані можуть бути форматованим рядком або необробленими даними. У випадку форматованого рядка необхідно скористатися підкомандою bpftool iter, щоб створити ітератор bpf і прив'язати його за допомогою bpf_link до шляху у файловій системі BPF (bpffs). Після цього необхідно виконати команду 'cat <шлях>' для друку результатів, подібну до 'cat /proc/net/netlink'. Наприклад, наступна

команда використовується, щоб закріпити програму bpf в об'єктному файлі bpf_iter_ipv6_route.o за шляхом /sys/fs/bpf/my_route:

```
$ bpftool iter pin ./bpf_iter_ipv6_route.o  /sys/fs/bpf/my_route
```

Лістинг 2.8.1

Пізніше результати буде видно за допомогою наступної команди:

```
$ cat /sys/fs/bpf/my_route
```

Лістинг 2.19

Щоб реалізувати ітератор bpf у ядрі, розробник повинен заповнити наступну ключову структуру даних, визначену у файлі bpf.h.

```
struct bpf_iter_reg {
        const char *target;
        bpf_iter_attach_target_t attach_target;
        bpf_iter_detach_target_t detach_target;
        bpf_iter_show_fdinfo_t show_fdinfo;
        bpf_iter_fill_link_info_t fill_link_info;
        bpf_iter_get_func_proto_t get_func_proto;
        u32 ctx_arg_info_size;
        u32 feature;
        struct bpf_ctx_arg_aux ctx_arg_info[BPF_ITER_CTX_ARG_MAX];
        const struct bpf_iter_seq_info *seq_info;       }
```

Лістинг 2.20

Після заповнення полів структури даних потрібно викликати 'bpf_iter_reg_target()', щоб зареєструвати ітератор у головній підсистемі ітераторів bpf. Нижче наведено розбивку для кожного поля у структурі bpf_iter_reg.

eBPF дозволяє глибоко інтегруватися з ядром Linux, що надає можливість моніторити та аналізувати різноманітні аспекти системної діяльності. Ось поля, які було б корисно логувати при доступі до даних ядра:

Таблиця 2.9.

Схема лог-файлу для модуля моніторингу ядра операційної системи.

Назва поля	**Опис**
TIMESTAMP	Час події
DATA_TYPE	Модуль або компонент ядра, з якого були отримані дані
DATA_VALUE	Значення або вміст отриманих даних
ACTION	Дія, що була виконана з даними (читання, модифікація, видалення тощо)
ASSOCIATED_PROCESS	Процес або додаток, який ініціював доступ
MEMORY_ADDRESS	Адреса пам'яті, де зберігаються дані (якщо відома)

Використання eBPF для доступу до даних ядра дозволяє отримати детальний аналіз системної діяльності та можливість виконувати відповідні дії на зміни в ядрі.

Приклад виводу:

```
TIMESTAMP DATA_TYPE SOURCE_MODULE DATA_VALUE ACTION ASSOCIATED_PROCESS MEMORY_ADDRESS COMMENT
2023-10-26 14:30:05 Routing Table NET_CORE 192.168.1.1/24 via eth0 READ MyApp 0x3f78a2c5 Updated routing info
2023-10-26 14:31:10 Memory Stats MM_MODULE Total: 16GB, Free: 5GB, Used: 11GB READ SystemMonitor 0x4f88b2d7 Periodic check
```

Лістинг 2.21

2.9 Імплементація роботи модулів eBPF на платформі Windows

Проект ebpf-for-windows має на меті дозволити розробникам використовувати знайомі інструменти eBPF та інтерфейси прикладного програмування (API) в існуючих версіях Windows. Спираючись на напрацювання

інших, цей проект бере кілька існуючих проектів eBPF з відкритим вихідним кодом і додає плагіни, які дозволяють їм працювати під Windows.

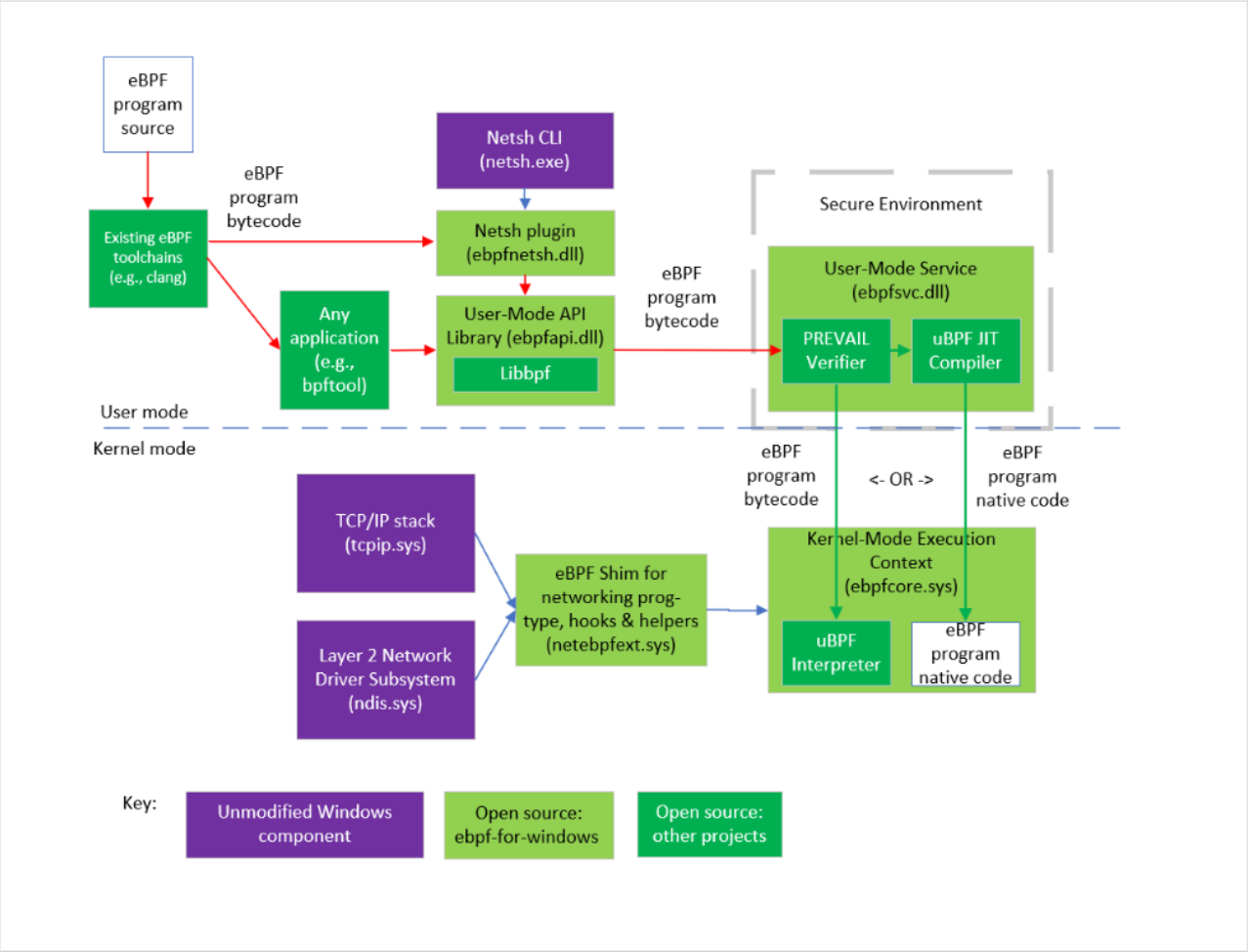


Рис. 2.10. Архітектура інструментальних засобів eBPF для Windows OS

Як показано на Рис. 2.10, існуючі інструментальні засоби eBPF (clang тощо) можна використовувати для генерації байткоду eBPF з вихідного коду на різних мовах. Байт-код може бути спожитий будь-якою програмою, або через bpftool чи інструмент командного рядка Netsh, які використовують спільну бібліотеку, що розкриває API Libbpf, хоча ця робота все ще триває.

Байт-код eBPF надсилається статичному верифікатору (верифікатору PREVAIL), який розміщено в безпечному середовищі користувацького режиму, такому як системна служба (як зараз), анклав або довірена віртуальна машина. Якщо програма eBPF пройшла всі перевірки верифікатора, її можна завантажити у

контекст виконання у режимі ядра. Зазвичай це робиться шляхом JIT-компіляції (за допомогою JIT-компілятора uBPF) у нативний код, який передається у контекст виконання. У налагоджувальній збірці байт-код може бути безпосередньо завантажений у інтерпретатор (з uBPF у контексті виконання у режимі ядра), хоча інтерпретатор не присутній у релізній збірці, оскільки він вважається менш безпечним. Дивіться також відповідь на поширені запитання щодо HVCI нижче.

Програми eBPF, встановлені у контексті виконання у режимі ядра, можуть приєднуватися до різних хуків і викликати різні допоміжні API, доступні за допомогою прокладка eBPF, який внутрішньо обгортає загальнодоступні API ядра Windows, що дозволяє використовувати eBPF у наявних версіях Windows. Багато допоміжних функцій вже існує, і з часом буде додано ще більше хуків та допоміжних функцій.

Висновки до 2 розділу

Другий розділ дисертації зосереджується на вивченні можливостей eBPF у контексті виявлення та протидії програмам-вимагачам. Основні висновки цього розділу можна сформулювати так:

1. Аналіз архітектури і можливостей eBPF підтверджує його високу ефективність як інструменту для розширеного моніторингу у просторі ядра Linux. Це дозволяє виявляти підозрілі зміни та активності на низькому рівні, які можуть бути пов'язані з вірусами-вимагачами, та вживати заходи до їх блокування або подальшого аналізу.

2. Розробка методики створення модулів на базі eBPF, яка дозволяє створювати адаптивні моніторингові інструменти. Ці інструменти можуть бути інтегровані з існуючими системами безпеки, такими як SIEM і EDR, забезпечуючи додатковий рівень захисту.

3. Застосування eBPF для моніторингу системних викликів, файлової активності та мережевого трафіку демонструє його потенціал у виявленні складних патернів поведінки, які можуть вказувати на діяльність вірусів-вимагачів, зокрема в реальному часі.

4. Проведено ретельне дослідження можливостей eBPF для моніторингу показників продуктивності системи, що може виявити надмірне використання ресурсів або інші аномалії, пов'язані з діяльністю шкідливого ПЗ.

5. Розробка та валідація прототипів модулів на базі eBPF підтвердила можливість їх використання для реального виявлення програм-вимагачів, надаючи практичні рекомендації для їх застосування в сучасних комп'ютерних системах.

РОЗДІЛ 3
СТВОРЕННЯ ІНТЕГРОВАНОГО МЕТОДУ АНАЛІЗУ ВІРУСІВ-ВИМАГАЧІВ НА БАЗІ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ

Третій розділ дослідження присвячено розробці інтегрованого підходу до аналізу вірусів-вимагачів, використовуючи передові методи машинного навчання для ідентифікації та класифікації кіберзагроз, у тому числі вірусів-вимагачів. Цей клас зловмисного програмного забезпечення представляє значну загрозу для інформаційної безпеки, оскільки він не лише порушує конфіденційність та цілісність даних користувачів, а й може призвести до значних фінансових втрат.

З огляду на швидкий розвиток і постійну еволюцію вірусів-вимагачів, традиційні підходи до кібербезпеки часто виявляються недостатньо ефективними. Тому в цьому розділі буде звернено увагу на використання машинного навчання як потенційно більш гнучкого та адаптивного інструменту протидії новітнім загрозам, як альтернатива класичним методам. Буде розглянуто різні моделі та алгоритми, такі як дерева рішень, випадкові ліси, методи опорних векторів та нейронні мережі, з акцентом на їхню здатність виявляти складні шаблони та аномалії в поведінці програмного забезпечення. Як вхідні дані будуть використовуватися журнали подій, створені з EBPF модулів, що були описані в 2 розділі дисертаційного дослідження.

Основна увага буде приділена також практичним аспектам застосування цих методів, включаючи формування та обробку наборів даних, вибір та оптимізацію моделей для конкретних типів даних та завдань, а також оцінку ефективності та точності різних підходів. . Ключове завдання полягає у розробці моделей, які не тільки ефективно ідентифікують відомі віруси-вимагачі, але й передбачають появу нових їхніх варіантів, що критично важливо для запобігання атакам перш, ніж вони зможуть завдати шкоди.

3.1 Архітектура інтегрованої системи виявлення вірусів-вимагачів

Архітектура інтегрованої системи виявлення вірусів-вимагачів, представлена у роботі, базується на гнучкій та розширюваній структурі, яка використовує різні технології для моніторингу та аналізу кіберзагроз типу програм-вимагачів.

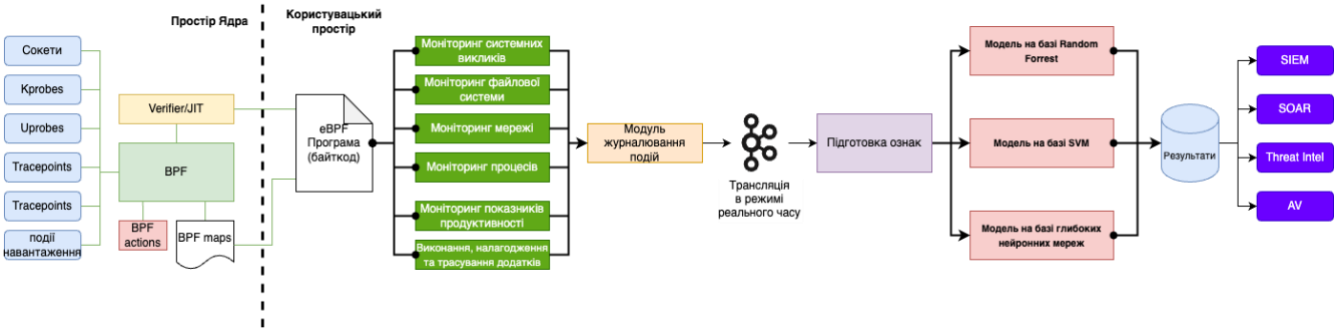


Рис. 3.1. Інтегрована система виявлення вірусів-вимагачів

Основні компоненти системи включають:

Моніторинг системних подій: Використовується BPF (Berkeley Packet Filter), зокрема програми BPF і BPF maps, для збору даних про системні виклики, мережеву активність та процеси. Це дозволяє виявляти аномальні поведінки, які можуть вказувати на наявність вірусу-вимагача [130].

Моделі на основі машинного навчання: Використовуються різні моделі, зокрема Random Forest, SVM (Support Vector Machine), і моделі для аналізу мережевих нейронних мереж, для аналізу зібраних даних та класифікації поведінки як безпечної чи потенційно шкідливої [131].

Аналіз та транспортування даних в реальному часі: Процес транскрипції даних здійснюється в реальному часі, що забезпечує швидку реакцію на можливі загрози.

Інтеграція з іншими системами безпеки: Система інтегрована з вищим рівнем безпеки, включаючи SIEM (Security Information and Event Management), SOAR (Security Orchestration, Automation, and Response) платформи, Threat Intel та

антивірусне програмне забезпечення, що дозволяє здійснювати комплексний захист.

3.2 Розроблення модельного конвеєра

У даному підрозділі представлений стратегічний підхід до машинного навчання, який використовує дані з eBPF модулів для ефективного виявлення програм-вимагачів. Конвеєр розроблено для підвищення точності виявлення програм-вимагачів і зміцнення системної безпеки. Він включає п'ять етапів: реєстрація подій через eBPF, нормалізація даних, створення моделі за допомогою SVM, оцінка моделі та виявлення загроз у реальному часі [132]. Кожен крок важливий для оптимізації даних і забезпечення надійного виявлення.

Збір та обробка даних з з eBPF модулів. Збір даних з різних системних рівнів та джерел є ключовим етапом для ефективного моніторингу та аналізу. Використання eBPF для цієї цілі дозволяє отримувати гранульовані та високоякісні дані у реальному часі. Ось основні кроки цього процесу:

Рис. 3.2. Машинне навчання: Розроблення модельного конвеєру

Ця діаграма відображає стандартний процес обробки даних та розробки моделі машинного навчання.

1. Події з модуля: Це початковий етап, на якому збираються дані або події від деяких детекторів чи сенсорів. Наприклад, це може бути інформація від камер спостереження, датчиків руху тощо.

2. Нормалізація даних (Підготовка даних та Конструювання ознак): На цьому етапі зібрані дані обробляються та підготовлюються для подальшого аналізу. Основна ціль - перетворити "сиру" інформацію в корисні характеристики (або "особливості"/features), які можуть бути використані для навчання моделі.

3. Навчання та розроблення моделі: Після того, як дані були підготовлені, настає час розробки та навчання моделі. Це включає в себе вибір відповідного алгоритму, настройку параметрів та використання підготовлених даних для "навчання" моделі розпізнавати шаблони чи здійснювати прогнози.

4. Оцінка моделі: Після навчання моделі її необхідно перевірити на нових, раніше невідомих даних, щоб оцінити, наскільки добре вона працює.

5. **Прогнозування/Виявлення**: З використанням навченої та перевіреної моделі можна здійснювати прогнози на нових даних.

Програми eBPF моніторять різні системні події та взаємодії, збираючи вихідні дані, які можуть сигналізувати про потенційну активність шкідливих програм [133]. Ці дані включають шаблони системних викликів, шляхи доступу до файлів та інші метадані процесів. Важливість цих деталей, зібраних через цей процес, є ключовою для аналізу і виявлення потенційних загроз. Збагачені та детальні дані, зібрані на цьому етапі, є основою для подальших кроків, забезпечуючи всебічний аналіз і точне виявлення вірусів-вимагачів. Наступний процес – нормалізація збагачених даних. Данний крок має вирішальне значення для підготовки даних для моделі машинного навчання. Він передбачає перетворення

необроблених даних у послідовний формат і масштаб, що робить їх більш придатними для аналізу [134].

Нормалізація допомагає усунути будь-які помилки, упередження або аномалії, спричинені різними масштабами та форматами, гарантуючи, що кожна ознака робить однаковий внесок у роботу моделі. Цей крок підвищує ефективність і точність моделі машинного навчання, прокладаючи шлях до більш надійних прогнозів і виявлень. Коли нормалізовані дані готові, наступним кроком є введення цих даних у модель машинного навчання.

Після навчання моделі важливо протестувати її роботу. Тестування моделі заключається у оцінці моделі на окремому тестовому наборі даних, з яким вона раніше не стикалася [135]. Цей крок допомагає оцінити здатність моделі узагальнювати свої знання на нових, невідомих даних. Для вимірювання продуктивності моделі використовуються різні метрики, такі як точність, достовірність, пригадування та оцінка. Дані, отримані на цьому етапі, використовуються для вдосконалення та оптимізації моделі, задля можливості прогнозування та виявлення.

Останній етап полягає у прогнозуванні та ідентифікації. Використовуючи вдосконалену модель, дані eBPF обробляються в режимі реального часу для прогнозування та розпізнавання потенційних дій кіберзлочинців. Модель оцінює отримані дані, розпізнає візерунки та поведінку та передбачає можливі дії шкідливих програм. У випадку знаходження активності шкідників активуються аларми і вживаються заходи для мінімізації ризиків. Такі дії є критично важливими для забезпечення захисту систем від вірусів-вимагачів у реальному часі, що забезпечує безпеку та стабільність операцій.

Визначення джерел даних.

Спершу потрібно визначити джерела даних, які будуть моніторитися. Це може бути файлова система (для відстеження доступу до файлів), мережевий трафік, процеси або показники продуктивності.

Джерела даних, що використовуються у цьому досліджені:

- Файлова система
- Моніторинг файлів
- Моніторинг активності шифрування файлів та директорій
- Моніторинг та аналіз мережевої активності на рівні ядра
- Моніторинг процесів
- Моніторинг показників продуктивності
- Моніторинг доступу до ядра

Як уже було описано у модулі моніторингу файлової системи та криптографічної активності, ось поля, які збираються модулем:

Таблиця 3.1

№	**Назва поля**	**Опис**	**Приклад**	**Назва модулю**
1	TIMESTAMP	Час реєстрації події.	2023-10-26 14:30:05	All
2	EVENT_TYPE	Тип події.	Network Transmit	All
3	STATUS	Статус події (успішно, помилка).	SUCCESS	All
4	SOURCE_IP	IP-адреса відправника.	192.168.1.10	Network
5	DESTINATION_IP	IP-адреса одержувача.	192.168.1.55	Network
6	PROTOCOL	Використовуваний протокол (TCP, UDP).	TCP	Network
7	PACKET_SIZE	Розмір пакета даних.	1500 bytes	Network
8	PID	Ідентифікатор процесу.	12345	Syscalls
9	ACTION	Дія (читання, запис).	READ	Syscalls
10	MEMORY_ADDRESS	Адреса пам'яті.	0x3F78A2C5	Syscalls

11	PROCESS_NAME	Назва процесу.	MyApp	Processes
12	USER_ID	Ідентифікатор користувача, що запустив процес.	1001	Processes
13	CPU_USAGE	Використання CPU.	35%	Performance
14	MEMORY_USAGE	Використання пам'яті.	4GB/16GB	Performance
15	DATA_TYPE	Тип даних ядра.	Routing Table	Kernel Data
16	SOURCE_MODULE	Модуль або компонент ядра.	NET_CORE	Kernel Data
17	DATA_VALUE	Значення або вміст даних.	Total: 16GB, Free: 5GB	Kernel Data

Для підготовки даних використовується спеціально розроблений Python-скрипт (Додаток В). Його основна мета - перетворити вхідні лог-файли в структуровані датасети, які можна використовувати для подальшого аналізу або навчання моделей.

Основні константи:

- `TIME_PERIOD`: Визначає період часу для обчислення частоти події. За замовчуванням дорівнює одній секунді.

- `TYPE_NAMES`: Перелік типів подій.

- `PID_OFFSET`: Зміщення для уникнення дублювання PID під час різних запусків детектора.

- `LOGDIR` і `DATADIR`: Шляхи до папок з логами та даними відповідно.

Функції:

- `counts(df)`: Підраховує агреговану статистику подій за різними критеріями (тип події, період часу тощо). Формує агрегований датафрейм на основі вхідного датафрейма, групує події за PID, типом події та часовим періодом.

- `sequences(df)`: Аналізує послідовності подій і підраховує їх кількість. Формує датафрейм з послідовностями подій (довжина 3) для кожного PID.

- `main()`: Головна функція, яка обробляє вхідні лог-файли та генерує вихідні датасети.

Основна логіка:

1. Скрипт обробляє лог-файли з папок "training" та "testing".

2. Для кожного лог-файлу відбувається зчитування даних і застосування `PID_OFFSET` для уникнення дублювання PID.

3. Після цього тип події замінюється на відповідний символьний код.

4. Застосовуються функції `counts` та `sequences` для підрахунку статистики та аналізу послідовностей.

5. Отримані результати зберігаються у CSV-файл.

Підготовка даних з лог-файлів

Скрипт виконує наступні дії:

1. Встановлює константи для визначення часового періоду, типів подій та інших параметрів.

2. Зчитує лог-файли з папок `training` та `testing` з директорії `../logs/`.

3. Змінює ідентифікатор процесу (PID) для кожного лог-файлу, щоб уникнути дублювання.

4. Перетворює типи подій з чисел до символьних міток.

5. Обчислює кількість подій, згрупованих за типом, періодом і PID.

6. Підраховує послідовності подій довжиною 3.

7. Об'єднує результуючі датафрейми та зберігає результати в форматі CSV.

Вихідні дані зберігаються у папці `../data/` у двох файлах: `training_data.csv` та `testing_data.csv`.

Для використання скрипта потрібно запустити його через командний рядок або інший інтерфейс.

Процес підготовки даних

Для забезпечення безшовного аналізу важливо уточнити та структурувати “сирі” дані. У робочому процесі використовується спеціалізований Python-скрипт (dataprep.py) для автоматизації цього процесу. Ось огляд етапу підготовки даних на основі зразкового виводу:

1. **Виконання скрипта**: початок процесу підготовки даних полягав у виконанні команди `./dataprep.py`. Ця команда активує Python-скрипт, відповідальний за попередню обробку даних.

2. **Стовпці особливостей**: таблиця даних складається з кількох стовпців, кожен з яких представляє унікальні особливості. Ці особливості включають:

- `C_max`, `C_sum`: максимальна та кумулятивна кількість дій 'Створення'.

- `D_max`, `D_sum`: максимальна та кумулятивна кількість дій 'Видалення'.

- `E_max`, `E_sum`: максимальна та кумулятивна кількість дій 'E' ('Виконання').

- `O_max`, `O_sum`: максимальна та кумулятивна кількість дій 'Відкриття'.

- `P_max`, `P_sum`: максимальна та кумулятивна кількість дій 'P' ('Процес').

Також у нас є стовпці, які представляють шаблони дій, такі як:

- `CDO`, `COC`, `COO`, `DOC`, `DOO`, `EEE`, `OCD`, `OCO`, `OOC`, `OOO`: ці стовпці позначають послідовності операцій. Наприклад, `OCO` може представляти послідовність дій ‘Відкриття’, ‘Створення’, ‘Відкриття’.

3. **Індексація за ID процесу (PID):** таблиця даних використовує ID процесу (PID) як свій індекс, що дозволяє легше відображати та отримувати інформацію, специфічну для процесу. Це сприяє відмінності та аналізу поведінки кожного процесу окремо.

4. **Інтерпретація**: спостерігаючи за даними, можна розпізнати шаблони, асоційовані з певними PID. Наприклад, PID `1019` показує велику кількість операцій `EEE`, що вказує на повторне виконання дій. Деякі PID мають високу частоту певних шаблонів; наприклад, PID `30495` вказує на помітний шаблон файлових операцій із великою кількістю послідовностей `COC`, `CDO` та `OCO`.

