НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»
УДК 004.056

«До захисту допущено»
Завідувач кафедри
Стіренко С.Г.
(підпис) (ініціали, прізвище)
“ ___ ” _____2020 р.

Магістерська дисертація

зі спеціальності: *123. Комп’ютерна інженерія*
(код та назва напряму підготовки або спеціальності)

Спеціалізація: *123. Комп’ютерні системи та мережі*

на тему: *«Об’єктно-орієнтований спосіб підвищення безпеки обчислювального вузла на базі підсистеми аудиту операційної системи»*

Виконав (-ла): студент (-ка) *2* курсу, групи *ІО-81мн*
(шифр групи)

Вернер Анна Ігорівна
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник *проф., д.т.н., проф. Сімоненко В. П.*
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант *нормоконтроль, професор, д.т.н., Кулаков Ю. О.*
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент *доц. каф. СКС, к.т.н., Орлова М. М.*
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність *123. Комп'ютерна інженерія*
(код і назва)

Спеціалізація *123. Комп'ютерні системи та мережі*
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Стіренко С.Г.
(підпис) (ініціали, прізвище)
" _____ " _____ *2020* р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Вернер Анні Ігорівні
(прізвище, ім'я, по батькові)

1. Тема дисертації *«Об'єктно-орієнтований спосіб підвищення безпеки обчислювального вузла на базі підсистеми аудиту операційної системи»*

Науковий керівник дисертації *Сімоненко В.П, проф., д.т.н.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «*24*» *березня 2020* р. № *910-с*

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження *Системи безпеки комп'ютерних вузлів*

4. Предмет дослідження *Підсистема аудиту операційних систем як основа засобів захисту інформації (ЗЗІ)*

5. Перелік завдань, які потрібно розробити *дослідження принципів роботи сучасних засобів захисту, аналіз структури підсистем аудиту як компонента засобів захисту, розробка алгоритмів аналізатора, моделювання роботи способу, аналіз отриманих результатів*

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	*Кулаков Ю.О., професор, д. т. н.*		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	*Аналіз літератури*	*16.11.2019*	*Викон.*
2	*Огляд існуючих рішень*	*23.12.2019*	*Викон.*
3	*Аналіз теоретичних методів*	*20.01.2020*	*Викон.*
4	*Проектування способу*	*17.02.2020*	*Викон.*
5	*Моделювання роботи способу*	*23.03.2020*	*Викон.*
6	*Оформлення документації*	*13.04.2020*	*Викон.*

Студент _____ *Вернер А. І.*
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ *Сімоненко В. П.*
(підпис) (ініціали, прізвище)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Об'єктно-орієнтований спосіб підвищення безпеки обчислювального вузла на базі підсистеми аудиту операційної системи

студенткою: Вернер Анною Ігорівною

Магістерська дисертація. 77 сторінок, 33 рисунки, 5 таблиць, 64 джерела.

Актуальність. Недосконалість сучасних засобів захисту зумовлює необхідність пошуку нових рішень для підвищення безпеки інформаційних систем.

Мета роботи. Підвищення безпеки обчислювальних вузлів шляхом застосування систем захисту, реалізованих з використанням принципів, закладених у запропонованому об'єктно-орієнтованому способі.

Задачі дослідження. Дослідження принципів роботи сучасних засобів захисту, аналіз структури підсистем аудиту як базового компонента засобів захисту, розробка алгоритмів аналізатора, моделювання роботи способу, аналіз отриманих результатів.

Об'єкт дослідження. Системи безпеки комп'ютерних вузлів

Предмет дослідження. Підсистема аудиту операційних систем як основа засобів захисту інформації (ЗЗІ).

Наукова новизна одержаних результатів. Новизна полягає у застосуванні підсистеми аудиту у якості механізму з перехоплення та контролю подій у системі, що може бути застосовано для розробки динамічного засобу захисту інформаційних систем.

Практичне значення одержаних результатів. Запропонований спосіб підвищення безпеки обчислювальних вузлів може бути застосований під час розробки сучасних систем захисту інформації.

Апробація результатів дисертації. Участь у міжнародній науковій конференції *«Наукові дослідження: парадигма інноваційного розвитку»* (Братислава - Відень), «25» березня 2020 року. Участь у міжнародній науково-технічній конференції *«The International Conference on Security, Fault Tolerance, Intelligence»* (ICSFTI2020), 13-14 травня 2020 року.

Публікації. Стаття у науковому фаховому виданні *«Інформаційний вісник»* випуск №66 *«Динамічно-аналізуюча програмна компонента реагування на події операційного вузла»*. Стаття у міжнародному науковому мультидисциплінарному журналі *«The scientific heritage»* випуск №20 том 1 *«Computer unit's security level increment using the audit mechanism of the Operating system»*. Тези до міжнародної наукової конференції *«Наукові дослідження: парадигма інноваційного розвитку» (Братислава - Відень),* «25» березня 2020 року *«Об'єктно-орієнтований спосіб підвищення безпеки обчислювальних вузлів на основі підсистеми аудиту»*. Тези до міжнародної науково-технічної конференції *«The International Conference on Security, Fault Tolerance, Intelligence» (ICSFTI2020)*, 13-14 травня *«Object-Oriented Method for Enhancing Computing Node Security by Means of Operating System Audit Subsystem»*.

Ключові слова: *засоби захисту інформації, інформаційна безпека, нейронні мережі, динамічні аналіз, аудит, операційні системи.*

ABSTRACT

of the master's thesis

«Object-Oriented Method for Enhancing Computing Node Security by Means of Operating System Audit Subsystem»

done by Anna Verner

Master's thesis. 77 pages, 33 figures, 5 tables, 64 sources.

Relevance. The imperfection of modern means of protection causes necessity of search for new decisions for the increase of information systems' safety.

Purpose of the paper. Increase of safety of computing knots by application of the protection systems realised with the use of the principles put in the offered object-oriented way.

Research tasks. Study of work principles of modern protection frames, the analysis of the structure of subsystems of audit as a basic component of protection frames, development of algorithms of the analyser, modelling of work of a method, the analysis of the received results.

The object of research. Security systems of computer nodes.

The subject of study. The audit subsystem of operating systems as a basis of information protection means (IPM).

The scientific novelty of the obtained results. The novelty consists in the application of audit subsystem as the mechanism on interception and control of events in a system that can be applied to working out a dynamic protection frame of information systems.

The practical value of the received results. The offered way of increase of computing knots safety can be applied during developing of modern systems of information protection.

Approbation of results of the thesis. Participation in the International Scienific Conference «*Scientific research: the paradigm of innovative development»* (Bratislava - Vienna), 25 March 2020. Participation in the of the International Scientific and Technical Conference *«The International Conference on Security, Fault Tolerance, Intelligence» (ICSFTI2020)*, 13-14 May 2020.

Publications. Article in the scientific professional journal *«Informational Bulletin»* issue No. 66 *«Dynamic and analytical software component of the response to the events of the operational node»*. Article in the international scientific multidisciplinary journal *«The scientific heritage»* issue No. 20 volume 1 *«Computer unit's security level increment using the audit mechanism of the operating system»*. Theses of the international scientific conference *«Scientific research: Paradigm of innovative development»* (Bratislava - Vienna), 25 March 2020 *«Object-oriented way to increase the computing nodes security based on the audit subsystem»*. Theses of the International Scientific and Technical Conference *«The International Conference on Security, Fault Tolerance, Intelligence» (ICSFTI2020)*, 13-14 May 2020 «*Object-Oriented Method for Enhancing Computing Node Security by Means of Operating System Audit Subsystem».*

Keywords: *information protection means, information security, neural networks, dynamic analysis, audit, operating systems.*

ЗМІСТ

ВСТУП 4

РОЗДІЛ 1. ДОСЛІДЖЕННЯ СУЧАСНИХ ЗАСОБІВ ПІДВИЩЕННЯ БЕЗПЕКИ ОБЧИСЛЮВАЛЬНИХ ВУЗЛІВ 6

1.1. Огляд стану інформаційної безпеки 6

1.2. Шкідливе програмне забезпечення. Класифікація та особливості 7

1.3. Способи виявлення загроз 12

1.4. Підходи до виявлення загроз на обчислювальних вузлах 15

1.5. Аудит операційних систем та його застосування для захисту обчислювальних вузлів 23

Висновки до розділу 1 25

РОЗДІЛ 2. АНАЛІЗ ПІДСИСТЕМИ АУДИТУ ОПЕРАЦІЙНОЇ СИСТЕМИ НА БАЗІ ЯДРА LINUX ЯК ОСНОВИ ЗЗІ 26

2.1. Реалізація підсистеми аудиту на базі операційних систем 26

2.2. Компоненти підсистеми аудиту на базі ядра Linux 28

2.3. Принцип роботи підсистеми аудиту 31

2.3.1. Архітектура підсистеми аудиту 31

2.3.2. Життєвий цикл підсистеми аудиту 31

2.4. Налаштування підсистеми аудиту та створення правил 37

2.4.1. Правила контролю 44

2.4.2. Правилами файлової системи 45

2.4.3. Робота з правилами системних викликів 46

2.5. Структура записів журналу аудиту 49

Висновки до розділу 2 53

РОЗДІЛ 3. СПОСІБ ПІДВИЩЕННЯ БЕЗПЕКИ ОБЧИСЛЮВАЛЬНОГО ВУЗЛА НА БАЗІ ПІДСИСТЕМИ АУДИТУ ОПЕРАЦІЙНОЇ СИСТЕМИ 54

3.1. Основні способи впливу на безпеку обчислювальних вузлів 54

3.2. Принцип роботи способу підвищення безпеки 55

3.2.1. Структура способу 55
3.2.2. Опис алгоритму роботи способу 59
3.2.3. Розробка механізму аналізатора 62
3.3. Моделювання способу підвищення безпеки обчислювального вузла на базі механізму аудиту 68
3.3.1. Підготовка наборів даних 69
3.3.2. Розробка програмної моделі аналізатора 70
3.3.3. Тестування роботи способу 72
Висновки до розділу 3 75
ВИСНОВКИ 76
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 78

СПИСОК СКОРОЧЕНЬ

БПЗ – безпечне програмне забезпечення

ЗЗІ - засіб захисту інформації

IoT - “Інтернет речей”

ІС – інформаційна система

ОВ - обчислювальний вузол

ОС - операційна система

ШПЗ – шкідливе програмне забезпечення

ВСТУП

Стрімкий розвиток технологій та обчислювальної техніки майже повністю змінив усі сфери сьогодення. Шляхом суцільної інтеграції інформаційних технологій досягається значне підвищення продуктивності та вдосконалення діяльності людства. Проте, внаслідок проведення глобальної цифровізації, значно збільшується та урізноманітнюється кількість небезпек, що впливають на цілісність інформаційних систем. Це зумовлює необхідність посилення захисту кінцевих вузлів та винайдення нових адаптивних засобів безпеки.

Серед засобів захисту інформації (ЗЗІ) найбільшого поширення набули: апаратні, програмно-апаратні та програмні засоби. Останні є найбільш універсальними та гнучкими, оскільки дозволяють виконувати налаштування під будь-які типи систем та архітектур, що є актуальним у зв'язку з активним поширенням «інтернету речей» (IoT). Саме цей сектор пристроїв є одним з найбільш уразливих та все частіше застосовується для проведення цільових атак [49]. Проте, лідерами з кількості розроблюваного шкідливо програмного забезпечення досі залишаються настільні персональні комп’ютери.

У роботі пропонується спосіб підвищення безпеки ОВ, заснований на застосуванні засобів аудиту операційної системи. Основне призначення аудиту - здійснення моніторингу подій періодично або у режимі реального часу, тобто під час роботи системи. Даний компонент присутній у більшості сучасних операційних систем (ОС). Останні наукові роботи [6], [60] демонструють ефективність застосування даних засобів з метою проведення дослідження щодо наявності стороннього впливу на інформаційну систему. Проте, такий аналіз з застосуванням аудиту здійснюється вже після реалізації ряду шкідливий дій відносно ОВ, маючи суто діагностичний характер. На противагу, розроблений спосіб виявлення загроз безпеці вузла використовує широкий спектр можливостей, наданих системою аудиту, шляхом застосування інтегрованої компоненти ядра для контролю процесів, що відбуваються у системі, дозволяючи запобігти виконанню шкідливого програмного забезпечення.

Даний спосіб не потребує додаткової розробки системи керування викликами, оскільки базується на використанні нативних засобів, які входять до базових модулів ядра. Завдяки проведенню аналізу в режимі реального часу надається можливість попередити небезпеку та вберегти дані у стабільному стані.

РОЗДІЛ 1

ДОСЛІДЖЕННЯ СУЧАСНИХ ЗАСОБІВ ПІДВИЩЕННЯ БЕЗПЕКИ ОБЧИСЛЮВАЛЬНИХ ВУЗЛІВ

1.1. Огляд стану інформаційної безпеки

В сучасному світі привалюють високі темпи розвитку та поширення технологій. Разом з тим, відповідно до статистичних даних [56], спостерігається тенденція щорічного експоненційного зростання кількості шкідливого програмного забезпечення, впливам якого піддаються інформаційні системи. Це зумовлює необхідність винайдення спектру нових засобів захисту, що мають адаптивний характер. Та, не зважаючи на сумісні зусилля спеціалістів, питання аналізу та виявлення шкідливого програмного забезпечення (ШПЗ) залишається відкритим.

Статистика використання операційних систем (ОС) [43], станом на початок 2020 року, свідчить про те, що найбільш поширеною серед користувачів настільних комп'ютерів незмінно залишається комерційна ОС Windows. В щорічному звіті щодо стану шкідливого програмного забезпечення [53] зазначена загальна кількість виявлених загроз в підсумку за 2019 рік, що сягнула 50510960, та зросла, в порівнянні з попереднім роком, на один відсоток, що свідчить про продовження загострення ситуації.

У зв'язку зі швидким ростом популярності вбудованих систем, головним чином використовуваних у так званому "Інтернеті речей" (IoT), можна спостерігати досить помітні зміни ландшафту шкідливого програмного забезпечення. Через високі темпи виходу продукції, з метою зайняти найбільшу нішу на ринку, компанії приділяють недостатньо уваги питанню безпеки їх продукції, через що досить часто спостерігається наявність великої кількості критичних вразливостей у таких системах.

Архітектурно вбудовані системи мають сильні відмінності від настільних персональних комп'ютерів, що зумовлено, в перу чергу, застосуванням різноманітних процесорів та досить обмежених ресурсів. З точки зору

використання операційних систем тут, безумовно, ситуація також є кардинально відмінною, оскільки за даними [55] розробники застосовують Unix-подібні ОС з різноманітними варіаціями ядра Linux. Відповідно до свідчень [34], майже половина розумних будинків з вбудованими системами мали в наявності критичні вразливості, що дозволяли зловмисникам їх легко атакувати. Окрім того, за цього даними джерела спостерігається зростання на 217% кількості атак у порівнянні з минулими роками. Найбільш активно застосовуються вразливості типу backdoor [35].

Аналізуючи приведені вище дані можна зробити висновок, що розробка засобів захисту для підвищення безпеки обчислюваних вузлів на базі ядра Linux є вкрай необхідною та, безперечно, актуальною проблемою в наш час.

1.2. Шкідливе програмне забезпечення. Класифікація та особливості

Під шкідливим програмним забезпеченням розуміють будь-яке програмне забезпечення, що може завдати шкоди цілісності інформаційної системи [41].

Через наявність великої кількості шкідливого програмного забезпечення та величезного спектру програм, кожного з видів ШПЗ, можна провести його однозначний розподіли за класами. ШПЗ включає: віруси, хробаків, троянських коней, руткіти, шпигунські програми, кейлоггери тощо. Відповідно до звіту [53], найбільш часто у 2019 році було виявлено рекламні віруси (близько 20000 виявлень у грудні 2019 року). Графік виявленого шкідливого програмного забезпечення за рік зображено на рис. 1.1.

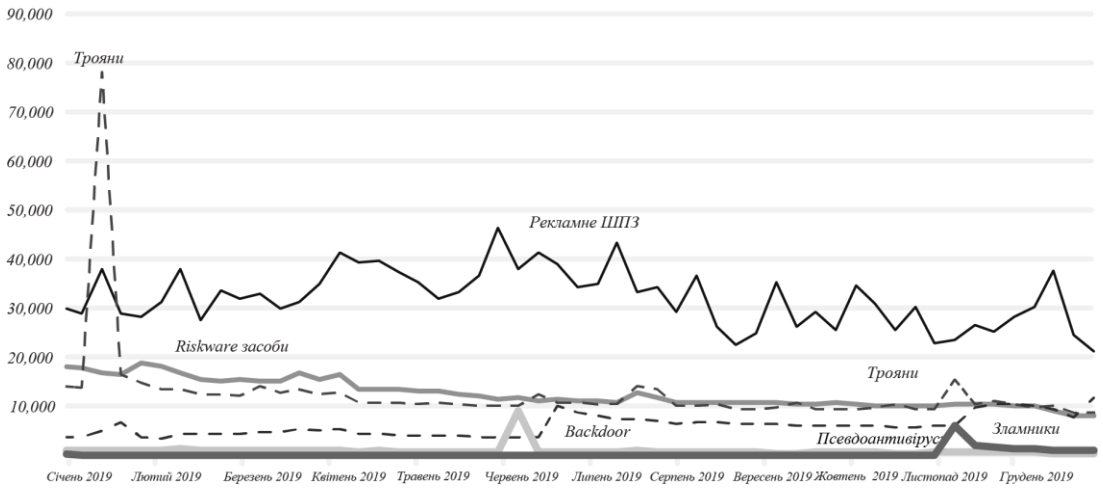


Рис. 1.1. *Топ-10 категорій загроз для споживачів у 2019 році* [53]

Розкриємо детальніше особливості роботи кожного з видів шкідливого програмного забезпечення.

Віруси є шкідливим програмним забезпеченням з механізмом самовідтворення (реплікації). Існують у вигляді виконуваного файлу та поширюються, копіюючи себе до інших хост-системи. Оскільки це пасивний вид ПЗ, зараження відбувається через файли, медіа або мережеві файли. Залежно від того, наскільки складний код програми, він навіть може змінювати свої репліковані копії [36]. Віруси можуть використовуватися не тільки для заподіяння шкоди обчислювальним вузлам та мережам, але і для крадіжки інформації, створення ботнетів, виведення реклами, тощо.

Хробаки також мають механізм реплікації, проте є активним шкідливим програмним забезпеченням, що поширюється мережею, використовуючи різні вразливості в операційній системі або встановленому програмному забезпеченні. Вони містить шкідливі процедури, що можуть також використовуватися для відкриття каналів зв'язку та слугують активними носіями. Цей клас значно понижую продуктивність роботи системи, оскільки виконує постійне сканування ресурсів [40], що призводить до нестабільності роботи вузла та у крайніх випадках призводить до краху системи. Також хробаки можуть містити корисне навантаження у вигляді представляє собою фрагментів коду, написаних для впливу на обчислювальний вузол шляхом крадіжки даних, видалення файлів або створення бота, який може призвести до зараженої системи частиною ботнету [36]. На відміну від вірусів, хробаки не потребують людської діяльності для поширення, оскільки мають здатність поширюватися та розмножуватися незалежно.

Троянський кінь - це програма, що являє собою законне програмне забезпечення, що під час завантаження та виконання виконує вбудовану шкідливу програму або файли. У якості корисного навантаження троянський кінь може встановлювати додаткове ШПЗ у вигляді вірусів, як у більшості випадків, або може не мати корисного навантаження. На відміну від вірусів та хробаків, трояни не мають механізму самостійної реплікації та активуються

виключно після запуску користувачами. Проте, у якості навантаження може бути код, що надаватиме віддалений доступ до обчислювального вузла зловмиснику, який потім може вчиняти будь-які шкідливі дії. Програми троянських коней мають різний вплив на ОВ, що залежить від доданого корисного навантаження та зазвичай поширюються за допомогою соціальної інженерії [36].

Шпигунське програмне забезпечення - це шкідлива програма, що використовує функції операційної системи для проведення шпіонажу за діяльністю користувача. Інколи таке програмне забезпечення має додаткові можливості, як, наприклад, перешкоджання мережевим підключенням чи навіть зміна параметрів безпеки інфікованої системи. Поширення відбувається шляхом приєднання до законного програмного забезпечення, троянських коней або навіть завдяки відомим вразливим програмам. Шпигунське програмне забезпечення може контролювати поведінку користувача, наприклад, збираючи натискання клавіш та надсилати інформацію на віддалений хост зловмиснику [61].

Adware, що є коротким терміном для ***рекламного програмного забезпечення***, виконує автоматичну демонстрацію реклами у вигляді спливаючих оголошень на веб-сайтах тощо. Більшість такого програмного забезпечення створена для створення рекламодавцями інструментів з метою отримання прибутку. Деякі рекламні програми можуть поставлятися в комплекті зі шпигунським програмним забезпеченням, що згодом може призводити до небезпечних наслідків у вигляді відстеження активності користувачів та викрадення інформації [29].

Руткіти використовують набір інструментів, для уникнення виявлення в системі. У якості інструментів вступають, зазвичай, передові та складні програми, створені для приховування звичайних процесів на інфікованому вузлі. Через свою непомітність вони дуже інвазивні та їх важко видалити. Розробляються з можливістю повного контролю над системою та отримання найвищих привілеїв на інфікованому вузлі [19]. Через методи маскування , що використовуються руткітами, більшість частина рішень програмного

забезпечення для захисту вузлів не є ефективними для їхнього виявлення та видалення. Засоби боротьби з ними супроводжуються моніторингом поведінки комп'ютерної системи щодо предмету незвичних дій, аналізу дампів-пам’яті та сканування підписів системних файлів.

Ботами називають програми, призначені для виконання конкретних операцій. Боти вперше були розроблені для управління каналами чату. Деякі з них використовуються в законних цілях, проте шкідливі боти призначені для формування ботнетів. Ботнет - мережа вузлових комп'ютерів (зомбі / бот), якими керує зловмисник або ботмайстер. Боти заражають і контролюють інший комп'ютер, який, в свою чергу, заражає інші підключені комп’ютери, формуючи мережу компрометованих комп'ютерів під назвою ботнет. Боти часто застосовують як спам-боти, для DDOS-атак, веб-розповсюджувачів для поширення зловмисних програм на файлообмінниках, тощо. Для боротьби застосовують тести CAPTCHA [19].

Програми-вимагачі (Ransomware) заражають обчислювальні вузли чи мережу та утримують систему в замкнутому стані, вимагаючи викупу у користувачів. Зазвичай, відбувається шифрування файлів в зараженій системі або блокування системи, з метою заборони доступу користувачеві. Потім відображаються повідомлення, які змушують проводити оплату для отримання доступу до даних. Використовує ті ж засоби розповсюдження, що і комп'ютерні хробаки.

