МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

КУЛЯС Іван

Латентні семантичні простори для виявлення шкідливого програмного забезпечення в умовах часткової інформації / Latent Semantic Spaces for Detecting Malicious Software in Conditions of Partial Information

спеціальність: 125 – Кібербезпека та захист інформації
освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи
КБзм -21
І. Куляс

Науковий керівник
І.З. Якименко

Кваліфікаційну роботу
допущено до захисту:

«____» _____ ___2024 р.

Завідувач кафедри
_____ **В.В.Яцків**

ТЕРНОПІЛЬ - 2024

Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки
Освітній ступінь «магістр»
спеціальність: 125 – Кібербезпека та захист інформації
освітньо-професійна програма – Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ В.В.Яцків
«____» _____ 20___ року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
КУЛЯСА Івана
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Латентні семантичні простори для виявлення шкідливого програмного забезпечення в умовах часткової інформації / Latent Semantic Spaces for Detecting Malicious Software in Conditions of Partial Information
керівник роботи к.т.н., доцент І.З. Якименко
затверджені наказом по університету від 12 грудня 2023 року № 753
2. Строк подання студентом закінченої кваліфікаційної роботи 12 грудня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити:
- провести аналіз основних підходів до виявлення шкідливих програм;
- проаналізувати існуючі методи побудови семантичних просторів і відображення в них об'єктів різної природи;
- розробити алгоритм і запропонувати архітектуру моделі побудови спільного ймовірнісного семантичного простору для різних типів подань виконуваних файлів;
- зробити опис запропонованого алгоритм виявлення шкідливого коду на основі довільного набору артефактів, а також отримано теоретичні гарантії стійкості запропонованого алгоритму;
- запропоновати рішення для низки важливих для індустрії інформаційної безпеки завдань, на основі навченої узагальненої ймовірнісної моделі.

5. Перелік графічного матеріалу у роботі:

– динамічний аналіз файлу;

– відносне розташування файлів і логів для різних сімейств шкідливих програм;

– альтернативні сценарії застосування об'єднаного ймовірнісного простору;

– архітектура загрози;

6. Консультанти розділів кваліфікаційної роботи

	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
	Якименко І.З.		
	Якименко І.З.		
	Якименко І.З.		
	Якименко І.З.		

7. Дата видачі завдання 12 грудня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Огляд методів виявлення шкідливих програм	12.2023 р. – 03.2024 р.	
2	Архітектура та методика навчання моделі відображення та відновлення зі схованого простору	03.2024 р. – 06.2024 р.	
3	Альтернативні сценарії застосування загального прихованого подання	06.2024 р. – 11.2024 р.	

Студент _____ Куляс І.
(підпис)

Керівник роботи _____ І.З. Якименко

АНОТАЦІЯ

Кваліфікаційна робота на тему «Латентні семантичні простори для виявлення шкідливого програмного забезпечення в умовах часткової інформації» на здобуття освітнього ступеня «Магістр» зі спеціальності 125 «Кібербезпека» освітньо-професійної програми «Кібербезпека» написана обсягом 71 сторінку і містить 7 ілюстрацій, 3 таблиці, 1 додаток та 43 джерело за переліком посилань.

Метою кваліфікаційної роботи є розробка методу побудови уніфікованого ймовірнісного семантичного простору, який дозволяє консолідувати потік різнорідної інформації з різних етапів аналізу шкідливого програмного забезпечення..

Методи досліджень. Для розв’язання поставлених задач у даній кваліфікаційній роботі використано: методи оптимізації, статистичного та динамічного аналізу, побудови латентних семантичних просторів, інтелектуального аналізу, математичного моделювання.

Результати дослідження: розроблено метод, який дозволяє підвищити ефективність виявлення шкідливого ПЗ шляхом інтеграції результатів статичного та динамічного аналізу в єдину модель і забезпечити високий рівень узагальнення даних і точності детектування.

Результати роботи можуть успішно застосовуватися при реалізації систем пов’язаних з виявленням та класифікацією нових типів шкідливих програм в умовах часткової інформації.

Ключові слова: ЛАТЕНТНІ СЕМАНТИЧНІ ПРОСТОРИ, ШКІДЛИВЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, МОДЕЛІ МАШИННОГО НАВЧАННЯ, СПІЛЬНИЙ ЙМОВІРНІСНИЙ СЕМАНТИЧНИЙ ПРОСТІР.

ABSTRACT

Qualification work on " Latent Semantic Spaces for Malware Detection Under Partial Information" for the degree of "Master" in the specialty 125 "Cybersecurity" educational and professional program "Cybersecurity" is written in 71 pages and contains 7 illustrations, 3 tables, 1 appendices and 43 source according to the list of links.

The aim of the qualification thesis is to develop a method for constructing a unified probabilistic semantic space that consolidates heterogeneous information flows from various stages of malware analysis.

Research methods. To solve the tasks set in this thesis, the following methods were used: optimization methods, statistical and dynamic analysis, latent semantic space construction, data mining, and mathematical modeling.

Research results: A method has been developed to improve the efficiency of malware detection by integrating static and dynamic analysis results into a unified model, ensuring a high level of data generalization and detection accuracy.

The results of this work can be successfully applied to implement systems for detecting and classifying new types of malware under conditions of partial information.

Keywords: LATENT SEMANTIC SPACES, MALWARE, MACHINE LEARNING MODELS, UNIFIED PROBABILISTIC SEMANTIC SPACE.