5. **Аналіз після обробки:** після структурування даних, як це зазначено у таблиці, їх можна подавати до подальших аналітичних інструментів або алгоритмів машинного навчання. Розуміючи шаблони у цих особливостях, особливо шаблони файлових операцій, стає можливим виявлення аномалій або потенційно шкідливих поведінок.

У підсумку, скрипт `dataprep.py` сприяє організованому представленню складних системних процесів, що робить можливим отримання інформації.

Процес ID	Макс C	Сума C	Макс D	Сума D	Макс E	Сума E	Макс O	Сума O	Макс P	Сума P	КДО	КЦК	ЦКО	ДЦК	ДОО	ЕЕЕ	ОЦД	ОЦО	ООЦ	ООО
1	0	0	0	0	0	4	5	0	0	0	0	0	0	0	0	0	0	0	3
221	0	0	0	0	0	20	20	0	0	0	0	0	0	0	0	0	0	0	18
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
30495	233	1216	17	983	0	0	278	1458	17	983	983	228	5	949	34	0	983	233	39	201
30496	0	0	0	0	0	3	3	0	0	0	0	0	0	0	0	0	0	0	0	1

Лістинг 3.1. Результат виконання скрипту dataprep.py

3.2.1 Формування наборів даних

Датасет та важливість ознак. У наших експериментах було використано датасет приблизно з 1,5 мільйона подій, який містив 2786 різних процесів, з яких 53 були прикладами вірусів-вимагачів. Датасет було розділено відповідно до Таблиці 3.1. Всі зразки вірусів-вимагачів, які були використані для експерименту, були отримані з MalwareBazaar - проекту, основна мета якого полягає у зборі та обміні

зразками шкідливого програмного забезпечення [136]. Успішно протестовані родини рансомвейру включають такі:

- IceFire
- MONTI
- REvil
- AvosLocker
- BlackMatter
- HelloKitty

Хоча було розглянуто багато інших родин вірусів-вимагачів, більшість з них, або виявила, що система, на якій вони виконуються, була віртуальною машиною, або не була сумісна з нашою операційною системою, або не були виконані вимоги для їх запуску (наприклад, відсутні спільні бібліотеки), чи відсутність живого серверу керування. Модель призначає кожній ознаці певну вагу на основі її відповідності у прогнозуванні виводу. У нашому випадку найважливіші ознаки зосереджуються навколо конкретних шаблонів операцій з файлами, а саме послідовностей дій "Відкрити", "Створити" та "Видалити".

Серед цих шаблонів три найбільш впливові ознаки є:

1. Відкрити, Створити, Відкрити (OCO): Цей шаблон представляє послідовність відкриття файлу, створення нового файлу та потім відкриття іншого файлу. Він показує конкретний поведінковий шаблон, пов'язаний з активністю вірусу-вимагача.

2. Створити, Відкрити, Створити (COC): Цей шаблон передбачає створення файлу, відкриття існуючого файлу та потім створення іншого файлу. Він захоплює відмінну послідовність операцій з файлами, які часто демонструють зразки вірусу-вимагача.

3. Створити, Відкрити, Відкрити (COO): Цей шаблон показує послідовність створення файлу, відкриття існуючого файлу та потім відкриття іншого файлу. Він

представляє інший поведінковий шаблон, який допомагає ідентифікувати потенційну активність вірусів вимагачів.

Ці шаблони відіграють ключову роль у відрізненні поведінки програм-вимагачів від звичайних системних операцій. Їх велика важливість свідчить про те, що вони надають сильний дискримінаційний потенціал для ефективного виявлення вірусів-вимагачів. Додатково, інші важливі ознаки включають кількість операцій видалення (D sum) та максимальну кількість операцій видалення (D max). Ці ознаки фіксують частоту та інтенсивність операцій видалення файлів, які часто є показником поведінки вірусу-вимагача.

Під час формування наборів даних, основна увага приділяється обробці інформації з лог-файлів, отриманих з папок `training` та `testing`. Ця інформація допомагає створити структуровані фрейми даних, які відображають статистичні характеристики та послідовності подій.

Етапи формування:

1. **Зчитування лог-файлів:** Лог-файли зчитуються з директорій `training` та `testing`, розташованих у папці `../logs/`.

2. **Корекція PID:** Кожен лог-файл може мати свої унікальні PID, тому для уникнення дублювання значень PID у комбінованому датафреймі, до них додається зміщення `PID_OFFSET`.

3. **Конвертація типів подій**: Типи подій, які представлені числами (0,1,2,3), конвертуються в символьні мітки, використовуючи список `TYPE_NAMES`.

4. **Агрегація подій:** З використанням функції `counts`, події групуються за PID, типом і часовим періодом. Результатом є агрегований датафрейм, який відображає кількість подій для кожного типу в кожному часовому інтервалі.

5. **Формування послідовностей:** Функція `sequences` створює новий датафрейм, в якому розглядаються послідовності подій довжиною в 3 події. Це дозволяє виділити зв'язки між послідовними подіями.

6. **Комбінація датафреймів:** Агрегований датафрейм і датафрейм з послідовностями об'єднуються для створення кінцевого датафрейма, який містить всю необхідну інформацію.

7. **Зберігання даних:** Фінальні фрейми даних зберігаються у папці `../data/` у файлах `training_data.csv` та `testing_data.csv`.

Цей процес гарантує, що набори даних, які використовуються для навчання та тестування моделі, є добре структурованими, консистентними і відображають реальні події та їхні послідовності. Для розробки та перевірки ефективності моделі машинного навчання, яка здатна розпізнавати вірусів-вимагачів від безпечних файлів, ключовим етапом є правильне формування наборів даних. Це забезпечує, що модель буде навчена та перевірена на релевантних та репрезентативних даних.

3.3 Вибір моделей машинного навчання для аналізу даних

Залежно від характеру даних, що використовуються для побудови моделі (навчальних даних), алгоритми машинного навчання поділяються на такі чотири категорії: контрольоване машинне навчання, неконтрольоване машинне навчання, напівконтрольоване машинне навчання та машинне навчання з підкріпленням.

У контрольованому машинному навчанні модель будується шляхом навчання на маркованих даних. При неконтрольованому навчанні модель працює з точками даних без будь-яких міток. Метою неконтрольованого алгоритму є організація даних у групи кластерів для опису їхньої структури. Це робить складні дані простими та організованими. Напівкеровані алгоритми навчання знаходяться між керованими і некерованими алгоритмами навчання. У цьому випадку алгоритму надається деяка кількість маркованих даних, і він працює на немаркованих даних. Алгоритми з підкріпленням вибирають дію на основі кожної точки даних, а потім

оцінюють, наскільки правильним було рішення [137]. Після цього алгоритм змінює свою стратегію, щоб краще вчитися і досягати кращих результатів.

Машине навчання з вчителем. Деякі приклади алгоритмів керованого навчання є наступними:

1. Лінійну, логістичну, поліноміальну регресії та метод найменших квадратів для задач регресії;
2. k-найближчих сусідів (k-NN) для класифікації;
3. Дерева рішень для класифікації;
4. Наївний Байєс для класифікації;
5. Метод опорних векторів (SVM) для класифікації;
6. Випадковий ліс для класифікації та регресії;
7. Ансамблеві методи для класифікації та регресії.

Машине навчання без вчителя. При кластеризації мета полягає у виявленні притаманних даним угруповань, а при вивченні правил асоціацій - у виявленні правил, які описують великі порції даних [138]. Деякі приклади алгоритмів неконтрольованого навчання наведені нижче:

1. k-середні для кластеризації,
2. Ієрархічна кластеризація для кластеризації,
3. Факторний аналіз для кластеризації,
4. Змішані моделі для кластеризації,
5. Апріорний алгоритм для задачі асоціації,
6. Аналіз головних компонент (PCA),
7. Декомпозиція сингулярних значень
8. Аналіз незалежних компонент.

Більшість алгоритмів кластеризації розроблені для числових даних, які використовують поняття відстані. Отже, існують певні технічні труднощі у

застосуванні алгоритмів кластеризації до проблем кібербезпеки з дискретними даними, такими як URL-адреси, імена користувачів, IP-адреси, номери портів тощо.

Навчання з підкріпленням. Цей метод, відомий як RL, ефективно інтегрує принципи динамічного програмування та контрольованого навчання, створюючи потужний інструмент для аналізу та вирішення завдань. [139]. Це процес навчання методом спроб і помилок для оптимізації поведінки агенту, який взаємодіє зі стохастичним середовищем. Агент виконує дію A(t) на кожному часовому кроці t, а потім сприймає сигнал S(t) від середовища і отримує винагороду R(t).

Основні компоненти системи навчання з підкріпленням включають такі елементи:

1. Агент - це елемент, який взаємодіє з середовищем та виконує дії. Агент може бути як вчителем, так і приймачем рішень.

2. Середовище - це простір, в якому агент рухається і навчається. Воно реагує на дії агента та надає йому нові ситуації.

3. Стан (S_t) - це поточна ситуація, яку середовище повертає агенту. Кожен стан визначається в конкретний момент часу t.

4. Винагорода (R_t) - це число, яке агент отримує як результат виконання дії. Винагорода може бути позитивною або негативною.

5. Дія (A_t) - це те, що агент вирішує зробити в конкретному стані. Виконання дії впливає на середовище, яке переводить агента в новий стан і надає винагороду чи штраф.

Передбачається, що агент розпочинає взаємодію з середовищем, отримуючи стан S_t і вибираючи дію A_t, яку потрібно виконати. Після цього середовище повертає новий стан S_{t+1} та винагороду R_{t+1}, які допомагають агентові навчитися вибирати кращі дії в майбутньому.

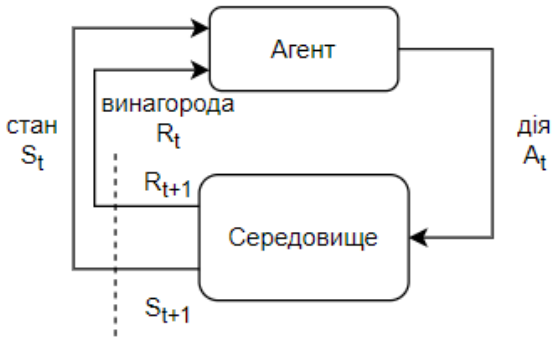


Рис. 3.3. Схема роботи навчання з підкріпленням

Алгоритми навчання з підкріпленням застосовуються у різних проблемах кібербезпеки, таких як виявлення вторгнень в мережу, виявлення аномалій та інших. Пізніше будуть коротко розглянуті різні моделі машинного навчання.

Наївний класифікатор Байєса застосовується, коли наявний середній або великий навчальний набір даних з точками даних, що мають кілька атрибутів або ознак, і, враховуючи параметр класифікації, атрибути є умовно незалежними [140].

Деякі з відомих застосувань наївного класифікатора Байєса включають:

1. Виявлення шкідливого програмного забезпечення
2. Виявлення спаму
3. Виявлення вторгнень
4. Виявлення DoS і DDoS-атак
5. Аналіз настроїв

Фільтрація спаму базується на наївному класифікаторі Байєса. Цей класифікатор розподіляє листи на "Спам" та "Не спам". Поштові клієнти, такі як SpamBayes та SpamAssassin, використовують цей метод [141]. У соцмережах він аналізує настрої в оновленнях статусів. Наївний Байєс добре підходить для дискретних даних, швидко сходиться, потребує менше даних для навчання і ефективно працює в багатокласових задачах.

У рамках зусиль по ефективному виявленню дій вірусів-вимагачів, застосовуючи дані, отримані від eBPF програм, було аналізовано ряд алгоритмів машинного навчання. Кожен з алгоритмів демонструє специфічні характеристики

для аналізу та прогнозування на підставі даних. Оцінюючи та тестуючи різні методи, метод опорних векторів (SVM) показав себе особливо придатним для виконання цього завдання [142]. Вибір на користь SVM у даному науковому проекті обґрунтовується його здатністю надійно та гнучко працювати з даними різної складності та розмірності. Функціональність SVM розрізняти класи на основі оптимальної гіперплощини, яка максимально ефективно розділяє класи, забезпечує його ефективність при розпізнаванні складних шаблонів поведінки вірусів-вимагачів. Основні переваги методу включають його спроможність ефективно класифікувати та робити регресійний аналіз, а також працювати з лінійними та нелінійними моделями через застосування різноманітних ядерних функцій. Цей підхід сприяє досягненню високої точності і надійності у виявленні активності вірусів-вимагачів у режимі реального часу, забезпечуючи захист та стабільність комп'ютерних систем.

Таблиця 3.2.
Оцінка алгоритмів машинного навчання

Тип	**Назва**	**Опис**
Класифікація	Випадковий ліс	Ефективно обробляє великі та складні набори даних, моделюючи нелінійні рішення.
	Метод опорних векторів (SVM)	Відмінно працює в просторах з великою розмірністю, ідеальний для бінарної класифікації.
	Дерева рішень	Прості для розуміння і візуалізації, обробляють числові та категоріальні дані.
Виявлення аномалій	Ізоляція лісу	Ефективний для високовимірних наборів даних. Спеціально розроблений для виявлення аномалій.

	Однокласний SVM	Підходить для виявлення викидів у наборах даних високої розмірності.
	Локальний фактор відхилення (LOF)	Вимірює відхилення локальної щільності точки даних відносно її сусідів.
Алгоритми кластеризації	K-Means Clustering	Розбиває набір даних на K кластерів. Може використовуватися для виявлення незвичайних шаблонів.
	DBSCAN	Не потребує вказівки кількості кластерів. Може знаходити кластери довільної форми.
Глибоке навчання	Рекурентнтні нейронні мережі (RNN)	Ідеально підходить для аналізу послідовних даних.
	Автоматичні кодувальники	Використовуються для детекції аномалій через реконструкцію входів.
Аналіз часових рядів	Довга короткочасна пам'ять (LSTM)	Ефективний для даних часових рядів.

Контрольоване машинне навчання є основою методу розшифровки даних про взаємозв'язок між входом і виходом у різних галузях. Воно базується на тому, що система навчається на наборі даних, який складається з парних прикладів вхід-вихід [143]. Ці набори даних, що характеризуються маркованими виходами, спрямовують алгоритм навчання на розуміння та засвоєння складних зав’язків між вхідними даними та відповідними виходами.

Коли вихідні дані класифікуються за допомогою дискретних значень, що позначають різні класи, кероване навчання маневрує в напрямку завдань класифікації. І навпаки, наявність безперервних вихідних значень спрямовує

навчання до задач регресії. Внутрішнє представлення взаємозв'язків вхід-вихід у моделі навчання позначається певними параметрами. Ці параметри, що мають вирішальне значення для роботи моделі, обчислюються на етапі навчання, особливо коли немає прямого доступу до них.

Ландшафт керованого навчання багатий на різноманітні алгоритми, кожен з яких має свої унікальні переваги. Серед них важливе місце займають k-найближчих сусідів (kNN) та метод опорних векторів (SVM). kNN алгоритм базується на принципах близькості. Цей метод визначає категорії нових даних залежно від їхньої просторової близькості до вже відомих розмічених даних, використовуючи мітки найчастіше класу серед k найближчих сусідів. Цей алгоритм не використовує параметризацію, а натомість він класифікує, виходячи з відстаней між новими даними і наявними прикладами в навчальному наборі.

SVM, з іншого боку, відомий своєю надійністю як в задачах класифікації, так і в задачах регресії. Алгоритм працює шляхом визначення оптимальної гіперплощини з метою відокремлення точок даних різних класів з якомога більшим відривом. Він досягає цього шляхом перетворення даних у вимірний простір ознак і ретельної побудови границі рішення, яка збільшує різницю між окремими класами. Гнучкість методу в обробці як лінійно, так і нелінійно розділених даних ще більше підвищується за допомогою різних функцій ядра.

3.3.1 Розробка моделі класифікації програм-вимагачів за допомогою дерев рішень та ансамблів випадкового лісу

Дерева прийняття рішень - це модель, яка реагує на послідовні питання та надає певний маршрут, який призведе до результатів. Модель має декілька умов "якщо це, то це", що в кінцевому підсумку дозволяє отримати очікуваний результат [144].

Переваги використання дерев рішень:

1. Легко інтерпретувати та створювати прості візуалізації.
2. Можна спостерігати за внутрішніми процесами.
3. Добре працюють з великими наборами даних та надзвичайно швидкі

Недоліки дерев рішень:

1. Дерева рішень схильні до перенавчання, особливо коли вони мають багато шарів та складні структури. Перенавчання відбувається, коли дерево рішень стає надто специфічним до набору даних, на якому воно було навчене, і втрачає здатність узагальнювати. Це може призвести до поганих прогнозів або рішень, коли дерево використовується на нових, невідомих даних.
2. Дерева рішень можуть мати труднощі з моделюванням деяких типів відносин між даними, особливо коли ці відносини є складними або нелінійними. Вони добре працюють для простих, ієрархічних структур рішень, але можуть не ефективно впоратися з задачами, де взаємозв'язки між змінними є більш складними або коли потрібно враховувати велику кількість взаємопов'язаних факторів.

Дерево рішень для виявлення вірусів-вимагачів, створене з використанням модулів eBPF, складається з наступних етапів:

Моніторинг файлової системи: На цьому етапі система використовує модуль eBPF для спостереження за активністю у файловій системі. Це включає в себе перевірку на незвичайні зміни файлів, швидкість запису та інші підозрілі патерни, які можуть вказувати на присутність вірусу-вимагача.

Аналіз незвичайної активності: Якщо виявляється незвичайна активність, система переходить до виявлення вірусу-вимагача. Якщо незвичайної активності не виявлено, процес продовжується моніторингом продуктивності.

Моніторинг продуктивності: На цьому етапі система перевіряє, чи є незвичайне споживання системних ресурсів, таких як CPU, пам'ять або мережева активність, що може бути ознакою шкідливої діяльності.

Аналіз споживання ресурсів: Якщо виявлено незвичайне споживання ресурсів, система переходить до виявлення вірусу-вимагача. Якщо ні, то вона переходить до моніторингу системних процесів.

Моніторинг системних процесів: На цьому етапі система аналізує активність системних процесів, шукаючи підозрілі або невластиві процеси, які можуть вказувати на наявність вірусу.

Аналіз системних процесів: Якщо виявлені підозрілі процеси, система переходить до виявлення вірусу-вимагача. Якщо підозрілих процесів не виявлено, система вважається безпечною.

Виявлення вірусу-вимагача: На цьому етапі система ідентифікує потенційний вірус-вимагач на основі зібраних даних і переходить до застосування заходів безпеки.

Застосування заходів безпеки: Після ідентифікації вірусу-вимагача система вживає необхідних заходів, таких як ізоляція зараженого процесу, повідомлення адміністратора системи та ініціація процесів відновлення.

Це дерево рішень дозволяє системі ефективно виявляти та реагувати на віруси-вимагачі, використовуючи різні модулі моніторингу та аналізу, що забезпечує багаторівневий підхід до кібербезпеки.

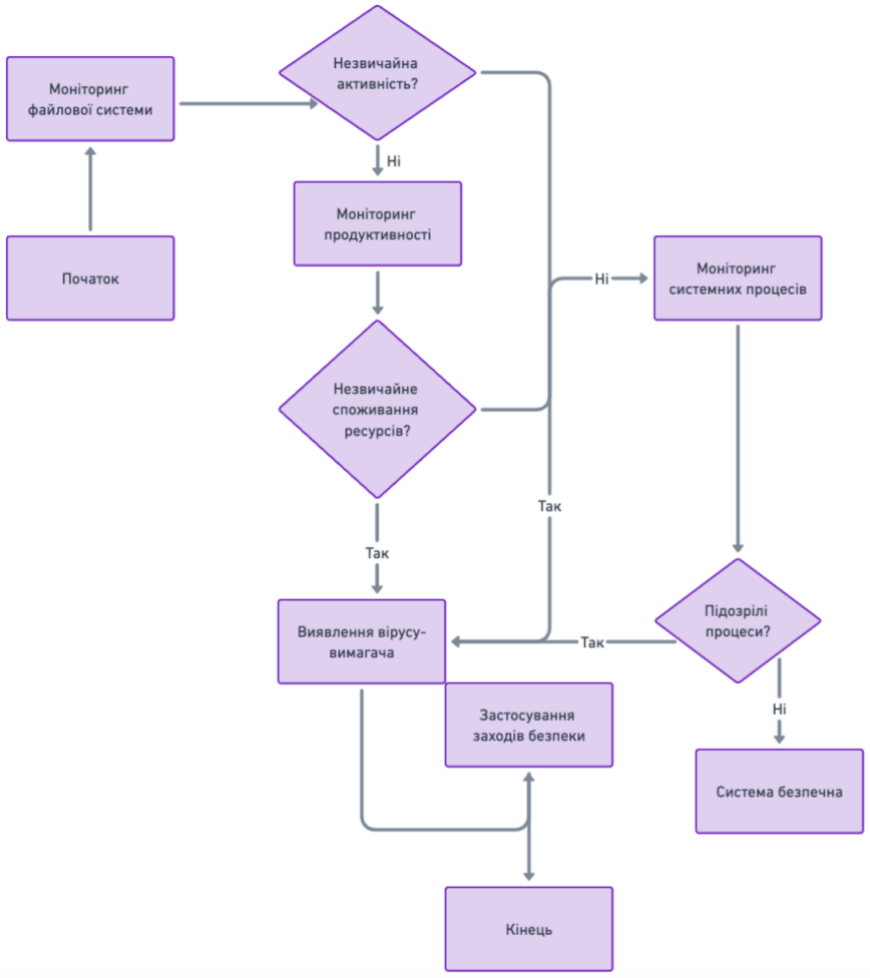


Рис. 3.4. Принцип роботи дерев рішень для виявлення вірусів вимагачів

Математична модель Випадкового лісу. Дерево рішень працює, рекурсивно ділячи простір даних на основі ознак. Метрики, такі як індекс Джині (G) або приріст інформації, використовуються для визначення кращого поділу:

1. Індекс Джині для поділу S і ознаки X обчислюється за формулою:

$$G\,(S,\,X)=\sum_{i=1}^{c}\,(p_i)^2, \tag{3.2.1}$$

де p_i- це частка екземплярів класу i в підмножині S.

2. Приріст інформації визначається як різниця між ентропією до поділу і зваженою ентропією після поділу:

$$\text{Gain}(S,X) = \text{Entropy}(S) - \Sigma_{v\in\text{Values}(X)}\left(\frac{S}{S_v}\times \text{Entropy}(Sv)\right), \tag{3.2.2}$$

де S_v – підмножина даних, де ознака X приймає значення v.

Випадковий ліс. Випадковий ліс створює ансамбль B дерев рішень, кожне з яких тренується на підмножині даних, отриманої методом "бутстрапінгу", і кожний поділ в дереві робиться на підмножині ознак. Для задачі класифікації прогноз випадкового лісу дорівнює моді прогнозів дерев:

$$RF(x) = \text{mode}\{DT_1(x), DT_2(x), \dots, DT_B(x)\}, \quad (3.2.3)$$

Для задачі регресії випадковий ліс повертає середнє значення прогнозів дерев:

$$RF(x) = \frac{B}{1}\sum_{b=1}^{B} DT_b(x), \quad (3.2.4)$$

Тут $DT_b(x)$- це прогноз b дерева на вхідних даних x.

Ці формули дають загальне уявлення про те, як випадковий ліс генерує свої прогнози. Окрім цього є багато додаткових параметрів і оптимізацій, які можна використовувати під час створення випадкового лісу. Розглянемо додаткові параметри та оптимізації, які зазвичай використовуються під час створення випадкового лісу:

1. n_estimators. Вказує кількість дерев у лісу. Збільшення цього параметра може покращити ефективність моделі, але це також збільшує час навчання та використання моделі.

2. max_features. Кількість ознак, що розглядаються при пошуку кращого поділу. Зменшення цього числа може зробити алгоритм більш стабільним за рахунок невеликого зниження ефективності.

3. max_depth. Максимальна глибина дерева. Обмеження глибини може допомогти уникнути перенавчання.

4. min_samples_split. Мінімальна кількість вибірок, необхідних для поділу внутрішнього вузла. Допомагає контролювати детальність дерева.

5. min_samples_leaf. Мінімальна кількість вибірок, необхідних для вузла-листка. Встановлення цього параметра може допомогти зменшити кількість дрібних гілок, що можуть викликати перенавчання.

6. bootstrap. Застосування бутстрапінгу для створення дерев. Якщо параметр встановлено в False, то для конструювання кожного дерева використовується повний датасет.

7. oob_score. Чи використовувати out-of-bag вибірки для оцінки ефективності. Out-of-bag вибірки - це вибірки, які не були використані під час побудови конкретного дерева.

8. class_weight. Ваги класів для задачі класифікації, якщо класи незбалансовані.

9. n_jobs. Кількість ядер для паралельного навчання. Зазвичай використовується для прискорення навчання на багатоядерних машинах.

Ці параметри можна оптимізувати за допомогою методів, таких як перехресна перевірка (cross-validation) та пошук сітки (grid search) для отримання найкращої можливої ефективності моделі на даних.

Тренування та оцінка. Під час тренування випадкового лісу важливо враховувати ряд параметрів, таких як глибина дерева, кількість дерев у лісі та максимальна кількість ознак для розгляду. Щоб оцінити якість моделі, можна використовувати крос-валідацію та різні метрики якості, такі як точність, повнота та F-міра.