Кейлоггери - це певний тип шпигунських програм. Відбувається невидимий для користувача запис клавіш, що він натискає на клавіатурі. Ці дані потім передаються зловмиснику через Інтернет. Ці програми застосовуються для крадіжки паролів, наприклад, у паролі онлайн-банкінгу. Вони також можуть використовувати шпигунські програми для викрадення іншої особистої інформації, наприклад, документів, що зберігаються на обчислювальному вузлі. Використовується досить широкий спектр прийомів, для завантажити шпигунських програм та реєстраторів клавіш на віддалений вузол. Найбільш поширеним є перехід за посиланням у спамі, що приходить на електронну пошту,

або шляхом відвідування веб-сторінок, створених виключно для зараження вузлів. Інколи, за такого поширення, ще застосовується термінологія троянського програмного забезпечення

Бекдор є таємним “входом” для отримання доступу. Іноді він заплановано створюється розробниками системи або постачальниками послуг у якості віддалений засобу для діагностики, усунення несправностей або інших системних тестів. Просте існування бекдорду - це величезний ризик для безпеки, оскільки його виявлення не складає великих труднощів . На базі безпечних бекдорів досить часто виникають зараження, наприклад, вірусом "backdoor". Це шкідливий код, що, використовуючи недоліки та вразливості системи, застосовується для полегшення віддаленого несанкціонованого доступу до комп'ютерної системи чи програми. Як і весь шкідливий код, він працює у фоновому режимі. Цей доступ надає повноту дій для проведення шкідливих операцій в системі. Обчислювальні вузли є дуже вразливими до незаконного копіювання файлів, модифікацій, викрадення даних шляхом застосування бекдорів [58].

Існують й інші різновиди шкідливого програмного забезпечення, проте ці є найбільш популярними на найчастіше зустрічаються сьогодні. Підсумовуючи звіт останніх десятьох років[5], можна зазначити тенденцію до спадання кількості створеного нового ШПЗ та швидкого нарощування загальної кількості ШПЗ. Найкраще це можна продемонструвати за допомогою графіків, представлених на малюнку 1.2.

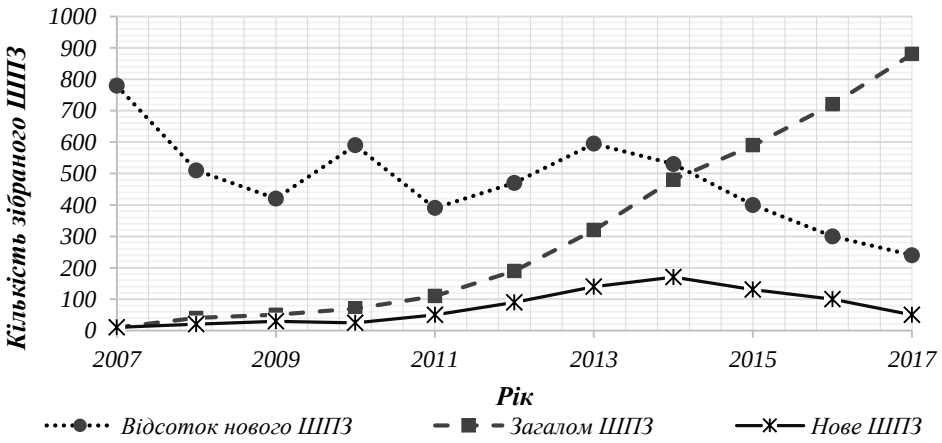


Рис. 1.2. *Кількість зібраного ШПЗ в проміжку 2007-2018 рр* [5]

Аналізуючи приведений графік можна дійти висновку, що у випадку винайдення способів для виявлення та запобігання більшості типів існуючих загроз, можливим є значне скорочення загально числа шкідливого програмного забезпечення , що на даний момент невпинно зростає.

1.3. Способи виявлення загроз

Для виявлення шкідливого програмного забезпечення застосовують різноманітні методи аналізу коду. Одним з них є статичний аналіз. Під статичний аналізом розуміють проведення аналізу шкідливого програмного забезпечення без його виконання. Шаблони виявлення, що використовуються під час статичному аналізу, включають сигнатури рядків, байт-послідовності, n-грами, синтаксичні виклики бібліотек, графи потоку управління, аналіз розподілу за частотою операційних кодів, тощо. Аналіз відбувається завдяки проведенню попереднього розпакування та декодування виконуваного файлу. Інструменти для дизасембляції та налагодження, аналізу дампу пам’яті застосовуються для зворотного перегляду принципу роботи ШПЗ. Процес реінжинірингу зображено на рис. 1.3.

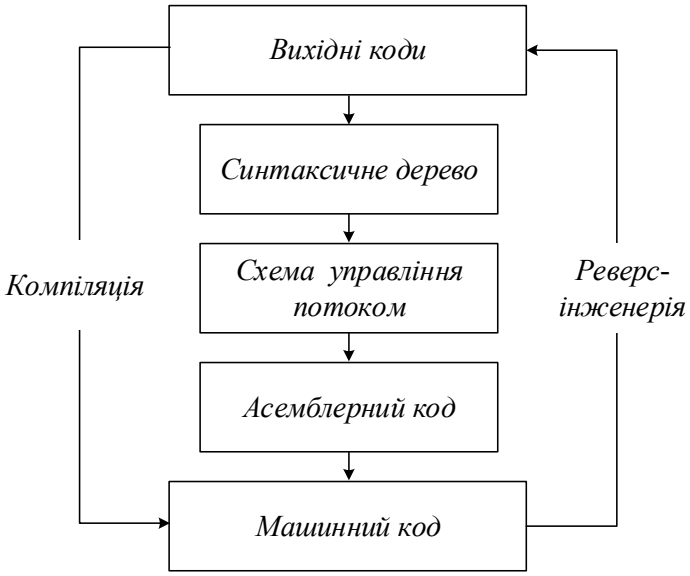


Рис. 1.3. *Аналіз шкідливого програмного забезпечення за використання підходу реінжинірингу* [2]

Дизасемблятори та дебагери дозволяють відображати код ШПЗ у вигляді асемблерних інструкцій, що надає велику кількість інформації відносно того, що

саме виконує шкідливе програмне забезпечення, а також допомагає визначити шаблони для ідентифікації зловмисників. Засоби дампів пам’яті, застосовуються для отримання захищеного коду, розміщеного в пам’яті системи, та скидання його у файл. Це корисна методика аналізу запакованих виконуваних файлів, які важко дизасемблювати. Бінарні методи обфускування, що перетворюють бінарні файли зловмисних програм у самостиснуті та єдино структуровані файли, розроблені таким чином, щоб протистояти змінам та вивченню, що не дозволяє проводити якісний аналіз. Крім того, за використання бінарних виконуваних файлів (отриманих шляхом компіляції вихідного коду) для статичного аналізу, така інформація, як розмір структур даних або змінних, втрачається, ще більше ускладнюючи аналіз коду програмного забезпечення, зазначається у [18].

Технічні методи ухилення зловмисниками від статичного аналізу призвели до розвитку динамічного аналізу. Мозер та ін. [3], досліджував недоліки методології статичного аналізу. Ним було запроваджено схему, засновану на проведенні обфускації коду, що доводить неодстатність статичного аналізу для виявлення або класифікації шкідливих програмного забезпечення. Відповідно до проведених досліджень затверджено, що динамічний аналіз є необхідним доповненням до статичного аналізу, оскільки він є менш вразливим до обфускації.

Динамічний аналіз шкідливих програм передбачає аналіз програми під час її роботи в системі [2120]. Шляхом виконання зловмисного програмного забезпечення у безпечному та контрольованому середовищі, для уникнення перенесення досліджуваного зловмисного програмного забезпечення до інших систем або мереж. Основний динамічний аналіз включає спостереження за зібраним зразком та його взаємодією з системою. Спочатку робиться знімок вихідного стану віртуальної машини, після цього шкідливе програмне забезпечення запускається для виконання у тестовій системі. Вихідний та вхідний стани порівнюються для виявлення змін. Отримані зі спостережень зміни в подальшому застосовуються для видалення ШПЗ із заражених вузлів та/або проведення моделювання ефективних сигнатур . Як і базовий статичний

аналіз шкідливого програмного забезпечення, динамічний аналіз є важливим початковим етапом аналізу ШПЗ, хоча він і не надає вичерпної інформації про зловмисне програмне забезпечення [27].

Розширений динамічний аналіз передбачає використання інструментів для вивчення стану шкідливої програми під час її запуску. Так, наприклад, вивчається внутрішній стан шкідливого коду. Застосування розширеної методики аналізу надає інформацію, яку неможливо зібрати за використання інших методів [40]. Динамічний аналіз завжди проводиться в ізольованому середовищі, аби гарантувати, що всі входи та вихідні дані системи відомі для подальшого врахування. Застосування додаткових інструментів дозволяє відстежувати API, що використовуються на цьому етапі, виклики функцій системи, викликані та видалені файли, зміни реєстру та дані, оброблені програмою, що аналізується, під час взаємодії із системою.

Аналіз параметрів, що використовуються під час викликів API та функцій, дозволяє групувати функції, що використовуються, семантично, тоді як аналіз даних, що обробляються та розповсюджуються в системі, дає розуміння файлів, які використовуються та виробляються шкідливим програмним забезпеченням. Вони призводять до визначення мети, з якою зловмисне програмне забезпечення було розроблене [39]. Розширений динамічний аналіз шкідливих програм дуже корисний для виявлення варіантів зловмисного програмного забезпечення та прихованих методів.

Для зручності застосовуються автоматизовані засоби динамічного аналізу зловмисних програм, що надають звіти, які можна використовувати для групування за поведінкою ШПЗ.

Підсумовуючи, слід виокремити особливості роботи кожного з типів аналізаторів. Системи динамічного аналізу виконують бінарні файли у віртуалізованому середовищі та записують поведінку вибірки, шукаючи показники зловмисної активності. З іншого боку, статичні аналізатори обробляють виконувані файли без їх запуску, витягуючи особливості, які

використовуються для класифікації, безпосередньо з двійкових файлів та їх метаданих.

Хоча обидва підходи мають позитивні та негативні аспекти, багато рішень безпеки на кінцевих точках, як правило, вирішується саме завдяки статичним аналізатори через встановлені суворі часові обмеження, необхідні, аби не впливати на продуктивність роботи системи.

1.4. Підходи до виявлення загроз на обчислювальних вузлах

Оскільки кількісті академічних досліджень з виявлення шкідливого програмного забезпечення зростає, з’являється велика кількість методів для виявлення загрозою. Розглянемо детальніше основні підходи виявлення ШПЗ на обчислювальних вузлах.

На ранніх етапах досить широко застосовувався метод виявлення, що базується на сигнатурах (підписах). Цей метод працює швидко і ефективно для визначення відомих видів шкідливого програмного забезпечення. Проте, даний метод не можна застосовувати для виявлення ШПЗ, що використовує вразливості нульового дня [4]. Пізніше з’явилися поведінкові, евристичні та інші методи.

Кожен з підходів визначення ключових ознак відрізняються один від одного та має свої переваги та недоліки, тому не можна однозначно стверджувати про те, що певний метод кращий.

Використовуючи підходи до виявлення загроз, що базуються на поведінковій моделі, евристиці та моделюванні, можна визначити велику кількість ШПЗ. Крім того, дані моделі також визначають нові шкідливі програми. Однак, вони не є універсальними та не в змозі виявити всі зловмисні програми. Тому, існує велика необхідність знаходження методу, що ефективно виявляє і складніші, невідомі програми.

Діаграма підходів до визначення ШПЗ та його ознак зображена на рис. 1.3.

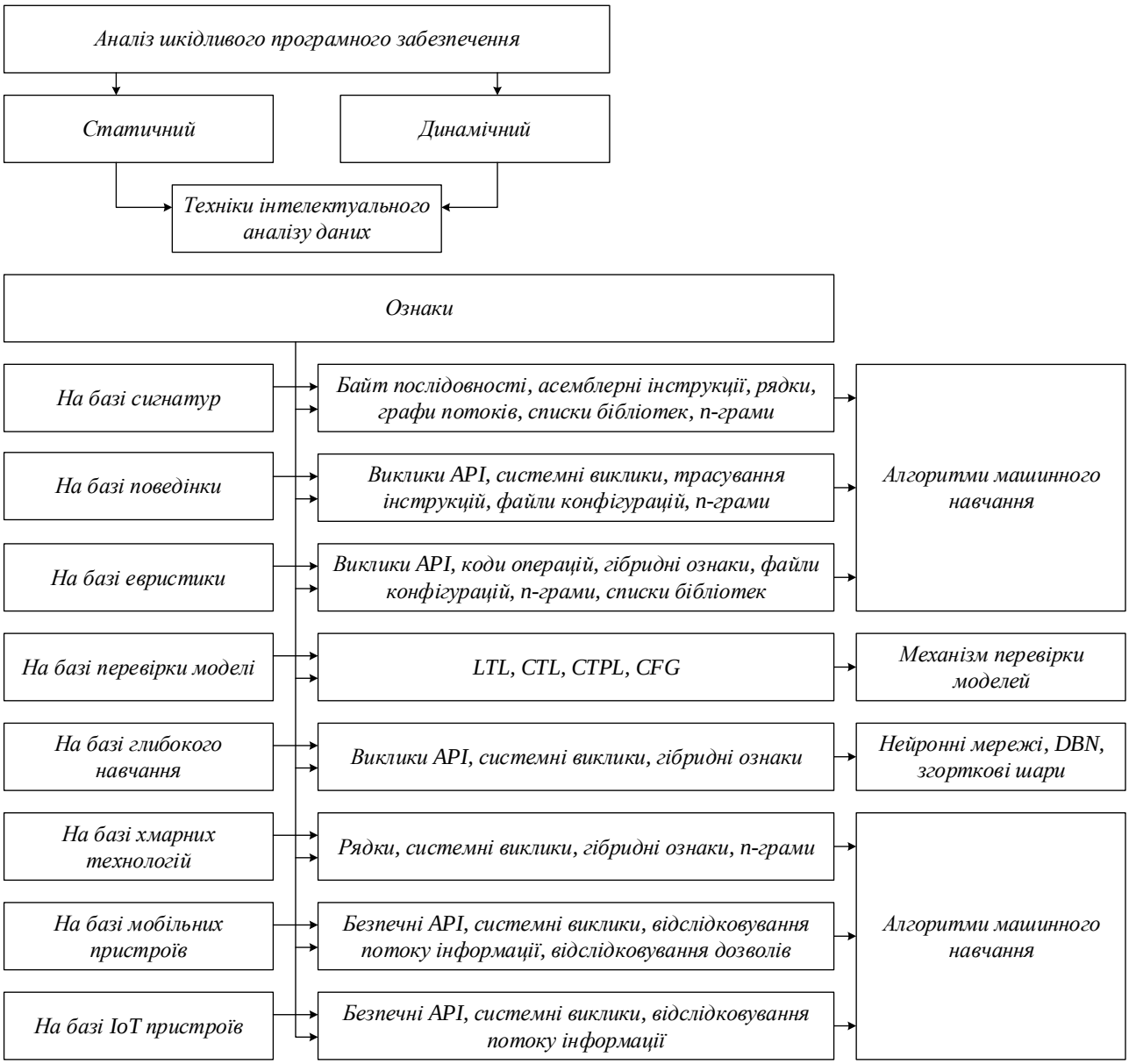


Рис. 1.3. *Діаграма підходів до визначення ознак ШПЗ* [2]

Сигнатурний підхід. Під сигнатурою розуміють функцію зловмисного програмного забезпечення, що інкапсулює структуру програми та визначає кожне зловмисне програмне забезпечення, як унікальне. Підхід до виявлення підписів широко використовується в комерційних антивірусах. Цей підхід досить швидкий та ефективний для виявлення відомих видів шкідливого програмного забезпечення, але недостатньо потужний для виявлення невідомих видів шкідливих програм. Крім того, зловмисне програмне забезпечення, що належить до однієї родини, може легко уникнути виявлення на базі підписів, шляхом застосування обфускації. Загальний вигляд схеми виявлення на основі підпису можна побачити на малюнку 1.4.

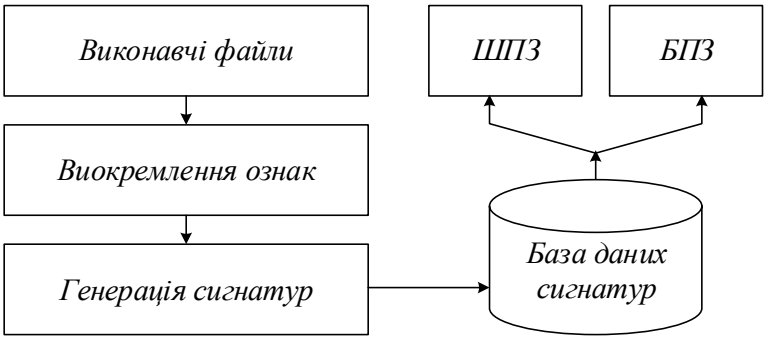


Рис. 1.4. *Схема виявлення загроз на базі сигнатурного підходу* [2]

Поведінковий підхід до виявлення зловмисного програмного забезпечення, дотримується поведінки програми за допомогою інструментів моніторингу та визначає, чи є програма зловмисною. Не зважаючи на зміну програмного коду поведінка залишається подібною тим самим дозволяючи застосування даного підходу для визначення більшості нових шкідливих програм [42]. Однак, деякі програми зловмисного програмного забезпечення не працюють належним чином у захищеному середовищі, а отже, зразок зловмисного програмного забезпечення може бути помилково позначений як безпечний . Схема визначення ШПЗ за поведінкового підходу зображена на рис. 1.5. За виявлення загроз на основі поведінки необхідно спочатку за допомогою однієї з методик, визначити поведінку, а потім, шляхом віднімання функцій створити датасет, після чого можливим є виведення специфічні особливостей з набору даних та здійснення їх класифікації за допомогою алгоритмів ML.

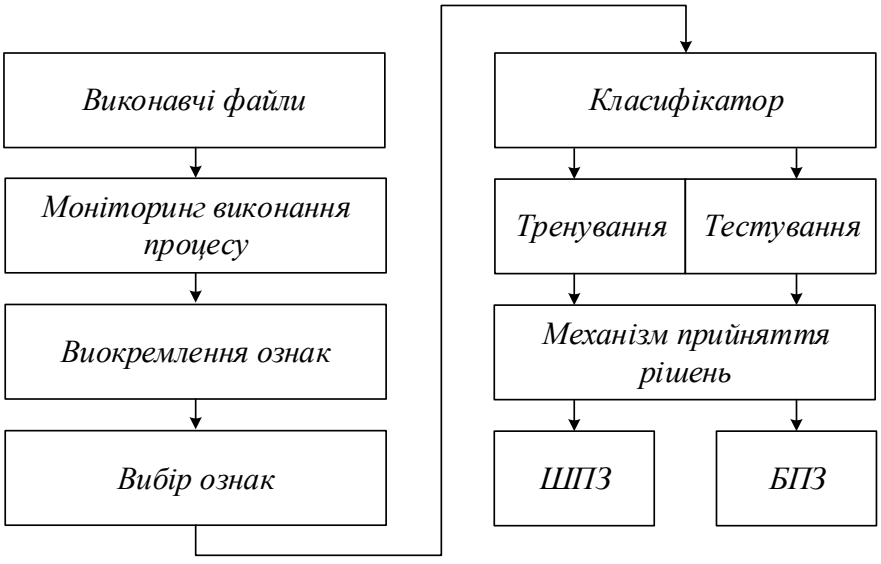


Рис. 1.5. *Схема визначення ШПЗ за поведінкового підходу* [2]

В останні роки часто застосовується ***евристичний підхід*** до виявлення ШПЗ. Це складний метод виявлення, який використовує досвід та різні методи, такі як правила та методики машинного навчання (ML) 37]. Даний підхід надає можливість визначати уразливості нульового дня, проте не здатний виявляти складне програмне забезпечення. Евристичний підхід виявлення ШПЗ представлений на рис. 1.6.

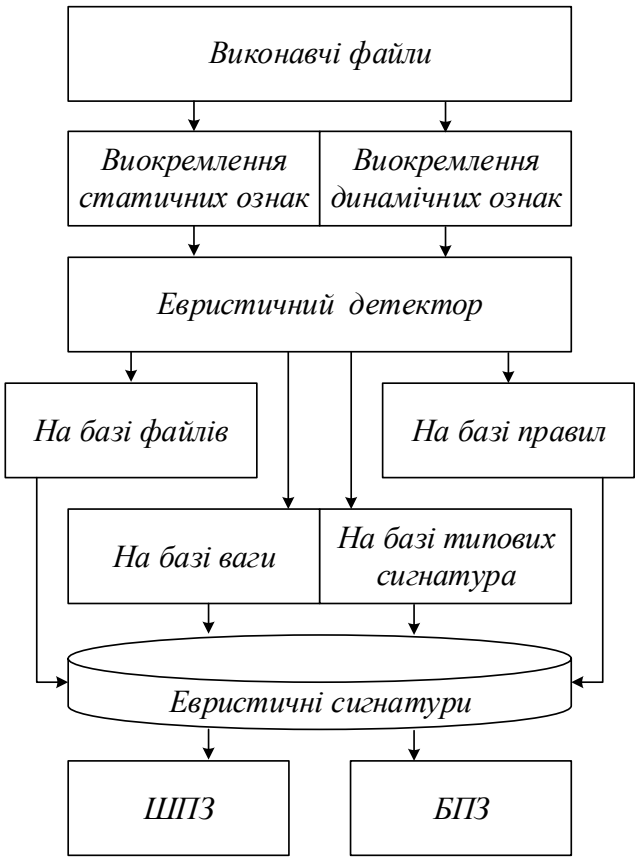


Рис. 1.6. *Евристичний підхід виявлення ШПЗ* [2]

Підхід, що базується на перевірці моделі. За такого підходу поведінку зловмисного програмного забезпечення визначається власноруч, а групи поведінки кодуються за допомогою лінійної часової логіки (LTL) для відображення відповідних особливостей.

Програмна поведінка створюється шляхом перегляду відносин потоку одного або декількох системних викликів та визначення поведінки за допомогою властивостей, таких як приховування, розповсюдження та ін'єктування [37]. Порівнюючи ці поведінки, визначається, чи програма є зловмисною. Таких спосіб дозволяє виявляти деяке нове ПЗ, проте, визначення нового покоління

ШПЗ за таким методом не є можливим.. Схему виявлення на основі перевірки моделі можна побачити на рис. 1.7.

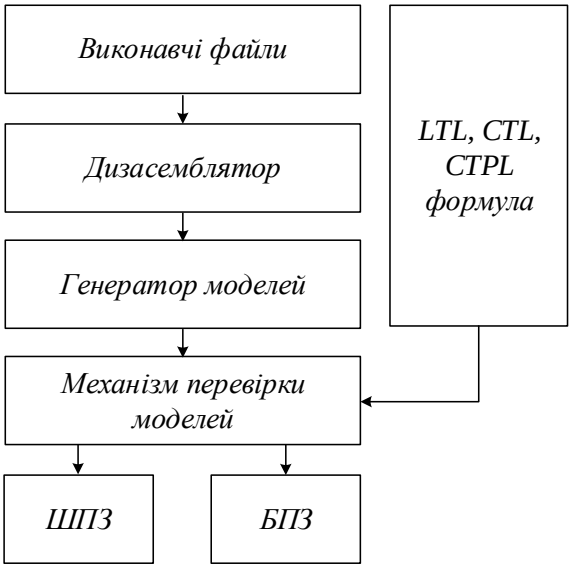


Рис. 1.7. *Схему виявлення на основі підходу перевірки моделі* [2]

Глибоке навчання є різновидом машинного навчання ML, яке успадковане від штучних нейронних мереж (ANN), що навчаються на прикладах. Це новий підхід, що широко використовується для обробки зображень, безпілотного контролю і голосового контролю; проте, він ще досі недостатньо використовується для виявлення шкідливого програмного забезпечення. Незважаючи на те, що він є досить ефективним, його основним недоліком є не стійкість до атак, що використовують ухилення. Схему на основі глибокого навчання можна побачити на рис. 1.8.

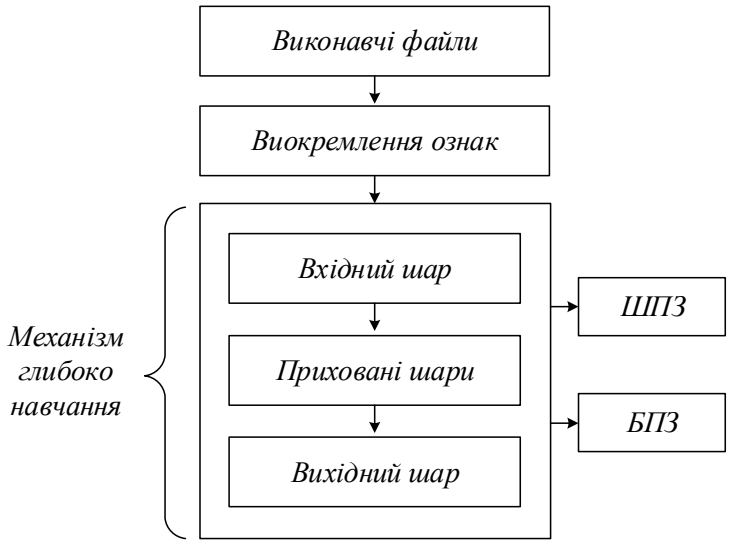


Рис1.8. *Схему роботи підходу, заснованого на глибокому навчанні* [2]

Хмарні обчислення швидко розвиваються, оскільки вони надають багато переваг, перш за все, маючи просту доступність, можливість зберігання за запитом та зменшення витрат.

Оскільки хмара є досить популярним рішення, її також використовують для виявлення шкідливих програм.

Хмарне виявлення шкідливих програм підвищує ефективність виявлення для персональних комп'ютерів та мобільних пристроїв зі значно більшими базами даних ШПЗ, що вимагають більш інтенсивного використання обчислювальних ресурсів.

Хмарне виявлення використовує різні типи агентів виявлення на хмарних серверах і пропонує безпеку як послугу (SecaaS). Користувач може завантажувати файли будь-якого типу та отримувати звіт, чи завантажений файл розпізнано як шкідливий, чи ні.

Схему виявлення на основі хмари можна побачити на малюнку 1.9. Деякі бінарні файли шкідливих програм не працюють належним чином у захищеному середовищі (віртуальні машини тощо).

Отже, зразки зловмисного програмного забезпечення можуть бути неправильно позначені як безпечні.

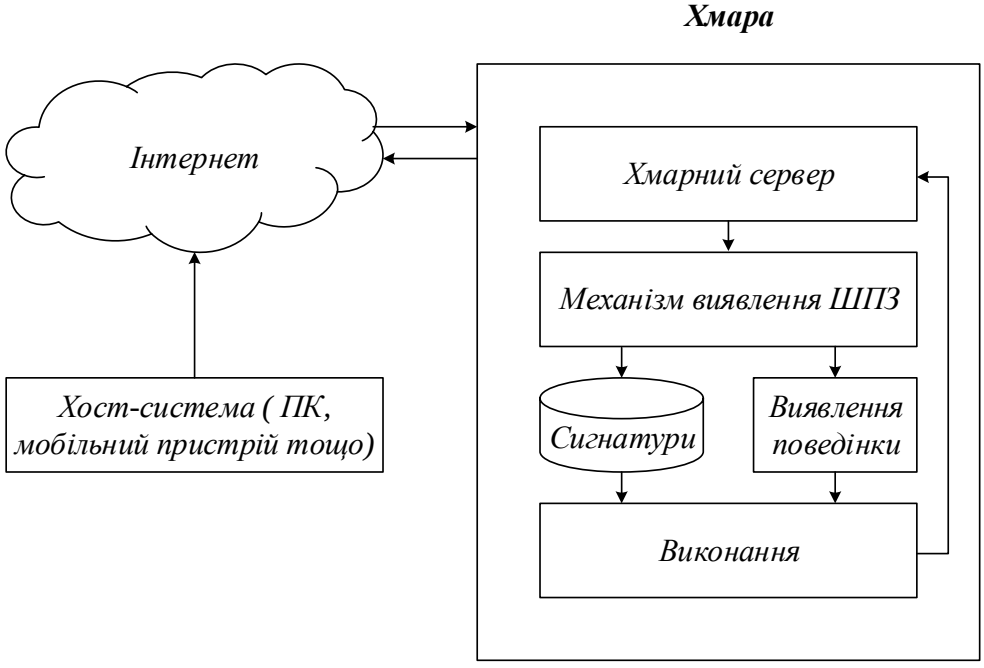


Рис. 1.9. *Схему виявлення ШПЗ на основі хмарного підходу* [2]

Платформа Android стала лідером на ринку. Згідно з останніми дослідженнями, нові шкідливі програми для Android впроваджуються кожні 10 секунд. Через це дослідники зосередилися саме на даній платформі.

Спеціально для платформи Android запропоновано численні методи виявлення зловмисних програм. Як правило, ці методи використовують алгоритми передачі даних та ML для виявлення шкідливих програм. Використовується ряд різних функцій, таких як системні виклики, чутливі до безпеки API, інформаційні потоки та структури потоків управління.

Не зважаючи на те, що поточні дослідження покращили виявлення традиційних шкідливих програм нового покоління виявлення складних зловмисних програм як і раніше залишаються складним завданням. Схему виявлення на основі мобільних пристроїв приведено на рис. 1.10.

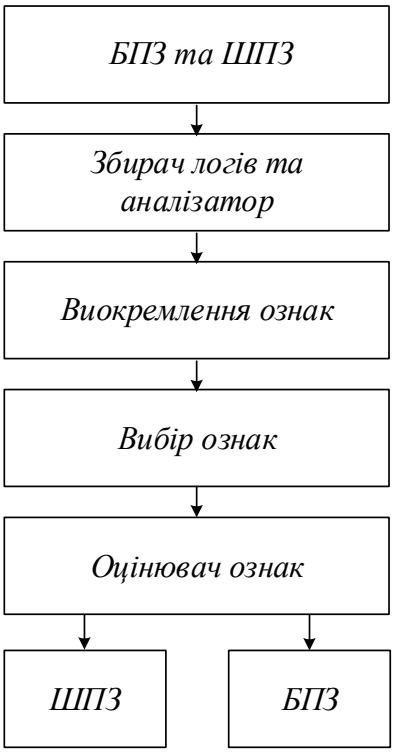


Рис. 1.10. *Схему виявлення ШПЗ на базі мобільних пристроїв* [2]

Архітектура Інтернету речей (IoT) зазвичай складається з широкого спектру підключених до Інтернету смарт-пристроїв, таких як побутова техніка, мережеві камери та датчики. IoT та мобільні пристрої почали домінувати в Інтернеті навіть більше, ніж на персональні комп’ютери.

Оскільки мобільні пристрої та пристрої IoT з кожним днем стають все більш популярними серед користувачів, вони також стають більш улюбленими мішенями для нападників. Через це схеми виявлення зловмисних програм змінюються від комп'ютерів до IoT та мобільних пристроїв. Схема виявлення на основі IoT видно на малюнку 1.11.

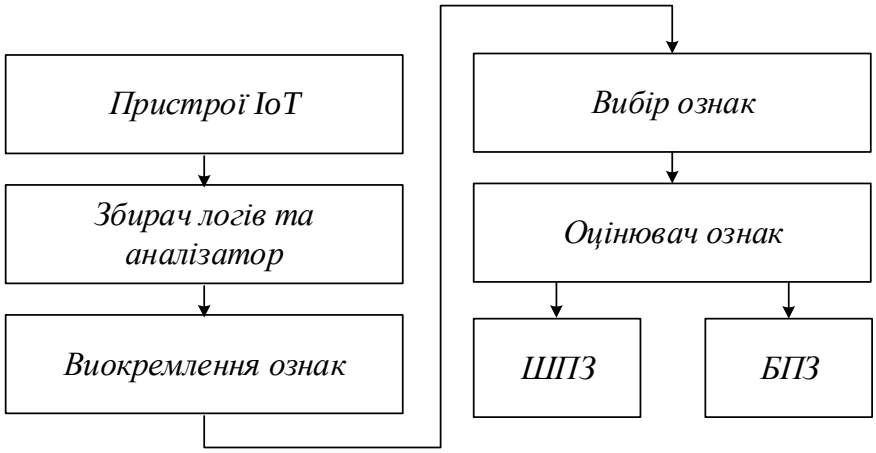


Рис. 1.11. *Схема виявлення ШПЗ на основі IoT* [2]

У якості висновків приведемо порівняльну таблицю для визначення того, який підхід краще застосовувати для відповідного типу ШПЗ.

Таблиця 1.1*. Порівняльна характеристика підходів виявлення ШПЗ*

Підхід до виявлення ШПЗ	***Виявлення невідомого ШПЗ***	***Стійкий до обфускації***	***Новий підхід***
Сигнатурний	-	-	-
Поведінковий	+	+	-
Евристичний	+	-	-
Перевірка моделі	+	+	-
Глибоке навчання	+	-	+
Хмарний	+	-	+
Мобільний	+	-	+
IoT	+	-	+

1.5. Аудит операційних систем та його застосування для захисту обчислювальних вузлів

Одним з ключових напрямків досліджень, спрямованих на захист інформаційних систем, є розроблення та вдосконалення систем виявлення вторгнень, відповідно англійською – Intrusion Detection Systems (IDS). Застосування таких засобів може попередити можливість порушення цілісності обчислювальних вузлів та мереж.

IDS виявляють шкідливу активність, серед якої: мережеві атаки вразливих сервісів, здійснення неавторизованого доступу, дії спричинені шкідливим ПЗ таким як черви, програми-вимагачі тощо. Виявлення вторгнень зазвичай відбувається шляхом аналізу двох основних компонент: даних мережевого трафіку (IDS мережевого рівня) та даних аудиту (IDS рівня хоста).

В термінах операційних систем під аудитом розуміють фіксацію спеціалізованим програмним забезпеченням подій, що відбуваються у системі з метою подальшого їх аналізу.

Задля підтримання необхідного рівня цілісності, доступності та конфіденційності ОВ передбачається створення політики безпеки - комплексу процедур та елементів управління, які визначають, кому може надаватися доступ до операційної системи, які ресурси (файли, програми) можуть використовуватися та які дії можуть бути виконані в системі.

Політика безпеки охоплює всі запобіжні заходи та методи для забезпечення захисту ОС, а також даних, які можуть бути викрадені, відредаговані чи видалені. Переважно, основними компонентами політики безпеки є відстеження: процедур входу, маркерів доступу, списків контролю та прав доступу.

Створена політика безпеки визначає спектр даних для контролю за допомогою підсистеми аудиту. Збір даних аудиту відбувається на окремих хост-системах з журналів, що утворюються в процесі роботи утиліти. Отримані дані містять велику кількість корисної інформації, що дозволяє не тільки детально

відслідковувати події, що відбуваються у системі, але і відновлювати хід подій у разі необхідності вивчення специфіки атак.

Початково, підсистема аудиту була створена для отримання діагностичних даних, проте у зв’язку з включенням підсистеми до модулів ядра та її роботи на найнижчому рівні з можливістю перехоплення системних викликів ядра, пропонується її застосування як базової компоненти для розробленого способу підвищення безпеки обчислювального вузла.

Висновки до розділу 1

В результаті проведення детального огляду сучасного стану інформаційної безпеки, видів загроз та основних підходів до виявлення загроз системі, можна зробити такі висновки:

1. Кількість зразків загроз інформаційним системам щорічно невпинно збільшується. Таке експоненційне зростання зумовлює необхідність посилення існуючого захисту. Доведено, що відсоток нових семантик загроз скорочується в результаті того, що нові екземпляри ШПЗ є лише модифікацією вже реалізованих механізмів загроз, до яких було застосовано поліморфізм та обфускацію з метою зміни їх сигнатур. Така тенденція є позитивною, оскільки дозволяє значно підвищити безпеку обчислювальних вузлів, шляхом запобігання виконанню близько 80% ШПЗ у випадку розроблення системи виявлення та запобігання загрозам стійким до таких модифікацій.

2. Найбільшим недоліком сучасних систем покладено саме нестійкість до обфускації (таблиця 1.1). Для усунення цього недоліку пропонується розробка системи захисту, що реалізується на основі ідей об’єктно-орієнтованого способу підвищення безпеки обчислювальних вузлів на базі підсистеми аудиту. Застосування в основі розробленого підходу механізмів глибокого навчання дозволить нівелювати головну вразливість найпоширенішого методу захисту (сигнатурного) – виявлення невідомих видів ШПЗ.

РОЗДІЛ 2
АНАЛІЗ ПІДСИСТЕМИ АУДИТУ ОПЕРАЦІЙНОЇ СИСТЕМИ НА БАЗІ ЯДРА LINUX ЯК ОСНОВИ ЗЗІ

2.1. Реалізація підсистеми аудиту на базі операційних систем

Підсистема аудиту входить до переліку основних компонент у сучасних операційних системах. Так у найбільш поширеній ОС для настільних комп'ютерів Windows вона реалізована як Event Tracing (ETW) - програмний інтерфейс для проведення трасування на рівні ядра, що дозволяє виконувати запис не тільки подій ядра, але і додатків в спеціалізований файл журналу у режимі реального часу та надає можливість додатково використовувати дані для налаштування застосунків та миттєвого визначення проблем з ними [28]. Проте, відсутність доступу до вихідних кодів програмного інтерфейсу підсистеми аудиту, що пояснюється комерційністю сімейства операційних систем від Microsoft, значно ускладнюють її вивчення.

Особливості реалізації структури та принципів роботи підсистеми аудиту можна дослідити на базі вільного програмного забезпечення у вигляді різноманітних дистрибутивів операційних систем з ядром Linux. За даними відкритих джерел [21], [38], [1] саме Linux дистрибутиви застосовуються для більшості серверних систем найбільших промислових гігантів, у якості рішення для хмарних інфраструктур та суперкомп'ютерів, що доводить актуальність розгляду саме в контексті даного ядра операційної системи.

Підсистема аудиту Linux є функцією ядра, що збирає дані необхідні типів системних подій для полегшення подальшого аналізу. Особливістю підсистеми аудиту є наявність інтегрованого модуля ядра для перехоплення системних викликів та демону простору користувача, що реєструє події системних викликів, виконуючи запис логів подій у пам'ять. Компонента, що представлена у просторі користувача має назву auditd. Налаштування демону аудиту відбувається відповідно до політики безпеки, шляхом додавання правил у один з

файлів підсистеми аудиту - audit.rules для подальшого застосування з метою визначення причин порушення політики безпеки та дій, що їх спричинили.

Підсистема аудиту контролює три основні типи подій:

1. системні виклики, дозволяючи переглядати: які виклики були виконані, з якою контекстною інформацією, які аргументи були передані, інформацію щодо користувача, який здійснив виклик та багато інших відомостей;

2. події доступу до файлів як альтернативний спосіб моніторингу активності доступу до файлів, а не прямого моніторингу відкритого системного виклику для пов’язаних з цією подією викликів [59];

3. попередньо налаштовані для відстеження події відповідно до приведеного у документації [45] списку. Серед них, наприклад, можна виокремити події надання файлу атрибуту виконання, невдалі спроби входу до системи, перебіг виконання криптографічних операцій, перехід до облікового запису адміністратора, спроби вимкнення підсистеми аудиту, створення віддаленого підключення, спроби блокування облікового запису користувача з віддаленого вузла, зміни у політиці безпеки, модифікація SELinux (модуля безпеки ядра (LSM), що надає механізм для підтримки політики безпеки контролю доступу, що включає примусовий контроль доступу (MAC) [48]) і інші.

Основні відомості, що можуть бути записані до журналу демоном аудиту відповідно до зазначених вище подій :

1. дата, час, тип та результат події;

2. асоціація події з ідентифікатором користувача, що ініціював подію;

3. усі зміни конфігурацій підсистеми аудиту та спроби доступу до файлів журналу аудиту;

4. усі види використання механізмів автентифікації, таких як SSH, Kerberos та інші;

5. внесення змін до системних файлів;

6. спроби імпорту та експорту інформації у двосторонньому напрямку відносно системи;

7. службова інформація відносно перехоплених подій на базі даних ідентифікатора користувача або інших атрибутів [54].

2.2. Компоненти підсистеми аудиту на базі ядра Linux

Окрім демону аудиту підсистема в просторі користувача містить також компоненти необхідні для перегляду записів аудиту (ausearch, aureport), утиліту для налаштування правил (auditctl), що задаються відповідно до політики безпеки (audit.rules), засоби перегляду останніх зареєстрованих у системі користувачів, компонента запису подій в журнал на сервері агрегованих журналів (audisp-remote), мультиплексор подій аудиту (audispd) та файли конфігурацій (audit.conf), що дозволяють кастомізувати роботу підсистеми. Взаємозв'язок усіх компонент підсистеми аудиту з ядром операційної системи зображено на рисунку 2.1.

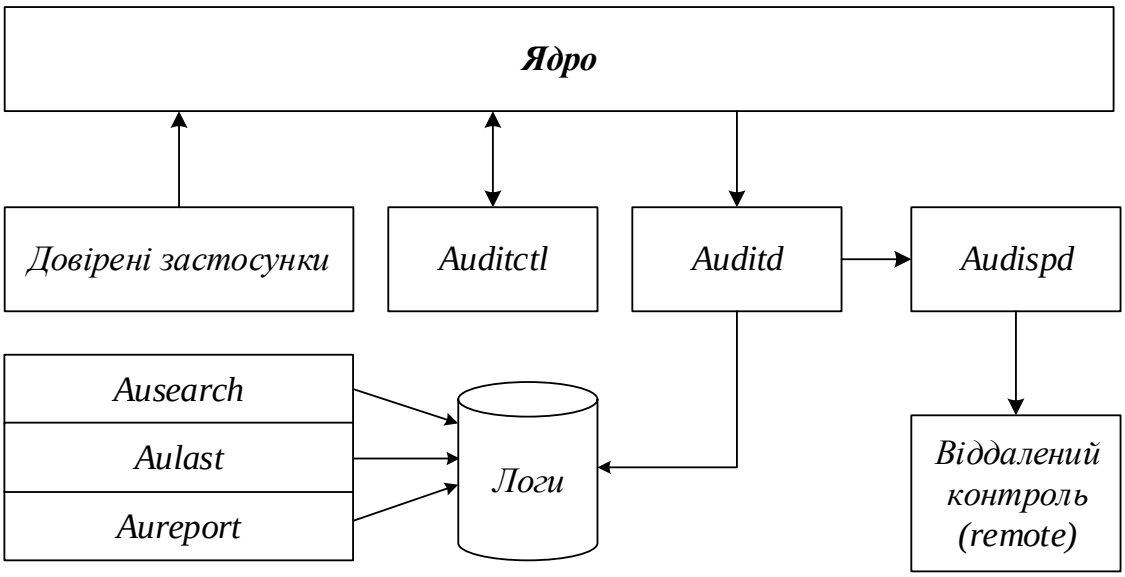


Рис. 2.1. *Компоненти підсистем аудиту ОС на базі ядра Linux*

Розглянемо детальніше призначення кожної з компонент для демонстрації широко спектру можливостей аудиту.

ausearch є інструментом, що застосовується для проведення пошуку всередині журналу аудиту. Дана компонента може також приймати дані зі стандартного потоку введення (stdin) доти, доки на вході будуть необроблені дані записів логів. Всі умови, зазначені в параметрах під час виклику компоненти, об’єднуються логічним І. Наприклад, за вказання ключів -m та -ui в

якості параметрів будуть відображені події, що відповідають заданому типу та коду користувача. Варто також зазначити, що кожний системний виклик ядра з простору користувача та повернення даних в простір користувачів мають один унікальний (для кожного системного виклику) ідентифікатор події. Різноманітні частини ядра можуть додавати додаткові записи вносячи незначні зміни кінцевої структури запису журналу аудиту. Так, наприклад, під виконання системного виклику з відкриття open() додається запис PATH з ім'ям файлу. ausearch показує всі записи події разом. Це означає, що при запиті окремих видів записів результат може містити записи даних системних викликів SYSCALL [16]

aureport - це інструмент, що генерує кінцеві звіти на базі логів демона аудиту. Компонента може також приймати дані зі стандартного потоку введення (stdin) доти, доки на вході будуть необроблені дані логів. Кожен звіт у верхній частині містить заголовок, що значно полегшує розуміння даних. Усі звіти, крім основного кінцевого звіту, містять номери подій аудиту. Використовуючи їх, можна знайти повні дані відносно подій за допомогою ausearch -a та номера події. Для зменшення кількості даних, можна задати час початку та час закінчення відстеження подій для уточнення часового проміжку. Звіти, що генеруються aureport, можуть бути використані як вихідний матеріал для отримання більш розгорнутих звітів.[15]

auditctl використовується для контролю поведінки, отримання стану і додавання або видалення правил аудиту. Серед можливостей контролю поведінки слід виокремити встановлення максимальної кількість доступних для аудиту буферів, прапору блокування для тимчасового вимкнення аудиту або захисту конфігурації аудиту від змін. Також можливим є встановлення способу обробки прапора збою, що дозволяє визначити яким чином ядро буде обробляти критичні помилки та фільтри прав доступу для точки спостереження, що визначають типи системних викликів, які виконують дії читання, запису та виконання [11].

aulast є засобом перегляду останніх зареєстрованих у системі користувачів. Компонента виконує пошук всередині журналу аудиту чи у

заданому файлі аудиту та відображає список усіх користувачів, що входили або виходили з системи відповідно до заданого діапазону часу у журналі аудиту. Додаткового у якості параметрів можуть бути задані імена користувачів та терміналів. В такому випадку утиліта виведе тільки ту інформацію, що відповідає заданим критеріям пошуку. Назви терміналів можуть бути скороченими [14].

audispd використовується для мультиплексування подій аудиту. Він має бути запущений безпосередньо демоном аудиту для отримання подій. На вхід приймаються події аудиту після чого відбувається їх розподіл між дочірнім програмами, що виконують аналіз подій у режимі реального часу. Коли демон аудиту отримує сигнал SIGTERM (спричинення припинення програми, сигнал можна блокувати, обробляти та ігнорувати) чи SIGHUP (розрив зв'язку між користувачем та терміналом), він також передає цей сигнал диспетчеру. Диспетчер, відповідно, передає ці сигнали свої дочірнім процесам [7].

audisp-remote це плагін для демона диспетчера подій аудиту audispd, котрий виконує попередній запис в журнал, що розміщений на сервері агрегованих журналів. Застосовується з сигналом SIGUSR1, що змушує програми записувати значення деяких внутрішніх прапорів в системний журнал. Прапор призупинки вказує на те, чи було призупинене ведення журналу. прапор транспортування вказує на справність з’єднання з віддаленим сервером, а прапор черги сповіщає про кількість записів, що було поставлено в чергу для відправлення на віддалений сервер. Використання SIGUSR2 призводить до поновлення запису до журналу агрегованого аудиту у випадку, якщо було раптом призупинене логування через наявність помилок [8].

autrace програма, що додає правила аудиту для відстеження процесу так само як і strace. Після трасування відбувається занесення інформації до файлів журналу якщо демон аудиту був запущений. Ця команда видаляє усі правила до та після виконання цільової під час виконання програми з передачею аргументів. В якості попередження безпеки дана компонента не працюватиме якщо не буде виконано видалення усіх правил через застосунок auditctl [17].