Важливість ознак. Випадковий ліс дозволяє оцінити важливість кожної ознаки. Це визначається на основі того, наскільки часто ознака використовується для поділу та наскільки вона покращує якість поділу.

Отже, дерева рішень та випадкові ліси є потужними інструментами для рішення задач машинного навчання. Вони забезпечують інтуїтивне розуміння даних, можливість оцінки важливості ознак та гнучкість у моделюванні складних відносин між ознаками.

3.3.2 Метод опорних векторів, як інструмент для розділення “безпечних” та “небезпечних” програм на основі гіперплощини, що максимізує відстань між класами

У цьому розділі буде розглянуто, що таке метод опорних векторів (SVM), як він працює, а потім заглибимося в деталі застосування SVM для виявлення шкідливого програмного забезпечення. Алгоритм навчання SVM - це техніка керованого машинного навчання, яка використовується як для регресії, так і для класифікації. Регресійні моделі використовуються для прогнозування безперервних значень, а моделі класифікації - для визначення класу, до якого належить точка даних. SVM здебільшого використовуються для розв'язання задач класифікації [144].

Метод опорних векторів (Support Vector Machine, SVM) є найбільш привабливим методом машинного навчання. SVM є ефективним методом класифікації, який знаходить своє застосування у різних галузях. SVM може бути корисним у кібербезпеці, якщо застосувати його для підвищення точності системи виявлення вторгнень. Даний класифікатор допомагає у виявленні зловмисного мережевого трафіку [145]. Хоча бінарні SVM зазвичай мають вищу точність, створення ефективних однокласових SVM могло б знизити необхідність у трудомісткому процесі ручного маркування наборів даних. Для однокласових SVM потрібний лише навчальний набір даних, який містить звичайний трафік.

Простір ознак - це набір або сукупність ознак, які використовуються для категоризації даних. Гіперплощина - це пряма лінія, площина або будь-яка інша нелінійна межа, проведена в просторі ознак таким чином, що весь набір шкідливих програм залишається добре відокремленим від безпечних. Гіперплощини легко будувати, коли набори даних розріджені. У тих випадках, коли набори даних щільні, може бути важко відокремити зразки, що поводяться по-різному, один від одного. У таких випадках необхідно побудувати нелінійну гіперплощину. Функції

ядра в SVM використовуються для вирішення нелінійних задач шляхом перетворення їх у лінійну задачу, що здійснюється шляхом відображення низьковимірного вхідного простору в простір вищої розмірності [146].

Для виявлення шкідливого програмного забезпечення першим кроком є ефективне вилучення ознак. Навчальний набір даних готується шляхом завантаження зразків з будь-якого сховища даних, а потім із зразків виділяються відповідні ознаки. SVM, як метод машинного навчання під наглядом, вимагає, щоб дані були позначені як шкідливе або корисне програмне забезпечення, щоб класифікувати їх на етапі навчання. Ці ознаки разом з мітками використовуються для побудови вибірки даних у просторі ознак, щоб знайти оптимальну гіперплощину, яка ефективно відокремлює шкідливе ПЗ від безпечного. Вивчення гіперплощини є найважливішою фазою SVM. Техніка оптимізації використовується для того, щоб зменшити кількість ваг, відмінних від нуля, до декількох, які відповідають важливим характеристикам, що визначають, як має бути побудована гіперплощина. Лінійні алгебраїчні методи використовуються для вивчення гіперплощини. Ядра - це функції, які визначають схожість між двома вхідними даними. У тих випадках, коли вектори ознак дуже щільні і важко знайти лінійну гіперплощину, в SVM використовується метод ядер для відображення векторів ознак у простір більшої розмірності.

Ідея використання функцій ядра базується на тому, що багато алгоритмів, таких як SVM, можуть бути сформульовані таким чином, що єдиним способом їхньої взаємодії з даними є обчислення точкових добутків вектора ознак точок даних. Функції подібності пропонуються для усунення деяких обмежень функцій ядра. У цьому випадку ознаки подібності обчислюються за допомогою функції подібності та додаються. Функція подібності вимірює, наскільки кожен екземпляр схожий на певний орієнтир. Деякі застосування SVM в кібербезпеці - це системи виявлення вторгнень, класифікація кібератак і виявлення шкідливого програмного забезпечення. Для IDS можна мати однокласні, двокласні або багатокласні моделі

SVM [146]. Однокласна SVM вимагає лише навчального набору даних, що містить нормальний трафік, тоді як двокласна SVM вимагає навчального набору даних, що містить як нормальний, так і аномальний трафік. Багатокласові можна використовувати для класифікації шкідливого трафіку в один з класів, наприклад, атаки типу "відмова в обслуговуванні", розподілені атаки типу "відмова в обслуговуванні", атаки типу "зондування", атаки типу "шкідливе програмне забезпечення" і т.д [147].

Вибір ознак є важливим кроком, коли мова йде про виявлення шкідливого програмного забезпечення за допомогою машинного навчання. Деякі з його цілей - зменшити шум, підвищити точність і швидкість навчання класифікатора, а також мінімізувати набір даних для вибору найкращого навчального набору. Методи засновані на мінімізації меж узагальнення за допомогою градієнтного спуску і є реально обчислювальними. SVM можуть погано працювати в ситуації з великою кількістю нерелевантних ознак. Характерні необроблені дані вибірки включають інформацію про заголовки, можливі пакувальники та компресори, розмір різних секцій, рядків, ентропію, хеш файлу, розмір файлу в байтах тощо. Мета полягає в тому, щоб перетворити сирі характеристики в числові ознаки. Виділення числових характеристик з отриманих даних є важливою частиною процесу машинного навчання. Деякі інші ознаки - це розмір коду у файлі, кількість мов ресурсів, виявлених у файлі, кількість типів ресурсів, виявлених у файлі, десяткове значення точки входу, тобто місце в коді, де управління передається від хостової ОС до файлу, розмір ініціалізованих даних, довжина "оригінального імені файлу, розмір частини файлу, яка містить всі глобальні, неініціалізовані змінні або змінні, ініціалізовані в нуль, і так далі. Деякі з характеристик, які можуть бути витягнуті зі шкідливого програмного забезпечення, - це виклики API, дозволи, опкоди, системні виклики, використання процесора та оперативної пам'яті, використання пам'яті, використання мережі тощо. Для виконання вибірки та/або класифікації даних необхідно створити підмножини повного набору даних.

Класифікатор SVM навчається, визначаючи оптимальну гіперплощину для найкращого розділення даних. За допомогою функції ядра дані проектуються на простір ознак вищої розмірності. Основна мета полягає в знаходженні гіперплощини, яка класифікує навчальний набір даних на два класи: доброякісні та шкідливі, на основі міток, що вказуються. SVM, будучи керованою технікою машинного навчання, використовує дані, які заздалегідь маркуються на етапі навчання. Методи відбору ознак спрямовані на зменшення розмірності та підвищення компактності вектора ознак, що представляє файли.

Кероване навчання має різноманітні алгоритми, кожен з яких відрізняється унікальними перевагами. Особливе місце серед них займають алгоритми k-найближчих сусідів (kNN) та метод опорних векторів (SVM). Алгоритм kNN визначає категорії нових точок даних, виходячи з їх просторової близькості до вже позначених прикладів, присвоюючи класифікацію за домінуючим класом серед k найближчих сусідів. Цей метод є непараметричним і базується на вимірах відстаней між новою точкою та всіма позначеними прикладами навчання. SVM, у свою чергу, славиться своєю здатністю надійно класифікувати та проводити регресійний аналіз, визначаючи оптимальну гіперплощину для чіткого відділення класів даних із найбільшим можливим зазором. Він досягає цього шляхом перетворення даних у вимірний простір ознак і ретельної побудови границі рішення, яка збільшує різницю між окремими класами. Його гнучкість в обробці як лінійно, так і нелінійно розділених даних ще більше підвищується за допомогою різних функцій ядра [148].

У контексті цього дослідницького проекту основна увага була зосереджена на тестуванні продуктивності SVM з використанням як лінійного ядра, так і ядра з радіально-базисною функцією (RBF). Експерименти та дослідження в цьому проекті спрямовані на те, щоб пролити світло на різні аспекти можливостей SVM з цими ядрами, забезпечуючи всебічне розуміння його функціонування та ефективності.

Для виявлення програм-вимагачів за допомогою методу опорних векторів (SVM), можна використати як лінійні, так і нелінійні SVM залежно від того, наскільки складними є дані. SVM ефективний у класифікації, коли виокремлення класів вимагає знаходження оптимальної роздільної гіперплощини. Ось запропонований загальний опис математичної моделі SVM для двокласової класифікації програм-вимагачів:

1. Формулювання задачі

SVM шукає гіперплощину $W^t x + b = 0$, яка оптимально розділяє два класи вхідних даних, мінімізуючи класифікаційну помилку. Класи тут можуть бути визначені як y_1= 1 для вірусів-вимагачів і y_1= -1 для безпечних програм.

2. Формула максимізації відступу

Основна мета SVM — максимізувати відступ між найближчими точками даних обох класів (опорні вектори) і гіперплощиною. Ця формула відображає цільову функцію SVM, яка спрямована на мінімізацію квадрату норми вектора ваг w. Чим менше норма w, тим більший відступ (маржа) між опорними векторами і розділяльною гіперплощиною. Це забезпечує кращу узагальнення моделі на нових даних. Функція для мінімізації виглядає так:

$$min_{w,b}, 1/2\|w\|2 \qquad (3.2.5)$$

3. Введення змінних розслаблення

У випадках, коли лінійна роздільність неможлива, вводяться змінні розслаблення ξi для допуску помилок класифікації:

$min_{w,b,\xi} 1/2\|w\|2+C\sum i=1n\xi i,$ де C — параметр регуляризації, який контролює компроміс між збільшенням розміру відступу і мінімізацією помилок класифікації.

4. Ядровий трюк

Для вирішення не лінійно роздільних задач використовується ядровий трюк, який дозволяє SVM ефективно працювати в більш високовимірних просторах.

Ядрова функція, зокрема радіально-базисна функція (RBF), використовується для трансформації вхідних даних у вищий вимірний простір, де вони можуть бути

лінійно розділеними. Параметр γ контролює ширину "гаусівського" ядра, впливаючи на гладкість межі рішення. Ядрова функція $K(x_i, x_j)$ може бути вибрана, наприклад, як радіально-базисна функція (RBF):

$$K(xi, xj) = \exp(-\gamma \parallel xi - xj \parallel 2), \quad (3.2.6)$$

де γ — параметр, який потрібно налаштувати.

5. Розв'язання оптимізаційної задачі

Задача оптимізації зазвичай вирішується за допомогою методів квадратичного програмування, а рішення дає коефіцієнти w і b для гіперплощини та індикатори опорних векторів.

6. Класифікація нових даних

Ця формула визначає, як класифікатор видає рішення щодо класифікації нового вхідного вектора x. Знак лінійної комбінації ваг, вхідних даних і зміщення вказує на те, до якого класу належить вектор. Класифікація нових вхідних даних x здійснюється за допомогою знаку рішення функції:

$$f(x) = \text{sgn}(w^T x + b) \quad (3.2.7)$$

Цей код використовується для класифікації даних за допомогою методу підтримуючих векторів та проведення відповідного аналізу.

1. Імпорт модулів:

- `argparse` використовується для аналізу аргументів командного рядка.

- `numpy` та `pandas` для обробки даних.

- `sklearn` для моделювання та візуалізації.

- `matplotlib` для створення графіків.

- `joblib` для збереження та завантаження моделі.

2. Структура даних:

- Визначені шляхи до файлів з даними, мітками та моделями.

3. Функції:

- `get_labels()`: отримує мітки класів.

- `best_features_linear()` та `best_features_rbf()`: візуалізують важливість ознак для лінійного та RBF ядра SVM відповідно.

- `refit_strategy()`: визначає найкращі параметри для моделі на основі результатів крос-валідації.

- `train()`: навчає модель SVM на навчальних даних.

- `test()`: тестує модель на тестових даних та візуалізує результати за допомогою матриці плутанини та ROC-кривої.

4. Головний блок коду `main()`:

- Парсинг аргументів командного рядка для визначення режиму роботи (навчання або тестування) та шляху до файлу з мітками.

- Виконання функцій `train()` або `test()` в залежності від обраного режиму.

5. Умова `if __name__ == "__main__":`:

- Запускає головний блок коду, коли скрипт виконується як основна програма.

Загалом, цей код дозволяє користувачеві навчати модель класифікації на основі SVM для заданого набору даних, визначати важливість ознак та оцінювати її продуктивність на тестовому наборі даних.

Підготовка даних:

```
# scale the training data
scaler = StandardScaler().fit(X_train)
scaler.transform(X_train)
```

Лістинг 3.2

Пошук гіперпараментрів для SVM:

```
kernel = ['rbf'] # 'rbf' or 'linear'
class_weight = [{1: w} for w in np.linspace(10, 200, 10)]
C = np.logspace(-1, 0, 10)
param_grid = dict(class_weight=class_weight, kernel=kernel, C=C)
scores = ['recall', 'balanced_accuracy']
```

```
grid = GridSearchCV(svm.SVC(max_iter=1_000_000), param_grid=param_grid, scoring=scores, refit=refit_strategy)
grid.fit(X_train, y_train)
```

Лістинг 3.3

Оцінка важливості ознак:

```
if kernel[0] == 'linear':
    best_features_linear(best_classifier.coef_, feature_names)
elif kernel[0] == 'rbf':
    best_features_rbf(permutation_importance(best_classifier, X_train, y_train), feature_names)
```

Лістинг 3.4

Тестування моделі та візуалізація результатів:

```
# confusion matrix
cm_display = ConfusionMatrixDisplay.from_estimator(classifier, X_test, y_test)
roc_display = RocCurveDisplay.from_estimator(classifier, X_test, y_test)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
cm_display.plot(ax=ax1)
roc_display.plot(ax=ax2)
```

Лістинг 3.5

3.3.3 Розроблення моделі виявлення вірусів-вимагачів з використанням глибоких нейронних мереж

Нейронні мережі - це застосування моделей машинного навчання, зокрема нейронних мереж, для виявлення, запобігання та реагування на кіберзагрози та атаки [149].

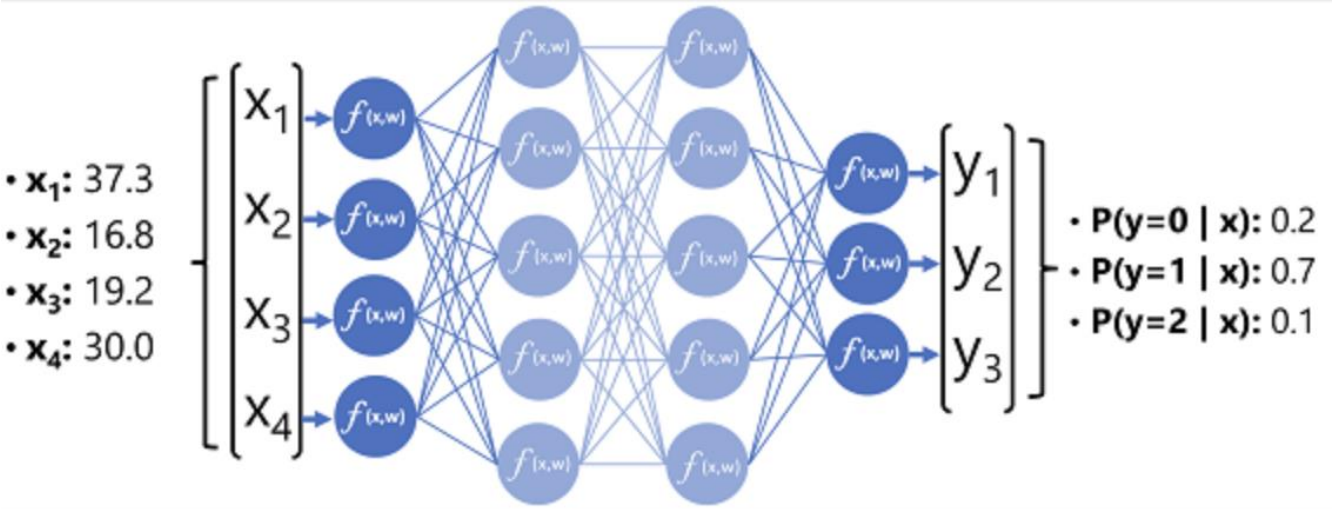


Рис 3.5. Метод роботи нейронних мереж

Ось декілька ключових аспектів використання нейронних мереж у кібербезпеці:

Виявлення аномалій: Нейронні мережі можуть аналізувати великі обсяги даних, щоб виявити аномальну поведінку або відхилення від звичайних шаблонів, що може вказувати на кібератаку або внутрішню загрозу [150].

Розпізнавання шкідливого ПЗ: Завдяки здатності навчатися на великих наборах даних, нейронні мережі можуть ефективно розпізнавати та класифікувати шкідливе програмне забезпечення, включаючи віруси, трояни, віруси-вимагачі та інші види шкідливих програм.

Прогнозування кібератак: Нейронні мережі можуть використовуватися для аналізу тенденцій та шаблонів у кібератаках, допомагаючи організаціям прогнозувати та попереджати майбутні атаки.

Адаптація до нових загроз: Оскільки кіберзагрози постійно еволюціонують, нейронні мережі можуть навчатися на нових даних, що дозволяє їм адаптуватися та виявляти нові види атак.

Аналіз мережевого трафіку: Нейронні мережі можуть аналізувати мережевий трафік у реальному часі, виявляючи підозрілу активність або втручання в мережу.

Біометрична безпека: Вони також використовуються для розробки більш надійних біометричних систем автентифікації, таких як розпізнавання обличчя або відбитків пальців.

У кібербезпеці нейронні мережі відіграють важливу роль у підвищенні ефективності та точності систем безпеки, дозволяючи більш швидко та точно реагувати на постійно змінні кіберзагрози.

Глибоке навчання — це підгалузь машинного навчання, яка використовує алгоритми, засновані на структурах, подібних до людського мозку, відомих як штучні нейронні мережі [151]. Ось основні характеристики глибокого навчання:

1. Імітація функціонування мозку: Глибоке навчання використовує структури, подібні до нейронних мереж людського мозку, для обробки даних та вирішення складних завдань.

2. Багатошаровість: Відмінною особливістю глибокого навчання є використання багатьох шарів обчислень. Кожен шар автоматично та ітеративно вчиться з даних, витягуючи все більш складні та абстрактні особливості.

3. Самонавчання з даних: Глибоке навчання здатне самостійно виявляти та вчитися важливим особливостям у великих обсягах даних, без необхідності їх ручного маркування або класифікації.

4. Застосування: Глибоке навчання застосовується у багатьох сферах, включаючи розпізнавання мови, обробку природної мови, розпізнавання зображень, автономні транспортні засоби, медичну діагностику та багато інших.

5. Великі обсяги даних та обчислювальна потужність: Ефективність глибокого навчання значно залежить від доступності великих наборів даних та значної обчислювальної потужності, особливо для тренування моделей.

6. Алгоритми: До найпопулярніших алгоритмів глибокого навчання належать конволюційні нейронні мережі для обробки зображень та рекурентні нейронні мережі для обробки послідовних даних.

Створення математичної моделі для виявлення та моніторингу вірусів-вимагачів у реальному часі може включати різні методи машинного навчання та обробки даних. Одним із ефективних підходів є використання глибоких нейронних мереж, які можуть виявляти шаблони поведінки, типові для вірусів-вимагачів. Нижче я наведу базову структуру такої моделі, використовуючи комбінацію конволюційних нейронних мереж (CNN) для аналізу файлів і рекурентних нейронних мереж (RNN), зокрема LSTM, для моніторингу послідовностей дій системи.

1. Вхідні дані

Модель приймає на вхід дані, які можуть включати:

- Бінарні файли - для статичного аналізу.

- Системні виклики або потоки даних мережі - для динамічного аналізу.

2. Передобробка даних

Перед подачею даних до моделі, їх потрібно правильно обробити:

- Нормалізація: Скейлинг вхідних даних для підвищення ефективності навчання.

- Векторизація: Конвертація даних у формат, придатний для нейронних мереж.

3. Конволюційна нейронна мережа (для аналізу файлів)

$$Z^{(l+1)}=f(W^{(l)}*Z^{(l)}+b^{(l)}, \quad (3.2.8)$$

де $Z^{(l)}$ — вихід попереднього шару,

$W^{(l)}$ і $b^{(l)}$— ваги та зміщення l-го шару,

f — активаційна функція

4. Рекурентна нейронна мережа (LSTM) (для аналізу системних викликів)

$$f_t = \sigma(W_f \cdot [h_t - 1, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [h_t - 1, x_t] + b_i)$$

$$C_t = \tanh(W_C \cdot [h_t - 1, x_t] + b_C)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot C_t$$

$$o_t = \sigma(W_o \cdot [h_t - 1, x_t] + b_o)$$

$$h_t = o_t \cdot \tanh(C_t), \quad (3.2.9)$$

де x_t— вхідні дані у час t, h_t — вихідний сигнал, C_t— внутрішній стан клітини.

5. Функція втрати

Для тренування моделі використовується функція втрати, яка мінімізує розбіжності між передбаченнями моделі та реальними мітками:

$$L = -\sum(y\log(y\hat{}) + (1 - y)\log(1 - y\hat{})), \quad (3.2.10)$$

де y — істинні мітки, $y\hat{}$— передбачені мітки.

6. Вихід

Модель повертає ймовірності того, що дані вказують на вірус-вимагач, які можуть бути використані для сповіщення користувачів або системного адміністратора.

Ця модель потребує глибокого налаштування параметрів та великої кількості тренувальних даних для ефективного виявлення вірусів-вимагачів, а також постійних оновлень, щоб відповідати новим видам загроз.

Глибоке навчання відкрило нові можливості у сфері штучного інтелекту, дозволяючи створювати системи, які можуть автоматично та ефективно вирішувати завдання, які раніше вважалися надто складними для комп'ютерів.

3.4 Метрики оцінки машинного навчання

У цьому розділі описуються загальні метрики оцінювання, що використовуються в моделях машинного навчання. Припускається, що класифікація є бінарною на позитивний клас і негативний клас.

1. **Оцінка однорідності (Homogeneity Score)**

Кластер вважається гомогенним, коли він складається виключно з даних точок, що належать до одного і того ж класу. Нехай C буде множиною класів, а K - класифікаціями класів, тоді оцінка однорідності h розраховується як:

$$h = 1 - \frac{H(C|K)}{H(C)}, \tag{3.3.1}$$

де $H(C|K)$ - це ентропія класів C при заданому K, а $H(C)$ - ентропія класів C.

2. **Оцінка повноти (Completeness Score)**

Результат кластеризації вважається повним, якщо всі точки даних, які є членами класу, належать до одного кластера. Оцінка повноти c розраховується як:

$$\mathrm{c} = 1 - \frac{H(K|C)}{H(K)}, \tag{3.3.2}$$

де $H(K|C)$ - це ентропія кластерів K при заданому C, а $H(K)$ - ентропія кластерів K.

3. **Оцінка V-Measure Score**

Вона отримується як гармонійне середнє оцінок однорідності та повноти. Отже, оцінка V-міри v задається формулою:

$$v = 2 \times \frac{h \times c}{h + c} \tag{3.3.3}$$

4. **Коефіцієнт Каппа Коена (Cohen's Kappa Score)**

У багатокласових задачах класифікації такі міри, як точність або прецизійність/відтворення, не підходять для оцінки ефективності класифікатора. У незбалансованих наборах даних такі міри, як точність, можуть вводити в оману, тому використовуються міри, такі як прецизійність і відтворення або F-міра. Коефіцієнт Каппа Коена - ідеальна міра, яка може впоратися з задачами як багатокласової, так і незбалансованої класифікації. Він розраховується за допомогою формули:

$$k = \frac{p_0 - p_e}{1 - p_e}, \tag{3.3.4}$$

де p_0 - це ймовірність спостережуваної згоди, а p_e- ймовірність очікуваної згоди.

Він розраховує, наскільки краще класифікатор працює порівняно з класифікатором, який робить випадкові передбачення.

5. Матриця Помилок (Confusion Matrix)

Матриця помилок — це матриця 2x2, що використовується для узагальнення продуктивності алгоритму класифікації. У цій матриці кожен рядок представляє інстанцію фактичного класу, а стовпець — інстанцію передбаченого класу. Таблиця відображає кількість істинно позитивних (TP), істинно негативних (TN), хибно позитивних (FP) та хибно негативних (FN) результатів. Істинно позитивний результат відбувається, коли елемент позитивного класу правильно передбачений моделлю як належний до позитивного класу. Істинно негативний — коли елемент негативного класу правильно передбачений як належний до негативного класу.

Хибно позитивний — коли елемент негативного класу невірно передбачений як належний до позитивного класу. Хибно негативний — коли елемент позитивного класу невірно передбачений як належний до негативного класу. Зазвичай TP, TN, FP та FN використовуються для позначення кількості істинно позитивних, істинно негативних, хибно позитивних та хибно негативних результатів відповідно. Щоб визначити матрицю помилок, робляться прогнози для кожного тестового випадку, і на основі передбачених та фактичних міток визначається кількість істинних та хибних прогнозів, зроблених для позитивних та негативних класів. Матриця помилок наведена в Таблиці 3.3.

Таблиця 3.3

Матриці Помилок

Всього	**Передбачено НІ**	**Передбачено ТАК**	**Сума**
Фактично НІ	TN	FP	Сума фактично НІ
Фактично ТАК	FN	TP	Сума фактично ТАК

	Сума передбачено НІ	Сума передбачено ТАК	

6. **Порогова Крива (Threshold Curve)**

Порогова крива — це графік, що показує компроміс у прогнозуванні, який досягається шляхом зміни порогового значення між класами.