2.3. Принцип роботи підсистеми аудиту

2.3.1. Архітектура підсистеми аудиту

Як було зазначено раніше, підсистема аудиту складається з двох частин : компоненти ядра та утиліт простору користувача. Компонент ядра отримує системні виклики від додатків простору користувача та фільтрує їх завдяки одному з трьох фільтрів: завдання, користувач чи вихід. Як тільки системний виклик проходить через один з цих фільтрів він відправляється для проходження через фільтр виключення, що на базі конфігурації правил аудиту відправляє його демону аудиту для подальшої обробки [24]. На рисунку 2.2 приведена ілюстрація даного процесу.

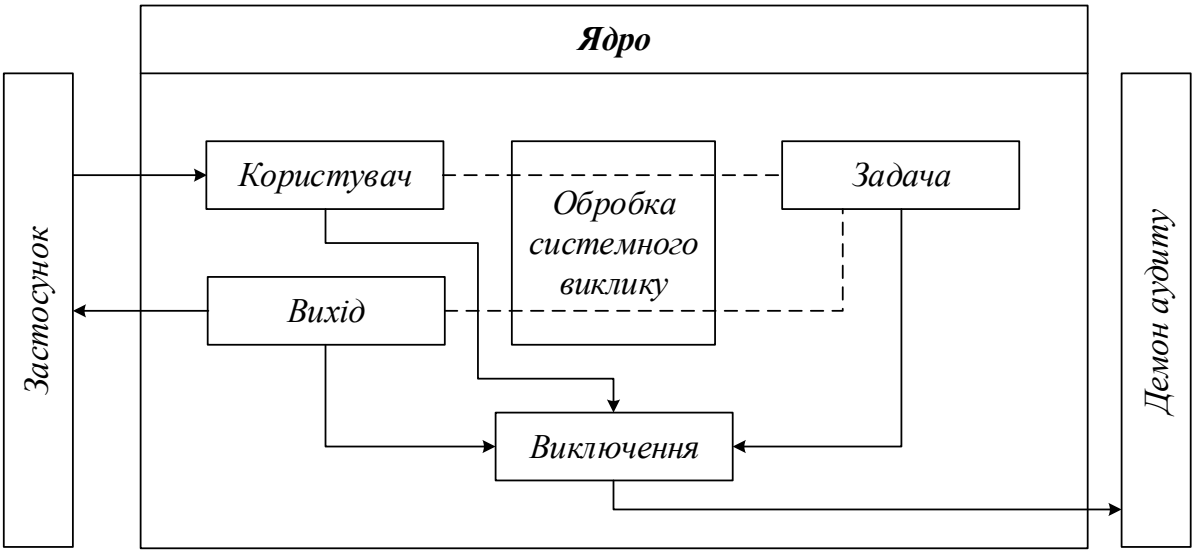


Рис. 2.2. *Архітектура під системи аудиту операційної системи*

Така проста схема дозволяє ефективно відслідковувати будь-який аспект операційної системи, а у випадку зламу системи виявляти підозрілі події та визначати їх причину.

2.3.2. Життєвий цикл підсистеми аудиту

Для опису життєвого циклу підсистеми аудиту необхідно попереднього надати відомості щодо подій, які відбуваються у системі.

Для цього наведемо таблицю викликів та, відповідно, їх призначень (табл. 2.1).

Таблиця 2.1. *Події, що складають життєвий цикл аудиту*

AUDIT_SYSTEM_BOOT	завантаження системи
AUDIT_SYSTEM_RUNLEVEL	зміни рівня виконання системи
AUDIT_DAEMON_START	запис запуску демона аудиту
AUDIT_DAEMON_ABORT	запис зупинки через помилки демона
AUDIT_SERVICE_START	запуск сервісу демона
AUDIT_SERVICE_STOP	зупинка сервісу демона
AUDIT_SYSTEM_SHUTDOWN	вимкнення системи
AUDIT_DAEMON_END	запис нормальної зупинки демона аудиту

Під час завантаження системи відбувається ініціалізація ядра, що виконує запуск першого процесу у системі - init. Цей породжувальний виклик відповідає за подальше завантаження системи шляхом запуску стартових сценаріїв, що виконують перевірку та монтування файлових систем, запуск демонів для завантаження модулів ядра та його налаштування, конфігурації мережевих інтерфейсів, завантаження графічної оболонки. Після закінчення процесу власної ініціалізації демон init має просигналізувати відправленням події AUDIT_SYSTEM_BOOT, для підтвердження завантаження модуля ядра підсистеми аудиту. Необхідно аби демон аудиту був запущеним одним з перших для того, щоб розпочати вилучення подій з ядра до моменту, коли вони будуть втрачені.

В результаті ініціалізації можуть завантажені різні режими роботи системи для інтерактивних сеансів. Init визначить якого саме рівня запуску намагається досягти система. У випадку успішного чи невдалого потрапляння він має виконати виклик події AUDIT_SYSTEM_RUNLEVEL, для того, аби вказати у якому саме режимі роботи він має знаходитися. За необхідності зміни режиму виконання на інший рівень має бути ініційований виклик AUDIT_SYSTEM_RUNLEVEL. Дана подія має зберігати попередній та новий рівні, що зазвичай мають вигляд числа або певної кількості пробілів в залежності від особливостей ініціалізації конкретної системи.

В процесі досягнення бажаного рівня виконання система init виконує запуск різноманітних системних служб та демонів. Під час старту демону програма init ініціює запуск події AUDIT_SERVICE_START. Дана подія має записувати повний шлях до демону або сервісу у шістнадцятковому форматі в полі з зазначеним іменем сервісу. Відносне вказання шляхів заборонене.

Зазвичай, демон аудиту є одним з перших, що запускається у системі. Він має виконати виклик події AUDIT_DAEMON_START, аби вказати, що демон був успішно запущений.

Якщо у його конфігурації буде виявлено помилку, що не дозволяє йому нормально працювати виконується виклик події AUDIT_DAEMON_ABORT з відповідним кодом причини помилки.

Якщо необхідно перезавантаження конфігурації сервісу, система init має ініціювати події AUDIT_SERVICE_STOP, що свідчить про зупинку сервісу демона та AUDIT_SERVICE_START.

За необхідності вимкнення системи ініціалізація має відбуватися з початкового відправлення події AUDIT_SYSTEM_RUNLEVEL, що демонструє поточний рівень запуску та новий рівень - шостий, що відповідає рівню вимкнення системи.

Після цього виконується виклик події AUDIT_SYSTEM_SHUTDOWN, що свідчить про початок вимкнення системи.

По закінченню вимкнення init має завершити роботу кожного працюючого демона та ініціювати подію AUDIT_SERVICE_STOP. Останнім сервісом, що має бути вимкненим має бути саме демон аудиту, що супроводжується подією AUDIT_DAEMON_END, що свідчить про його нормальну зупинку [51].

Отримання доступу до системи користувачем відбувається завдяки застосуванню облікового запису.

За внесення змін до паролів, домашніх каталогів, тощо оболонка та інші сервіси можуть потребувати інформацію щодо того ким були ініційовано внесення змін до системи. В таблиці 2.2 приведені події, що відпадають життєвому циклу облікового запису користувача.

Таблиця 2.2. *Події життєвого циклу облікового запису користувача*

AUDIT_ADD_USER	створення відображення користувача
AUDIT_USER_MGMT	зміна атрибутів облікового запису користувача
AUDIT_USER_CHAUTHTOK	зміна токена автентифікації користувача
AUDIT_ROLE_ASSIGN	призначення ролі облікового запису користувача
AUDIT_ROLE_REMOVE	видалення ролі з облікового запису користувача
AUDIT_DEL_USER	видалення відображення облікового запису користувача
AUDIT_ADD_GROUP	виконано співставлення групи
AUDIT_GRP_MGMT	зміна атрибуту групи
AUDIT_GRP_CHAUTHTOK	зміна токена автентифікації групи
AUDIT_DEL_GROUP	видалення співставлення групи

Користувач з точки зору системи реалізований у вигляді простого числа, що визначає його ідентифікатор. Для здійснення перевірки дозволу виконане розділення процесів на дві категорії: привілейовані, що мають ідентифікатор користувача 0 та належать root - привілейованому користувачеві (адміністратору системи) та непривілейовані, чий ідентифікатор відмінний від нульового. Привілейовані процеси оминають перевірки дозволів ядра, в той час як непривілейовані процеси підлягають повній перевірці дозволів на основі облікових даних процесу (UID - ефективного ідентифікатора користувача, GID - ефективного ідентифікатора групи та додаткового списку груп). Привілеї поділяються на окремі блоки, відомі як capabilities, що відображаються у якості атрибутів для кожного потоку.

Одним з таких привілеїв є CAP_SETUID , що дозволяє виконувати довільні маніпуляції з UID процесів (setuid, setreuid, setresuid, setfsuid); підробляти UID при передачі облікових даних сокета через сокети домену UNIX; записувати відображення ідентифікатора користувача у просторі імен користувачів [23].

Якщо процес має привілеї CAP_SETUID, він може здійснювати виклик setuid, використовуючи майже будь-яке число. Ідентифікація відбувається шляхом зіставлення даних атрибутів пов'язаних з обліковим записом у таблиці /etc/passwd.

Коли відбувається додавання користувача до системи відбувається генерація події типу AUDIT_ADD_USER у випадку успішності операції. Необхідність у реєстрації події створення домашнього каталогу чи поштової черги відсутня, якщо не відбувається збій у роботі. Проте, якщо все ж потрібно проводити відстеження певного роду подій слід застосовувати тип AUDIT_USER_MGMT. Якщо між запуском програми та співставленням будуть помилки, то відбувається реєстрації події AUDIT_ADD_USER з результатами збою.

За використання тіньових паролів з файлу /etc/shadow має застосовуватися тип події AUDIT_USER_MGMT, оскільки обліковий запис вже існує, відбувається звичайне співставлення користувачів з метою модифікації запису. Під час зміни паролю та інших секретних даних необхідно застосовувати події типу AUDIT_USER_CHAUTHTOK. Якщо у системі підтримується поняття ролей, тобто для систем, що застосовують модуль безпеки selinux подія додавання ролі до облікового запису користувача відбувається шляхом фіксації AUDIT_ROLE_ASSIGN. У випадку ануляції ролі застосовують події AUDIT_ROLE_REMOVE.

Видалення відображення користувача з системного файлу має супроводжуватися записом події AUDIT_DEL_USER. Дані події фіксують збої тільки якщо усі атрибути були правильно змінені, але з якихось причин запис до системних файлів груп неможлива. Отже, за створення чи видалення облікового запису користувача має відбуватися лише одна подія - AUDIT_ADD_USER чи AUDIT_DEL_USER, що може супроводжуватися декількома подіями AUDIT_USER_MGMT у випадку необхідності модифікації системних файлів та за наявності помилок [52].

Слід також розглянути життєвий цикл груп, що є дуже схожим на цикл користувачів. Створення відображення відбувається шляхом додавання відповідного рядка до файлу груп та реєстрації події AUDIT_ADD_GROUP у системі. Помилки фіксуються завдяки подіям AUDIT_ADD_GROUP. Після створення відображення групи зміни в атрибути мають реєструватися подією AUDIT_GRP_MGMT. Події, що генеруються за використання тіньової групи повинні приводити до подій з застосуванням типу AUDIT_GRP_MGMT, оскільки відображення групи вже існує. За ануляції рядка групи у відповідному системному файлі, необхідно відправити подію AUDIT_DEL_GROUP. Зміна паролю групи та інших даних супроводжуються AUDIT_GRP_CHAUTHTOK [52].

На рис. 2.3 приведено відображення повного життєвий цикл підсистеми аудиту операційної системи на базі ядра Linux з перебування в усіх її можливих станах.

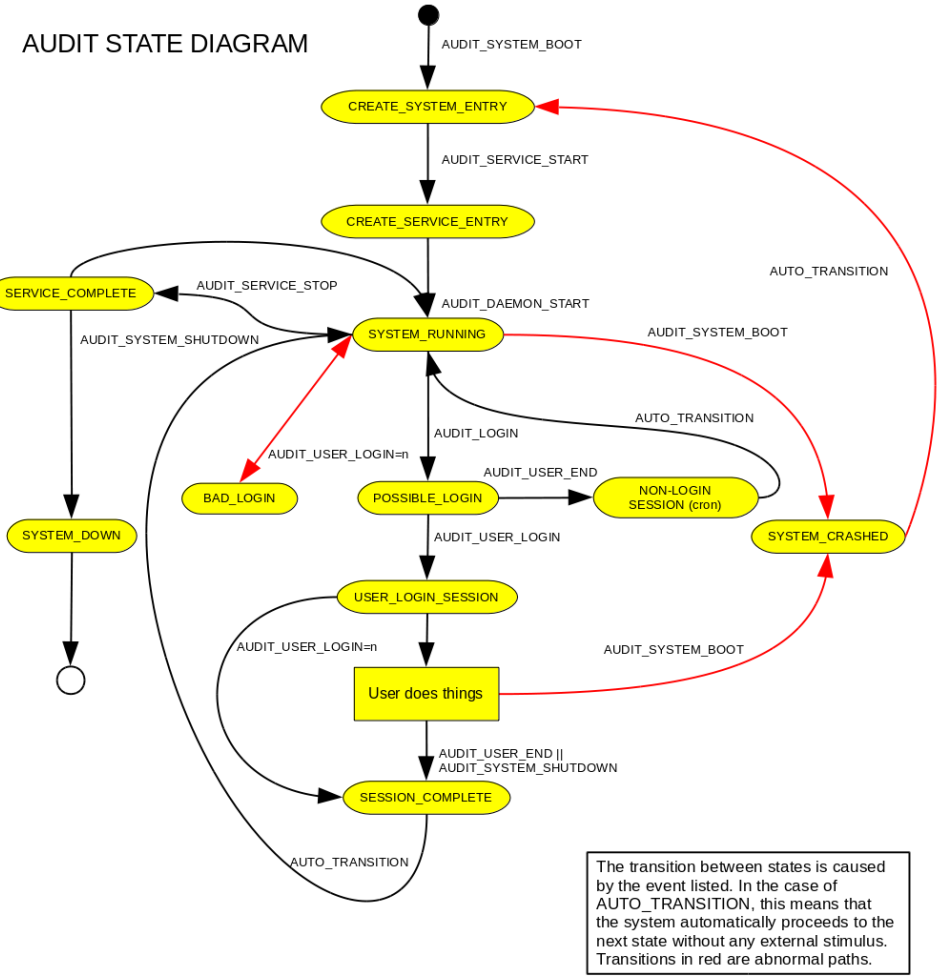


Рис. 2.3. *Діаграма станів підсистеми аудиту операційної системи* [50]

Переходи між станами зумовлені наявністю зовнішніх стимулів у вигляді перелічених вище подій. У випадку переходу за AUTO_TRANSITION система автоматично виконує перехід до наступного стану без необхідності застосування зовнішньої стимуляції. Переходи, зображені червоним кольором відображають анормальні шляхи розвитку подій.

2.4. Налаштування підсистеми аудиту та створення правил

Зміна системних налаштувань демону аудиту відбувається шляхом проведення модифікації файлу конфігурацій, що за замовчанням розміщений у директорії /etc/audit/. Файл /etc/audit/auditd.conf має бути змінений таким чином, щоб відповідати такій структурі: одне ключове слово конфігурації, знак рівності, відповідне значення поля. Усі назви опцій та значення є регістронезалежними. Кожен рядок повинен бути обмежений 160 символами, інакше його буде пропущено. Розглянемо детальніше ключові поля для здійснення налаштувань:

local_events застосовується для ввімкнення відстеження локальних подій. За замовчанням встановлене значення - “так”. Проте, можна встановити значення у “ні” для проведення у випадку необхідності проведення агрегації даних з мережі, що є корисним, якщо демон працює у контейнері. Ця опція може бути встановлена тільки одного разу під час запуску демона аудиту. Перезавантаження файлу конфігурацій не матиме жодного ефекту.

log_file задає повне ім'я шляху до файлу журналу, де будуть зберігатися записи аудиту. Це повинен бути звичайний файл.

write_logs визначає необхідність запису журналу на диск. За замовчуванням виконується запис тому значення поля - так.

log_format - формат журналу, що описує, як інформація повинна зберігатися на диску. Існують два варіанти: RAW (сирий) і ENRICHED (розширений). Якщо встановлено значення RAW, записи аудиту будуть зберігатися у форматі, в якому ядро надсилає інформацію. Параметр ENRICHED дозволить попередньо визначити інформацію щодо адреси uid, gid, syscall,

архітектури та сокетів перед записом події на диск. Це допомагає аналізувати логи подій, створених в інших системах.

log_group визначає групу, що застосовується до атрибутів дозволу файлу журналу. За замовчуванням - root. Назва групи може бути задана числом або словом .

prior_boost - невід'ємне число, яке повідомляє демону аудиту, яке значення прийняти у якості пріоритету. Зазвичай пріоритет процесу визначається у межах від -20 до 19. Найбільший проіоритет має процес з найменшим числом. За замовчуванням для демону аудиту - 4.

flush визначає порядок синхронізації з диском та серед допустимих значень приймає: none, incremental, incremental_async, data та sync. Якщо встановлене значення none, не приймається жодних зусиль щодо перенесення записів аудиту на диск. За incremental, використовується параметр freq для визначення того, наскільки часто необхідно виконувати скидання даних на диск. Параметр incremental_async дуже схожий на попередній відрізняючись тим, що очищення відбувається асинхронно для підвищення продуктивності. Параметр data вказує демону аудиту постійно виконувати синхронізацію частини даних файлу диска. Опція синхронізації повідомляє демону аудиту про необхідність повної синхронізації даних та метаданих під час кожного з записів на диск. За замовчання застосовується incremental_async.

freq повідомляє демону аудиту скільки записів необхідно записати до того як виконати явно команду запису даних на диск. застосовується тільки якщо flush встановлене в значення incremental чи incremental_async.

num_logs вказує кількість файлів журналу якщо rotate задано у якості max_log_file_action. За замовчанням 0 означає відсуніть ротації.

disp_qos визначає потрібність блокування зв’язку між демоном аудиту та диспетчером. Між демоном аудиту і диспетчером існує буфер 128 КБ. Якщо обрано параметр lossy, то події, що відправляються диспетчеру, відкидаються, коли черга заповнена. В іншому випадку демон чекатиме, доки в черзі не з'явиться порожнє місце, перш ніж реєструватися на диску. Існує ризик того, що

поки демон чекає мережевого введення-виведення, подія не записується на диск. Lossy застосовується у якості значення за замовчуванням.

dispatcher програма, що запускається демоном аудиту під час старту. Передає копію усіх подій аудиту потоку введення-виведення додатку.

name_format визначає як здійснюється запис імені вузла в потоці подій аудиту. можливо застосування таких варіантів: none, hostname, fqd, numeric, user. None відзначає, що ім’я комп'ютера не вставляється у подію аудиту. hostname - це ім'я, що повертається системним викликом gethostname. Fqd означає, що він перехоплює ім'я хоста та розпізнає його за допомогою dns . Numeric є аналогом fqd, проте відрізняється тим, що розпізнає IP-адресу машини. Для того, аби використовувати цю опцію необхідно застосувати 'hostname -i' чи 'domainname -i' та переконатись, що він перевертає числову адресу. Не рекомендується до застосування за наявності dhcp, через можливість зміни адреси вузла на іншу через певний проміжок часу. User рядок визначений адміністратором з опції name. Значенням за замовчанням є none.

name задається адміністратором та ідентифікує вузол якщо user задається як значення опції name_format.

max_log_file визначає максимальний розмір файлу у МБ. За досягнення максимального розміру спрацьовує тригер налаштованої опції. Задається у вигляді числового значення.

max_log_file_action визначає системну дію що має бути застосована якщо досягається межа максимально допустимого значення розміру журналу. Можливі значення: ignore, syslog, suspend, rotate та keep_logs. Якщо обрано ignore демон аудиту нічого не робитиме. За syslog буде зафіксоване спеціальне попередження у syslog. Suspend викликає зупинку записів демоном аудиту на диск. Демон аудиту буде продовжувати роботу. Rotate опція спричиняє ротацію логів. Має бути зафіксовано, що логи з більшими номерами старіші, ніж логи з меншими номерами. keep_logs аналогічна опції rotate , проте відрізняється тим, що застосовує налаштування опції num_logs. Це захищає логи від перезапису. В результаті логи акумулюються та не видаляються, спричиняючи активацію

тригеру space_left_action за переповнення розділу. Така комбінація вважається найкращою з застосуванням зовнішніх скриптів для архівації логів на періодичній основі.

verify_email визначає чи зазначений домен адреси, що використовується у action_mail_acct може бути розпізнаний. Дана опція має бути вказана до action_mail_acct чи інакше буде застосоване значення за замовчанням - так.

action_mail_acct має містити достовірну (валідну) адресу чи псевдонім. За замовчанням використовується root. Якщо задана адреса пошти не є локальною необхідно переконатися у правильності конфігурації мережі та поштової служби. Також дана опція потребує обов'язкового існування /usr/lib/sendmail на вузлі.

space_left числове значення у мегабайтах, що повідомляє демон аудиту коли виконувати налаштовані дії під час закінчення місця на диску. Можливо зададвати відсоткове значення, яке в подальшому буде вирахувано, базуючись на розмірі розділу диску.

space_left_action параметр, що задає яку саме системну дію необхідно виконувати у випадку коли закінчується вільне місце на диску. Можливі значення: ignore, syslog, rotate, email, exec, suspend, single, and halt. За ignore демон аудиту не виконує жодних дій. За syslog демон залишить попередження у системному журналі подій. Опція rotate спричинить ротацію логів, видаляючи найстарші для звільнення місця. За email буде надіслано сповіщення на електронну пошту користувача зазначену у опції action_mail_acct та залишить попередження у системному журналі. Exec (/шлях до скрипта) спричинить виконання скрипта, розміщеного за заданим шляхом. suspend спричинить зупинку запису демоном аудиту логів на диск. Демони аудиту продовжить свою роботу без зупинки. Застосування single спричинить переведення демони аудиту вузла до режиму single user. Опція halt виключення демоном аудиту комп'ютерного вузла.

admin_space_left - числове значення у мегабайтах що вказує демону аудиту коли виконувати налаштовану дію у зв'язку з закінчення вільного місця на диску. Це має сприйматися як останній шанс зробити щось, коли закінчується місце на

диску. Числове значення має бути меншим за кількість місця, що залишилася. Можливо також задати відсоткове значення для підрахунку кількості місця на базі розміру відповідного розділу.

admin_space_left_action - параметр, що задає дію, яку необхідно виконати у випадку коли залишається мала кількість місця на диску. Можливі значення: ignore, syslog, rotate, email, exec, suspend, single, halt. Дані опції виконують ті самі дії, що і у параметрі space_left_action.

disk_full_action - параметр, що задає яку дію має виконати система, коли виявляє, що розділ на який записуються логи демону аудиту - повний. Можливі значення: ignore, syslog, rotate, exec, suspend, single, halt. За значення ignore буде залишене повідомлення у системному журналі подій, але більше не буде прийнято жодних дій. Syslog спричинить залишення попередження у системному журналі подій. Rotate спричиняє ротацію логів, заміщуючи старі логи новими для звільнення місця. Подальші опції виконують ті самі дії, що описано у space_left_action.

disk_error_action параметр, що вказує системі що необхідно робити у випадку наявності помилки під час запису подій аудиту на диск чи ротації логів. Можливі значення: ignore, syslog, rotate, exec, suspend, single, halt. За значення ignore демон аудиту не проводить жодних дій. За syslog буде залишене менше п’яти послідовних попереджувальних сповіщень у системному журналі подій. Exec спричинить виконання заданого за адресою скрипта, проте передавати параметр у нього буде неможливо. Останні дії матимуть той самий результат, що і у попередніх опціях.

tcp_listen_port задаєься у вигляді числа з проміжку 1..65535, який, якщо зазначений, спричиняє прослуховування демоном аудиту відповідного TCP порту для записів аудиту з віддалених систем. Демон аудиту може бути зв’язаний за допомогою tcp_wrappers. Якщо необхідно здійснювати контроль доступу, то це можна зробити за допомогою додаткових налаштувань у файлах hosts.allow та deny.

tcp_listen_queue - числове значення, що визначає яка кількість підключень в стані очікування (виконано запит, що ще не прийнято) може бути дозволена. За замовчання - 5. Використання занадто малого числа може причинити відхилення підключень, якщо занадто багато вузлів будуть намагатися виконати підключення одночасно, як, наприклад, після збою живлення.

tcp_max_per_addr числове значення, що визначає яка кількість паралельних підключень з одної IP адрес дозволена. За замовчуванням дозволяється одне підключення, проте максимальне значення - 1024. За зазначення достатньо великого числа можливе спричинення атаки відмови в обслуговуванні (DoS) на сервері, що проводить логування. Також слід зазначити, що є обмеження зі сторони ядра операційної системи, що спричинить відділення не зважаючи на налаштування у демону аудиту. Значення за замовченням має бути адекватним в більшості випадків, якщо користувацький сценарій відновлення не запускається для невідправлених подій. В цьому випадку необхідне збільшення даного числа.

use_libwrap - цей параметр визначає чи необхідно застосовувати tcp_wrappers для розпізнавання спроб підключення від дозволених вузлів. Валідні значення: так або ні. За замовчанням встановлене значення так. Дозволено використання значень у проміжку 1..65535. Наприклад, для використання привілейованих портів необхідно зазначити 1-1023 у якості ключа параметра. Необхідно також встановити опцію local_port у audisp-remote.conf файлі.

tcp_client_max_idle визначає кількість секунд, впродовж якх користувач може знаходити в простої (відсутність будь-яких даних з боку користувача). застосовується для закривання неактивних підключень, якщо наявні проблеми з можливістю нормального завершення підключення. Є глобальним параметром, тому впливає на усі підключення одночасно. Це необхідно врахувати, оскільки різні клієнти мають різні налаштування та рівень продуктивності. За замовчання встановлено 0, що відхиляє необхідні цієї перевірки.

enable_krb5 за значення “так” Kerberos 5 буде застосовано для проведення автентифікації та шифрування. Значення за замовчанням - ні.

krb5_principal встановлює значення принципалу (унікальне значення імені користувача для якого дозволяється автентифікація у Kerberos) для серверу. За замовчання використовується значення "auditd". За використання значення за замовчанням сервер буде шукати ключ, названий у форматі auditd/hostname@EXAMPLE.COM , що зберігається у /etc/audit/audit.key для проведення власної автентифікації де у якості значення імені хоста(hostname) виступатиме канонічне значення (CN) хоста сервера, що повертається у результаті виконання DNS перегляду за IP адресою вузла.

krb5_key_file визначає розміщення ключа принципіалу клієнті. Варто зазначити, що файл ключа має мати права доступа 0400, власником якого має бути зазначений привілейований користувач. За замовчанням /etc/audit/audit.key.

distribute_network якщо встановлене значення - так, то вихідні події у мережі будуь передані диспетчеру аудиту для обробки. За замовчанням використовується значення ні [12].