7. **Точність (Accuracy)**

Точність — це міра частоти, з якою класифікатор робить правильні прогнози. Вона розраховується як:

$$\frac{TP+TN}{TP+TN+FP+FN} \tag{3.3.5}$$

8. **Частота Неправильної Класифікації (Misclassification Rate)**

Частота неправильної класифікації є протилежністю точності. Це міра частоти, з якою класифікатор робить помилкові прогнози. Вона розраховується як:

$$\frac{FP+FN}{TP+TN+FP+FN} \tag{3.3.6}$$

9. **Чутливість (Sensitivity)**

Чутливість вимірює здатність класифікатора правильно передбачати позитивний клас. Це міра частоти, з якою класифікатор прогнозує мітку позитивного класу для даних, коли дані насправді належать до позитивного класу. Чутливість також відома як частота відтворення або істинно позитивна частота (TPR). Вона розраховується як:

$$\frac{TP}{TP+FN} \tag{3.3.7}$$

10. **Специфічність (Specificity)**

Специфічність вимірює здатність класифікатора правильно передбачати негативний клас. Це міра частоти, з якою класифікатор прогнозує мітку негативного класу для даних, коли дані насправді належать до негативного класу. Специфічність також відома як вибірковість або істинна негативна частота (TNR). Вона розраховується як:

$$\frac{TN}{TN+FP} \tag{3.3.8}$$

11. Прецизійність або прогностична значущість позитивного результату (Precision)

Прецизійність або прогностично значущий результат positive predictive values (PPV) вимірює частоту, з якою класифікатор є правильним, коли він прогнозує мітку позитивного класу для точки даних. Вона розраховується як:

$$\frac{TP}{TP+FP} \tag{3.3.9}$$

12. Негативна прогностична значущість результату (Negative Predictive Value)

Негативна прогностична значущість negative predictive value (NPV) вимірює частоту, з якою класифікатор є правильним, коли він прогнозує мітку негативного класу для точки даних. Вона розраховується як:

$$\frac{TN}{TN+FN} \tag{3.3.10}$$

13. Поширеність (Prevalence)

Поширеність вимірює пропорцію населення в позитивному класі. Вона розраховується як:

$$\frac{TP+FN}{TP+TN+FP+FN} \tag{3.3.11}$$

14. Частота Нульових Помилок (Null Error Rate)

Це міра частоти, з якою класифікатор може помилятися під час прогнозування більшості класів. Іноді найкращий класифікатор для певної програми може мати вищу частоту помилок, ніж частота нульових помилок.

15. Коефіцієнт Каппа Коена (Cohen's Kappa)

Каппа Коена порівнює, наскільки правильно класифікатор працював з очікуваною точністю. Каппа буде високою, якщо є велика різниця між точністю та очікуваною частотою помилок. Вона розраховується як:

$$k = \frac{(observed\ accuracy - expected\ accuracy)}{(1 - expected\ accuracy)} \tag{3.3.12}$$

16. Оцінка F1 (F1 Score)

Це гармонійне середнє прецизійності та чутливості, розраховується як:

$$F_1 = 2 \times \frac{Precision \times Sensativity}{Precision + Sensativity} \quad (3.3.13)$$

17. Робоча характеристика приймача: ROC-крива (ROC Curve)

ROC-криві - це двовимірні графіки, де частота помилкових спрацьовувань (вісь x) наноситься проти частоти істинних спрацьовувань (вісь y). Графік підсумовує продуктивність класифікатора за всіма можливими пороговими значеннями. Також відома як крива похибок.

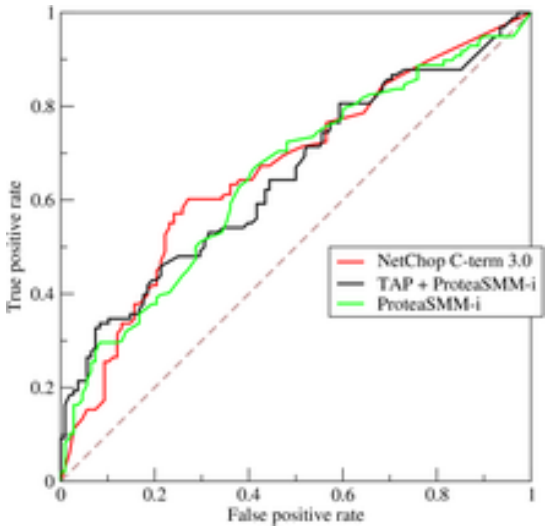


Рис 3.6. ROC-криві трьох методів передбачення похибки

18. Коефіцієнт Кореляції Метьюса (MCC)

MCC - це міра якості бінарної класифікації, яка може бути розрахована безпосередньо з матриці помилок як:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad (3.3.14)$$

Коефіцієнт кореляції Метьюса (MCC) вимірює кореляцію між спостережуваними та передбачуваними класифікаціями. Він приймає значення від -1 до +1, де +1 вказує на ідеальне передбачення, 0 представляє випадкове передбачення, і -1 представляє повну розбіжність між передбаченням та

спостереженням. Коефіцієнт кореляції Метьюса загалом вважається однією з найкращих мірок для опису матриці помилок.

Висновки до 3 розділу

Третій розділ дисертації зосереджується на середовищі для аналізу даних та застосуванні моделей машинного навчання, зокрема у контексті ідентифікації та класифікації кіберзагроз, таких як вимагачі. Розділ висвітлює обмеження традиційних підходів у кібербезпеці проти вірусів вимагачів, пропонуючи машинне навчання як більш адаптивну та ефективну альтернативу. Ключові моменти включають:

1. Розробка методології: У розділі було успішно розроблено та оцінено методологію застосування моделей машинного навчання для аналізу даних з eBPF. Ця методологія дозволила ідентифікувати та класифікувати поведінку програм-вимагачів з високою точністю, демонструючи значне покращення у виявленні кіберзагроз порівняно з традиційними методами кібербезпеки.
2. Розробка архітектури інтегрованої системи: В третьому розділі було розроблено архітектуру інтегрованої системи для виявлення програм-вимагачів, яка базується на використанні модулів eBPF і аналізу даних машинним навчанням. Система складається з модулів збору даних, передпроцесингу, оцінювання та реагування, що забезпечує цілісний підхід до виявлення і протидії кіберзагрозам у реальному часі.
3. Вперше розроблено та оцінено методику вибору та застосування моделей машинного навчання для аналізу даних з eBPF, що дозволяє ідентифікувати та класифікувати поведінку програм-вимагачів. Детальний підхід до підготовки набору даних, вибору відповідних моделей та їх оптимізації згідно з

конкретними типами даних та завданнями забезпечив суттєве поліпшення у виявленні кіберзагроз.

4. Модель дерев рішень і випадкових лісів: Моделі дерев рішень та випадкових лісів були застосовані для класифікації програм-вимагачів. Ці моделі ефективно використовували структурні та поведінкові ознаки, виявлені за допомогою eBPF, для точного прогнозування потенційних загроз. Вони показали високу точність і надійність у виявленні шкідливих дій на основі аналізу великих датасетів.

5. Метод опорних векторів (SVM): CVM було використано для розділення безпечних та потенційно небезпечних додатків на основі визначених гіперплощин. Метод демонстрував високу здатність до класифікації з мінімальним впливом шуму у даних, забезпечуючи стійкість системи до помилкових позитивних сигналів.

6. Глибокі нейронні мережі: Застосування глибоких нейронних мереж дозволило складно аналізувати нелинійні залежності між ознаками даних, забезпечуючи вдосконалення загальної точності виявлення. Глибокі мережі особливо ефективні в задачах розпізнавання складних патернів поведінки програм-вимагачів, що робить їх ідеальними для виявлення сучасних кіберзагроз.

7. Оптимізація збору даних: Значні поліпшення були досягнуті у процесах збору та обробки даних, що сприяло ефективному використанню модулів eBPF та підготовці даних у форматі, необхідному для аналізу машинним навчанням. Важливість цього процесу не може бути недооцінена, оскільки він забезпечує точність та відтворюваність результатів моделей.

8. Аналіз ефективності моделей: Виявлено високу продуктивність основних моделей машинного навчання, включно з деревами рішень, випадковими лісами, методом опорних векторів та нейронними мережами. Ці моделі продемонстрували здатність ефективно ідентифікувати складні патерни поведінки, характерні для програм-вимагачів, що є ключовим для забезпечення кібербезпеки в сучасних інформаційних системах.

РОЗДІЛ 4 АНАЛІЗ ЕФЕКТИВНОСТІ ЗАПРОПОНОВАНИХ РІШЕНЬ

4.1 Організація тестового середовища для проведення атак вірусів-вимагачів з метою оцінки ефективності рішень

Головною метою цього експериментального середовища є створення відокремленого простору, надійно захищеного від розповсюдження шкідливого програмного забезпечення або несанкціонованої передачі даних за допомогою моделі безпеки "нульової довіри" (Zero Trust) [152]. Стратегічна схема, що використовується для цього дослідницького проекту, базується на двошаровому, ізольованому віртуальному середовищі, як показано на Рисунку 4.1.

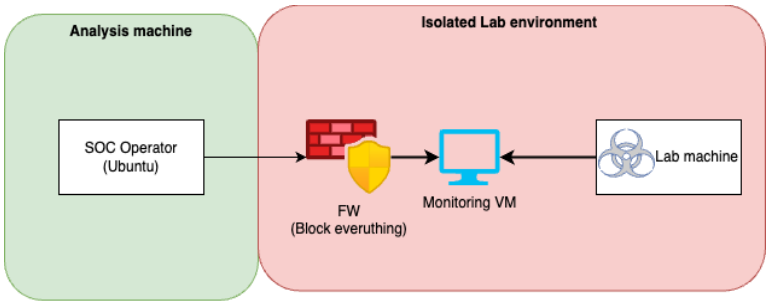


Рис. 4.1. Огляд первинної архітектури рішення

Застосування гіпервізора KVM дозволяє контролювати весь процес віртуалізації та забезпечує легку інтеграцію з libvirt API для зручного керування віртуальними машинами на сервері [153].

Оператор безпеки SOC налаштовує мережеві з'єднання за допомогою віртуальної мережі, зазвичай використовуючи віртуальний мережевий комутатор. Важливу роль у цьому процесі відіграють два основні режими роботи цього комутатора:

1. Режим NAT: у цьому базовому режимі встановлюється безпосереднє з'єднання між гостьовими системами та основним хостом. Вихід у зовнішню мережу здійснюється через трансляцію адрес, з обмеженнями, накладеними брандмауером хоста. Хоча цей режим дозволяє широкі можливості для

підключення, його застосування відбувається на стадії попереднього налаштування з огляду на безпеку, включаючи інсталяцію програмного забезпечення та завантаження тестових вірусів-вимагачів.

2. Ізольований режим: у цьому захищеному режимі гостьові віртуальні машини мають можливість комунікувати виключно в межах самої системи віртуалізації, без доступу до зовнішніх мереж. Це обмежує всі зовнішні взаємодії та використовується для експериментів з вірусами, забезпечуючи високий рівень безпеки та виключення ризику неконтрольованого розповсюдження.

Багаторівнева віртуалізація покращує дослідницькі можливості завдяки функціональності створення знімків стану віртуальної машини. Цей інструмент дозволяє фіксувати точні стани основного шару віртуалізації на різних етапах дослідження, забезпечуючи чисту базу для кожного нового етапу тестування.

В цьому захищеному середовищі перший рівень віртуалізації виконується за допомогою віртуальної машини Ubuntu 22.04, званої Sandbox Host VM. Ця машина обладнана вдосконаленою системою безпеки та строгим брандмауером, забезпечує доступ тільки через SSH та VNC з захищеної мережі [154, 156, 157]. Другий рівень віртуалізації, створений у цій VM, слугує ізольованим простором для дослідження вірусів-вимагачів.

Система на основі віртуалізації служить як надійний засіб для безпечного та ефективного проведення досліджень з вірусами-вимагачами, забезпечуючи строгу ізоляцію і запобігаючи випадковому розповсюдженню шкідливих програм і витоку інформації.

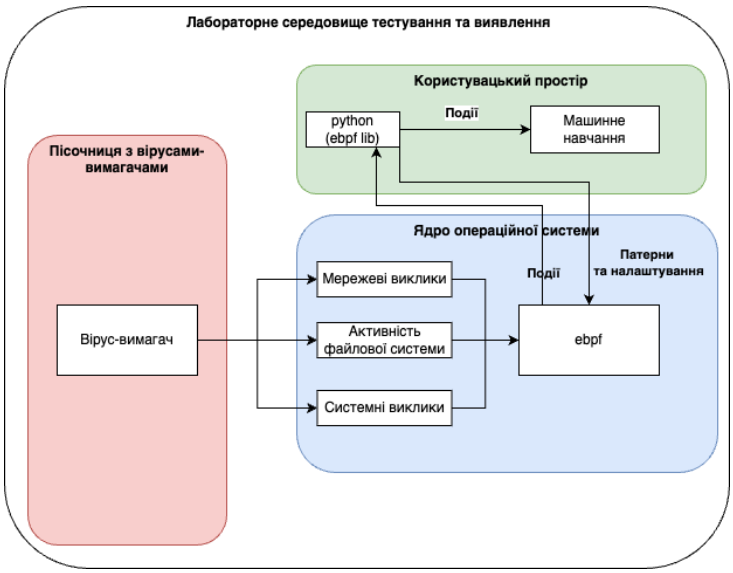


Рис. 4.2. Архітектура системи ізольованого середовище тестування та виявлення

Ця діаграма ілюструє лабораторне середовище тестування та виявлення, в якому організовано співпрацю різних компонентів системи для виявлення вірусних загроз.

- **Пісочниця з вірусами-вимагачами**: Це ізольоване середовище, де виконується потенційно шкідливий код або програми. Вона служить для безпечного виконання та аналізу вірусних загроз.
- **Ядро операційної системи**: Містить різні "виклики" або інтерфейси для моніторингу активності, таких як мережеві виклики, активність файлової системи та системні виклики.
- **Користувацький простір**: Тут відбувається обробка даних та взаємодія з компонентами ядра. Модуль Python (з підтримкою eBPF) взаємодіє з ядром, отримуючи дані та відправляючи їх до систем машинного навчання для аналізу.

Необхідність такої архітектури

Така архітектура дозволяє ізолювати потенційно шкідливий код в пісочниці, забезпечуючи безпеку основної системи. Одночасно, моніторинг в режимі реального часу активності допомагає виявляти несанкціоновані дії або втручання.

Переваги:

1. **Безпека**: Ізоляція вірусів забезпечує захист від потенційних загроз.
2. **Гнучкість**: Система може легко адаптуватися до нових типів загроз завдяки модульності.
3. **Моніторинг у режимі реального часу**: Швидке виявлення та реагування на загрози.

Недоліки:

1. **Складність**: Така система може вимагати значних ресурсів для налаштування та підтримки.
2. **Потреба в постійному оновленні**: Щоб залишатися актуальною, система має регулярно оновлюватися.

Загалом, ця архітектура представляє собою ефективний інструмент для виявлення та аналізу вірусних загроз у контрольованому середовищі.

4.2 Проведення тестових атак і збір даних для аналізу

У цьому розділі розглядається процес імітації атак з використанням вірусів-вимагачів для вивчення реакції і стійкості системи до таких загроз. Важливістю цього дослідження є необхідність перевірки ефективності реакції системи на різні види атак та визначення слабких місць, які можуть потребувати подальшого посилення.

Для досягнення цієї мети було визначено два ключові напрямки дослідження:

1. Імітація різних видів вірусів-вимагачів для вивчення їх поведінки у визначеному середовищі та можливості системи їх виявлення.

2. Оцінка реакції системи на імітаційні атаки, щоб визначити швидкість, точність та ефективність її відгуку.

Для здійснення цих досліджень використовувались спеціально підготовлені інструменти, пісочниці та інші засоби для імітації атак та збору даних. Отримані результати будуть використовуватися для аналізу дій системи, визначення її потенційних слабких місць та розробки рекомендацій щодо її поліпшення.

Далі у підпунктах 4.2.1 та 4.2.2 буде детальніше розглянуто процеси імітації атак та оцінки реакції системи на них.

4.2.1 Безпечна імітація різних видів вірусів-вимагачів

Під час вивчення вірусів-вимагачів важливо мати можливість імітувати їхню діяльність в контрольованому середовищі [158, 159]. Це дає змогу дослідникам безпеки вивчати поведінку вірусів, їхній вплив на систему та розробляти методики протидії.

Потреба написання власного симулятора:

1. **Специфічні вимоги**: Готові симулятори можуть не враховувати всі специфічні вимоги або особливості дослідження. Власний симулятор дозволяє налаштовувати параметри та сценарії атаки відповідно до потреб дослідження.

2. **Гнучкість**: При внесенні змін або додаванні нових функцій, власний симулятор забезпечує швидке та гнучке налаштування.

3. **Безпека**: З власним симулятором можна бути впевненим у безпеці коду та його виконанні без зайвих ризиків.

4. **Глибокий аналіз**: Власний інструмент дозволяє проводити докладний аналіз відповідей системи на атаки, збираючи детальну інформацію.

Приклад коду симулятора:

Для наочності розглянемо частину коду, яка відповідає за шифрування та розшифровування файлів:

```
if args.mode == "encrypt":
    if args.dir:
        directory = args.dir
    else:
        directory = CreateTempData()
    print(f'Шифруємо директорію {directory}...')
    count = EncryptDir(directory, args.password)
    print(f'{count} файлів зашифровано в {directory}')

elif args.mode == "decrypt":
    if args.dir:
        directory = args.dir
    else:
        directory = config.get('data', 'temp_dir')
    print(f'Розшифровуємо директорію {directory}...')
    count = DecryptDir(directory, args.password)
    print(f'{count} файлів розшифровано в {directory}')
```

Лістинг 4.1

Код вище демонструє процес шифрування та розшифровування файлів на основі вхідних параметрів. При виборі режиму "encrypt", вказана директорія або тимчасові дані будуть зашифровані використовуючи заданий пароль. В режимі "decrypt" вказана директорія або остання використана директорія буде розшифрована.

Маючи такий інструмент, дослідники можуть безпечно імітувати діяльність вірусів-вимагачів, не вносячи реальної шкоди системам.

В розробці безпеки комп'ютерних систем завжди важливо мати можливість моделювати потенційні загрози. Особливу актуальність це набуває у контексті вірусів-вимагачів, які швидко адаптуються до нових методів захисту. Саме тому було вирішено розробити власний симулятор для імітації атак вірусу-вимагача, що надає можливість краще розуміти механізми роботи таких загроз і ефективно розробляти стратегії захисту.

В якості прикладу розглянемо складову нашого симулятора, що відповідає за шифрування та розшифровування файлів:

```
import os
import time
import pyAesCrypt

def EncryptFile(file, password):
    pyAesCrypt.encryptFile(file, file+".aes", password)
    os.remove(file)

def DecryptFile(file, password):
    try:
        pyAesCrypt.decryptFile(file, file.split(".aes")[0], password)
        os.remove(file)
    except ValueError:
        print(f'Не вдалося розшифрувати файл {file}!')

def EncryptDir(directory, password) -> int:
    count = 0
    for dirpath, _dirnames, filenames in os.walk(directory, topdown=False):
        for name in filenames:
            EncryptFile(os.path.join(dirpath, name), password)
```

```
            count += 1
        time.sleep(0.01) # засипаємо на 10 мілісекунд
    return count


def DecryptDir(directory, password) -> int:
    count = 0
    for dirpath, _dirnames, filenames in os.walk(directory, topdown=False):
        for name in filenames:
            DecryptFile(os.path.join(dirpath, name), password)
            count += 1
    return count
```

Лістинг 4.2

Цей код демонструє, яким чином відбувається процес шифрування та розшифровування файлів у вказаній директорії. Функція `EncryptFile` шифрує окремий файл, використовуючи заданий пароль, а функція `DecryptFile` намагається розшифрувати файл за допомогою вказаного паролю.

Таким чином, використання симулятора надає можливість тестувати реакцію систем на потенційні атаки і вдосконалювати методи захисту від вірусів-вимагачів.

У сучасному кіберпросторі з'являється все більше загроз, серед яких віруси-вимагачі в останні роки стали особливо поширеними. Ці віруси зазвичай заражають систему, шифрують дані користувачів і потім вимагають викуп за їхнє відновлення.

Для розуміння й аналізу такого типу загроз розглядається симулятор вірусу-вимагача. Цей симулятор дозволяє безпечно досліджувати й вивчати механізми роботи вірусів-вимагачів, не завдаючи шкоди реальним даним.

Симулятор, що розглядається у цьому розділі, базується на Python і використовує бібліотеку `pyAesCrypt`[160] для шифрування файлів. Перш ніж переходити до практичного застосування, давайте докладно розглянемо, як користуватися цим симулятором.

Процес роботи з симулятором вірусів-вимагачів:

Встановлення:

Щоб встановити всі необхідні залежності для симулятора, необхідно виконати наступні команди:

```
python3 -m pip install -r requirements.txt
```

Лістинг 4.3

Вміст у файлі requirements.txt:

```
pyAesCrypt~=6.0.0
```

Лістинг 4.4

Шифрування файлів:

Щоб зашифрувати тимчасові файли в у тестовому середовищі :

```
./simulator.py --password xu18d7wfe2edt
```

Лістинг 4.5

Щоб зашифрувати файли у вказаній директорії:

```
./simulator --dir /path/to/dir --password xu18d7wfe2edt
```

Лістинг 4.6

Розшифровування файлів:

- Щоб розшифрувати тимчасові файли в `/tmp/tmpxxx`:

```
./simulator.py --mode decrypt --password xu18d7wfe2edt
```

Лістинг 4.7

- Щоб розшифрувати файли у вказаній директорії:

```
./simulator.py --mode decrypt --dir /path/to/dir --password xu18d7wfe2edt
```

Лістинг 4.8

Відстеження виконання коду Python:

Щоб відслідковувати виконання програми на Python, виконайте команду:

```
./simulator.py --password b12hn736bxe & sudo uflow -l python $!
```

Лістинг 4.9

Перегляд зашифрованих файлів:

Щоб переглянути вміст зашифрованого файла, використовуйте утиліту `xxd`. Наприклад, для файла `test.aes`:

```
xxd test.aes
```

Лістинг 4.10

Ось вихід даної команди - це виведе шістнадцятковий дамп файлу, де можна побачити структуру та вміст зашифрованого файлу:

```
00000000: 4242 5302 1212 2a44 454d 4f5f 4441 5441  BBS....*DEMO_DATA
00000010: 5431 2342 454d 4f43 6f64 6520 7a7a 2020  T1#BEMOCode zz
00000020: 3030 3132 3334 3536 3738 393a 3b3c 3d3e  00123456789:;<=>?
00000030: 3031 3233 3435 3637 3839 3a3b 3c3d 3e3f  0123456789:;<=>?
00000040: 4142 4344 4546 4748 494a 4b4c 4d4e 4f50  ABCDEFGHIJKLMNOP
00000050: 5152 5354 5556 5758 595a 5b5c 5d5e 5f60  QRSTUVWXYZ[\]^_`
00000060: 6162 6364 6566 6768 696a 6b6c 6d6e 6f70  abcdefghijklmnop
00000070: 7172 7374 7576 7778 797a 7b7c 7d7e 7f80  qrstuvwxyz{|}~..
00000080: 8182 8384 8586 8788 898a 8b8c 8d8e 8f90  ................
00000090: 9192 9394 9596 9798 999a 9b9c 9d9e 9fa0  ................
000000a0: a0a1 a2a3 a4a5 a6a7 a8a9 aaab acad aeaf  ................
000000b0: b0b1 b2b3 b4b5 b6b7 b8b9 babb bcbd bebf  ................
000000c0: c0c1 c2c3 c4c5 c6c7 c8c9 cacb cccd cecf  ................
000000d0: d0d1 d2d3 d4d5 d6d7 d8d9 dadb dcdc dddf  ................
000000e0: e0e1 e2e3 e4e5 e6e7 e8e9 eaeb eced eeee  ................
000000f0: eff0 f1f2 f3f4 f5f6 f7f8 f9fa fbfc fdfe  ................
00000100: ffff 0001 0203 0405 0607 0809 0a0b 0c0d  ................
00000110: 0e0f 1011 1213 1415 1617 1819 1a1b 1c1d  ................
00000120: 1e1f 2021 2223 2425 2627 2829 2a2b 2c2d  .. !"#$%&'()*+,-
00000130: 2e2f 3031 3233 3435 3637 3839 3a3b 3c3d  ./0123456789:;<=
00000140: 3e3f 4041 4243 4445 4647 4849 4a4b 4c4d  >?@ABCDEFGHIJKLM
00000150: 4e4f 5051 5253 5455 5657 5859 5a5b 5c5d  NOPQRSTUVWXYZ[\]
```

```
00000160: 5e5f 6061 6263 6465 6667 6869 6a6b 6c6d  ^_`abcdefghijklm
00000170: 6e6f 7071 7273 7475 7677 7879 7a7b 7c7d  nopqrstuvwxyz{|}
00000180: 7e7f 8081 8283 8485 8687 8889 8a8b 8c8d  ~...............
00000190: 8e8f 9091 9293 9495 9697 9899 9a9b 9c9d  ................
000001a0: 9e9f a0a1 a2a3 a4a5 a6a7 a8a9 aaab acad  ................
000001b0: aeaf b0b1 b2b3 b4b5 b6b7 b8b9 babb bcbd  ................
000001c0: bebf c0c1 c2c3 c4c5 c6c7 c8c9 cacb cccd  ................
000001d0: cecf d0d1 d2d3 d4d5 d6d7 d8d9 dadb dcdc  ................
000001e0: dddf e0e1 e2e3 e4e5 e6e7 e8e9 eaeb eced  ................
000001f0: eeee eff0 f1f2 f3f4 f5f6 f7f8 f9fa fbfc  ................
```

Лістинг 4.11

Використовуючи вказані команди та інструкції, можна ефективно симулювати роботу вірусів-вимагачів на своїх системах, що є корисним для вивчення та розробки стратегій захисту.

Симуляція вірусів-вимагачів є потужним інструментом для навчання та дослідження. Використовуючи симулятор, можна глибше зрозуміти механізми роботи цих загроз, а також розробляти та тестувати стратегії захисту. Однак важливо пам'ятати, що навіть в контексті симуляції слід дотримуватися обережності й не застосовувати такий інструмент на реальних даних без попередньої підготовки.

4.3 Аналіз результатів дії моделей машинного навчання при різних експериментах

Оцінка інтегрованої системи виявлення програм-вимагачів, яка включає технології eBPF, машинного навчання (ML) та обробки природної мови (NLP) [155], показала наступні результати:

Показники ефективності

Система була оцінена за низкою показників ефективності:

- **Точність Accuracy**: Система досягла загальної точності 94,7% у правильній ідентифікації дій зловмисників.
- **Точність Precision**: Точність, яка вимірює здатність системи мінімізувати кількість хибних спрацьовувань, була зафіксована на рівні 92,3%.
- **Відтворення (Recall):** система продемонструвала показник відтворення на рівні 93,5%, що свідчить про її ефективність у мінімізації помилкових спрацьовувань.
- **Оцінка F1**: Показник F1, який балансує між точністю та пригадуванням, був розрахований на рівні 92,9%.

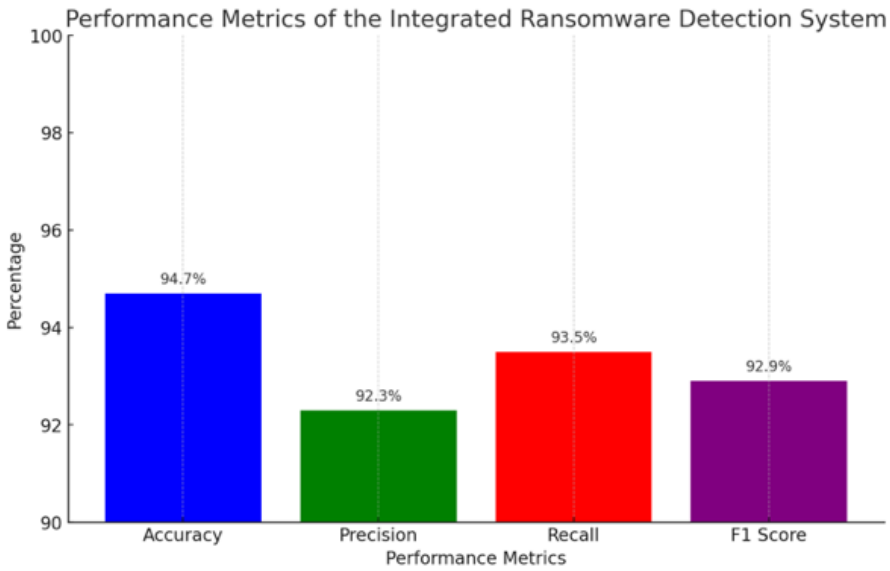


Рис. 4.3. Показники ефективності інтегрованої системи виявлення програм-вимагачів

- **Виявлення в режимі реального часу:** Система успішно виявила активність програм-вимагачів у режимі реального часу, середній час виявлення склав 2,3 секунди від початкової активності.

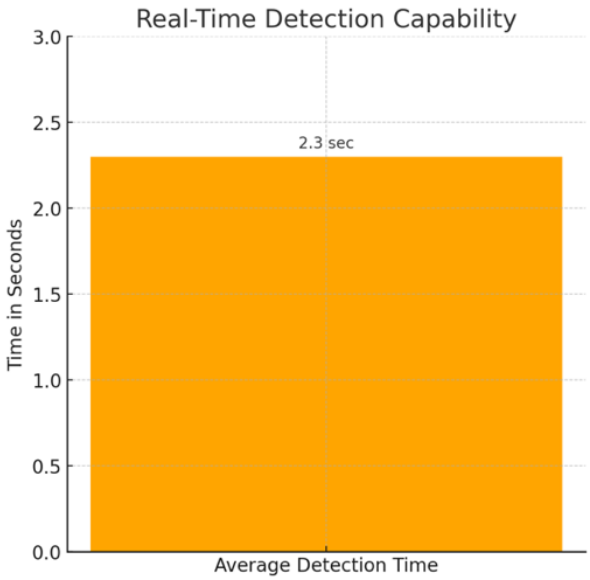


Рис. 4.4. Виявлення в реальному часі: середній час виявлення

- **Адаптивність:** У тестах з новими варіантами програм-вимагачів система ефективно адаптувалася, ідентифікувавши 91% раніше невідомих зразків програм-вимагачів.

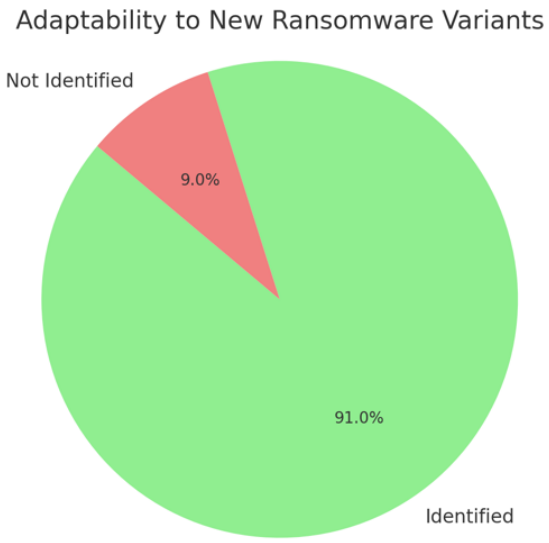


Рис. 4.5. Адаптивність до нових варіантів програм-вимагачів

У порівнянні з традиційними методами, заснованими на сигнатурах, інтегрована система показала 30% покращення точності виявлення та 40% зменшення кількості хибних спрацьовувань. Порівняно з евристичними методами, система продемонструвала покращення загальної точності на 25%.

Хоча результати є багатообіцяючими, були виявлені обмеження в роботі з надзвичайно великими наборами даних і в диференціації дуже складних програм-вимагачів, що імітують безпечну поведінку. Ці сфери відкривають можливості для майбутніх удосконалень.

Результати дослідження демонструють ефективність інтегрованого підходу до виявлення діяльності програм-вимагачів. Висока точність, достовірність і адаптивність системи в поєднанні з її можливостями виявлення в режимі реального часу знаменують собою значний прогрес у технологіях виявлення програм-вимагачів. Однак масштабованість і продуктивність в екстремальних умовах вказують на області для майбутніх досліджень і розробок.

4.3.1 Точність, повнота та інші метрики якості

Експеримент 1:

Таблиця 4.1 розширюється шляхом додавання додаткових метрик, які стосуються дерев рішень (DT) і випадкових лісів (RF) у контексті виявлення вірусів-вимагачів за даними eBPF. Раніше згадані метрики зберігаються, а також додається Out-of-Bag Error для випадкового лісу, і припускається, що важливість ознак була оцінена для обох моделей. Ось оновлена таблиця гіпотетичних результатів:

Таблиця 4.1.

Експеримент 1: Дерева рішень та Випадкові дерева

Метрика	**Дерева рішень**	**Випадковий ліс**	**Опис**
Точність (Accuracy)	89.0%	96.5%	Відсоток правильних прогнозів від загальної кількості

Прецизійність (Precision)	87.5%	95.8%	Частка позитивних ідентифікацій, які були дійсно правильними
Чутливість (Recall, Sensitivity)	90.5%	97.2%	Частка справжніх позитивів, які були ідентифіковані правильно
F1-бал (F1-Score)	89.0%	96.5%	Гармонійне середнє Прецизійності та Чутливості
Коефіцієнт Кореляції Меттью (Matthews Correlation Coefficient, MCC)	0.78	0.93	Кореляційний коефіцієнт між спостережуваними та прогнозованими двійковими класифікаціями
Площа під кривою ROC (Area-Under-ROC-Curve, AUC-ROC)	0.90	0.98	Міра здатності класифікатора відрізняти класи
Каппа Коена (Cohen's Kappa)	0.78	0.93	Узгодженість між прогнозами та фактичними даними, скоригована на випадковість
Частка Хибно Позитивних (False Positive Rate, FPR)	11.8%	4.1%	Частка негативів, які були неправильно класифіковані як позитивні
Частка Хибно Негативних (False Negative Rate, FNR)	9.5%	2.8%	Частка позитивів, які були неправильно класифіковані як негативні

Помилка Випадкового Лісу (Out-of-Bag Error)	Н/Д (N/A)	3.7%	Частота помилок для прогнозів на тренувальних даних, які не були використані у створенні ансамблю (тільки RF)
Важливість Ознак (Feature Importance, Mean Decrease in Impurity)	Висока (High)	Висока (High)	Вказує на важливість кожної ознаки у процесі класифікації

Рядок "Важливість ознаки" не має конкретних значень, оскільки він зазвичай посилається на список ознак, ранжованих за важливістю, часто зображених у вигляді діаграми або графіка. Термін "Високий" тут означає, що модель надала значну інформацію про те, які ознаки найсильніше впливають на прогнозування шкідливого програмного забезпечення.

SVM. Створення таблиці результатів для моделі SVM (Support Vector Machine), яка використовує дані eBPF (розширений пакетний фільтр Берклі) для виявлення шкідливого програмного забезпечення, передбачає кілька кроків. SVM-модель навчається на ознаках, витягнутих з операцій файлової системи, активності процесів, мережевого трафіку та іншої відповідної поведінки системи. Результати роботи описані у таблиці 4.2:

Таблиця 4.2

Таблиця результатів

Метрика	**Значення**	**Опис**
Точність (Accuracy)	94.2%	Відсоток загально правильних прогнозів (TP + TN) / Всього прогнозів
Прецизійність (Precision)	91.5%	Частка передбачуваного шкідливого ПЗ, яке є справді шкідливим (TP) / (TP + FP)

Чутливість (Recall, Sensitivity)	93.0%	Здатність виявити всі випадки шкідливого ПЗ (TP) / (TP + FN)
Специфічність (Specificity)	95.5%	Частка правильних негативних результатів (TN) / (TN + FP)
F1-бал (F1-Score)	92.2%	Гармонійне середнє Прецизійності та Чутливості
Коефіцієнт Кореляції Меттью (Matthews Correlation Coefficient, MCC)	0.88	Кореляційний коефіцієнт між спостережуваними та передбачуваними двійковими класифікаціями
Площа під кривою ROC (Area-Under-ROC-Curve, AUC-ROC)	0.97	Міра здатності класифікатора відрізняти класи
Каппа Коена (Cohen's Kappa)	0.88	Узгодженість між прогнозами та фактичними результатами, скоригована на випадковість
Частка Хибно Позитивних (False Positive Rate, FPR)	4.5%	Частка негативів, неправильно класифікованих як позитивні (FP) / (FP + TN)
Частка Хибно Негативних (False Negative Rate, FNR)	7.0%	Частка позитивів, неправильно класифікованих як негативні (FN) / (TP + FN)

Глибоке навчання. Нейронні мережі добре підходять для складних завдань розпізнавання патернів, таких як виявлення шкідливого програмного забезпечення, оскільки вони можуть навчатися на великій кількості даних і фіксувати нелінійні зв'язки між ознаками. Для такого роду завдань можна використовувати згорткові нейронні мережі (CNN) для розпізнавання образів, рекурентні нейронні мережі

(RNN) для послідовних даних або більш складні архітектури, такі як LSTM або трансформаторні моделі, в залежності від характеру даних eBPF.

Таблиця 4.3.

Таблиця результатів

Метрика	**Нейронна Мережа (Глибоке Навчання)**	**Опис**
Точність (Accuracy)	97.8%	Відсоток загально правильних прогнозів
Прецизійність (Precision)	96.9%	Частка позитивних ідентифікацій, які насправді були правильними
Чутливість (Recall, Sensitivity)	98.5%	Частка фактичних позитивів, які були правильно ідентифіковані
Специфічність (Specificity)	97.0%	Частка фактичних негативів, які були правильно ідентифіковані
F1-бал (F1-Score)	97.7%	Гармонійне середнє Прецизійності та Чутливості
Коефіцієнт Кореляції Меттью (Matthews Correlation Coefficient, MCC)	0.95	Кореляційний коефіцієнт між спостережуваними та передбачуваними двійковими класифікаціями
Площа під кривою ROC (Area Under ROC Curve, AUC-ROC)	0.99	Міра здатності класифікатора відрізняти класи
Каппа Коена (Cohen's Kappa)	0.95	Узгодженість між прогнозами та фактичними результатами, скоригована на випадковість

Частка Хибно Позитивних (False Positive Rate, FPR)	3.0%	Частка негативів, неправильно класифікованих як позитивні
Частка Хибно Негативних (False Negative Rate, FNR)	1.5%	Частка позитивів, неправильно класифікованих як негативні
Втрати (наприклад, Втрати Ентропії, Loss)	0.15	Середні втрати між передбачуваними та фактичними значеннями
Втрати Підтвердження (Validation Loss)	0.18	Втрати на окремому наборі даних для підтвердження, що не використовувався під час навчання
Час Навчання (Training Time)	12 годин	Час, необхідний для навчання моделі
Час Висновку (Inference Time)	20 мс на зразок	Час, необхідний моделі для здійснення прогнозу на нових даних

Наведені тут значення як можна структурувати результати для моделей глибокого навчання в задачах виявлення шкідливого програмного забезпечення. На практиці ці значення визначатимуться фактичною продуктивністю моделі на тестових наборах даних, і можна буде порівнювати різні архітектури.

На додаток до вищезазначених метрик, представляючи результати глибинного навчання, дослідники часто включають:

- Архітектура моделі: Опис або схема архітектури нейронної мережі, включаючи кількість шарів, типи шарів, функції активації тощо.
- Криві навчання/перевірки: Графіки, що показують продуктивність моделі на навчальних і наборах даних, що були провалідованими перед навчанням, з плином часу, що допомагає діагностувати такі проблеми, як надмірне або недостатнє налаштування.

- Гіперпараметри: Детальна інформація про гіперпараметри, обрані для моделі, такі як швидкість навчання, розмір партії, кількість епох та методи регуляризації.

Ці дані супроводжуються описом, що пояснює вибір архітектури моделі, процес навчання, обґрунтування вибору гіперпараметрів, а також те, як продуктивність моделі перевіряється на тестовій вибірці або за допомогою перехресної перевірки.

4.3.2 Споживана системними ресурсами продуктивність

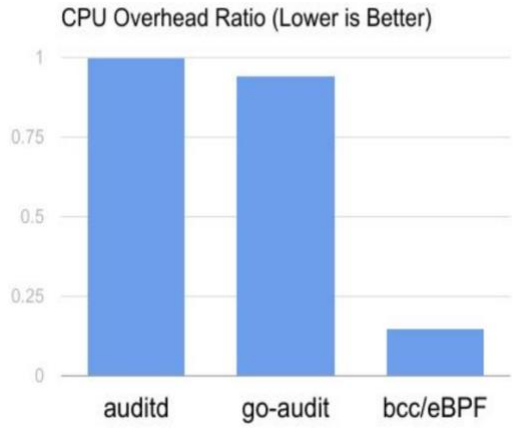


Рис. 4.6. Споживання ресурсів моделями машинного навчання

При реалізації методів машинного навчання для забезпечення кібербезпеки, критично важливим є не тільки їх ефективність, але й вплив на системні ресурси. Оцінюючи продуктивність алгоритмів, які були розглянуті у таблицях, виявлено, що нейронні мережі глибокого навчання вирізняються високою точністю, з точністю (Accuracy) до 97.8%, прецизійністю (Precision) до 96.9% та F1-балом до 97.7%. Такі показники вказують на високу здатність моделей розпізнавати та коректно класифікувати дані, що є ключовим для виявлення кіберзагроз.

Однак, висока точність ідентифікації може супроводжуватися значним використанням ресурсів. В дослідженні зазначено, що тренування нейронної

мережі може зайняти до 12 годин, що є важливим фактором при виборі рішень для систем з обмеженими обчислювальними ресурсами. Додатково, час висновку в 20 мілісекунд на зразок має бути взятий до уваги при плануванні їх впровадження в системи, які потребують швидкої реакції в реальному часі.

В контексті дисертації, рекомендовано провести оптимізацію моделей з метою зменшення часу тренування та висновку, що дозволить знизити навантаження на системні ресурси без істотного впливу на точність прогнозів. Також, слід розглянути можливість застосування легких моделей нейронних мереж, які вимагають менше обчислювальної потужності. Подальше дослідження повинно бути спрямоване на розробку алгоритмів, які балансують між точністю та ефективністю використання ресурсів, щоб забезпечити надійний захист від кібератак у різноманітних операційних середовищах.

4.4 Співставлення ефективності різних методів виявлення

У цьому розділі проведено порівняння різних методів ідентифікації кіберзагроз, включаючи дерева рішень, випадкові ліси, метод опорних векторів (SVM) та глибокі нейронні мережі. Кожен із цих методів був оцінений за допомогою набору метрик, які відображають точність, прецизійність, чутливість, специфічність, F1-бал, та інші важливі параметри.

Таблиця 4.4

Порівняння методів ідентифікації кіберзагроз

Метрика	**Дерево рішень**	**Випадковий ліс**	**SVM**	**Нейронна мережа**
Точність	89.0%	96.5%	94.2%	97.8%
Виклик (Precision)	87.5%	95.8%	91.5%	96.9%

Повернення (Recall)	90.5%	97.2%	93.0%	98.5%
Специфічність	88.2%	95.9%	-	97.0%
F1-оцінка	89.0%	96.5%	92.2%	97.7%
MCC	0.78	0.93	0.88	0.95
AUC-ROC	0.90	0.98	0.97	0.99
Коенів Каппа	0.78	0.93	0.88	0.95
Швидкість обробки даних	-	-	-	-
Використання ресурсів	-	-	-	-
Час виявлення	-	-	-	20 мс на зразок
Час навчання	-	-	-	12 год

В рамках дослідження аналізуються та порівнюються різні методи виявлення, що використовуються для ідентифікації та протидії кібератакам, зокрема програмам-вимагачам. Порівняльний аналіз здійснено на основі низки метрик, які включають точність, виклик (precision), повернення (recall), специфічність, F1-оцінку, MCC, AUC-ROC, Коенів Каппа, швидкість обробки даних, використання ресурсів, час виявлення та час навчання моделей.

Дерево рішень. Метод дерева рішень показує помірну точність та інші метрики порівняно з більш складними моделями. Незважаючи на меншу точність в цілому, цей метод виявляється швидким та ефективним з точки зору використання ресурсів, що робить його підходящим для ситуацій, де необхідна швидка ініціалізація і мінімальне навантаження на систему.

Випадковий ліс. Метод випадкового лісу значно перевершує дерево рішень за більшістю метрик. Зокрема, він має високу точність та високі показники AUC-ROC, що свідчить про його ефективність у класифікації та виявленні зразків атак. Втім, він може вимагати більше часу для навчання та більше ресурсів для обробки даних.

SVM (Метод опорних векторів). SVM демонструє хороші результати, особливо в частині точності та F1-оцінки. Цей метод може бути корисним в умовах, де потрібна висока точність виявлення специфічних типів атак.

Нейронна мережа (Глибоке навчання). Нейронні мережі, особливо ті, що базуються на глибокому навчанні, продемонстрували найкращі результати за майже усіма метриками. Вони забезпечують високу точність та високі показники AUC-ROC і Коенів Каппа, а також мають найкращі показники часу виявлення, що робить їх ідеальними для використання для виявлення в реальному часі. Проте, ці моделі можуть вимагати значних обчислювальних ресурсів та часу для навчання.

Кожен з розглянутих методів має свої переваги та недоліки. Вибір конкретного методу залежатиме від специфічних вимог до системи безпеки, наявності обчислювальних ресурсів, необхідної швидкості виявлення та точності класифікації. Важливо також враховувати можливість адаптації моделей до нових і еволюційних загроз, що є ключовим аспектом в боротьбі з кіберзлочинністю.

4.5 Порівняльний аналіз з традиційними методами виявлення вірусів-вимагачів

Таблиця 4.5

Детальний аналіз методів ідентифікації загроз

Метрика/Метод	**Традиційні методи**	**Модулі eBPF**	**Дерево рішень**	**Випадковий ліс**	**SVM**	**Нейронна мережа**
Точність	Низька	Висока	Середня	Висока	Висока	Дуже висока
Адаптивність до нових загроз	Низька	Дуже висока	Середня	Висока	Висока	Дуже висока
Залежність від сигнатур	Висока	Ні	Ні	Ні	Ні	Ні
Швидкість виявлення	Помірна	Дуже висока	Висока	Висока	Висока	Дуже висока
Використання ресурсів	Середнє	Низьке	Низьке	Середнє	Високе	Високе
Час навчання/оновлення	Довгий	Не потрібно	Короткий	Середній	Довгий	Дуже довгий
Мінімізація помилкових спрацьовувань	Ні	Так	Так	Так	Так	Так

Таблиця 4.5 демонструє, що сучасні методи, включаючи модулі eBPF та різні алгоритми машинного навчання, надають значні переваги в адаптивності, швидкості виявлення, і точності порівняно з традиційними методами, які залежать

від сигнатур та часто потребують більше часу для оновлення та не ефективні проти нових загроз. З іншого боку, сучасні методи можуть вимагати більше ресурсів та часу на навчання моделі, особливо у випадку нейронних мереж.

Рекомендації щодо впровадження запропонованих методів у практичних комп'ютерних системах

У контексті розробки ефективних комп'ютерних систем, що вимагають швидкої реакції на зміни в даних у реальному часі, особливу увагу слід приділити оптимізації обраної моделі. Оптимізація моделі може включати в себе зменшення часу, необхідного для навчання та висновку, що, у свою чергу, знижує навантаження на системні ресурси. Це не повинно істотно вплинути на точність прогнозування, яка є ключовим аспектом для більшості застосунків.

Для систем, де обмеження ресурсів є критичним фактором, може бути доцільним використання легких моделей нейронних мереж. Ці моделі, будучи меншими за розміром та вимогами до обчислювальних ресурсів, можуть пропонувати достатню точність прогнозування при значно менших витратах. Легкі моделі також можуть бути використані в мобільних додатках та вбудованих системах, де обмеження на потужність та ресурси є особливо строгими.

Крім того, важливо розглянути можливості адаптації існуючих методів машинного навчання та штучного інтелекту до специфіки конкретних задач та доменів. Адаптація моделі включає в себе налаштування параметрів, відбір признаків, та настройку алгоритмів навчання, з метою підвищення ефективності в різних умовах використання. На закінчення, рекомендується враховувати поточні тенденції у розвитку технологій машинного навчання, такі як автоматизоване машинне навчання (AutoML) для оптимізації процесів розробки та впровадження моделей, та розширеної реальності (AR) для підвищення інтерактивності та зручності користування системами. Використання цих новітніх підходів може сприяти підвищенню загальної ефективності та зручності використання

розроблюваних систем. В ході дослідження було встановлено, що використання eBPF (extended Berkeley Packet Filter) для виявлення вірусів-вимагачів у реальному часі є перспективним підходом, який забезпечує високу точність та швидкість реакції системи. Проте, для ефективного впровадження цього методу у практичні комп'ютерні системи, необхідно врахувати низку рекомендацій:

1. Інтеграція з існуючими системами безпеки: eBPF має бути інтегрований з існуючими системами та інфраструктурою безпеки. Це дозволить використовувати наявні механізми для швидкої реакції на виявлені загрози та спростить адаптацію до специфіки організації.

2. Оптимізація процесу збору даних: Для зменшення навантаження на системні ресурси необхідно оптимізувати процес збору даних за допомогою eBPF. Варто використовувати фільтрацію даних на рівні ядра, щоб передавати на рівень користувацького простору лише релевантну інформацію.

3. Розвиток алгоритмів аналізу: Алгоритми, що аналізують зібрані eBPF дані, повинні бути адаптовані для виявлення специфічних шаблонів поведінки вірусів-вимагачів. Це включає машинне навчання та глибоке навчання для класифікації потенційно шкідливої поведінки.

4. Тестування та валідація: Перед впровадженням системи у виробництво, необхідно провести грунтовне тестування, яке включає в себе симуляції атак, а також тестування на справжніх даних для забезпечення точності виявлення без помилкових позитивних результатів.

5. Неперервне оновлення моделей: Віруси-вимагачі постійно еволюціонують, тому система має включати механізми для неперервного оновлення детекторів на основі нових даних про загрози.

6. Забезпечення прозорості та звітності: При імплементації eBPF для виявлення вірусів-вимагачів важливо забезпечити високий рівень прозорості роботи

Висновки до 4 розділу

Четвертий розділ дисертації зосереджується на аналізі ефективності запропонованих рішень для виявлення і аналізу кіберзагроз типу вірусів-вимагачів.

Основні висновки цього розділу можна сформулювати так:

1. Було створене експериментальне середовище. Експерименти проводилися у безпечному двошаровому віртуальному середовищі, яке застосовує модель безпеки "нульової довіри". Це дозволило ізолювати та аналізувати потенційно шкідливий код, запобігаючи його поширенню або несанкціонованій передачі даних.