Для визначення типу подій що мають фіксуватися у ході роботи системи необхідно виконати попередні налаштування підсистеми аудиту, створивши набір правил, що задовольняє вимогам обраної політики безпеки.

Можуть бути вказані такі типи правил аудиту:

1. ***правила контролю:*** дозволяють відслідковувати зміни поведінки системи та деяких її налаштувань;

2. ***правила файлової системи:*** дозволяють контролювати доступ до файлів та каталогів;

3. ***правила системних викликів:*** дозволяють реєстрацію системних викликів, що виконуються будь-якою вказаною програмою.

Правила можуть бути встановлені двома шляхами: у командному рядку за допомогою застосування утиліти auditctl та за допомогою файлу audit.rules. Перші не зберігаються після перезавантаження операційної системи, в той час як останні, задані у файлі, діють на постій основі.

Розглянемо детальніше особливості встановлення правил, що діють на постійній основі та не є залежними від циклів перезапуску операційної системи. Можливо також застосувати ці правила завдяки прямому використанню augenrules, що виконує зчитування правил, що знадяться у каталозі /etc/audit/rules.d/. Файл audit.rules використовує той самий синтаксис, що і утиліта командного рядка auditctl.

Усі команд, що взаємодіють з демоном аудиту та файлами журналу аудиту перебувають під привілеями суперкористувача. Тому необхідно завчасно переконатися, що запуск відбувається від імені root. Окрім того, необхідно мати привілеї (capabilities) CAP_AUDIT_CONTROL для налаштування служб аудиту, та CAP_AUDIT_WRITE - для реєстрації користувацьких повідомлень [26].

2.4.1. Правила контролю

Для зміни поведінки підсистеми аудиту можна застосовувати такі опції:

1. -b – встановлює максимальну кількість існуючих буферів аудиту у ядрі.

2. -f [0..2] – встановлює дію, що виконується ядром при виявленні критичної помилки. За значення 0 застосовується тихий режим, 1 - printk, 2 - панікування.

3. -e [0..2] – застосовується для відключення підсистем аудиту чи блокування її конфігурацій. 0 використовується для тимчасового вимкнення аудиту. 1 передається у якості аргументу коли вмикається аудит, а для блокування конфігурації аудиту з подальшою неможливістю її зміни застосовують флаг 2. Будь-яка спроба зміни конфігурації аудиту буде перевірена та відхилена. Зміна у конфігурації можлива тільки шляхом перезапуск вузла.

4. -r – встановлює швидкість генерації повідомлень у секунду.

5. -s – повідомляє стан підсистем аудит.

6. -l – застосовується для перелічення усіх завантажених в даний час правил аудиту.

7. -D – використовується для видалення завантажених в даний момент правил аудиту [11].

2.4.2. Правилами файлової системи

Застосування даних правил для контролю доступу до файлів та каталогів допомагає під час визначення спроб несанкціонованого доступу до системи. Наприклад, завдяки відстеженню доступу до файлу /etc/sudoers, що задає політику модуля користувацьких привілеїв. Якщо вказаний у правилі шлях є каталогом, то використовується рекурсивний обхід до кінця дерева каталогів, виключаючи усі каталоги, що є точками монтування у системі. Синтаксис даних правил описується таким шаблоном:

-w шлях до файлу -p права доступу -k ключ-ім'я,

де у якості прав доступу можуть бути використані такі опції:

1. r – для читання файлів;
2. w – для запису у файл;
3. x – для запуску файлів;
4. a – для зміни атрибутів файлу [9].

Ключ ім'я є не обов'язковою оцією, що лише допомагає визначити яке саме правило чи набір правил згенерували запис до журналу аудиту.

Приклади застосування [25]

Для визначення правила, що реєструє всі права на запис та будь-яку зміну атрибутів файлу /etc/passwd, необхідно задати таке правило:

-w / etc / passwd -p wa -k passwd_changes

passwd_changes - довільний рядок, що використовується для ідентифікації правила.

Для визначення правило, що реєструє усі права на запис та кожну зміну атрибутів файлів у каталозі /etc/selinux/, необхідно додати такий запис:

-w / etc / selinux / -p wa -k selinux_changes

Для фіксації виконання будь-якої команди, наприклад, однієї з каталогу з утилітами для системного адміністрування - /sbin/insmod, що виконує вставку модуля у ядро Linux, необхідно задати таке правило:

-w /sbin/insmod -p x -k module_insertion

2.4.3. Робота з правилами системних викликів

Шляхом задавання цих правил виконується перехоплення кожного системного виклику, що виконують програми у системі у системі. Тому принципово важливим є застосування правил лише у тому випадку, коли це є вкрай необхідним, оскільки дані правила сильно впливають на продуктивність роботи системи. Підвищення продуктивності можна досягти завдяки виконанню поєднання декількох правил у одне, якщо це є можливим.

Ядро операційної системи містить п’ять фільтрів відповідності правилам. ***Серед них:*** *задача(task), вхід(entry), вихід(exit), користувач(user) та виключення (exclude).*

Список *задач* перевіряється тільки під час системних викликів fork чи clone, що є породжуючими. Він не часто застосовується на практиці.

Список *входів* використовуються під час кожного входу системного виклику [9].

Список *виходів* також перевіряється за кожного виходу системного виклику. Основна відмінність цих двох списків у тому, що деякі речі можуть бути недоступними та не можуть бути перевіреними, оскільки, наприклад виникають тільки на виході, як от значення входу (exit value), що повертає виклик. Правила для фільтра виходу зустрічаються набагато частіше, перш за все тому, що усі поля є доступними на виході системного виклику. Існує думка, про виключення необхідності застосування вхідних фільтрів через їх невисоке застосування.

Фільтр *користувача* застосовується для фільтрації деяких подій, що відбуваються у просторі користувача. Поля, що є допустимими для використання: uid, auid, gid и pid.

Фільтр *виключення* використовується для виключення опрацювання (emit) певних подій.

Поле msgtype застосовується для того аби повідомити ядру які саме типи повідомлень не потрібно записувати.

Правила системних викликів мають таку загальну форму:

-a дія, список -S системний виклик -F поле = значення -k ключ-і'мя

Опція -a повідомляє компоненті ядра, що займається обробкою правил, що необхідно додати правило у кінець списку правил. Але необхідно задати до якого саме списку відносить правило та яку дію необхідно прийняти під час обробки. У якості дій можуть виступати дві:

1. *always* - завжди створювати подію
2. *never* - ніколи не створювати подію

Дія та список завжди відокремлюються комою без використання пробілів між ними. У якості списку застосовуються значення: task, entry, exit, user, exclude.

Опція -s повідомляє який системний виклик застосовується. Його можна задати за допомогою номера або імені. Кожна архітектура операційної системи має власну таблицю системних викликів. В даній роботі розглядається архітектура x86_64. Приклад таблиці системних викликів даної архітектури приведено у таблиці 2.3 .

Таблиця 2.3. *Приклад таблиці системних викликів для архітектури x86_64* [47]

%rax	***I'мя***	***Точка входу***	***Реалізація***
0	read	sys_read	fs/read_write.c
1	write	sys_write	fs/read_write.c
2	open	sys_open	fs/open.c
3	close	sys_close	fs/open.c
8	lseek	sys_lseek	fs/read_write.c
9	mmap	sys_mmap	arch/x86/kernel/sys_x86_64.c
12	brk	sys_brk	mm/mmap.c
60	exit	sys_exit	kernel/exit.c

Для зручності користувача майже завжди використовується ім'я виклику. Можна задавати більше одного виклику, вказавши іншу опцію -S. Під час відправлення у ядро усі поля системного виклику перетворюють у маску, аби при порівнянні можна було визначити чи необхідно обробляти даний системний

виклик. Таким чином додавання декількох системних викликів у одне правило є дуже ефективним.

Після вказання імені системного виклику auditctl буде виконувати його пошук у відповідному файлі та отримає номер відповідного системного виклику. Проте досить часто це призводить до проблеми через те, що в залежності від архітектури, що застосовується таблиця системних викликів відрізняється. Вирішення цієї проблеми полягає у можливості розбиття одного правила на два та застосування поля -F та вказанням типу архітектури. Наприклад: -F arch = b32, та інше -F arch = b64. Це необхідно розмістити перед опцією -S.

Після вказання системного виклику необхідно використати один або декілька параметрів з -F, що точно задають налаштування з чим саме необхідно виконувати співставлення.

Система аудиту переглядає ідентифікатори як числа без знаків. Якщо застосовується число -1 це трактується як відсутність встановленого логіну. Для коректності написання правила необхідно переглянути файл /etc/login.defs, аби побачити де саме починаються облікові записи користувачів. Наприклад, якщо UID_MIN дорівнює 1000, необхідно, враховуючи беззнакове представлення числа, за якого -1 більше 1000, встановити додатковий фрагмент правила, що має такий вигляд:

-F auid> = 1000 -F auid! = 4294967295

Останнє поле, що використовується у правилах системних викликів являє собою текстовий рядок довільної форми, який за необхідності можна використовувати для вставляння у логи події та подальшої її ідентифікації серед записів у файлі журналу подій аудиту [9].

Приклад застосування:

Для визначення правила, яке створює запис журналу кожного разу, коли файл видаляється або відбувається модифікація назви файлу користувачем системи, ідентифікатор якого 1000 або більше, необхідно застосувати таку команду:

-a always,exit -S unlink -S unlinkat -S rename -S renameat -F auid>=1000 -F auid!=4294967295 -k delete

Параметр -F auid! = 4294967295 використовується для виключення користувачів, для яких не встановлено UID для входу

Для того аби створити правило, що повністю записує виконання програми, наприклад, /bin/id необхідно:

-a always,exit -F exe=/bin/id -F arch=b64 -S execve -k execution_bin_id

Можливо також застосовувати правила системних викликів замість правил файлової системи за допомогою синтаксису правила системного виклику[25].

2.5. Структура записів журналу аудиту

Файли журналу зберігаються у каталозі /var/log/audit/audit.log. Кожен запис події у журналі складається з чотирьох підзаписів, які мають однакову марку часу та порядковий номер. Записи завжди починаються з ключового слова type =. Кожен запис складається з декількох пар ім'я = значення, розділених пробілом або комою. Проведемо детальний аналіз останнього запису з журналу аудиту для вивчення структури запису. Запис журналу має вигляд:

type=SYSCALL msg=audit(1583083123.679:2662): arch=c000003e syscall=316 success=yes exit=0 a0=ffffff9c a1=7ffc8fd3f55f a2=ffffff9c a3=7ffc8fd3f561 items=4 ppid=4109 pid=4121 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts5 ses=2 comm="mv" exe="/usr/bin/mv" subj==unconfined key="rename_file"

type=CWD msg=audit(1583083123.679:2662): cwd="/root"

type=PATH msg=audit(1583083123.679:2662): item=3 name="2" inode=9965541 dev=08:12 mode=0100644 ouid=0 ogid=0 rdev=00:00 nametype=CREATE cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0

type=PROCTITLE msg=audit(1583083123.679:2662): proctitle=6D7600310032

Події з ідентифікатором 2662, що відбулася відповідає шість записів.

type задає тип записаної події. Серед викликів події 2662 наявні такі:

SYSCALL - системному виклику ядра

CWD - запис поточної робочої директорії

PATH - генерується для кожного переданого системному виклику шляху

PROCTITLE - надає повний командний яжой, що було виконано для запуску події аудиту, спричинену системним викликом ядра [46].

Параметр ***msg*** задає час виконання події та її ідентифікатор. Час зазначається відповідно до часу епохи Unix. Усі події, що реєструються з системного виклику одного додатку мають однаковий ідентифікатор. За виконання другого системного виклику надається інший ідентифікатор.

arch задає архітектуру процесора для системного виклику

syscall показує тип системного виклику. Для ідентифікації виклику дані беруться зі списку системних викликів у файлі usr/include/asm/unistd.h та можуть відрізнятися в залежності від архітектури процесора [57]. В даному прикладі виконується виклик 316, що відповідає "renameat2", а саме зміні імені файлу.

success повідомляє чи був успішним системний виклик.

exit значення виходу, що повертається системним викликом [13]

. Для системного виклику renameat2 повернення значення 0 означає успішність завершення операції, -1 у випадку невдачі [63].

a[0..3] перші чотири аргументи системного виклику у шістнадцятковому форматі. Їх значення залежать від того який саме застосовувався системний виклик.

items задає кількість рядків, що передаються у додаток

ppid вказує на ідентифікатор батьківського процесу, що використовується

pid вказує ідентифікатор процесу, що аналізується

auid ідентифікатор аудиту. під час входу користувача процесу надається ідентифікатор. Потім цей ідентифікатор передається будь-якому дочірньому процесу, що запускається початковим процесом користувача. навіть якщо користувач змінює обліковий запис ідентифікатор аудиту залишається без змін.

Таким чином можна відслідковувати дії першого користувача, що зайшов у систему.

uid ідентифікатор користувача що виконав запуск процесу. для користувача root - 0.

gid ідентифікатор групи користувача, що виконав запуск процесу. В даному випадку 0, оскільки привілейований користувач виконував запуск.

euid, suid, fsuid ефективний ідентифікатор користувача, встановлений ідентифікатор користувача та ідентифікатор користувача файлової системи, що виконав запуск процесу

egid, sgid, fsgid ефективний ідентифікатор групи, ідентифікатор встановленої групи та ідентифікатор групи, що використовується ядром для доступу до файлової системи

tty - термінал з якого було виконано запуск додатку

ses - ідентифікатор сеансу входу. Цей атрибут процесу встановлюється коли користувач виконує вхід до системи та може зв’язати будь-який процес з відповідним ім’ям користувача

comm - назва додатку під якою він з’являється у списку задач. В даному випадку під час перейменування файлу застосовувалася команда "mv"

exe - визначений шлях до бінарної програми. В прикладі застосовується бінарний файл, розміщений за таким шляхом: "/usr/bin/mv"

subj - запис, що вказує чи підпорядковується процес контексту безпеки, наприклад, AppArmor. В прикладі процес не підпорядковується жодному контексту, тому значення - unconfined

key - присвоєна назва для правила, необхідна для полегшення пошуку у журналі аудиту

item посилається на аргумент, що пов’язаний з вихідним повідомленням SYSCALL. За наявності більше одного аргументу відбувається додаткова фіксація події PATH

name вказує на ім’я, що передається у якості аргументу системному виклику[57]

inode вказує на номер індексного дескриптора файлу, що відповідає name. У даному прикладі індексний дескриптор рівний 9965541. Індексний дескриптор містить метадані файлу [33]

dev вказує на пристрій на якому зберігається файл.

mode визначає права доступу до файлу. В даному випадку привілейований користувач має права на читання та запис, а група та інші користувачі лише на читання (0100644).

ouid, ogid посилається на UID та GID індексного дескриптора

rdev використовується за доступу до блочних або символьних пристроїв, не файлів.[57]

Висновки до розділу 2

У даному розділі обґрунтовано вибір операційної системи для подальшої розробки способу підвищення безпеки обчислювального вузла на базі механізму аудиту.

1. Огляд повного циклу життя підсистеми, її компонентів та реалізації модуля дозволив виокремити основні недоліки та переваги пов’язані з особливостям реалізації кореневого модуля ядра. Це зумовлює необхідність нівелювання виявлених недоліків. Для цього було додатково розглянуто особливості конфігурації підсистеми аудиту та надано рекомендації щодо оптимізації роботи демона з метою покращення продуктивності роботи системи.

2. Аналіз структури записів журналу аудиту з визначенням призначення кожного з полів вихідних записів, дозволив оцінити можливості, що надає підсистема аудиту та виокремити дані, необхідні для збору та використання в моделі аналізатора.

3. Проведення детального аналізу структури підсистеми аудиту операційної системи на конкретній реалізації з урахуванням особливостей взаємодії з ядром дозволило розробити алгоритм способу підвищення безпеки обчислювальних вузлів, представлений у Розділі 3.

РОЗДІЛ 3

СПОСІБ ПІДВИЩЕННЯ БЕЗПЕКИ ОБЧИСЛЮВАЛЬНОГО ВУЗЛА НА БАЗІ ПІДСИСТЕМИ АУДИТУ ОПЕРАЦІЙНОЇ СИСТЕМИ

3.1. Основні способи впливу на безпеку обчислювальних вузлів

Для розроблення способу підвищення безпеки обчислювальних вузлів необхідним є визначення методів безпосереднього зовнішнього впливу на безпеку системи. Перш за все, реалізація впливу може здійснюватися шляхом використання вразливостей системи з метою порушення цілісності, доступності та конфіденційності оброблюваної інформації. Саме завдяки усуненню вразливостей системи можна запобігти виконанню атаки на кінцеву систему. Проте, оскільки повного усунення вразливостей досягти все одно неможливо, через наявність шансу реалізації атаки нульового дня, додатково мають бути реалізовані інтегровані у систему засоби захисту, здатні до виявлення порушення цілісності роботи системи.

Як попередньо було зазначено, сучасні засоби захисту мають в основі прості та не зовсім ефективні механізми. Для захисту від шкідливого програмного забезпечення активно застосовують сигнатурний аналіз з елементам евристики. Такий підхід не дозволяє виявляти інваріанти шкідливого програмного забезпечення, оскільки є нестійким до обфускації, поліморфізму, а також призводить до появи великої кількості хибних спрацювань. Окрім цього, він є суцільно залежним від оновлення баз даних сигнатур, що необхідна для виявлення загроз. Широкого поширення даний метод набув завдяки швидкості аналізу та спричиненню незначного впливу на продуктивність роботи системи.

Слід зазначити, що концепція сигнатур є ключовою для детермінації шкідливого програмного забезпечення в запропонованому способі. Однак, під сигнатурою будемо розуміти не хеш-суму, отриману шляхом виконання криптографічних операцій над ділянками зловмисного коду, а наявність певної множини системних викликів, необхідних для реалізації окремих видів зловмисного програмного забезпечення. Саме такий підхід дозволить

абстрагуватися від залежності, спричиненої прив’язаністю до структури шкідливого ПЗ. Приведемо детальний структури способу.

3.2. Принцип роботи способу підвищення безпеки

3.2.1. Структура способу

Спосіб підвищення безпеки обчислювальних вузлів базується на застосуванні трьох основних компонент: механізму аудиту операційних систем методів машинного навчання та віртуальних файлових систем.

Основне призначення підсистеми аудиту - отримання детальної інформації щодо стану процесів системи, що відповідають певній політиці, та проведення журналювання будь-яких типів подій в режимі реального часу (спостереження за доступом до файлів, моніторинг системних викликів, запис команд запущених користувачем, запис подій безпеки, пошук подій тощо). Отримання даних відбувається шляхом проходження кожного процесу через систему фільтрів підсистеми аудиту. Схема реалізації викликів з простору користувача приведена на рис. 3.1.

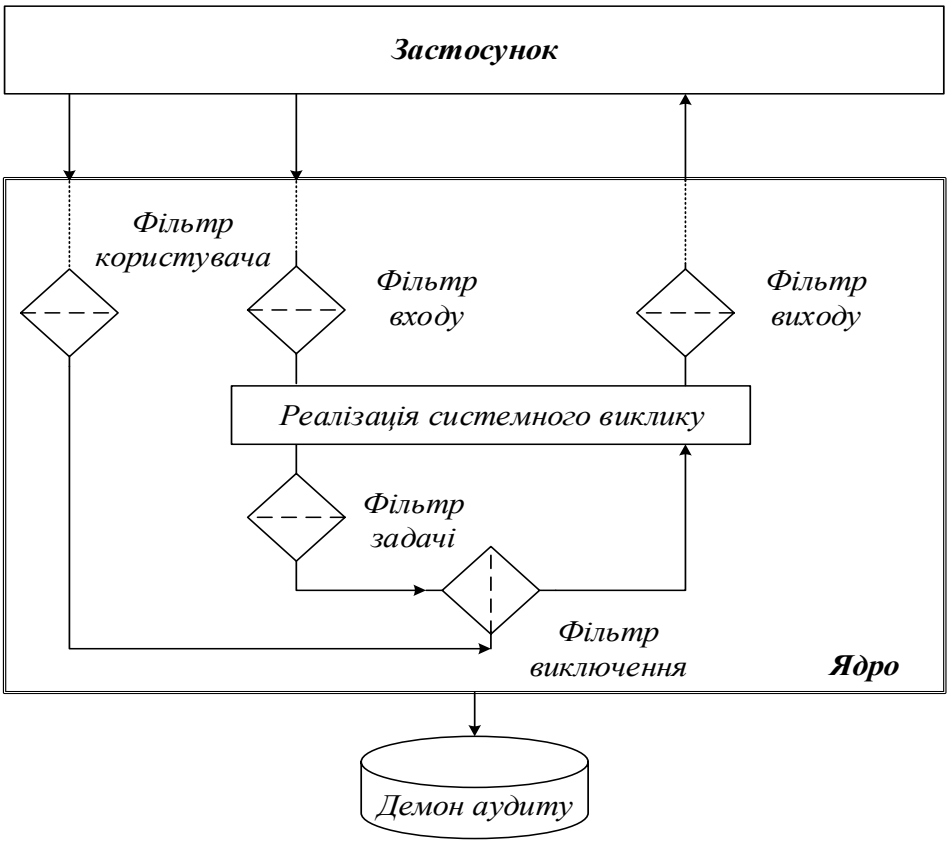


Рис. 3.1. *Схема реалізації викликів з простору користувача*

Наявність прошарку у вигляді модуля ядра аудиту дозволяє побудувати універсальний засіб з перехоплення та керування системними викликами та процесами, а також забезпечує можливість проведення збору даних, необхідних для подальшого застосування в процесі аналізу роботи системи на предмет наявності шкідливої активності. Відповідно до цього, обов'язковим є надання підсистемі аудиту можливості запису таких даних: дата та час події, тип події, ідентифікатор суб'єкта, результат дій; асоціація події з особою користувача, що її спричинив; усі спроби модифікації файлів конфігурації підсистеми аудиту та доступу до логів; використання механізмів автентифікації; змін у довірених джерелах та базах даних; спроби імпорту та експорту інформації. Також важливим є забезпечення можливість здійснення включення або виключення подій, що базується на ідентифікаторі користувача, мітках об'єктів та інших атрибутах.

Для виконання аналізу роботи системи на базі отриманих даних пропонується побудувати аналізатор з використанням засобів машинного навчання, а саме - нейронних мереж. Такий підхід дозволить створити систему, здатну до самонавчання та покращення якості роботи в процесі появи нових загроз, шляхом використання даних з баз знань та накопиченням досвіду. Перевагою такого підходу є відносна стійкість до вразливостей нульового дня.

Перехоплення усіх системних викликів, що відбуваються в ОС, може спричинити значний вплив на продуктивність роботи системи. Тому, для покращення показників використання системних ресурсів пропонується виконувати подальше трасування лише для викликів, що збуджують тригери аналізатора, як показано на рисунку 3.2. Для цього усі точки входу у будь-яке програмне забезпечення мають містити особистий ідентифікатор входу. Також необхідним є створення списку довірених застосунків, які матимуть доступ до файлів баз даних знань та журналів підсистеми аудиту з метою їх оновлення та модифікації(серед них також слід виокремити і службові застосунки, що входять до списку утиліт та демонів безпосередньо необхідних для нормального функціонування системи).

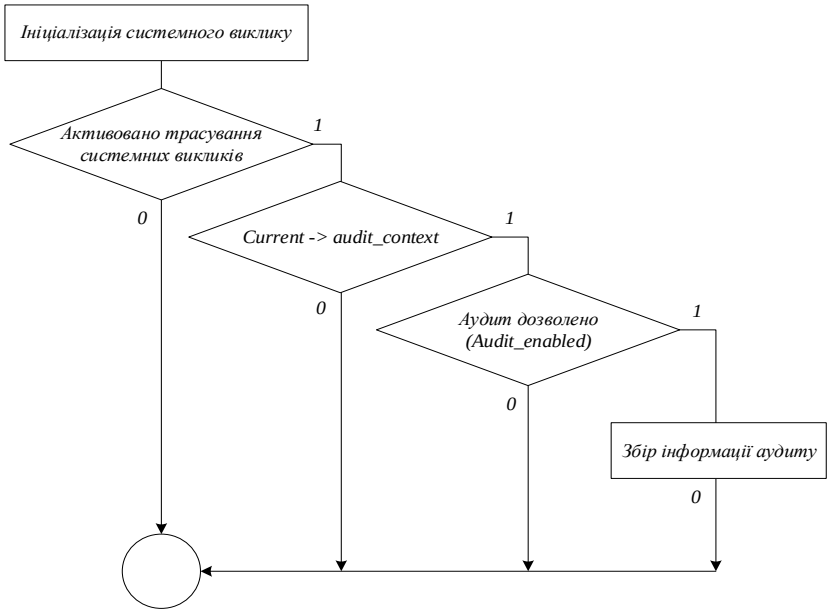


Рис. 3.2. *Схема прийняття рішень щодо збору даних підсистемою аудиту*

За даними наведеної каскадної схеми, визначення необхідності журналювання відбувається лише за умови відповідності поточного суб'єкта політиці аудиту.

Для забезпечення обмеження впливу суб'єктів стороннього програмного забезпечення, пропонується їх ізоляція, шляхом використання віртуальної файлової системи (ФС). Цей крок є необхідним для визначення зловмисності програмного забезпечення. Приведена нижче діаграма (рис. 3.3) демонструє принцип взаємодії простору користувача з різними типами файлових систем.

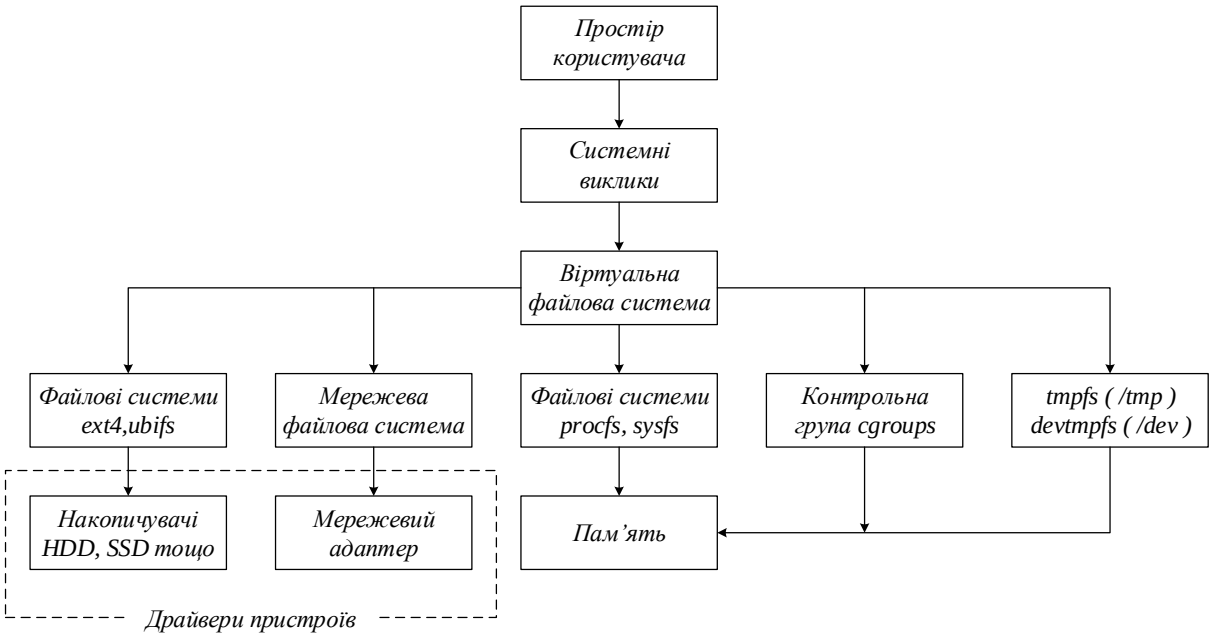


Рис. 3.3. *Принцип взаємодії простору користувача з різними типами ФС*

Абстракція у вигляді віртуальної файлової системи дозволяє користувачу не тільки безпечно взаємодіяти зі сторонніми сутностями, але і надає можливість використання структур даних інших файлових систем. Завдяки проведенню ізоляції на основі часткової віртуалізації файлової системи відбувається відокремлення простору користувача. Це дозволяє запобігти втратам у випадку спроб модифікації даних об’єктів, оскільки відбуватиметься лише зміна дублікатів об’єктів файлової системи всередині обмеженого середовища. Обмеження прав всередині такої системи не дає можливості для проведення атаки на основну систему зсередини віртуальної ФС через інтерфейси операційної системи [44].

Завдяки застосуванню машинного навчання, розроблений спосіб дозволяє проводити не тільки ідентифікацію роботи шкідливого програмного забезпечення в системі, але і надає можливість додавання векторів атак для подальшої детермінації в системі, імітуючи засіб захисту внутрішнього периметру. Цільові об’єкти вторгнення до системи приведені на рис. 3.4.

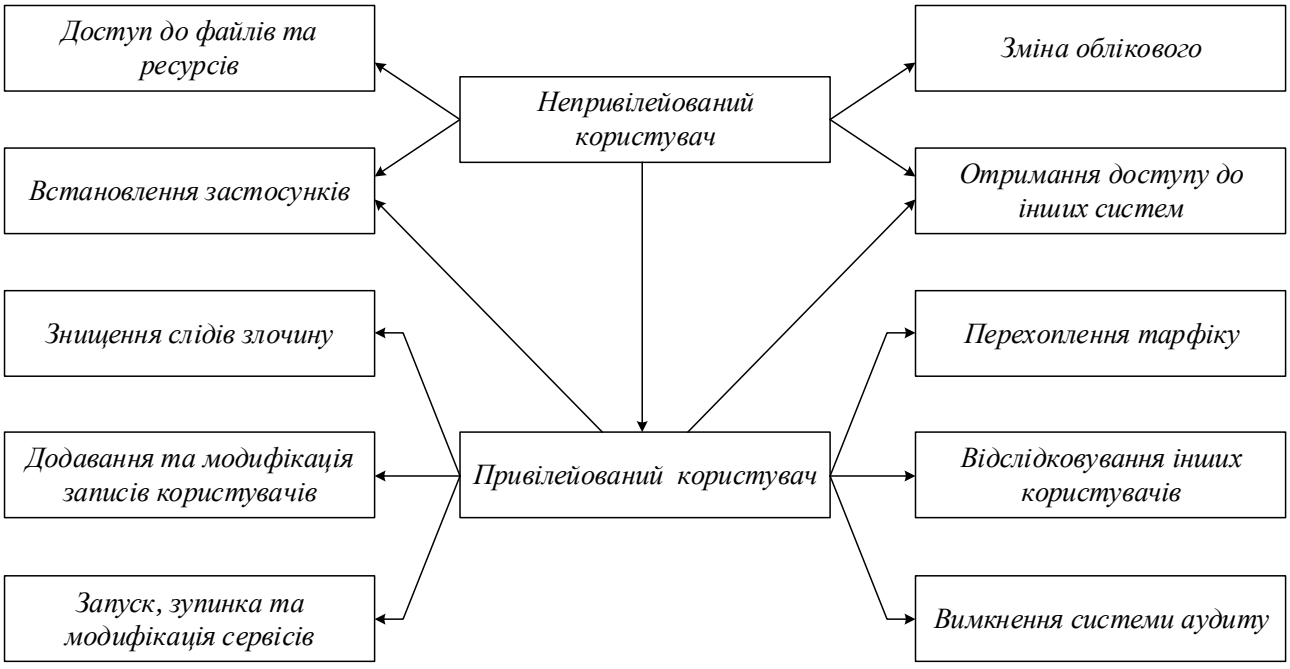


Рис. 3.4. *Цілі вторгнення до інформаційних систем*

Виявлення вторгнень досягається завдяки тому, що, як і у випадку зі ШПЗ, не зважаючи на наявність корисного навантаження, поліморфізму та обфускації,

множина ключових системних викликів вектору атак залишається незмінною, а на результат роботи аналізатора не впливає порядок даних у векторі, важливим є лише контекст. Приклад дерева реалізації вторгнень приведено рис. 3.5.

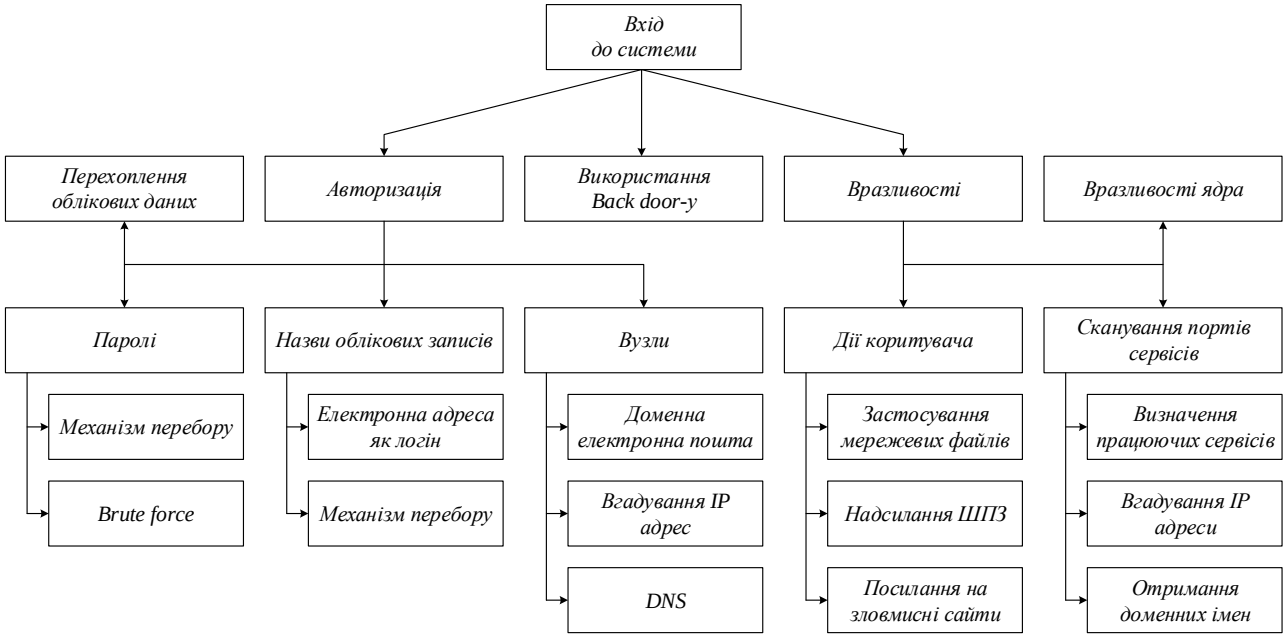


Рис. 3.5. *Основні вектори атак для реалізації вторгнення у систему* [31]

Інтеграція цих трьох компонент можлива у випадку зміни програмного коду відкритого програмного забезпечення модуля аудиту ядра операційної системи та додавання до його механізму розробленого аналізатора та додаткового шару з віртуальної файлової системи.

3.2.2. Опис алгоритму роботи способу

Оскільки спосіб базується на застосуванні трьох компонент (віртуальна ФС, нейромережевий аналізатор, підсистема аудиту) та бази знань, що містить інформацію відносно критичних з точки зору системи викликів (сигнатур ШПЗ), необхідно виконати огляд специфіки реалізації кожного з них в контексті роботи системи.

Підсистема аудиту. Дана компонента завантажується однією з перших під час запуску системи, для того аби не втратити важливі дані, зібрані під час старту роботи системи. За замовчання підсистема перехоплює усі виклики, тому, відповідно до необхідної політики безпеки на заданому вузлі, виконується її

налаштування шляхом задавання набору правил для зменшення навантаження. Під час виконання системного виклику з простору користувача дані надходять одразу до підсистеми аудиту та пропускаються через систему, що складається з п’яти фільтрів, як показано на рис.3.1. Найбільш важливим з точки зору аналізу є фільтр виходу, оскільки він містить усю інформацію щодо системного виклику. Під час виконання системного виклику ядро працює у контексті користувача та може здійснювати перехід у стан призупинення, наприклад, якщо виклик блокується. Можливість переходу до призупиненого стану дозволяє системному виклику використовувати більшу частину функціональних можливостей ядра.

Нейромережевий аналізатор. Отриманий виклик надходить на входи аналізатора (нейронної мережі), що на виході формує скалярне значення: 0 - отриманий виклик не є шкідливим або 1 - виклик належить до класу небезпечних. У випадку наявності позитивної відповіді процес помічається як підозрілий. Для кожного підозрілого процесу створюється власний запис, що дозволяє переглянути послідовність дій з моменту ідентифікації шкідливого виклику. Після цього відбувається повернення виклику до простору користувача з продовженням виконання процесу.

Якщо помічений як підозрілий процес в ході накопичення системних викликів перестає ідентифікуватися як шкідливий, його відслідковування припиняється, а запис видаляється. У іншому разі процес, та усі його дочірні компоненти зупиняються та блокується. Шляхом відслідковування послідовностей дій можна зменшувати хибні спрацювання системи. Можливим також є варіант ізоляції процесу шляхом перенаправлення його до виконання у віртуальній файловій системі. Однак, це може вкрай негативно відобразитися на продуктивності роботи системи.

Віртуальна файлова система. Для запуску застосунків, що не належать до довірених пропонується ізоляція процесу, шляхом запуску у віртуальній файловій системі. Ці дії є необхідними під час першого запуску оскільки таким чином дозволяють мінімізувати втрати. Після запуску застосунків у віртуальній ФС виконується описаний аналіз системних викликів. В разі відсутності масивів

системних викликів, що можуть бути ідентифіковані як шкідливі застосунок помічається як довірений та його процес повертається до основної файлової системи. В зворотному випадку відбувається блокування роботи процесу з відповідним сповіщенням, а дані логу роботи процесу додаються до бази знань, збільшуючи загальну кількість зразків та уточнюючи результати роботи аналізатора.

Блок-сема алгоритму роботи способу підвищення безпеки обчислювального вузла на базі підсистем аудиту операційної системи приведена на рис. 3.6

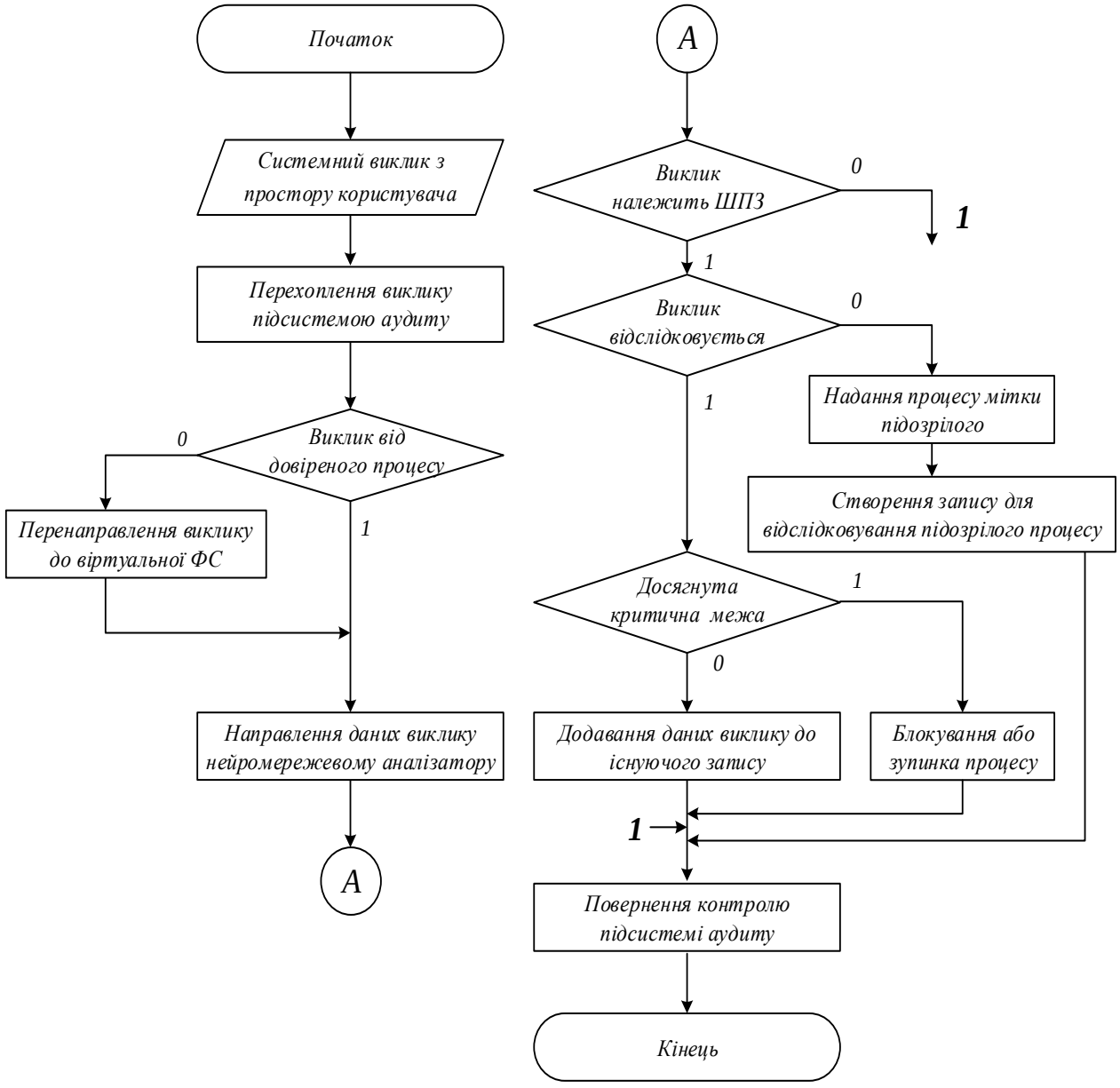


Рис. 3.6. *Блок-сема алгоритму роботи способу підвищення безпеки обчислювального вузла на базі підсистем аудиту операційної системи*

3.2.3. Розробка механізму аналізатора

В основі механізму аналізатора системних викликів покладено застосування методів машинного навчання, що дозволяють виокремити ознаки, необхідні для класифікації об’єктів, відповідно до статистичної структури вхідних даних. Для вирішення задачі детермінації системних викликів було обрано підхід, що широко застосовується під час класифікації текстів, оскільки таким чином можна запобігти впливам порядку слідування даних та визначати шкідливість зразку відповідно до наявного контексту.

Для передачі токенів на входи нейронної мережі необхідно виконати ряд операцій для перетворення початкових даних у числові представлення тензорів. Це перетворення текстових значень називають векторизацією, що може бути виконана у декілька способів:

– шляхом виокремлення n-грам, що відповідають групам з декількох слів або символів;

– розбиттям тексту на слова та перетворенням кожного слова на вектор;

– розбиттям тексту на символи з їх подальшим перетворенням.

Розроблений спосіб не передбачає застосування n-грамного підходу (мішків слів) до заданих послідовностей . Такий інструмент хоч і є досить потужним для конструювання ознак моделями, що використовують логістичну регресію, яка демонструє гарні результати в процесі аналізу системних викликів [22], та все ж обмежений та досить ламкий.

Для того аби пов'язати утворений вектор з відповідним йому токеном можна застосовувати пряме кодування токенів або векторне представлення токенів (слів).

Пряме кодування є одним з найбільш поширених методів, що дозволяє співставляти кожному слову унікальний індекс з наступним цього перетворенням у бінарний вектор необхідної розмірності. Розмірність можна коригувати, обмежуючи кількість елементів, що допомагає уникнути створення занадто широкого простору векторів. Для цього застосовується статистична міра

TF-IDF важливості слова в контексті певного зразка, що є часиною корпусу. Розглянемо детальніше цю міру.