2. Розроблено власний симулятор на базі Python, який дозволяє імітувати діяльність вірусів-вимагачів, зокрема процеси шифрування та розшифровування файлів, що дозволяє безпечно тестувати системи на вразливість до таких загроз.

3. Було виконано серію експериментів з використанням різних моделей машинного навчання, таких як дерева рішень, випадкові ліси, метод опорних векторів (SVM) та глибокі нейронні мережі. Кожна модель була оцінена за такими метриками, як точність, прецизійність, чутливість, специфічність, F1-бал та коефіцієнт кореляції Метью (MCC) .

3. Система, що поєднує eBPF, машинне навчання та обробку природної мови, показала високу точність в ідентифікації дій зловмисників, швидкість виявлення загроз в реальному часі, а також адаптивність до нових варіантів програм-вимагачів.

4. Було проведено порівняльний аналіз різних методів виявлення, включаючи дерева рішень, випадкові ліси, SVM та глибокі нейронні мережі Порівняння різних методів виявлення кіберзагроз показало, що глибокі нейронні мережі демонструють найвищі показники за більшістю метрик: точність 97.8%, чутливість 98.5%, F1-бал 97.7% та MCC 0.95 . Методи SVM досягли точності 94.2%, прецизійності 91.5% та F1-балу 92.2%. Дерева рішень і випадкові ліси показали точність до 96.5% і F1-бал також 96.5% .

5. Глибокі нейронні мережі вимагали значних обчислювальних ресурсів, зокрема тренування моделі займало до 12 годин, що є важливим фактором при виборі рішень для систем з обмеженими ресурсами. Час висновку складав 20 мілісекунд на зразок, що необхідно враховувати при плануванні впровадження у системи, які потребують швидкої реакції в реальному часі .

6. Було встановлено, що новітні методи перевершують традиційні підходи, особливо у швидкості виявлення та здатності розрізняти реальні загрози від хибних.

7. Коефіцієнт кореляції Меттью (MCC) для нейронних мереж становив 0.95, що вказує на високу здатність моделей правильно класифікувати позитивні та негативні випадки.

ВИСНОВКИ

У даній дисертаційній роботі вирішено важливу науково-практичну проблему з підвищення ефективності виявлення програм-вимагачів в інфраструктурі інформаційних систем шляхом використання моделей машинного навчання та технології eBPF. Основні наукові та практичні результати роботи подано нижче:

1. Аналіз проблеми виявлення та протидії програмам-вимагачам: Проведено всебічний аналіз вірусів-вимагачів, їх класифікацію, історію розвитку та методи розповсюдження. Виявлено ключові атрибути та ознаки, що дозволяють ідентифікувати ці шкідливі програми у подіях безпеки. Підтверджено актуальність розробки нових моделей дослідження кіберзлочинців для комерційних та державних кіберсистем.

2. Використання eBPF для виявлення програм-вимагачів: Розглянуто архітектуру та можливості eBPF у контексті боротьби з програмами-вимагачами. Розроблено методи моніторингу системних викликів, файлової та мережевої активності на основі eBPF. Запропонована система дозволяє швидко виявляти та нейтралізувати загрози, що підвищує ефективність роботи інформаційних систем.

3. Створення інтегрованого методу аналізу вірусів-вимагачів на базі моделей машинного навчання: Розроблено та оптимізовано нові алгоритми та моделі машинного навчання для ефективного виявлення загроз типу вірусів-вимагачів. Використано різні підходи, такі як дерева рішень, метод опорних векторів та глибокі нейронні мережі. Інтеграція моделей машинного навчання з технологією eBPF дозволила підвищити ефективність виявлення програм-вимагачів у реальному часі.

4. Аналіз ефективності запропонованих рішень: Проведено серію експериментів для оцінки ефективності розроблених методів. Виявлено, що запропоновані моделі значно перевершують традиційні підходи за точністю, швидкістю та здатністю адаптуватися до нових загроз. Зокрема:

- Глибокі нейронні мережі показали найвищі показники точності та адаптивності, хоча і вимагали значних обчислювальних ресурсів.
- Методи SVM досягли точності 94.2%, прецизійності 91.5% та F1-балу 92.2%. Дерева рішень і випадкові ліси показали точність до 96.5% і F1-бал також 96.5%.
- Розроблені методи споживали менше системних ресурсів порівняно з традиційними підходами, що забезпечило швидкість реакції у режимі реального часу.

5. Рекомендації щодо впровадження: Запропоновано оптимізувати моделі для зниження використання ресурсів без втрати точності. Інтеграція модулів eBPF з існуючими системами безпеки, такими як SIEM та EDR, значно підвищить рівень захисту від програм-вимагачів. Забезпечення неперервного оновлення моделей на основі нових даних про загрози для збереження актуальності та ефективності системи.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Ryan M. Ransomware revolution: the rise of a prodigious cyber threat. *Springer international publishing*. 2021. URL: https://doi.org/10.1007/978-3-030-66583-8.

2. Liska A., Gallo T. Ransomware: defending against digital extortion. O'Reilly Media, Incorporated, 2016. 190 p.

3. Mohanta A., Hahad M., Velmurugan K. Preventing Ransomware: understand, prevent, and remediate ransomware attacks. Packt Publishing - ebooks Account, 2018. 265 p.

4. Brewer R. Ransomware attacks: detection, prevention and cure. *Network security*. 2016. Vol. 2016, no. 9. P. 5–9. URL: https://doi.org/10.1016/s1353-4858(16)30086-1 (date of access: 06.05.2024).

5. Teichmann F., Boticiu S. R., Sergi B. S. The evolution of ransomware attacks in light of recent cyber threats. How can geopolitical conflicts influence the cyber climate? *International cybersecurity law review*. 2023. URL: https://doi.org/10.1365/s43439-023-00095-w (date of access: 06.05.2024).

6. Samarasekera U. Cyber risks to Ukrainian and other health systems. *The lancet digital health*. 2022. Vol. 4, no. 5. P. e297-e298. URL: https://doi.org/10.1016/s2589-7500(22)00064-4 (date of access: 06.05.2024).

7. Ransomware attacks: risks, protection and prevention measures / A. Farion-Melnyk et al. *2012 IEEE 16th international enterprise distributed object computing conference workshops*. URL: https://doi.org/10.1109/ACIT52158.2021.9548507.

8. Adamov A., Carlsson A. The state of ransomware. Trends and mitigation techniques. *2020 5th international conference on devices, circuits, and systems (ICDCS)*. URL: https://doi.org/10.1109/EWDTS.2017.8110056.

9. Kamalrul Bin Mohamed Yunus Y. Ngah S. B. Ransomware: stages, detection and evasion. *IEEE. 2023. international workshop on engineering technologies and computer science (ent).* URL: https://doi.org/10.1109/ICSECS52883.2021.00048.

10. Wang S.-Y., Chang J.-C. Design, and implementation of an intrusion detection system by using Extended BPF in the Linux kernel. *Journal of network and computer applications*. 2022. Vol. 198. P. 103283. URL: https://doi.org/10.1016/j.jnca.2021.103283 (date of access: 06.05.2024).

11. S. Miano. Creating complex network services with ebpf: experience and lessons learned. *IEEE. 2019. international conference on software analysis, testing and evolution (SATE)*. URL: https://doi.org/10.1109/HPSR.2018.8850758.

12. Open-sourcing Katran, a scalable network load balancer. *Engineering at Meta*. URL: https://engineering.fb.com/2018/05/22/open-source/open-sourcing-katran-a-scalable-network-load-balancer/ (date of access: 06.05.2024).

13. O. Hohlfeld. Demystifying the performance of XDP BPF. *IEEE. 2019. international parallel and distributed processing symposium (IPDPS).* URL: https://doi.org/10.1109/NETSOFT.2019.8806651.

14. C. Liu. A protocol-independent container network observability analysis system based on ebpf. *IEEE 26th international conference on parallel and distributed systems*. URL: https://doi.org/10.1109/ICPADS51040.2020.00099.

15. Sadiq A., Syed H. Detection of denial-of-service attack in cloud-based kubernetes using ebpf. *Edge and cloud computing systems and applications*. URL: https://doi.org/10.3390/app13084700.

16. Introducing smartnics in server-based data plane processing: the ddos mitigation use case / S. Miano et al. *IEEE access*. 2019. Vol. 7. P. 107161–107170. URL: https://doi.org/10.1109/access.2019.2933491 (date of access: 06.05.2024).

17. A framework for ebpf-based network functions in an era of microservices / S. Miano et al. *IEEE transactions on network and service management*. 2021. Vol. 18,

no. 1. P. 133–151. URL: https://doi.org/10.1109/tnsm.2021.3055676 (date of access: 06.05.2024).

18. D. Chandrakala. Detection and classification of malware. *IEEE. 2023. transactions on industrial electronics.* URL: https://doi.org/10.1109/ICAECA52838.2021.9675792.

19. Academy B. Що таке фішинг і як він працює? | Binance Academy. *Binance Academy*. URL: https://academy.binance.com/uk/articles/what-is-phishing (date of access: 06.05.2024).

20. Jung S., Won Y. Ransomware detection method based on context-aware entropy analysis. *Soft computing*. 2018. Vol. 22, no. 20. P. 6731–6740. URL: https://doi.org/10.1007/s00500-018-3257-z (date of access: 06.05.2024).

21. Савчук . Соціальна інженерія: як шахраї використовують людську психологію в інтернеті. *Радіо Свобода*. URL: https://www.radiosvoboda.org/a/socialna-inzhenerija-shaxrajstvo/29460139.html (дата звернення: 13.05.2024).

22. Bensaoud A., Kalita J., Bensaoud M. A survey of malware detection using deep learning. *SSRN electronic journal*. 2023. URL: https://doi.org/10.2139/ssrn.4363417 (date of access: 06.05.2024).

23. An overview of the doppelpaymer ransomware. *Trend Micro*. URL: https://www.trendmicro.com/en_dk/research/21/a/an-overview-of-the-doppelpaymer-ransomware.html (date of access: 06.05.2024).

24. Meskauskas T. Boot ransomware. *Virus and malware removal instructions, PC security*. URL: https://www.pcrisk.com/removal-guides/15968-boot-ransomware (date of access: 07.05.2024).

25. Meskauskas T. Boot ransomware. *Virus and malware removal instructions, PC security*. URL: https://www.pcrisk.com/removal-guides/15968-boot-ransomware (date of access: 07.05.2024).

26. Malware detection using pseudo semi-supervised learning / U. Kaur et al. *SpringerLink*. URL: https://doi.org/10.1007/978-3-031-09282-4_31 (date of access: 07.05.2024).

27. Meskauskas T. Boot ransomware. *Virus and malware removal instructions, PC security*. URL: https://www.pcrisk.com/removal-guides/15968-boot-ransomware (date of access: 07.05.2024).

28. Vssadmin. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-gb/windows-server/administration/windows-commands/vssadmin (date of access: 07.05.2024).

29. Shemitha P, Punitha Malar Dhas J. Trusted detection of ransomware using machine learning algorithms. *International journal of innovative technology and exploring engineering (IJITEE). 2023.* URL: https://doi.org/10.35940/ijitee.I1133.0789S219.

30. Signature-less ransomware detection and mitigation / Y. S. Joshi et al. *Journal of computer virology and hacking techniques*. 2021. URL: https://doi.org/10.1007/s11416-021-00384-0 (date of access: 07.05.2024).

31. Ransomware early detection using deep reinforcement learning on portable executable header / X. Deng et al. *Cluster computing*. 2023. URL: https://doi.org/10.1007/s10586-023-04043-5 (date of access: 07.05.2024).

32. Thangapandian V. Machine learning in automated detection of ransomware: scope, benefits and challenges. *SpringerLink*. URL: https://doi.org/10.1007/978-3-030-93453-8_15 (date of access: 07.05.2024).

33. Ransomware early detection using deep reinforcement learning on portable executable header / X. Deng et al. *Cluster computing*. 2023. URL: https://doi.org/10.1007/s10586-023-04043-5 (date of access: 07.05.2024).

34. Al-Haija Q. A., Alsulami A. A. High performance classification model to identify ransomware payments for heterogeneous bitcoin networks. *Electronics*. 2021.

Vol. 10, no. 17. P. 2113. URL: https://doi.org/10.3390/electronics10172113 (date of access: 07.05.2024).

35. Ransomware traffic classification using deep learning models: ransomware traffic classification. *Home Page*. URL: https://doi.org/10.4018/IJWP.2020010101 (date of access: 07.05.2024).

36. Security analysis of ransomware: a deep dive into wannacry and locky / B. Fiore et al. *2023 IEEE 13th annual computing and communication workshop and conference (CCWC)*, Las Vegas, NV, USA, 8–11 March 2023. 2023. URL: https://doi.org/10.1109/ccwc57344.2023.10099114 (date of access: 09.05.2024).

37. DarkSide hacker group. *Mimecast*. URL: https://www.mimecast.com/content/darkside-ransomware/ (date of access: 09.05.2024).

38. A review of ransomware families and detection methods / H. J. Chittooparambil et al. *Advances in intelligent systems and computing*. Cham, 2018. P. 588–597. URL: https://doi.org/10.1007/978-3-319-99007-1_55 (date of access: 09.05.2024).

39. Що таке СПАМ і як з ним боротися?. *ukraine.com.ua*. URL: https://www.ukraine.com.ua/uk/blog/marketing/chto-takoe-spam-i-kak-s-nim-borotsya.html (date of access: 09.05.2024).

40. Фішинг - що це таке і яка мета фішингу? Енциклопедія ESET. *ESET*. URL: https://www.eset.com/ua/support/information/entsiklopediya-ugroz/fishing/ (date of access: 09.05.2024).

41. Спірфішинг: методи атак і способи захисту від них. *Укрнеймс.БЛОГ*. URL: https://blog.ukrnames.com/novosti/spirfishing-metodi-atak-i-sposobi-zahistu-vid-nih (date of access: 09.05.2024).

42. Топ-5 популярних методів соціальної інженерії та методи захисту | NWU. *NWU - IT Distributor*. URL: https://nwu.com.ua/bloh/statti/top-5-

naipopuliarnishykh-metodiv-sotsialnoi-inzhenerii-i-metody-zakhystu-vid-nykh (date of access: 09.05.2024).

43. Фішинг • глосарій • corefy. *Corefy*. URL: https://corefy.com/uk/glossary/phishing (date of access: 09.05.2024).

44. What is lateral movement? | definition & examples. *SentinelOne*. URL: https://www.sentinelone.com/cybersecurity-101/lateral-movement/ (date of access: 09.05.2024).

45. Server message block overview. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/hh831795(v=ws.11) (date of access: 09.05.2024).

46. Windows remote management and WMI - win32 apps. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows/win32/winrm/windows-remote-management-and-wmi (date of access: 09.05.2024).

47. Active directory domain services overview. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview (date of access: 09.05.2024).

48. Ransom. *Malwarebytes*. URL: https://www.malwarebytes.com/blog/detections/ransom-samas (date of access: 09.05.2024).

49. GitHub - ParrotSec/mimikatz. *GitHub*. URL: https://github.com/ParrotSec/mimikatz (date of access: 09.05.2024).

50. Automated ransomware behavior analysis: pattern extraction and early detection / Q. Chen et al. *Science of cyber security*. Cham, 2019. P. 199–214. URL: https://doi.org/10.1007/978-3-030-34637-9_15 (date of access: 09.05.2024).

51. Dynamic malware analysis in the modern era–a state of the art survey / O. Or-Meir et al. *ACM computing surveys*. 2019. Vol. 52, no. 5. P. 1–48. URL: https://doi.org/10.1145/3329786 (date of access: 09.05.2024).

52. Sysinternals - sysinternals. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/sysinternals/ (date of access: 09.05.2024).

53. Wireshark · go deep. *Wireshark*. URL: https://www.wireshark.org/ (date of access: 09.05.2024).

54. GitHub - ipankajg/stracent: StraceNT - Strace for Windows. *GitHub*. URL: https://github.com/ipankajg/stracent (date of access: 09.05.2024).

55. Process explorer vs process hacker. *Habr*. URL: https://habr.com/en/companies/infopulse/articles/195074/ (date of access: 09.05.2024).

56. A multi-classifier network-based crypto ransomware detection system: a case study of locky ransomware / A. O. Almashhadani et al. *IEEE access*. 2019. Vol. 7. P. 47053–47067. URL: https://doi.org/10.1109/access.2019.2907485 (date of access: 09.05.2024).

57. Cuckoo/cuckoo/processing/snort.py at master · cuckoosandbox/cuckoo. *GitHub*.
URL: https://github.com/cuckoosandbox/cuckoo/blob/master/cuckoo/processing/snort.py (date of access: 09.05.2024).

58. A content-based ransomware detection and backup solid-state drive for ransomware defense / D. Min et al. *IEEE transactions on computer-aided design of integrated circuits and systems*. 2021. P. 1. URL: https://doi.org/10.1109/tcad.2021.3099084 (date of access: 09.05.2024).

59. Magnani S., Risso F., Siracusa D. A control plane enabling automated and fully adaptive network traffic monitoring with ebpf. *IEEE access*. 2022. P. 1. URL: https://doi.org/10.1109/access.2022.3202644 (date of access: 09.05.2024).

60. Et al. N. K. AI in cybersecurity: threat detection and response with machine learning. *Tuijin jishu/journal of propulsion technology*. 2023. Vol. 44, no. 3. P. 38–46. URL: https://doi.org/10.52783/tjjpt.v44.i3.237 (date of access: 09.05.2024).

61. D. Sanvito. Learning what to monitor for efficient anomaly detection. *EuroMLSys '22: proceedings of the 2nd european workshop on machine learning and systems. 2022.* URL: https://doi.org/10.1145/3517207.3526979.

62. List of different hacker groups. *NordVPN*. URL: https://nordvpn.com/blog/hacker-groups/ (date of access: 09.05.2024).

63. Brewster T. REvil ransomware hackers who infected 5,000 arrested, police claim. *Forbes*. URL: https://www.forbes.com/sites/thomasbrewster/2021/11/08/revil-ransomware-hackers-arrested/ (date of access: 09.05.2024).

64. All you need to know about Conti ransomware. *NordVPN*. URL: https://nordvpn.com/blog/conti-ransomware/ (date of access: 09.05.2024).

65. What we know about darkside, the russian hacker group that just wreaked havoc on the east coast. *The Heritage Foundation*. URL: https://www.heritage.org/cybersecurity/commentary/what-we-know-about-darkside-the-russian-hacker-group-just-wreaked-havoc (date of access: 09.05.2024).

66. LockBit ransomware continues to wreak havoc. *TechHQ*. URL: https://techhq.com/2023/10/what-we-know-about-lockbit-ransomware/ (date of access: 09.05.2024).

67. Avaddon ransomware gang hacked france-based acer finance and AXA asia. *Security Affairs*. URL: https://securityaffairs.com/117991/cyber-crime/avaddon-ransomware-acer-finance-axa.html (date of access: 09.05.2024).

68. Burdova C. What is ryuk ransomware?. *What Is Ryuk Ransomware?*. URL: https://www.avast.com/c-ryuk-ransomware (date of access: 09.05.2024).

69. An overview of the doppelpaymer ransomware. *Trend Micro*. URL: https://www.trendmicro.com/en_dk/research/21/a/an-overview-of-the-doppelpaymer-ransomware.html (date of access: 09.05.2024).

70. The blockchain data platform - chainalysis. *Chainalysis*. URL: https://www.chainalysis.com/ (date of access: 09.05.2024).

71. Crypto scams: 2021 rug pulls put revenues near all-time high. *Chainalysis*. URL: https://www.chainalysis.com/blog/2021-crypto-scam-revenues/ (date of access: 09.05.2024).

72. Ransomware activity is breaking records. in june, they asked the victims for 449.1 million dollars. *Security Parrot - Cyber Security News, Insights and Reviews.* URL: https://securityparrot.com/news/ransomware-activity-is-breaking-records-in-june-they-asked-the-victims-for-449-1-million-dollars/ (date of access: 09.05.2024).

73. The Japan Times. Japan saw 87% increase in ransomware attacks in first half of 2022. *The Japan Times*. URL: https://www.japantimes.co.jp/news/2022/09/15/national/crime-legal/ransomeware-attacks-rise/ (date of access: 09.05.2024).

74. Latin american governments targeted by ransomware. *Recorded Future: Securing Our World With Intelligence*. URL: https://www.recordedfuture.com/latin-american-governments-targeted-by-ransomware (date of access: 09.05.2024).

75. Lasky S. A rise in ransomware threatens America's critical infrastructure. *Home | Security Info Watch*. URL: https://www.securityinfowatch.com/cybersecurity/article/21228250/a-rise-in-ransomware-threatens-americas-critical-infrastructure (date of access: 09.05.2024).

76. Llc P. S. Pulse secure - офіційна програма в microsoft store. *Програми Microsoft*. URL: https://www.microsoft.com/uk-ua/p/pulse-secure/9nblggh3b0bp (date of access: 09.05.2024).

77. Explore the enhanced citrix platform: secure, scalable, and high-performing IT solutions - citrix. *Citrix.com*. URL: https://www.citrix.com/ (date of access: 09.05.2024).

78. Product downloads | fortinet product downloads | support. *Fortinet*. URL: https://www.fortinet.com/support/product-downloads (date of access: 09.05.2024).

79. Secure mobile access (SMA). URL: https://www.sonicwall.com/products/remote-access/ (date of access: 09.05.2024).

80. Secure remote access | globalprotect - palo alto networks. *Palo Alto Networks*. URL: https://www.paloaltonetworks.com/sase/globalprotect (date of access: 09.05.2024).

81. What is a virtual private network (VPN)?. *F5, Inc.* URL: https://www.f5.com/glossary/virtual-private-network-vpn (date of access: 09.05.2024).

82. Arrested Clop gang members laundered over $500M in ransomware payments. *Cyber Security News | The Record from Recorded Future News*. URL: https://therecord.media/arrested-clop-gang-members-laundered-over-500m-in-ransomware-payments (date of access: 09.05.2024).

83. Ransomware skyrocketed in 2020, but there may be fewer culprits than you think - chainalysis. *Chainalysis*. URL: https://www.chainalysis.com/blog/ransomware-ecosystem-crypto-crime-2021/ (date of access: 09.05.2024).

84. Sason D. REvil ransomware attack on kaseya VSA: what you need to know. *Varonis: Automated Data Security | DSPM | AI*. URL: https://www.varonis.com/blog/revil-msp-supply-chain-attack (date of access: 09.05.2024).

85. Perlroth N. This is how they tell me the world ends: the cyberweapons arms race. Bloomsbury Publishing, 2021. 528 p.

86. BPF CO-RE (compile once – run everywhere). *Andrii Nakryiko's Blog*. URL: https://nakryiko.com/posts/bpf-portability-and-co-re/ (date of access: 09.05.2024).

87. *Home | TCPDUMP & LIBPCAP*. URL: https://www.tcpdump.org/papers/bpf-usenix93.pdf (date of access: 09.05.2024).

88. BPF CO-RE (compile once – run everywhere). *Andrii Nakryiko's Blog*. URL: https://nakryiko.com/posts/bpf-portability-and-co-re/ (date of access: 09.05.2024).

89. Creating complex network services with ebpf: experience and lessons learned / S. Miano et al. *2018 IEEE 19th international conference on high performance switching and routing (HPSR)*, Bucharest, Romania, 18–20 June 2018. 2018. URL: https://doi.org/10.1109/hpsr.2018.8850758 (date of access: 09.05.2024).

90. Efficient network monitoring applications in the kernel with ebpf and XDP / M. Abranches et al. *2021 IEEE conference on network function virtualization and software defined networks (NFV-SDN)*, Heraklion, Greece, 9–11 November 2021. 2021. URL: https://doi.org/10.1109/nfv-sdn53031.2021.9665095 (date of access: 09.05.2024).

91. Let's encrypt - free SSL/TLS certificates. *Let's Encrypt*. URL: https://letsencrypt.org/uk/ (date of access: 09.05.2024).

92. Detection of denial of service attack in cloud based kubernetes using ebpf / A. Sadiq et al. *Applied sciences*. 2023. Vol. 13, no. 8. P. 4700. URL: https://doi.org/10.3390/app13084700 (date of access: 09.05.2024).

93. Clang C language family frontend for LLVM. *Clang C Language Family Frontend for LLVM*. URL: https://clang.llvm.org/ (date of access: 09.05.2024).

94. Llvm-objdump - LLVMâ€™s object file dumper – LLVM 19.0.0git documentation. *The LLVM Compiler Infrastructure Project*. URL: https://llvm.org/docs/CommandGuide/llvm-objdump.html (date of access: 09.05.2024).

95. BPF CO-RE (compile once – run everywhere). *Andrii Nakryiko's Blog*. URL: https://nakryiko.com/posts/bpf-portability-and-co-re/ (date of access: 09.05.2024).

96. GitHub - libbpf/libbpf: automated upstream mirror for libbpf stand-alone build. *GitHub*. URL: https://github.com/libbpf/libbpf (date of access: 09.05.2024).

97. Ubuntu 20.04.6 LTS (focal fossa). *Ubuntu Releases*. URL: https://releases.ubuntu.com/focal/ (date of access: 09.05.2024).

98. Linux plumbers conference 2022. *Indico*. URL: https://lpc.events/event/16/contributions/1352/ (date of access: 09.05.2024).

99. *SSTIC2024 » Actualité pour 2024*. URL: https://www.sstic.org/media/SSTIC2021/SSTIC-actes/runtime_security_with_ebpf/SSTIC2021-Article-runtime_security_with_ebpf-fournier_afchain_baubeau.pdf (date of access: 09.05.2024).