Частота терму (TF) є відношенням числа кількості входжень певного терму до загальної кількості термів в певному зразку, що дозволяє оцінити його важливість в межах окремого зразку. Дана оцінка обчислюється за такою формулою:

$$tf(t,d) = \frac{n_t}{\sum_k n_k}$$

n_t - кількість входження термів в зразок, а велична в знаменнику - загальна кількість слів у всьому зразку .

Зворотня частота документа (IDF) є інверсією частоти, з якою деяке слово зустрічається у зразку корпусу. Врахування даного компонента зменшує вагу загальновживаних термів. Дана характеристика є унікальною для кожного з термів в межах корпусу та визначається відповідно до формули:

$$idf(t,D) = log\frac{|D|}{|\{ d_i \in D \mid t \in d_i \}|}$$

$|D|$ - кількість зразків у корпусі, а в знаменнику вказане число зразків корпусу D, в яких зустрічається t за умови, що n_t не є рівним 0 [64].

Векторне представлення слів також є потужним способом для встановлення зв’язку між вектором та початковим термом завдяки використанню густих векторних слів та векторного представлення слів. Такі представлення є малорозмірними та, на відміну від прямого кодування, представлення слів формуються безпосередньо з даних.

Найбільшою перевагою є можливість розміщення великого об’єму даних інформації в меншій кількості розмірності (рис. 3.7).

Для проведення моделювання роботи способу було застосовано обидва варіанти кодування.

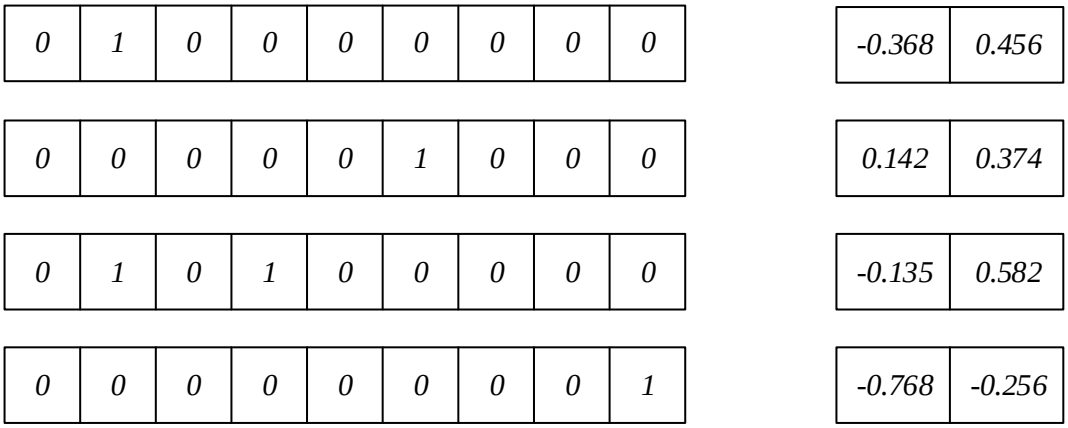


Рис. 3.7. *Порівняння структури векторного та прямого представлення*

Розглянемо детально структуру нейронної мережі, для класифікації системних викликів. У якості моделі нейронної мережі було обрано мережу з прямим поширенням - ***перцептрон Розенблата***.

Навчання у такій мережі здійснюється завдяки застосуванню методу зворотного поширення помилки таким чином, аби мінімізувати середньоквадратичну помилку роботи мережі на навчальній вибірці.

Основними будівельними блоками у нейронних мережах є шари (чи рівні), модулі обробки даних, які можна розглядати як фільтри даних.

На входи рівнів поступають дані (тензори), що після обробки транслюються у більш корисному форматі. Саме на шарах моделі відбувається виокремлення ознак. Ідея глибоко навчання полягає у комбінації декількох простих шарів з метою здійснення послідовної фільтрації ознак. Запропонована мережа має структуру приведену на рис. 3.8.

Створена модель нейронної мережі містить три послідовно з’єднаних шари Dense, що є повнозв’язними: перші два місять однакову кількість нейронів, а вихідний шар містить лише один нейрон, оскільки реалізує бінарну класифікація

.

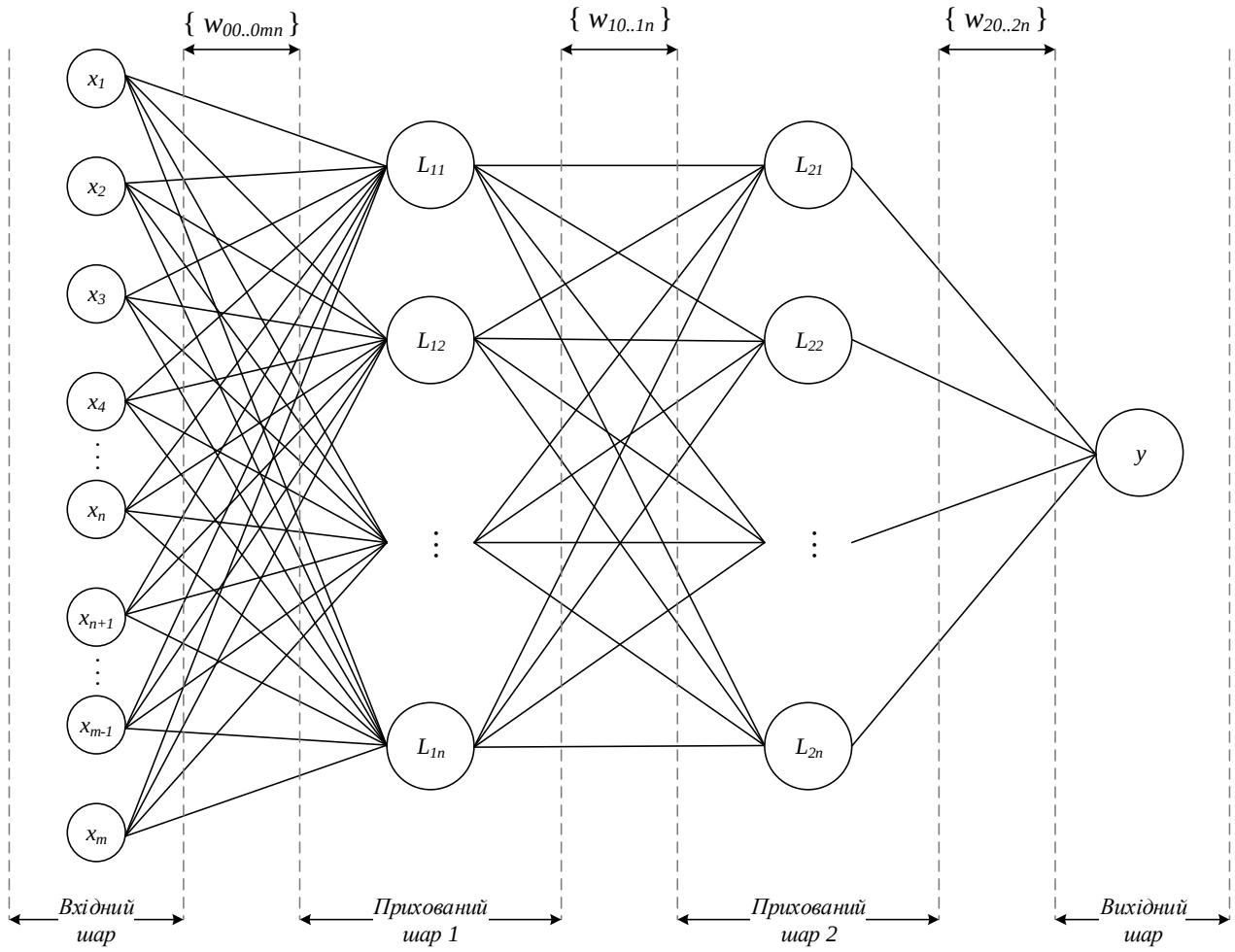


Рис. 3.8. *Структура нейронної мережі для класифікації системних викликів*

Для ***оптимізації функції помилки*** під час навчання застосовується стохастичний градієнтний спуск, що дозволяє апроксимувати значення градієнта градієнтом функції вартості, що вираховується для невеликого набору навчальних зразків (mini batch). Після цього параметри змінюються, пропорційно наближеному градієнту. Такий підхід надає значну перевагу відносно швидкості навчання в порівнянні зі стандартним градієнтним спуском.

Приведемо алгоритм методу стохастичного градієнта:

Вхідні дані:

X_m – навчальна вибірка ;

η – темп навчання;

r - темп забування;

Вихідні дані:

Синаптчні ваги $w_j, j = 1, 2, \dots, n$;

Хід роботи алгоритму:

1. Ініціалізувати ваги: $w_j, j = 1, 2, \dots, n$;

2. Ініціалізувати поточну оцінку функціонала якості:

$$Q(w; X_m) = \sum_{i=1}^{m} \lambda(\langle w, x_i \rangle, y_i)$$

3. Повторювати:

- вибрати об'єкт x_i з X_m
- обчислити вихідне значення алгоритму *a* (x_i, *w*):

$$a(w, x_i) = \varphi\left(\sum_{j=1}^{n} w_j x_i^j - w_0\right)$$

- обчислити помилку

$$\varepsilon_i := \lambda(\langle w, x_i \rangle, y_i)$$

- зробити крок градієнтного спуску

$$w^{(t+1)} := w^{(t)} - \eta\lambda`\left(\langle w^{(t)}, x_i \rangle, y_i\right)$$

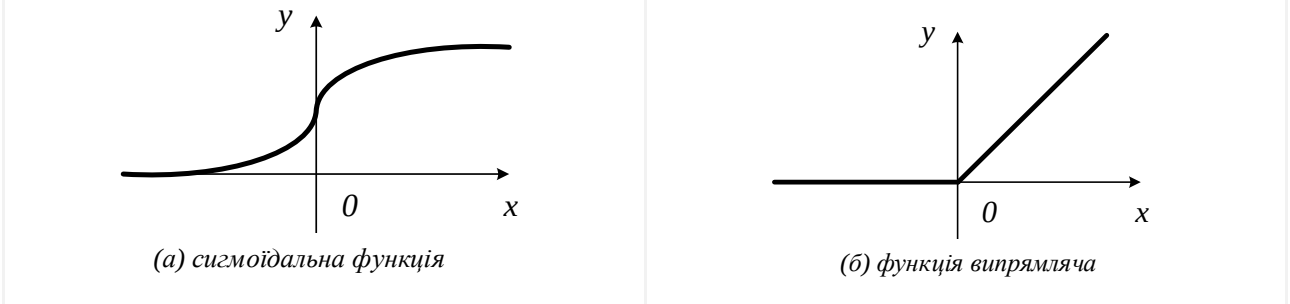
- оцінити значення функціонала:

$$\overline{Q} := (1 - r)\overline{Q} + \lambda\varepsilon_i$$

4. Поки значення $\overline{Q}$ функціонала не стабілізується і/або ваги не перестануть змінюватися;

Параметр можна покласти рівним $\frac{1}{m}$. У випадку надлишково довгої вибірки його рекомендується збільшувати.

У якості функції активації нейрону застосовано: на вхідному та прихованому шарі - випрямляч (ReLU), а на вихідному шарі обрано сигмоїдальну (Sigmoid) функцію. Графіки функцій активації наведено на рисунку 3.9.



(а) сигмоїдальна функція *(б) функція випрямляча*

Рис. 3.9. *Функції активації нейронів мережі*

Функція випрямляча здобула найбільшого поширення серед усіх функцій активації завдяки своїй простоті. Реалізуючи звичайний пороговий перехід у нулі, вона має таку формулу:

$$f_{ReLU}(x) = max(0, x).$$

Застосування цієї функції на внутрішніх шарах містить такі переваги:

1. На відміну від виконання ресурсомістких операцій з обчислення гіперболічного тангенса та сигмоїдальної функцій, *ReLU* може бути реалзіований за допомогою звичайного порогового перетворення матриці активації у нулі, значно зменшуючи час обчислень.

2. *ReLU* не піддається насиченню, а отже не впливає негативно на якість навчання.

Основним недоліком функції випрямляча є його недостатня надійність під час застосування у процесі швидкого навчання. Це спричинено можливостю раптового виходу нейронів з ладу (їх деактивації). Проходження великого градієнта через функцію випрямляча зумовлює кардинальну зміну ваг, призводячи до того, що нейрон втрачає здатність активуватися та містить постійний нуль на виході, здійснюючи опосередкований вплив на роботу інших нейронів. Таким чином, загальна кількість нейронів, що можуть бути деактивовані, складає близько 40% від загальної кількості у випадку надто високої швидкості навчання. Проте, цьому можна досить легко запобігти шляхом вибору правильних значень коефіцієнта швидкості навчання [32].

Сигмоїдальна функція застосовується досить рідко через наявність значних недоліків. На вхід функції надходять довільні дійсні числа. В результаті роботи функції на її виході формується дійсне число в інтервалі від 0 до 1. Великі за модулем значення від'ємних чисел перетворюються на 0 , а додатні значення - на 1. Формула функції:

$$f_{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

Застосування сигмоїдальної функції спричинено легкістю інтерпретації даних виходу функції, як рівня активації нейрона: від відсутності активації - 0, до повного насичення активації - 1.

Основною негативною якістю сигмоїдальної функції є її насичення. За насичення функції зі сторони 0 чи 1 значення градієнта стає близьким до нуля, що призводить до згасання градієнтів. Під час зворотного поширення помилки даний локальний градієнт множиться на загальний градієнт, спричинюючи неможливість проходження сигналу через нейрон до його ваг та даних.

Початкові ваги сигмоїдних нейронів мають бути не занадто великими аби запобігти насиченню, інакше мережа буде дуже погано навчатися. Також негативною якістю є відсутність центрування сигмоїди відносно нуля. Оскільки нейрони у наступних шарах отримують не центровані значення, це впливає на динаміку градієнтного спуску. Якщо значення, що надходить до нейрона є завжди додатним , то в процесі зворотного поширення помилки усі градієнти ваг будуть або додатними, або від'ємними, що призводить до небажаної зигзагоподібної динаміки оновлення ваг. Однак, слід також відмітити, що за підсумовування цих градієнтів, підсумкове оновлення ваг може мати різноманітні знаки, що дозволяє нівелювати даний недолік функції. Таким чином, відсутність центрування має не досить серйозні наслідки [62].

3.3. Моделювання способу підвищення безпеки обчислювального вузла на базі механізму аудиту

Для проведення моделювання роботи способу необхідним є попереднє виконання таких дій:

1. Збір зразків шкідливого програмного забезпечення
2. Підготовка ізольованого віртуального середовища з метою проведення трасування ШПЗ
3. Автоматизація збору та обробки даних трасування
4. Підготовка тестового та тренувального набору даних
5. Побудова моделі механізму аналізатора
6. Навчання моделі аналізатора на підготовленому наборі даних

Після завершення підготовчого етапу можна переходити до аналізу роботи об’єктно-орієнтованого способу та проводити його порівняння з іншими реалізованими моделями.

Розробка моделі роботи способу відбувалася на мові програмування Pyhton із застосуванням бібліотеки машинного навчання Keras та безкоштовної платформи для розробки проектів з ML - GoogleColaboratory.

3.3.1. Підготовка наборів даних

Для створення тренувального та тестового наборів даних було виконано збір зразків шкідливого програмного забезпечення в мережі Інтернет. Оскільки у відкритому доступі відсутні бази даних ШПЗ, пошук здійснювавсяя на публічних платформах. Загалом було зібрано **110** зразків різноманітного ШПЗ.

У якості безпечного програмного забезпечення (БПЗ) було обрано нявні вбудовані за замовчанням у системі застосунки. Кількість зібраних зразків безпечного програмного забепечення становила **337** одиниць.

Для підготування ізольованого віртуального середовища, необхідного для запуску зібраних зразків ШПЗ та БПЗ, було інстальовано відкрите програмне забезпеченя у вигляді системи віртуалізації - Vbox.

Для автоматизації збору даних в ході виконання ШПЗ та БПЗ у ізольованому середовищі було розоблено bash-скрипт та bat файл. Даний bat файл дозволив виконувати автоматичний запуск та запис логів системних викликів у відповідні до типу забезпечення каталоги з необхідним ступенем деталізації події. Bash-скрипт було застосовано з метою очищення файлів логів від надлишкових даних. В результаті роботи такої підсистеми автоматизованого збору даних було зібрано кількість системних викликів, наведену у таблиці 3.1.

Таблиця 3.1. *Кількість системних викликів в зібраних даних*

Загальна кількість викликів	**Кількість викликів БПЗ**	**Кількість викликів ШПЗ**
6 233 791	*917 272*	*5 316 519*

В результаті проведення такого ряду маніпуляцій з даними було отримано два каталоги (benign та malware) з логами кожного з виконаних застосунків. Проте, отримані файли не можуть застосовувався у розробленій моделі, доки не будуть інтерпретовані як числа. Отже, необхідним є проведення векторизації. Аби пов'язати утворений вектор з відповідним йому токеном було виконано застосовування прямого кодування токенів.

Для реалізації прямого кодування необхідно виконати зчитування кожного з файлів по порядку з подальшим створенням двовимірного масиву, що міститиме масиви системних викликів відповідних зразків з мітками їх приналежності до бінарного класу. Далі, проводиться частотний аналіз усіх наявних системних викликів, в результаті якого формується впорядкований за спаданням кількості разів зустрічання викликів масив. Після формування такого масиву можна виконати обхід матриці з метою заміни назви кожного з системних викликів на відповідний йому індекс з щойно утвореного частотного масиву. Отримання такого двовимірного масиву, закодованого відповідно до прямого кодування дозволяє здійснити розподіл у відношенні 3:1 даних на тренувальний та тестовий набори, та виконати подачу даних на входи аналізатора.

3.3.2. Розробка програмної моделі аналізатора

Для розробки програмної моделі аналізатора було застосовано бібліотеку Keras, що надає високорівневі будівельні блоки для конструювання моделей глибокого навчання. Низькорівневі операції, як, наприклад, маніпуляція з тензорами та диференціювання реалізовані в низькорівневій бібліотеці, що підключається модулем до Keras - TensorFlow. Застосування TensorFlow дозволяє виконувати обчислення на не тільки на CPU але і на GPU [30].

Для зручності завантаження даних до моделі надано інтерфейс для вибору файлу з логами (рис. 3.10).

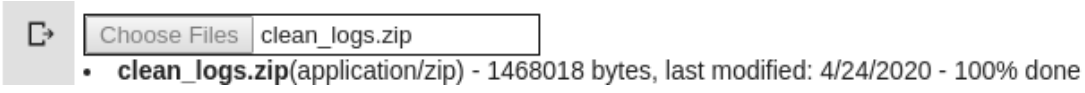


Рис. 3.10. *Інтерфейс завантаження файлу даних*

Приведемо структуру нейронної мережі (рис. 3.11.), що створюється під час моделювання роботи способу і виступає у якості аналізатора системних викликів. Функція model.summary надає детальну інформацію щодо створеної структури. А саме інформацію щодо того, що на кожному шарі експериментальної системи знаходиться 128 нейронів, а на результуючому - 1.

```
model.summary()

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 128)               68224
_________________________________________________________________
dense_4 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 84,865
Trainable params: 84,865
Non-trainable params: 0
```

Рис. 3.11. *Структура нейронної мережі аналізатора*

На вхід аналізатора подаються перетворені дані в форматі прямого кодування(рис. 3.12), що зберігаються у масивах визначеної довжини. В даному випадку довжина масиву - 532. Саме така кількість різноманітних системних викликів застосовувалися під час виконання усіх зразків зібраного програмного забезпечення.

```
[1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 0. 0. 0. 0. 1. 0. 1. 1. 1. 1. 0. 0. 0. 0.
 0. 1. 0. 1. 0. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1.
 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.
 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1. 0. 1. 1. 0. 1. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 1. 1.
 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

Рис. 3.12. *Представлення даних у прямому кодуванні*

Для визначення коректності роботи аналізатора застосовується метод вбудованих циклів оцінки та навчання model.evaluate (рис. 3. 13).

```
scores = model.evaluate(x_test, y_test, verbose=1)
4/4 [==============================] - 0s 2ms/step - loss: 0.0071 - accuracy: 1.0000
```

Рис. 3.13. *Реалізація перевірки коректності роботи аналізатора*

3.3.3. Тестування роботи способу

З метою перевірки можливості реалізації об'єктно-орієнтованого способу підвищення безпеки обчислювального вузла на базі підсистеми аудиту операційної системи необхідно виконати запуск моделі аналізатора. Результати роботи моделі на створених наборах даних наведено на рис. 3.14.

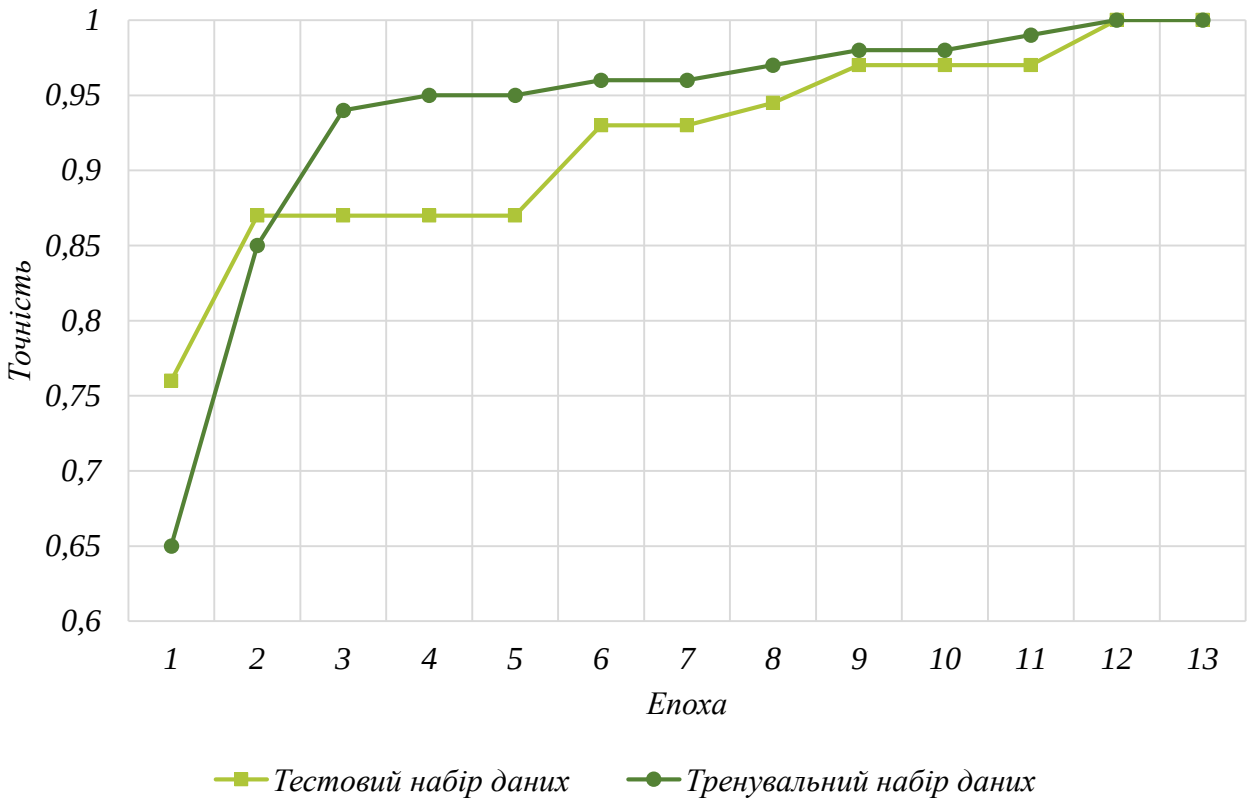


Рис. 3.14. *Результати роботи моделі на створених наборах даних*

Отриманий в процесі роботи графік демонструє коректність роботи запропонованого способу, оскільки відповідно до результатів на тестовому наборі точність розпізнавання ШПЗ та БПЗ зростає з 70% до 100%. Для тестування роботи моделі можна було обрати меншу кількість епох, оскільки, як

видно з графіка система навчається приблизно за 10 епох. Остаточна перевірка на тестовому прикладі (рис. 3.13) моделі свідчить про правильність напрямку проведеного дослідження.

Було також додатково проведено тестування розробленої моделі з даними, що відповідають формату густих векторних представлень. Це спричинило необхідність внесення додаткових змін у структуру самої мережі. Відповідно до отриманих даних, система навчається менш швидко ніж за першого способу (рис. 3.15).

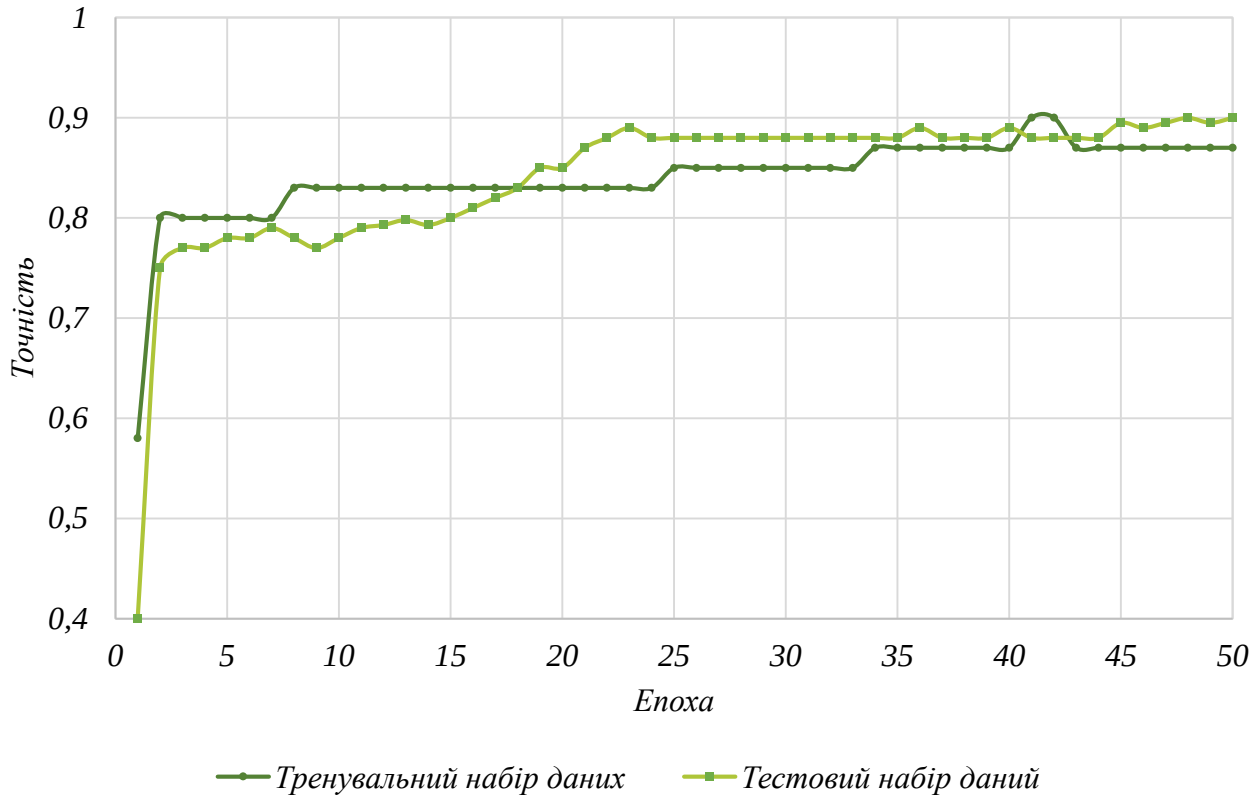


***Рис. 3.15**. Результати роботи моделі на створених наборах даних за застосування векторного представлення у якості кодування даних*

Проте, такий підхід дозволив інтерпретувати розподіл векторів у просторі (рис. 3.16), демонструючи наявність геометричного відношення між ними, що свідчить про наявність зв’язку між викликами.

Наявність на графіку досить щільного розміщення точок свідчить про те, що виклику, що потрапили до даної множини є граничними, тобто активно застосовуються як у ШПЗ, так і у БПЗ. Найбільш важливими з точки зору

ідентифікації шкідливої активності є точки на графіку, розташовані на периферії від центру.

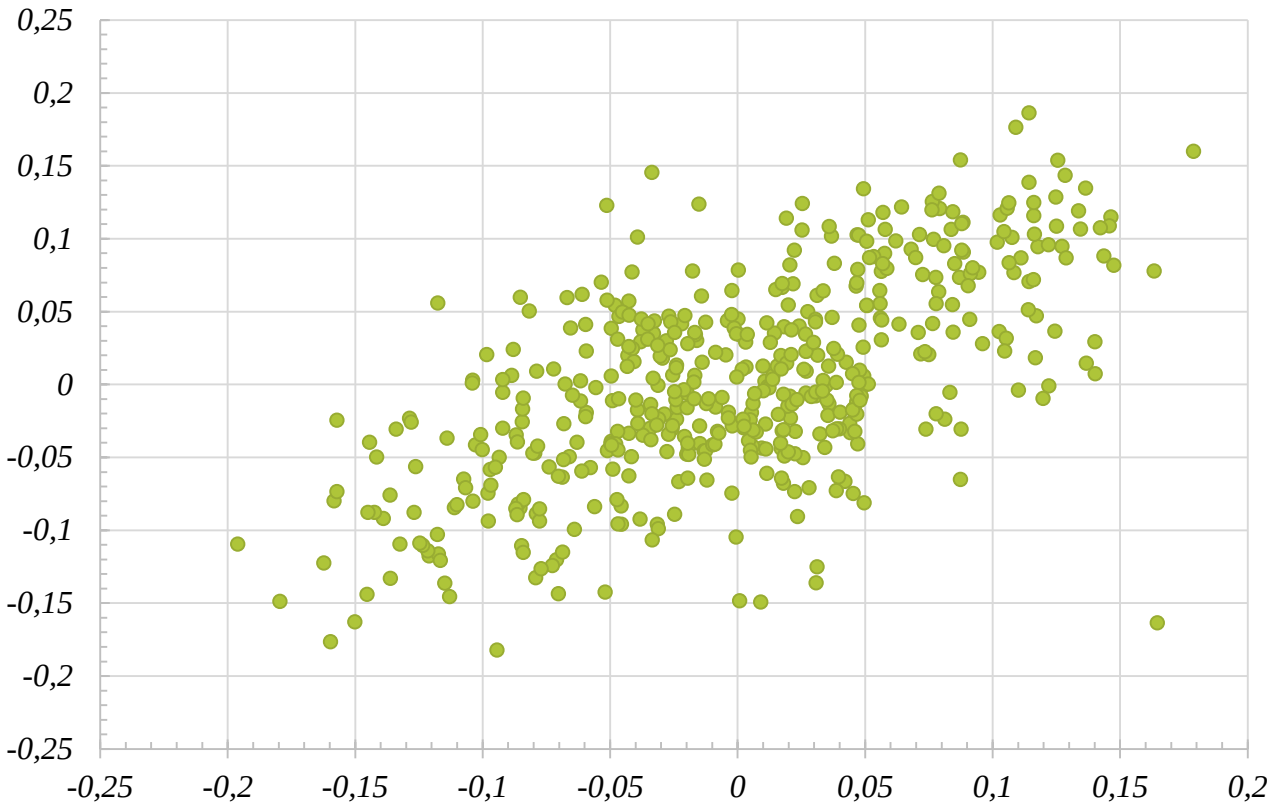


Рис. 3.16. *Отриманий векторний простір системних викликів*

У випадку збереження отриманого векторного представлення його можна буде надалі застосовувати для демонстрації наявності відношень між компонентами вибірки без необхідності повторного вирішення задачі.

Висновки до розділу 3

В результаті проведення моделювання роботи способу було отримано позитивні результати з розпізнавання загроз, шляхом аналізу записів системних викликів, отриманих під час роботи шкідливих та безпечних зразків.

Застосування двох видів кодування даних, що подаються на входи аналізатора дозволило провести порівняльний аналіз, в результаті якого визначено:

1. Формат представлення даних, а саме тип кодування значно впливає на швидкість навчання нейромережевого аналізатора. В процесі аналізу великих об’ємів даних, що є розрідженими масивами, спостерігається зниження ефективності роботи системи. З цього витікає необхідність застосування компактних способів представлення вхідних даних. Одним з варіантів вирішення даної проблеми може бути застосування густих векторів.

2. Зміна структури нейронної мережі та правильний вибір функцій активації дозволяють покращити точність розпізнавання та ефективність роботи системи. В ході моделювання з застосуванням прямого кодування підвищення швидкості навчання було зумовлене використанням моделі нейронної мережі у вигляді багатошарового перцепторна з повнозв’язних шарів та використанням функції активації ReLU.

3. В ході аналізу спосіб векторного представлення, що застосовується для виявлення семантичних залежностей в текстах, дозволив виявити наявність залежностей між системними викликами, завдяки чому було отримано простір їх векторного представлення.

Отримані в ході моделювання результати свідчать про коректність висунутих припущень.

ВИСНОВКИ

В ході реалізації дослідження за темою магістерської дисертації було досягнуто поставленої мети з підвищення безпеки обчислювальних вузлів шляхом вирішення таких задач:

1. Дослідження принципу роботи сучасних засобів захисту та особливостей реалізації шкідливого програмного забезпечення надали можливість виокремити ключові недоліки сучасних способів виявлення загроз та розробити власний спосіб, що базується на застосуванні підсистеми аудиту та методів глибокого навчання з метою нівелювання нестійкості до вразливостей нульового дня.

2. Проведення аналізу структури підсистеми аудиту з метою застосування її як основи для розробки засобів захисту інформації дозволило виявити недоліки, які мають бути попередньо усунені, а також надало можливість розробити алгоритм способу з урахуванням особливостей реалізації взаємодії підсистеми аудиту з ядром операційної системи.

3. В ході розробки алгоритму виявлено компоненти, що впливають на продуктивність роботи системи та вимагають подальшого доопрацювання з метою зниження навантаження на систему, оскільки передбачається робота способу у режимі реального часу.

4. Моделювання роботи системи дало змогу оцінити реальний вплив представлення даних та особливостей реалізації структури нейронної мережі для розпізнавання загроз. Результати моделювання дозволили підтвердити гіпотезу відносно можливості застосування векторного представлення даних під час проведення проектування структури аналізатора.

5. Перевірка коректності роботи здійснювалася на сторонніх зразках. Відповідно до даних перевірки розпізнавання шкідливих зразків відбувалося коректо. Для оцінки коректності було застосовано хмарні платформи для аналізу ШПЗ, що надають свідчення відносно зловмисності зразків.

Отримані результати дослідження свідчать про можливість застосування запропонованого способу як основного механізму у сучасних системах захисту.

Розроблений спосіб має свої переваги та недоліки. Перевагою даного способу є використання вбудованого засобу у вигляді підсистеми аудиту у якості системи з перехоплення та контролю подій, завдяки чому непотрібно розробляти додатковий механізм взаємодії з системними викликами. Головним недоліком слід вважати підвищення навантаження на продуктивність роботи системи, а отже необхідним є подальше проведення досліджень, які б дозволили усунути цей недолік та оптимізувати роботу системи захисту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 111+ Linux Statistics and Facts – Linux Rocks! [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://hostingtribunal.com/blog/linux-statistics/#gref.

2. ASLAN Ö. A Comprehensive Review on Malware Detection Approaches / Ö. ASLAN, R. SAMET. // IEEE Access. – 2020. – №8. – С. 6249–6271.

3. Arora A. Permpair: Android malware detection using permission pairs / A. Arora, K. Peddoju, M. Conti. // IEEE Transactions on Information Forensics and Security. – 2019.

4. Souri A. A state-of-the-art survey of malware detection approaches using data mining techniques / A. Souri, R. Hosseini. // Hum.-Centric Comput. Inf. Sci. – 2018. – №8. – С. 3.

5. Malware [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.av-test.org/en/statistics/malware.

6. Bates J. Analysis of Computer Audit Data to Create Indicators of Compromise for Intrusion Detection / J. Bates, S. Millett, M. Toolin. // DATASCIENCEREVIEW. – 2019. – №2.

7. Grubb S. AUDISPD:(8) [Електронний ресурс] / Steve Grubb. – 2007. – Режим доступу до ресурсу: http://man7.org/linux/man-pages/man8/audispd.8.html.

8. Grubb S. audisp-remote [Електронний ресурс] / Steve Grubb – Режим доступу до ресурсу: https://linux.die.net/man/8/audisp-remote.

9. Grubb S. audit.rules [Електронний ресурс] / Steve Grubb – Режим доступу до ресурсу: https://linux.die.net/man/7/audit.rules.

10. Grubb S. auditctl [Електронний ресурс] / Steve Grubb – Режим доступу до ресурсу: https://linux.die.net/man/8/auditctl.

11. Kolotov A. auditctl(8) [Електронний ресурс] / Alexander Kolotov. – 2007. – Режим доступу до ресурсу: https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=auditctl&category=8.

12. auditd.conf [Електронний ресурс] – Режим доступу до ресурсу: https://linux.die.net /man/ 8/auditd.conf.

13. Yemelianov A. Auditing System Events in Linux [Електронний ресурс] / Andrej Yemelianov. – 2017. – Режим доступу до ресурсу: https://blog.selectel.com/auditing-system-events-linux/.

14. Grubb S. aulast [Електронний ресурс] / Steve Grubb – Режим доступу до ресурсу: https://linux.die.net/man/8/aulast.

15. Kolotov A. aureport(8) [Електронний ресурс] / Alexander Kolotov. – 2007. – Режим доступу до ресурсу: https://www.opennet.ru/cgi-bin/opennet/man.cgi ?topic=aureport &category=8.

16. Шафоростов Н. ausearch(8) [Електронний ресурс] / Николай Шафоростов. – 2007. – Режим доступу до ресурсу: https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=aus earch&category=8.

17. Grubb S. AUTRACE [Електронний ресурс] / Steve Grubb // Linux Programmer's Manual. – 2007. – Режим доступу до ресурсу: http://man7.org/linux/man-pages/man8/autrace.8.html.

18. Raghuraman C. Static and dynamic malware analysis using machine learning / C. Raghuraman, S. Suresh, S. Shivshankar. // First International Conference on Sustainable Technologies for Computational Intelligence. – 2020. – С. 793–806.

19. Elisan C. Malware, Rootkits & Botnets A Beginner’s Guide / C. C. Elisan., 2012. – 385 с

20. Casey E. Malware Forensics: Investigating and Analyzing Malicious Code / E. Casey, J. Aquilina, C. Malin. // Syngress. – 2008.

21. Vaughan-Nichols S. Can the Internet exist without Linux? [Електронний ресурс] / Steven J. Vaughan-Nichols. – 2015. – Режим доступу до ресурсу: https://www.zdnet.com/article/can-the-internet-exist-without-linux/.

22. Canzanese R. System Call-Based Detection of Malicious Processes / R. Canzanese, S. Mancoridis, K. Moshe. // Software Quality, Reliability and Security (QRS), IEEE International Conference. – 2015. – С. 119–124.

23. CAPABILITIES(7) [Електронний ресурс] // Linux Programmer's Manual. – 2019. – Режим доступу до ресурсу: http://man7.org/linux/man-pages/man7/capabilities.7.html

24. CHAPTER 6. SYSTEM AUDITING [Електронний ресурс] – Режим доступу до ресурсу: https://access.redhat.com/documentation/en-us/red_hat_ enterprise_linux /7/html/security_guide/chap-system_auditing.

25. 6.5. DEFINING AUDIT RULES [Електронний ресурс] – Режим доступу до ресурсу: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux /7/html/security_guide/sec-Defining_Audit_Rules_and_Controls#sec-Defining_Audit _ Rules_and_Controls_in_the_audit.rules_file.

26. 6.5. DEFINING AUDIT RULES [Електронний ресурс] – Режим доступу до ресурсу: https://access.redhat.com/documentation/en-us/red_ hat_enterprise_linux /7/html/security_guide/sec-Defining_Audit_Rules_and_Controls#bh-defining_contro ls_with_auditctl.

27. Eilam E. Reversing: secrets of reverse engineering / E. Eilam, E. Chikofsky. – Indianapolis: Wiley, 2005.

28. Event Tracing [Електронний ресурс] // 2018 – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/windows/win32/etw/event-tracing-portal.

29. F. Thomas, Adware: The Only Book You'll Ever Need. Lulu Press,Inc, 2015

30. Francois Chollet. 2017. Deep Learning with Python (1st. ed.). Manning Publications Co., USA

31. Grubb S. Audit and IDS [Електронний ресурс] / Grubb. – 2008. – Режим доступу до ресурсу: https://github.com/linux-audit/audit-documentation/blob/ master/presentations/2008-rh_summit/audit-ids.pdf.

32. Сверточная нейронная сеть, часть 1: структура, топология, функции активации и обучающее множество [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://habr.com/ru/post/348000/.

33. INODE(7) [Електронний ресурс] // Linux Programmer's Manual. – 2019. – Режим доступу до ресурсу: man7.org/linux/man-pages/man7/inode.7.html.

34. Gatlan S. IoT Attacks Escalating with a 217.5% Increase in Volume [Електронний ресурс] / Sergiu Gatlan. – 2019. – Режим доступу до ресурсу: https://www.bleepingcomputer.com/news/security/iot-attacks-escalating-with-a-2175 -percent-increase-in-volume/.

35. IT threat evolution Q3 2019. Statistics [Електронний ресурс] / [V. Chebyshev, F. Sinitsyn, D. Parinov та ін.]. – 2019. – Режим доступу до ресурсу: https://securelist.com/it-threat-evolution-q3-2019-statistics/95269/.

36. Bachaalany E. The antivirus hacker's handbook / E. Bachaalany, J. Koret. – Indianapolis: John Wiley & Sons, 2015.

37. Alzarooni K. Malware variant detection : дис. канд. техн. наук / Alzarooni – London, 2012.

38. Prakash A. Linux Runs on All of the Top 500 Supercomputers, Again! [Електронний ресурс] / Abhishek Prakash. – 2020. – Режим доступу до ресурсу: https://itsfoss.com/linux-runs-top-supercomputers/.

39. A survey on automated dynamic malware-analysis techniques and tools / M.Egele, T. Scholte, E. Kirda, C. Kruegel. // ACM Comput. Surv. CSUR. – 2012. – №44. – С. 6.

40. Sikorski M. Practical Malware Analysis / M. Sikorski, A. Honig. // Network Security. – 2012. – №12. – С. 4–12.

41. Rouse M. Malware (malicious software) [Електронний ресурс] / Margaret Rouse // Techtarget – Режим доступу до ресурсу: https://searchsecurity .techtarget.com/definition/malware.

42. Aslan Ö. Investigation of possibilities to detect malware using existing tools / Ö. Aslan, R. Samet. // IEEE/ACS 14th Int. Conf. Comput. Syst. Appl. (AICCSA). – 2017.

43. Operating System Market Share Worldwide [Електронний ресурс] // Statcounter. – 2020. – Режим доступу до ресурсу: https://gs.statcounter.com/os-market-share.

44. IRabinovich. Антивирусные песочницы. Введение [Електронний ресурс] / IRabinovich. – 2010. – Режим доступу до ресурсу: https://habr.com/ru /post/105581/.

45. RHEL Audit System Reference. Audit-record-types [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://access.redhat.com/articles/ 4409591

46. RHEL Audit System Reference. Audit-record-types [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://access.redhat.com/articles/4409591 #audit-record-types-2.

47. Searchable Linux Syscall Table for x86 and x86_64 [Електронний ресурс] – Режим доступу до ресурсу: https://filippo.io/linux-syscall-table/.

48. Security-Enhanced Linux [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Security-Enhanced_Linux.

49. Sobers R. 10 Must-Know Cybersecurity Statistics for 2020 [Електронний ресурс] / Rob Sobers. – 2020. – Режим доступу до ресурсу: https://blogvaronis2.wpengine.com/cybersecurity-statistics/.

50. Moore P. SPEC Audit System State Diagram [Електронний ресурс] / Paul Moore. – 2016. – Режим доступу до ресурсу: https://github.com/linux-audit/audit-documentation/wiki/SPEC-Audit-System-State-Diagram.

51. Grubb S. SPEC System Lifecycle Events [Електронний ресурс] / Steve Grubb. – 2017. – Режим доступу до ресурсу: https://github.com/linux-audit/audit-documentation/wiki/SPEC-System-Lifecycle-Events.

52. Moore P. SPEC User Account Lifecycle Events [Електронний ресурс] / Paul Moore. – 2016. – Режим доступу до ресурсу: https://github.com/linux-audit/audit-documentation/wiki/SPEC-User-Account-Lifecycle-Events.

53. 2020 State of Malware Report [Електронний ресурс] // 2020 State of Malware Report. – 2020. – Режим доступу до ресурсу: https://resources .malwarebytes.com/files/2020/02/2020_State-of-Malware-Report. pdf

54. System Auditing [Електронний ресурс] – Режим доступу до ресурсу: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/chap-system_auditing.

55. Kuppusamy K. S. The Five Most Popular Operating Systems for the Internet of Things [Електронний ресурс] / Kuppusamy. – 2019. – Режим доступу до ресурсу: https://opensourceforu.com/2019/10/the-five-most-popular-operating-systems-for-the-internet-of-things/.

56. The Ultimate List Of Cyber Security Statistics For 2019 [Електронний ресурс] – Режим доступу до ресурсу: https://purplesec.us/resources/cyber-security-statistics/.

57. Understanding Linux Audit [Електронний ресурс] – Режим доступу до ресурсу: https://doc.opensuse.org/documentation/leap/archive/42.3/security/html/book.security/cha.audit.comp.html.

58. Griffin L. What is a Backdoor Virus? - Definition, Removal & Example [Електронний ресурс] / Lyna Griffin – Режим доступу до ресурсу: https://study.com/academy/lesson/what-is-a-backdoor-virus-definition-removal-example.html.

59. Shortridge K. What is the Linux Auditing System [Електронний ресурс] / Kelly Shortridge – Режим доступу до ресурсу: https://capsule8.com/blog/auditd-what-is-the-linux-auditing-system/.

60. Yu H. Needle in a Haystack: Attack Detection from Large-Scale System Audit / H. Yu, A. Li, R. Jiang. // IEEE 19th International Conference on Communication Technology (ICCT). – 2019. – С. 1418–1426.

61. Zuo Z. On the time complexity of computer viruses / Z. Zuo, Q. Zhu, M. Zhou. // IEEE Trans. Inf. Theory. – 2005. – №51. – С. 2962–2966.

62. Это нужно знать: Ключевые рекомендации по глубокому обучению (Часть 2) [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: http://datareview.info/article/eto-nuzhno-znat-klyuchevyie-rekomendatsii-po-glubokomu-obucheniyu-chast-2/.

63. RENAME(2) [Електронний ресурс] – Режим доступу до ресурсу: http://man7.org/linux/man-pages/man2/rename.2.html

64. TF-IDF [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/TF-IDF.