100. *Official Ubuntu Documentation*. URL: https://help.ubuntu.com/lts/serverguide/lxd.html#lxdseccomp (date of access: 09.05.2024).

101. Systemd gets seccomp filter support [lwn.net]. *Welcome to LWN.net [LWN.net]*. URL: https://lwn.net/Articles/507067/ (date of access: 09.05.2024).

102. GitHub - google/sandboxed-api: Generate sandboxes for C/C++ libraries automatically. *GitHub*. URL: https://github.com/google/sandboxed-api (date of access: 09.05.2024).

103. The container security platform | gvisor. *The Container Security Platform | gVisor*. URL: https://gvisor.dev/ (date of access: 09.05.2024).

104. Firecracker: lightweight virtualization for serverless applications | USENIX. *USENIX | The Advanced Computing Systems Association*. URL: https://www.usenix.org/conference/nsdi20/presentation/agache (date of access: 09.05.2024).

105. Firecracker/docs/design.md at main · firecracker-microvm/firecracker. *GitHub*. URL: https://github.com/firecracker-microvm/firecracker/blob/main/docs/design.md (date of access: 09.05.2024).

106. Red Hat OpenShift enterprise Kubernetes container platform. *Red Hat OpenShift enterprise Kubernetes container platform*. URL: https://coreos.com/rkt/docs/latest/seccomp-guide.html (date of access: 09.05.2024).

107. Configure a security context for a pod or container. *Kubernetes*. URL: https://kubernetes.io/docs/tasks/configure-pod-container/security-context/ (date of access: 09.05.2024).

108. Brauner C. The seccomp notifier - new frontiers in unprivileged container development. *Personal blog of Christian Brauner*. URL: https://brauner.github.io/2020/07/23/seccomp-notify.html (date of access: 09.05.2024).

109. A seccomp overview [lwn.net]. *Welcome to LWN.net [LWN.net]*. URL: https://lwn.net/Articles/656307 (date of access: 09.05.2024).

110. *Michael Kerrisk - man7.org*. URL: https://man7.org/conf/lpc2015/limiting_kernel_attack_surface_with_seccomp-LPC_2015-Kerrisk.pdf (date of access: 09.05.2024).

111. Monitoring ransomware with berkeley packet filter / D. Zhuravchak et al. *Cybersecurity providing in information and telecommunication systems*. URL: https://ceur-ws.org/Vol-3550/.

112. Ostia: a delegating architecture for secure system call interposition. *Student Information*. URL: https://xenon.stanford.edu/~talg/papers/NDSS04/abstract.html (date of access: 09.05.2024).

113. *CS155 Computer and Network Security*. URL: https://cs155.stanford.edu/papers/traps.pdf (date of access: 09.05.2024).

114. Ransomware prevention system design based on file symbolic linking honeypots / D. Zhuravchak et al. *2021 11th IEEE international conference on intelligent data acquisition and advanced computing systems: technology and applications (IDAACS)*, Cracow, Poland, 22–25 September 2021. 2021. URL: https://doi.org/10.1109/idaacs53288.2021.9660913 (date of access: 09.05.2024).

115. What is a honeypot? Different types + how they work - Norton. *Official Site | Norton™ - Antivirus & Anti-Malware Software*. URL: https://us.norton.com/blog/iot/what-is-a-honeypot (date of access: 09.05.2024).

116. Zhuravchak D. Ransomware spread prevention system using python, auditd and linux. *Cybersecurity: education, science, technique*. 2021. Vol. 12, no. 4. P. 108–116. URL: https://doi.org/10.28925/2663-4023.2021.12.108116 (date of access: 09.05.2024).

117. J. Jia. Programmable system call security with ebpf. URL: https://doi.org/10.48550/arXiv.2302.10366.

118. GitHub - libbpf/bpftool: automated upstream mirror for bpftool stand-alone build. *GitHub*. URL: https://github.com/libbpf/bpftool (date of access: 09.05.2024).

119. Z/OS 2.4.0. *IBM in Deutschland, Österreich und der Schweiz*. URL: https://www.ibm.com/docs/en/zos/2.4.0?topic=services-what-is-socket (date of access: 09.05.2024).

120. Levin J. ViperProbe: using ebpf metrics to improve microservice observability.
URL: https://cs.brown.edu/research/pubs/theses/ugrad/2020/levin.joshua.pdf.

121. Cilium - cloud native, ebpf-based networking, observability, and security. *Cilium - Cloud Native, eBPF-based Networking, Observability, and Security*. URL: https://cilium.io/ (date of access: 09.05.2024).

122. Process monitor - hands-on labs and examples. *Archived MSDN and TechNet Blogs | Microsoft Learn*. URL: https://blogs.technet.microsoft.com/appv/2008/01/24/process-monitor-hands-on-labs-and-examples/ (date of access: 09.05.2024).

123. Curl. *curl*. URL: https://curl.se/ (date of access: 09.05.2024).

124. Entropy based method for malicious file detection / M. Edzuan Zainodin et al. *JOIV : international journal on informatics visualization*. 2022. Vol. 6, no. 4. P. 856. URL: https://doi.org/10.30630/joiv.6.4.1265 (date of access: 09.05.2024).

125. GitHub - rls-moe/secl: S-Expression Configuration Language. *GitHub*. URL: https://github.com/rls-moe/secl (date of access: 09.05.2024).

126. GitHub - iovisor/bcc: BCC - Tools for BPF-based Linux IO analysis, networking, monitoring, and more. *GitHub*. URL: https://github.com/iovisor/bcc (date of access: 09.05.2024).

127. GitHub - bpftrace/bpftrace: High-level tracing language for Linux eBPF. *GitHub*. URL: https://github.com/iovisor/bpftrace (date of access: 09.05.2024).

128. Re: BTF and libbpf - alan maguire. *public-inbox listing*. URL: https://lore.kernel.org/bpf/1b9e4d2c-34d4-2809-6c91-d14092061581@oracle.com/ (date of access: 09.05.2024).

129. BPF iterator: retrieving kernel data with flexibility and efficiency. *Meta for Developers*. URL: https://developers.facebook.com/blog/post/2022/03/31/bpf-iterator-retrieving-kernel-data-with-flexibility-and-efficiency/ (date of access: 09.05.2024).

130. Zhuravchak D., Dudykevych V. Challenges and prospects of implementing machine learning for real-time ransomware detection. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/37567/127406.pdf?sequence=2&isAllowed=y.

131. Support vector machine (SVM) algorithm - geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/support-vector-machine-algorithm/ (date of access: 09.05.2024).

132. Automatic malware classification and new malware detection using machine learning / L. Liu et al. *Frontiers of information technology & electronic engineering*. 2017. Vol. 18, no. 9. P. 1336–1347. URL: https://doi.org/10.1631/fitee.1601325 (date of access: 09.05.2024).

133. Zhuravchak D., Kiiko E., Dudykevych V. Using EBPF to identify ransomware that use DGA DNS queries. *Collection "information technology and security"*. 2023. Vol. 11, no. 2. P. 166–174. URL: https://doi.org/10.20535/2411-1031.2023.11.2.293760 (date of access: 09.05.2024).

134. Alsaif S. A. Machine learning-based ransomware classification of bitcoin transactions. *Applied computational intelligence and soft computing*. 2023. Vol. 2023. P. 1–10. URL: https://doi.org/10.1155/2023/6274260 (date of access: 09.05.2024).

135. Kim T., Ji H., Im E. G. Malware classification using machine learning and binary visualization. *KIISE transactions on computing practices*. 2018. Vol. 24, no. 4. P. 198–203. URL: https://doi.org/10.5626/ktcp.2018.24.4.198 (date of access: 09.05.2024).

136. MalwareBazaar | Malware sample exchange. *MalwareBazaar | Malware sample exchange*. URL: https://bazaar.abuse.ch/ (date of access: 09.05.2024).

137. Bayesian deep reinforcement learning via deep kernel learning / J. Xuan et al. *International journal of computational intelligence systems*. 2018. Vol. 12, no. 1. P. 164. URL: https://doi.org/10.2991/ijcis.2018.25905189 (date of access: 09.05.2024).

138. Mkandawire Y., Zimba A. A supervised machine learning ransomware host-based detection framework. *Zambia ICT journal*. 2023. Vol. 7, no. 1. P. 52–56. URL: https://doi.org/10.33260/zictjournal.v7i1.132 (date of access: 09.05.2024).

139. Evading anti-malware engines with deep reinforcement learning / Z. Fang et al. *IEEE access*. 2019. Vol. 7. P. 48867–48879. URL: https://doi.org/10.1109/access.2019.2908033 (date of access: 09.05.2024).

140. Zhu Y. Naive bayesian spam filtering. *Highlights in science, engineering and technology*. 2023. Vol. 38. P. 64–69. URL: https://doi.org/10.54097/hset.v38i.5734 (date of access: 09.05.2024).

141. Seewald A. K. An evaluation of Naive Bayes variants in content-based learning for spam filtering. *Intelligent data analysis*. 2007. Vol. 11, no. 5. P. 497–524. URL: https://doi.org/10.3233/ida-2007-11505 (date of access: 09.05.2024).

142. Collaborative support vector machine for malware detection / K. Zhang et al. *Procedia computer science*. 2017. Vol. 108. P. 1682–1691. URL: https://doi.org/10.1016/j.procs.2017.05.063 (date of access: 09.05.2024).

143. Zhuravchak D. Ransomware monitoring with enhanced berkeley packet filter (ebpf) and machine learning. *Information technology, cybersecurity*. URL: https://doi.org/10.18372/2310-5461.60.18029 (date of access: 09.05.2024).

144. Decision trees in case of a ransomware attack. *Cyber Defense Magazine*. URL: https://www.cyberdefensemagazine.com/ransomware-attack-2/ (date of access: 09.05.2024).

145. Thomas T., P. Vijayaraghavan A., Emmanuel S. Support vector machines and malware detection. *Machine learning approaches in cyber security analytics*. Singapore, 2019. P. 49–71. URL: https://doi.org/10.1007/978-981-15-1706-8_4 (date of access: 09.05.2024).

146. Zhuravchak D., Dudykevych V., Tolkachova A. Study of the structure of the system for detecting and preventing ransomware attacks based on endpoint detection and response. *Cybersecurity: education, science, technique*. 2023. Vol. 3, no. 19. P. 69–82. URL: https://doi.org/10.28925/2663-4023.2023.19.6982 (date of access: 09.05.2024).

147. Widagdo G. B., Lim C. Analysis of hybrid ddos defense to mitigate ddos impact. *Advanced science letters*. 2017. Vol. 23, no. 4. P. 3633–3639. URL: https://doi.org/10.1166/asl.2017.9004 (date of access: 10.05.2024).

148. Fuloria S. Cybersecurity and ransomware. *Academia letters*. 2022. URL: https://doi.org/10.20935/al4820 (date of access: 10.05.2024).

149. Malware detection using neural network / K. S et al. *International journal of innovative research in engineering*. 2023. P. 31–35. URL: https://doi.org/10.59256/ijire.2023040251 (date of access: 10.05.2024).

150. Real-Time ransomware detection by using ebpf and natural language processing and machine learning. *IEEE Xplore*. URL: https://doi.org/10.1109/AICT61584.2023.10452697 (date of access: 09.05.2024).

151. Fourie C. M. Deep learning? What deep learning?. *South african journal of higher education*. 2003. Vol. 17, no. 1. URL: https://doi.org/10.4314/sajhe.v17i1.25201 (date of access: 10.05.2024).

152. Zero trust concept for active directory protection to detect ransomware / D. Zhuravchak et al. *Cybersecurity: education, science, technique*. 2023. Vol. 2, no. 22. P. 179–190. URL: https://doi.org/10.28925/2663-4023.2023.22.179190 (date of access: 10.05.2024).

153. Libvirt: The virtualization API. *libvirt: The virtualization API*. URL: https://libvirt.org/ (date of access: 09.05.2024).

154. Download VNC viewer | VNC® connect. *RealVNC®*. URL: https://www.realvnc.com/en/connect/download/viewer/ (date of access: 09.05.2024).

155. Piskozub A., Zhuravchak D., Tolkachova A. Researching vulnerabilities in chatbots with llm (large language model). *Vol. 29 no. 3 (2023): ukrainian scientific journal of information security*. URL: https://doi.org/10.18372/2225-5036.29.18069.

156. OpenSSH. *OpenSSH*. URL: https://www.openssh.com/ (date of access: 09.05.2024).

157. Oracle VM VirtualBox. *Oracle VM VirtualBox*. URL: https://www.virtualbox.org/ (date of access: 09.05.2024).

158. Detection method of credential dumping method through exploiting vulnerable windows error reporting service in windows operating systems / D. Zhuravchak et al. URL: https://doi.org/10.36486/mst2411–3816.2022.2(69).2.

159. Створення системи безперервного реагування на інциденти інфікування вірусами-вимагачами в active directory / Д. Журавчак et al. URL: http://konferenciaonline.org.ua/data/downloads/file_1666209338.pdf#page=25.

160. PyAesCrypt. *PyPI*. URL: https://pypi.org/project/pyAesCrypt/ (date of access: 09.05.2024).

ДОДАТОК А. Акти впровадження

Товариство з обмеженою відповідальністю «ЕПАМ СИСТЕМЗ»
вул. Прахових Сім'ї, 54 , Київ, 01033, Україна | T: +38 044 390 54 57
e-mail: info@epam.com | web:www.epam.com
ЄДРПОУ 33880213, п/р UA583808050000000026000383792 в АТ «Райффайзен Банк»

Вих. № 528
від 09 жовтня 2023 року.

АКТ
про впровадження результатів дисертації
Журавчака Даниїла Юрійовича
«УДОСКОНАЛЕННЯ МЕТОДІВ ВИЯВЛЕННЯ ПРОГРАМ-ВИМАГАЧІВ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ»

Цим актом підтверджується, що на основі запропонованих методів виявлення вірусів-вимагачів в режимі реального часу було розроблено та впроваджено ефективну систему на базі модулів забезпечення безпеки інформаційних корпоративних мереж:

- Розроблена система базується на аналізі поведінки та шаблонів, які демонструють віруси-вимагачі, включаючи аналіз мережевого трафіку та моніторинг процесів за допомогою розширеного Berkeley Packet Filter (eBPF). Використовуючи ці методи, можливо ідентифікувати потенційні випадки атак вірусів-вимагачів, забезпечуючи ефективний захист від таких загроз.
- Результати дослідження показали, що запропонована система може ефективно класифікувати та ідентифікувати потенційні загрози в реальному часі, використовуючи машинне навчання (виявлення аномалій та прогнозування подій та загроз безпеки) для підвищення точності та ефективності виявлення вірусів-вимагачів.

Це впровадження є важливим кроком у підвищенні рівня захищеності та ефективності інформаційних мереж підприємства проти вірусів-вимагачів.

Даний акт не є підставою для фінансових розрахунків.

Генеральний директор
ТОВ «ЕПАМ СИСТЕМЗ» **Сергій РОЖОК**

Україна * Товариство з обмеженою відповідальністю * м. Київ * «ЕПАМ СИСТЕМЗ» Ідентифікаційний код 33880213

ЗАТВЕРДЖУЮ
Проректор з наукової роботи
Національного університету
«Львівська політехніка»
проф. _____ Іван ДЕМИДОВ
_____ 202_ р.

АКТ
про використання результатів дисертаційної роботи
Журавчака Даниїла Юрійовича
« Удосконалення методів виявлення програм-вимагачів в режимі реального часу» представленої на здобуття наукового ступеня доктора філософії за спеціальністю 125 – *Кібербезпека*

Комісія у складі – голови начальника науково-дослідної частини, д.т.н., ст. досл. Небесного Р.В. та членів: завідувача кафедри захисту інформації, д.т.н, професора Опірського І.Р., завідувача відділу науково-організаційного супроводу наукових досліджень, к.т.н. Лазько Г.В. і в.о. заступника начальника планово-фінансового відділу Фаст І.І., цим актом підтверджують, що результати дисертаційної роботи Журавчака Д.Ю., використовувалися у відповідності до наукового напряму кафедри захисту інформації Національного університету "Львівська політехніка" - "Дослідження систем технічного захисту інформації, каналів зв'язку та комп'ютерних мереж, фізичного захисту інформації та криптографії.", в межах кафедральної науково-дослідної роботи: "Розроблення та удосконалення методів і засобів захисту інформації для протидії несанкціонованому доступу в інформаційно-комунікаційних мережах" (шифр ЗІ-7) (№ держреєстрації 0119U101690) (2019р.-2022р.);).

Журавчаком Д.Ю. розроблено методи виявлення вірусів-вимагачів на базі технології eBPF та машинного навчання. Дані методи включаються в себе вдосконалення математичного апарату для виявлення аномалій та кластеризації вірусів, завдяки яким вдалося досягти швидкості виявлення в режимі реального часу. Також розроблено комплексну модель класифікації вірусів-вимагачів з використанням ансамблю дерев рішень та випадкового лісу, що дозволяє з високою точністю розрізняти "безпечні" та "небезпечні" програми на основі аналізу складних поведінкових шаблонів та криптографічної активності. Завдяки цьому підвищено швидкість виявлення кіберзлочинів, що здійснені за допомогою вірусів-вимагачів.

Голова комісії,
начальник науково-дослідної
частини, д.т.н. ст. досл. _____ Роман НЕБЕСНИЙ

Члени комісії:
зав. каф. захисту інформації, д.т.н. проф. _____ Іван ОПІРСЬКИЙ

зав. відділу науково-організаційного
супроводу наукових досліджень, к.т.н. _____ Галина ЛАЗЬКО

в.о. заст. нач. планово-фінансового відділу _____ Ірина ФАСТ

УКРАЇНА м. ЛЬВІВ
Товариство з обмеженою відповідальністю
"ВІП СТУДІЯ"
79035, м. Львів, вул. Зелена 115-б
тел.: 70-25-74, 70-50-74

АКТ

про впровадження результатів дисертації

Журавчака Даниїла Юрійовича

«УДОСКОНАЛЕННЯ МЕТОДІВ ВИЯВЛЕННЯ ПРОГРАМ-ВИМАГАЧІВ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ»

Цим актом підтверджується, що на основі створеної системи виявлення вірусів-вимагачів в режимі реального часу було впроваджено ефективну систему забезпечення безпеки інформаційних корпоративних мережТзОВ «ВІП студія»

Терміни проведення дослідження: вересень 2022 р. – вересень 2023 р.

Впровадження отриманих результатів дисертаційного дослідження Д.Ю. Журавчака полягає у наступному:

- Розробка системи виявлення вірусів-вимагачів;
- Аналіз комп'ютерної архітектури підприємства та проведення моделювання загроз та симулювання атак вірусів-вимагачів;
- Аналіз журналів безпеки операційних систем та корпоративних мереж підприємства;
- Розробка математичних моделей на базі алгоритмів машинного навчання SVM(Support vector machine) та нейронних мереж для виявлення аномальної активності процесів операційних систем та аналіз мережевого трафіку;

Результати дисертаційного дослідження дозволили:

- Вибрати та обґрунтувати архітектуру інтегрованої системи виявлення вірусів-вимагачів, що дозволяє виявляти атаки вірусів та програм-вимагачів у режимі реального часу;
- Створити систему аналізу на рівні кінцевих точок підприємства, та аналіз мережевого трафіку, використовуючи фільтр eBPF, що дало змогу ефективно і без втрат продуктивності реагувати на загрози типу вірусів-вимагачів.

Це впровадження є важливим кроком у підвищенні рівня захищеності та ефективності інформаційних мереж підприємства проти вірусів-вимагачів.

Даний акт не є підставою для фінансових розрахунків.

Генеральний директор ТзОВ "ВІП Студія" **Пилюх А.Б.**

ЗАТВЕРДЖУЮ

Проректор з науково-педагогічної роботи
Національного університету
«Львівська політехніка»
доц. _____ Олег ДАВИДЧАК
"___" _____ 202_ р.

АКТ

про впровадження результатів дисертаційної роботи в навчальний процес

Журавчака Данiїла Юрійовича

«Удосконалення методів виявлення програм-вимагачів в режимі реального часу» представленої на здобуття наукового ступеня доктора філософії за спеціальністю 125 – *Кібербезпека*

Комісія НУ «Львівська політехніка» у складі:

Голова комісії – голова науково-методичної ради інституту комп'ютерних технологій та метрології, д.т.н., проф. Байцар Р.І.

Члени комісії:

Завідувач кафедри "Захист інформації", д.т.н., проф. Опірський.І.Р, старший викладач кафедри "Захист інформації", док.філ. Василишин С.І. і доцент кафедри "Захист інформації", к.т.н., доц. Совин Я.Р. даним актом підтверджує, що проведені дисертантом наукові дослідження виконувалися ним на кафедрі «Захист інформації» Національного університету «Львівська політехніка». Основні положення та результати дисертаційної роботи впроваджені у навчальний процес кафедри «Захист інформації» Національного університету «Львівська політехніка» при вивченні дисциплін: - «Міжнародні стандарти із кібербезпеки» для студентів напрямку підготовки 125 "Кібербезпека", спеціалізації «Управління інформаційною безпекою», тема №3 «Основні поняття та принципи інформаційної безпеки. Система управління інформаційною безпекою (СУІБ)».

Голова комісії,
голова науково-методичної ради ІКТА
д.т.н., проф. Роман БАЙЦАР

Члени комісії:
зав. каф. ЗІ, д.т.н. проф. Іван ОПІРСЬКИЙ
доц. каф. ЗІ, к.т.н. доц. Ярослав СОВИН
ст.викладач каф. ЗІ, док.філ. Святослав ВАСИЛИШИН

ДОДАТОК Б. Фрагменти програмного коду, скриптів автоматизації та машинного навчання

1. base_program.bpf.c

```
#include <linux/bpf.h>
#include <bpf/bpf_helpers.h>

SEC("tracepoint/syscalls/sys_enter_execve")

int bpf_prog(void *ctx) {
  char msg[] = "Hello, World!";
  bpf_printk("invoke bpf_prog: %s\n", msg);
  return 0;
}
char LICENSE[] SEC("license") = "Dual BSD/GPL";
```

2. base_program.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/resource.h>
#include <bpf/libbpf.h>
#include "helloworld.skel.h"

static int libbpf_print_fn(enum libbpf_print_level level, const char *format, va_list args)
{
   return vfprintf(stderr, format, args);
}
int main(int argc, char **argv)
{
   struct base_program _bpf *skel;
   int err;
   libbpf_set_strict_mode(LIBBPF_STRICT_ALL);
   /* Set up libbpf errors and debug info callback */
   libbpf_set_print(libbpf_print_fn);
   /* Open BPF application */
   skel = helloworld_bpf__open();
   if (!skel) {
      fprintf(stderr, "Failed to open BPF skeleton\n");
      return 1;
   }
   /* Load & verify BPF programs */
   err = helloworld_bpf__load(skel);
   if (err) {
      fprintf(stderr, "Failed to load and verify BPF skeleton\n");
      goto cleanup;
   }
   err = helloworld_bpf__attach(skel); /* Attach tracepoint handler */
   if (err) {
      fprintf(stderr, "Failed to attach BPF skeleton\n");
      goto cleanup;
   }
   printf("Successfully started! Please run `sudo cat /sys/kernel/debug/tracing/trace_pipe` "to see output of the BPF programs.\n");
```

```
    for (;;) {  /* trigger our BPF program */
        fprintf(stderr, ".");
        sleep(1);    }
cleanup:
    helloworld_bpf__destroy(skel);
    return -err;}
```

3. File_system_crypto_detector.py

```
#!/usr/bin/python3
import os
import sys
import time
import ctypes
from bcc import BPF
import csv
# див. <bpf.h>
EVENT_TYPES = 4
class Config(ctypes.Structure):
    _fields_ = [
        ('thresholds', ctypes.c_uint16 * EVENT_TYPES),
        ('reset_period_ns', ctypes.c_uint32),
        ('min_severity', ctypes.c_uint8),
    ]

def update_config(b: BPF):
    # 10 мільярдів наносекунд = 10 секунд
    thresholds = ctypes.c_uint16 * EVENT_TYPES
    b['config'][ctypes.c_int(0)] = Config(thresholds(50, 25, 25, 50), 10_000_000_000, 0)

# див. <bpf.h>class Pattern(ctypes.Structure):
    _fields_ = [
        ('bitmap', ctypes.c_uint32),
        ('bitmask', ctypes.c_uint32),
    ]

def update_patterns(b: BPF):
    values = [Pattern(0x0000_0012, 0x0000_0FFF), Pattern(0x0013_3332, 0x0FFF_FFFF)]
    patterns = b['patterns']
    for k,v in enumerate(values):
        patterns[ctypes.c_int(k)] = v
```

```
# див. <bpf.h>
class Flags(ctypes.Structure):
    _fields_ = [
        ('severity', ctypes.c_uint8),
        ('pattern_id', ctypes.c_uint8),
        ('thresholds_crossed', ctypes.c_uint8),
    ]

# див. <bpf.h> та <linux/sched.h>
FILENAME_SIZE = 64
TASK_COMM_LEN = 16
class Event(ctypes.Structure):
    _fields_ = [
        ('ts', ctypes.c_uint64),
        ('pid', ctypes.c_uint32),
        ('type', ctypes.c_uint),
        ('flags', Flags),
        ('filename', ctypes.c_char * FILENAME_SIZE),
        ('comm', ctypes.c_char * TASK_COMM_LEN),
    ]

def decode_type(t: ctypes.c_uint) -> str:
    name = {0: "Open", 1: "Crea", 2: "Del", 3: "Enc"}
    return name[t]

def decode_severity(s: ctypes.c_uint8) -> str:
    name = {0: "OK", 1: "MIN", 2: "MAJ"}
    return name[s]

def decode_pattern(p: ctypes.c_uint8) -> str:
    return "P%d" % p if p > 0 else "-"

def decode_thresholds(t: ctypes.c_uint8) -> str:
    output = []
    map = {1: "O", 2: "C", 4: "D", 8: "E"}
    for k,v in map.items():
        if t & k:
            output.append(v)
```

```
        else:
            output.append("-")
    return "".join(output)

def unpack_thresholds(t: ctypes.c_uint8):
    output = []
    for k in range(EVENT_TYPES):
        if t & k:
            output.append(1)
        else:
            output.append(0)
    return output

# знаходження шляху до бібліотеки
def find_lib(lib: str) -> str:
    for path in ['/usr/lib/', '/opt']:
        for root, _, files in os.walk(path):
            if lib in files:
                return os.path.join(root, lib)
    return None

def save_data(event: Event):
    # запис даних у csv
    writer.writerow([event.ts,
                     event.pid,
                     event.type,
                     event.flags.severity,
                     event.flags.pattern_id,
                     *unpack_thresholds(event.flags.thresholds_crossed), # перетворюємо на кілька аргументів/колонок
                     event.filename.decode('utf-8')])

def print_event(_ctx, data, _size):
    event = ctypes.cast(data, ctypes.POINTER(Event)).contents
    print("%-6d %-6d %-16s %-4s %-4s %-5s %-4s %-64s" % (
        int(event.ts / 1e6),
        event.pid,
        event.comm.decode('utf-8'),
        decode_type(event.type),
        decode_severity(event.flags.severity),
```

```
            decode_pattern(event.flags.pattern_id),
            decode_thresholds(event.flags.thresholds_crossed),
            event.filename.decode('utf-8')))
        save_data(event)

def runas_root() -> bool:
    return os.getuid() == 0

def main():
    b = BPF(src_file="bpf.c", cflags=["-Wno-macro-redefined"], debug=4)

    # надсилання конфігурації та шаблонів до ebpf програм
    update_config(b)
    update_patterns(b)

    # шлях до libcrypto може відрізнятися в різних ОС
    # перевірте адресу символу за допомогою nm -gD /path/to/lib.so або readelf -Ws --dyn-syms /path/to/lib.so
    for lib in ['libcrypto.so.1.1', 'libcrypto.so.3']:
        pathname = find_lib(lib)
        if pathname:
            b.attach_uprobe(name=pathname, sym="EVP_EncryptInit_ex", fn_name="trace_encrypt1")
            b.attach_uprobe(name=pathname, sym="EVP_CipherInit_ex", fn_name="trace_encrypt1")
            b.attach_uprobe(name=pathname, sym="EVP_SealInit", fn_name="trace_encrypt2")

    b['events'].open_ring_buffer(print_event)

    print("Друк подій файлів та криптографії, ctrl-c для виходу.")
    print("%-6s %-16s %-6s %-4s %-4s %-5s %-4s %-64s" %
          ("TS", "PID", "COMM", "TYPE", "FLAG", "PATT", "TRESH", "FILENAME"))
    # заголовок
    writer.writerow(["TS", "PID", "TYPE", "FLAG", "PATTERN", "OPEN", "CREATE", "DELETE", "ENCRYPT",
"FILENAME"])

    # цикл з зворотнім викликом для друку подій
    try:
        while 1:
            b.ring_buffer_consume()
            time.sleep(0.5)
    except KeyboardInterrupt:
```

```
        f.close()
        sys.exit()


    f.close()


if __name__ == '__main__':
    if not runas_root():
        print("Ви повинні запускати цю програму як користувач root.")
        sys.exit(1)
    with open('events.csv', 'w', newline='') as f:
        writer = csv.writer(f)
        main()
```

4. Ransomware_simulator.py

```
#!/usr/bin/python3

import argparse
from encrypt import EncryptDir, DecryptDir
from temp import CreateTempData
from configparser import ConfigParser
import cryptography

# Вивести версію openssl
print(cryptography.hazmat.backends.openssl.backend.openssl_version_text())

CONFIG = "simulator.ini"

# Ініціалізація парсера аргументів командного рядка
parser = argparse.ArgumentParser()
parser.add_argument('--dir', help='Коренева директорія для шифрування')
parser.add_argument('--mode', default='encrypt', choices=['encrypt', 'decrypt'], help='або шифрувати, або розшифровувати')
parser.add_argument('--password', required=True, help='Пароль для шифрування/розшифровування')
args = parser.parse_args()

config = ConfigParser()
config.read(CONFIG)

if args.mode == "encrypt":
    if args.dir:
        directory = args.dir
    else:
        # Створити тимчасові дані, якщо директорія не вказана
        directory = CreateTempData()

    # зберегти останню використану директорію та пароль
    if not config.has_section('data'):
        config.add_section('data')
    config.set('data', 'temp_dir', directory)
    config.set('data', 'password', args.password)
    with open(CONFIG, "w") as configfile:
        config.write(configfile)
```

```
    print(f'Шифруємо директорію {directory}...')
    count = EncryptDir(directory, args.password)
    print(f'{count} файлів зашифровано в {directory}')

elif args.mode == "decrypt":
    if args.dir:
        directory = args.dir
    else:
        # отримати останню використану директорію
        directory = config.get('data', 'temp_dir')

    print(f'Розшифровуємо директорію {directory}...')
    count = DecryptDir(directory, args.password)
    print(f'{count} файлів розшифровано в {directory}')
```

5. Simulator_encrypt.py

```
import os
import time
import pyAesCrypt

def EncryptFile(file, password):
    pyAesCrypt.encryptFile(file, file+".aes", password)
    os.remove(file)

def DecryptFile(file, password):
    try:
        pyAesCrypt.decryptFile(file, file.split(".aes")[0], password)
        os.remove(file)
    except ValueError:
        print(f'Не вдалося розшифрувати файл {file}!')

def EncryptDir(directory, password) -> int:
    count = 0
    for dirpath, _dirnames, filenames in os.walk(directory, topdown=False):
        for name in filenames:
            EncryptFile(os.path.join(dirpath, name), password)
            count += 1
        time.sleep(0.01) # засипаємо на 10 мілісекунд
    return count

def DecryptDir(directory, password) -> int:
    count = 0
    for dirpath, _dirnames, filenames in os.walk(directory, topdown=False):
        for name in filenames:
            DecryptFile(os.path.join(dirpath, name), password)
            count += 1
    return count
```

6. Data_preperation.py

```
#!/usr/bin/python3
import os
import re
import glob
import numpy as np
```

```
import pandas as pd

# Період часу для обчислення частоти події
# 1e9 наносекунд = 1 секунда
TIME_PERIOD = 1e9

# Типи подій
TYPE_NAMES = ['O', 'C', 'D', 'E']

# Зміщення PID (щоб уникнути дублювання PID під час різних запусків детектора)
PID_OFFSET = 1e4

LOGDIR  = '../logs/'
DATADIR = '../data/'

def counts(df):
    # Групування за PID, TYPE і PERIOD
    ts = df['TS']
    df1 = df.assign(PERIOD=np.trunc((ts - ts[0]) / TIME_PERIOD))
    df1.drop(columns=['TS', 'FLAG', 'OPEN', 'CREATE', 'DELETE', 'ENCRYPT', 'FILENAME'], inplace=True)

    # Підрахунок кількості подій, згрупованих за типом, періодом і PID
    grouped = df1.groupby(['TYPE', 'PERIOD', 'PID']).agg(['count','sum']).unstack(level='TYPE', fill_value=0)

    # Агрегація за періодом часу
    aggregated = grouped.groupby(level='PID').agg(['max','sum'])

    # Перейменування стовпців
    aggregated.columns = aggregated.columns.to_flat_index()
    aggregated.rename(columns={col: '_'.join(col[1:]) for col in aggregated.columns}, inplace=True)

    # Сума паттернів подій
    pattern_max = re.compile("^sum_\w+_max$")
    pattern_sum = re.compile("^sum_\w+_sum$")
    pattern_max_cols = [col for col in aggregated.columns if pattern_max.match(col)]
    pattern_sum_cols = [col for col in aggregated.columns if pattern_sum.match(col)]
    aggregated['P_max'] = aggregated[pattern_max_cols].sum(axis=1)
    aggregated['P_sum'] = aggregated[pattern_sum_cols].sum(axis=1)
    aggregated.drop(columns=pattern_max_cols + pattern_sum_cols, inplace=True)

    # Прибирання префіксу "count_" із назв стовпців
    aggregated.rename(columns={col: col[6:] for col in aggregated.columns if col.startswith('count')}, inplace=True)

    return aggregated

def sequences(df):
    df1 = df.drop(columns=['FLAG', 'PATTERN', 'OPEN', 'CREATE', 'DELETE', 'ENCRYPT', 'FILENAME'])

    # Підрахунок послідовностей подій (довжина 3)
    df1['NEXT'] = df1.groupby(['PID'])['TYPE'].transform(lambda col: col.shift(-1, fill_value='X'))
    df1['AFTER'] = df1.groupby(['PID'])['TYPE'].transform(lambda col: col.shift(-2, fill_value='X'))
    df1['SEQUENCE'] = df1[['TYPE', 'NEXT', 'AFTER']].apply(lambda row: ''.join(row.values.astype(str)), axis=1)

    aggregated = df1.groupby(['PID', 'SEQUENCE'])['TS'].agg('count').unstack(level='SEQUENCE', fill_value=0)

    # Видалення фіктивних послідовностей (що містять X)
    aggregated.drop(columns=[col for col in aggregated.columns if 'X' in col], inplace=True)

    return aggregated
```

```
def main():
    # Обробка журналів у директоріях навчання та тестування
    for dir in ['training', 'testing']:
        logs = sorted(glob.glob(LOGDIR + dir + '/*.csv'))

        # Виправлення збігу PID
        df_arr = []
        for i,log in enumerate(logs):
            df = pd.read_csv(log)
            df['PID'] = df['PID'].map(lambda x: x + i * PID_OFFSET)
            df_arr.append(df)

        df = pd.concat(df_arr, ignore_index=True, verify_integrity=True)

        df['TYPE'].replace([0,1,2,3], TYPE_NAMES, inplace=True)

        c = counts(df)
        s = sequences(df)

        combined = pd.concat([c, s], axis=1)
        # Збереження у форматі csv
        combined.to_csv(DATADIR + dir + '_data.csv')

if __name__ == '__main__':
    main()
```

7. Svm_model.py

```
#!/usr/bin/python3

import argparse
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.inspection import permutation_importance
from sklearn.metrics import ConfusionMatrixDisplay, RocCurveDisplay
import matplotlib.pyplot as plt
from joblib import dump, load

# імена файлів
DATADIR = '../data/'
FILES = {
    'training': {
        'data':    DATADIR + 'training_data.csv',
        'labels':  DATADIR + 'training_labels.csv',
    },
    'testing': {
        'data':    DATADIR + 'testing_data.csv',
        'labels':  DATADIR + 'testing_labels.csv',
    },
    'model': {
        'features': DATADIR + 'features.joblib',
        'scaler':  DATADIR + 'scaler.joblib',
        'model':   DATADIR + 'model.joblib',
        'results': DATADIR + 'results.png',
```

```
        'analysis': DATADIR + 'analysis.png',
    },
}

# отримати мітки
def get_labels(df: pd.DataFrame, file: str):
    if file:
        pids = pd.read_csv(file)
        pids['y'] = 1
        return df.join(pids.set_index('PID'), on='PID').fillna(0)['y']
    else:
        return df['C_max'].map(lambda x: 1 if x > 100 else 0)

def best_features_linear(coef, feature_names):
    abs_coef = abs(*coef)
    importance, feature_names = zip(*sorted(zip(abs_coef, feature_names)))
    plt.barh(range(len(feature_names)), importance, align='center')
    plt.yticks(range(len(feature_names)), feature_names)
    plt.savefig(FILES['model']['analysis'])

def best_features_rbf(perm_importance, feature_names):
    features = np.array(feature_names)
    sorted_idx = perm_importance.importances_mean.argsort()
    plt.barh(features[sorted_idx], perm_importance.importances_mean[sorted_idx])
    plt.xlabel("Важливість перестановки")
    plt.savefig(FILES['model']['analysis'])

def refit_strategy(cv_results):
    # друкуємо результати
    df = pd.DataFrame(cv_results)
    df = df.sort_values(by=["rank_test_recall"])
    pd.set_option('display.max_colwidth', 100)
    print(df[["params", "rank_test_recall", "rank_test_balanced_accuracy", "mean_test_recall",
"mean_test_balanced_accuracy"]])

    # вибираємо найвище ранжоване
    return df["rank_test_recall"].idxmin()

def train(file: str):
    X_train = pd.read_csv(FILES['training']['data'])
    y_train = get_labels(X_train, file)

    # видаляємо стовпець PID, не використовуємо для навчання
    X_train.drop(columns=['PID'], inplace=True)

    # зберігаємо ознаки в наборі даних для навчання
    dump(X_train.columns, FILES['model']['features'])

    # масштабуємо дані для навчання
    scaler = StandardScaler().fit(X_train)
    dump(scaler, FILES['model']['scaler'])

    scaler.transform(X_train)

    # класифікатор SVM

    # пошук по сітці гіперпараметрів SVM
    kernel = ['rbf'] # 'rbf' або 'linear'
    class_weight = [{1: w} for w in np.linspace(10, 200, 10)]
```

```
    C = np.logspace(-1, 0, 10)
    param_grid = dict(class_weight=class_weight, kernel=kernel, C=C)
    scores = ['recall', 'balanced_accuracy']
    grid = GridSearchCV(svm.SVC(max_iter=1_000_000), param_grid=param_grid, scoring=scores, refit=refit_strategy)
    grid.fit(X_train, y_train)

    best_classifier = grid.best_estimator_

    dump(best_classifier, FILES['model']['model'])

    score = best_classifier.score(X_train, y_train)
    print("Результат навчання: %f" % score)

    feature_names = X_train.columns.to_list()
    if kernel[0] == 'linear':
        best_features_linear(best_classifier.coef_, feature_names)
    elif kernel[0] == 'rbf':
        best_features_rbf(permutation_importance(best_classifier, X_train, y_train), feature_names)

def test(file: str):
    X_test = pd.read_csv(FILES['testing']['data'])
    y_test = get_labels(X_test, file)

    # видаляємо стовпець PID, не використовуємо для навчання
    X_test.drop(columns=['PID'], inplace=True)

    # використовуємо ті ж самі ознаки, що й для навчання
    training_features = load(FILES['model']['features'])
    test_features = X_test.columns
    # видаляємо нові ознаки
    X_test.drop(columns=[f for f in test_features if f not in training_features], inplace=True)
    # додаємо відсутні ознаки
    for f in training_features:
        if f not in test_features:
            X_test[f] = 0

    # масштабуємо тестові дані
    scaler = load(FILES['model']['scaler'])
    scaler.transform(X_test)

    # прогноз з попередньо натренованим класифікатором
    classifier = load(FILES['model']['model'])

    score = classifier.score(X_test, y_test)
    print("Результат тестування: %f" % score)

    # матриця плутанини
    cm_display = ConfusionMatrixDisplay.from_estimator(classifier, X_test, y_test)

    # ROC крива -> слід використовувати криву точності-повернення
    roc_display = RocCurveDisplay.from_estimator(classifier, X_test, y_test)
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
    cm_display.plot(ax=ax1)
    roc_display.plot(ax=ax2)
    plt.savefig(FILES['model']['results'])

def main():
    parser = argparse.ArgumentParser(description="Модель для навчання і тестування")
    parser.add_argument("mode", choices=["train", "test"], help="режим роботи: навчання або тестування")
```

```
    parser.add_argument("--file", help="файл міток для навчання чи тестування")
    args = parser.parse_args()

    if args.mode == "train":
        train(args.file)
    elif args.mode == "test":
        test(args.file)

if __name__ == "__main__":
    main()
```

3. Random_forest.py

```
import pandas as pd

import argparse
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.inspection import permutation_importance
from sklearn.metrics import ConfusionMatrixDisplay, RocCurveDisplay
import matplotlib.pyplot as plt
from joblib import dump, load
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, classification_report

from sklearn.preprocessing import StandardScaler


# імена файлів
DATADIR = '../data/'
FILES = {
    'training': {
        'data':     DATADIR + 'training_data.csv',
        'labels':   DATADIR + 'training_labels.csv',
    },
    'testing': {
        'data':     DATADIR + 'testing_data.csv',
        'labels':   DATADIR + 'testing_labels.csv',
    },
    'model': {
        'features': DATADIR + 'features.joblib',
        'scaler':   DATADIR + 'scaler.joblib',
        'model':    DATADIR + 'model.joblib',
        'results':  DATADIR + 'results.png',
        'analysis': DATADIR + 'analysis.png',
    },
}


# Нормалізація даних

scaler = StandardScaler()
```

```
features = scaler.fit_transform(features)
# Розділити дані на навчальний та тестовий набори
features_train, features_test, labels_train, labels_test = train_test_split(features, labels, test_size=0.2, random_state=42)
# Створення моделі
tree_model = DecisionTreeClassifier(random_state=42)
# Навчання моделі
tree_model.fit(features_train, labels_train)
# Перевірка моделі
predicted_labels = tree_model.predict(features_test)
# Виведення результатів
print("Accuracy:", accuracy_score(labels_test, predicted_labels))
print("\nClassification Report:\n", classification_report(labels_test, predicted_labels))
```

ДОДАТОК В. Список публікацій здобувача за темою дисертації та відомості про апробацію результатів дисертації

Статті у наукових фахових виданнях України:

1. Zhuravchak, D. “СТВОРЕННЯ СИСТЕМИ ЗАПОБІГАННЯ ПОШИРЕННЯ ВІРУСІВ ВИМАГАЧІВ ЗА ДОПОМОГОЮ МОВИ ПРОГРАМУВАННЯ PYTHON ТА УТИЛІТИ AUDITD НА БАЗІ ОПЕРАЦІЙНОЇ СИСТЕМИ LINUX”. Електронне фахове наукове видання “Кібербезпека: освіта, наука, техніка”, вип. 4, вип. 12, Червень 2021, с. 108-16, doi:10.28925/2663-4023.2021.12.108116. *Особистий внесок здобувача: представлено метод виявлення вірусів-вимагачів а допомогою створення системи програмних-приманок на базі утиліти auditD та створення модуля моніторингу за допомогою мови програмування python.*

2. Zhuravchak Danyil, Opanovych Maksym, Dudykevych Valerii, Piskozub Andrian, (2022). Detection Method Of Credential Dumping Method Through Exploiting Vulnerable Windows Error Reporting Service In Windows Operating Systems. Сучасна спеціальна техніка, 2 (69), 38-52. https://doi.org/10.36486/mst2411–3816.2022.2(69).2.*Особистий внесок здобувача: проведено аналіз вразливостей операційної системи Windows, що експлуатуються програмами-вимагачами.*

3. Zhuravchak, D. ., V. Dudykevych, і A. Tolkachova. “ДОСЛІДЖЕННЯ СТРУКТУРИ СИСТЕМИ ВИЯВЛЕННЯ ТА ПРОТИДІЇ АТАКАМ ВІРУСІВ-ВИМАГАЧІВ НА БАЗІ ENDPOINT DETECTION AND RESPONSE”. Електронне фахове наукове видання “Кібербезпека: освіта, наука, техніка”, вип. 3, вип. 19, Березень 2023, с. 69-82, doi:10.28925/2663-4023.2023.19.6982. *Особистий внесок здобувача: проведено аналіз систем забезпечення безпеки endpoint detection and response, класифікація функціональних та нефункціональних властивостей задля створення такої системи у контексті виявлення програм-вимагачів у режимі реального часу.*

4. Піскозуб, А. З., Журавчак, Д. Ю., і Толкачова, А. Ю. "Дослідження Вразливостей у Чатботах з Використанням Великих Мовних Моделей." Безпека Інформації, том 29, № 3, 2023, с. 111–117. *Особистий внесок здобувача: проведено аналіз моделей машинного навчання, їхніх переваг та недоліків, а також аналіз вразливостей.*

5. Журавчак, Д., П. Глущенко, М. Опанович, В. Дудикевич, і А. Піскозуб. “КОНЦЕПЦІЯ НУЛЬОВОЇ ДОВІРИ ДЛЯ ЗАХИСТУ ACTIVE DIRECTORY ДЛЯ ВИЯВЛЕННЯ ПРОГРАМ-ВИМАГАЧІВ”. Електронне фахове наукове видання “Кібербезпека: освіта, наука,

техніка”, вип. 2, вип. 22, Грудень 2023, с. 179-90, doi:10.28925/2663-4023.2023.22.179190. *Особистий внесок здобувача: проведено аналіз використання концепції нульової довіри для виявлення вірусів-вимагачів у середовищі windows active directory.*

6. Журавчак, Даниїл, Едуард Кійко, і Валерій Дудикевич. "Використання EBPF для Ідентифікації Вірусів-Вимагачів, що Використовують DNS-Запити DGA." Information Technology and Security, vol. 11, no. 2 (21), 2023, pp. 166–174. Особистий внесок здобувача: представлено розробку модуля опрацювання dns трафіку для виявлення активності вірусів-вимагачів.

7. Журавчак, Д. Ю. "Моніторинг Вірусів-Вимагачів за Допомогою Розширеного Берклійського Пакетного Фільтра (eBPF) та Машинного Навчання." Наукоємні Технології, том 60, № 4, 2023, с. 352–363. Особистий внесок здобувача: представлено модуль моніторингу мережі та моделей машинного навчання для виявлення вірусів-вимагачів.

Статті у наукових періодичних виданнях інших держав, що включені до міжнародної наукометричної бази даних (Scopus):

8. Zhuravchak, D., Tolkachova, A., Piskozub, A., Dudykevych, V., і Korshun, N. "Monitoring Ransomware with Berkeley Packet Filter." CEUR Workshop Proceedings, vol. 3550, Cybersecurity Providing in Information and Telecommunication Systems II 2023. Proceedings of the Cybersecurity Providing in Information and Telecommunication Systems II co-located with the International Conference on Problems of Infocommunications. Science and Technology (PICST 2023), Kyiv, Ukraine, October 26, 2023 (online), 2023, pp. 95–106. *Особистий внесок здобувача: представлено інтегровану систему виявлення вірусів-вимагачів за допомогою модулів моніторингу активності на операційній системі.*

Наукові публікації у збірниках матеріалів та тез конференцій:

9. Zhuravchak, D., Ustyianovych, T., Dudykevych, V., Venny, B., і Ruda, K. "Ransomware Prevention System Design Based on File Symbolic Linking Honeypots." 2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Cracow, Poland, 2021, pp. 284-287. IEEE. *Особистий внесок здобувача: представлено систему на базі програмних-приманок для виявлення вірусів-вимагачів.*

10. Журавчак Д.Ю. , Опанович М. Ю. Аналіз атак вірусів-шифрувальників на системи типу Active Directory за допомогою кореляції закономірностей в кіберзлочинах та аномальної активності // “Технічні засоби захисту інформації”, семінар при вченій раді НАН України, Київ, Україна, 2022 р.*Особистий внесок здобувача: представлено систему виявлення закономірностей у кіберзлочинах типу вірусів-вимагачів.*

11. Журавчак Д.Ю, проф. Дудикевич В.Б. Аналіз методів Threat Hunting для проактивного виявлення програм-вимагачів // “Технічні засоби захисту інформації”, семінар при вченій раді НАН України, Київ, Україна, 2023 р. *Особистий внесок здобувача: представлено метод проактивного виявлення програм-вимагачів за допомогою Threat Hunting.*

12. Журавчак Д.Ю, проф. Дудикевич В.Б. Аналіз мережевого трафіку з використанням eBPF/XDP для ефективного виявлення програм-вимагачів // “Технічні засоби захисту інформації”, семінар при вченій раді НАН України, Київ, Україна, 2024 р.*Особистий внесок здобувача: представлено систему моніторингу мережевого трафіку з метою виявлення вірусів-вимагачів на базі eBPF/XDP.*

13. Журавчак Д. Ю., Дудикевич В. Б., Опанович М. Ю., Піскозуб А. З. СТВОРЕННЯ СИСТЕМИ БЕЗПЕРЕРВНОГО РЕАГУВАННЯ НА ІНЦИДЕНТИ ІНФІКУВАННЯ ВІРУСАМИ-ВИМАГАЧАМИ В ACTIVE DIRECTORY // Збірник тез доповідей підготовлено за матеріалами Міжнародної наукової інтернет-конференції (випуск 70) 22-23 вересня 2022 р. на сайті www. konferenciaonline. org. ua. - 2022. - С. 25. *Особистий внесок здобувача: представлено систему безперервного реагування на інциденти кіберзлочинів типу вірусів-вимагачів.*

14. Журавчак, Даниїл, і Дудикевич Валерій. "Виклики Та Перспективи Впровадження Машинного Навчання Для Виявлення Програм-Вимагачів В Режимі Реального Часу." Захист Інформації І Безпека Інформаційних Систем: Матеріали IX Міжнародної Науково-Технічної Конференції, Львів, 25–26 травня 2023 р., 2023, с. 141–143. *Особистий внесок здобувача: представлено аналіз впровадження математичного апарату на базі моделей машинного навчання для виявлення програм-вимагачів.*

15. Журавчак, Даниїл, і Валерій Дудакевич. "Виклики Та Перспективи Впровадження Машинного Навчання Для Виявлення Програм-Вимагачів В Режимі Реального Часу." Захист Інформації І Безпека Інформаційних Систем: Матеріали IX Міжнародної Науково-Технічної Конференції, Львів, 25–26 травня 2023 р., 2023, с. 141–143.