ЗМІСТ

ВСТУП

Актуальність. Зловмисне програмне забезпечення продовжує залишатися однією з головних загроз для інформаційної безпеки. Щодня створюється велика кількість нових шкідливих програм, які мають складні механізми обходу існуючих засобів захисту. Класичні методи виявлення, такі як аналіз сигнатур, часто не справляються із задачами виявлення нових типів шкідливого коду, що значно ускладнює забезпечення безпеки інформаційних систем.

У цьому контексті виникає потреба у впровадженні сучасних підходів, заснованих на машинному навчанні, для аналізу шкідливих програм. Використання латентних семантичних просторів дозволяє не лише виявляти нові зразки шкідливого ПЗ, але й аналізувати їх поведінку в умовах часткової або неповної інформації. Це забезпечує можливість адаптивного реагування на загрози, навіть у випадках, коли шкідливий код використовує поліморфізм або методи обфускації.

Актуальність роботи обумовлена необхідністю розробки універсальних підходів, що дозволяють інтегрувати дані з різних джерел, враховуючи статичний і динамічний аналіз шкідливого програмного забезпечення. Вирішення цих задач має ключове значення для підвищення ефективності захисту інформаційних систем у сучасних умовах.

Метою кваліфікаційної роботи є розробка методу побудови уніфікованого ймовірнісного семантичного простору, який дозволяє консолідувати потік різнорідної інформації з різних етапів аналізу шкідливого програмного забезпечення.

В основі методу лежить навчання комбінації нейромережевих автокодувальників, які описують спільний розподіл різнорідних даних і дозволяють, зокрема, передбачати очікувану поведінку файлу на основі його

структури. Для досягнення мети необхідно вирішити низку взаємноповязаних задач:

- провести аналіз основних принципів підходів до виявлення шкідливих програм;

- проаналізувати існуючі методи побудови семантичних просторів і відображення в них об'єктів різної природи;

- розробити алгоритм і запропонувати архітектуру моделі побудови спільного ймовірнісного семантичного простору для різних типів подань виконуваних файлів;

- зробити опис запропонованого алгоритм виявлення шкідливого коду на основі довільного набору артефактів, а також отримано теоретичні гарантії стійкості запропонованого алгоритму;

- запропоновати рішення для низки важливих для індустрії інформаційної безпеки завдань, на основі навченої узагальненої ймовірнісної моделі.

Об’єктом дослідження є процеси аналізу шкідливого програмного забезпечення.

Предметом дослідження є латентні семантичні простори для виявлення шкідливого програмного забезпечення в умовах часткової інформації.

Методи дослідження. Для досягнення поставленої мети та вирішення завдань дослідження використано такі методи:

1. Методи інтелектуального аналізу даних – для виявлення шаблонів у великих обсягах різнорідних даних, таких як байт-код або поведінкові логи виконуваних файлів.

2. Моделі машинного навчання – для створення узагальнених семантичних просторів, зокрема, використання автокодувальників і варіаційних автокодувальників для побудови прихованих представлень об’єктів різної природи.

3. Методи побудови латентних семантичних просторів – для формалізації прихованих зв’язків між об’єктами та їх подання у вигляді узгоджених векторних представлень.

4. Аналіз ймовірнісних просторів – для створення моделей, які дозволяють оцінювати взаємозв'язок між виконуваними файлами та їх поведінковими логами в умовах часткової інформації.

5. Методи статичного та динамічного аналізу – для оцінки характеристик виконуваних файлів, визначення їхньої структури, виявлення вразливих місць і аналізу їх поведінки в контрольованих середовищах.

6. Методи оптимізації – для навчання моделей і побудови алгоритмів, що забезпечують максимальну правдоподібність узгоджених розподілів даних у семантичному просторі.

7. Математичне моделювання – для формалізації задачі побудови спільного прихованого простору та створення методів для оцінки ймовірностей різних сценаріїв поведінки шкідливого програмного забезпечення.

8. Експериментальні дослідження – для перевірки працездатності запропонованих моделей і алгоритмів на тестових наборах даних, що включають приклади як шкідливих, так і безпечних файлів.

Наукова новизна отриманих результатів:

1. Розроблено метод побудови уніфікованого ймовірнісного семантичного простору, що дозволяє інтегрувати різнорідні дані з різних джерел, включаючи результати статичного та динамічного аналізу виконуваних файлів.

2. Удосконалено методику навчання моделей машинного навчання для виявлення шкідливого програмного забезпечення в умовах часткової інформації, що забезпечує підвищення точності та адаптивності класифікації.

Практичне значення виконаної роботи полягає у створенні методу побудови уніфікованого ймовірнісного семантичного простору для виявлення шкідливого програмного забезпечення в умовах часткової інформації. Запропонований метод дозволяє:

1. Підвищити ефективність виявлення шкідливого ПЗ шляхом інтеграції результатів статичного та динамічного аналізу в єдину модель. Це дозволяє забезпечити високий рівень узагальнення даних і точності детектування.

2. Забезпечити стійкість до змін умов аналізу**,** таких як поліморфізм, обфускація або різнорідність вихідної інформації. Система може адаптивно працювати з різними форматами вхідних даних, такими як байт-код, логи дій або мережеві дані.

3. Оптимізувати процес навчання моделей машинного навчання завдяки використанню узагальнених представлень, що знижують вимоги до обсягу і якості початкових даних.

4. Спростити впровадження в промислові рішення**.** Запропонована архітектура дозволяє інтегрувати модель у вже існуючі антивірусні або інформаційно-захисні системи без значних змін їхньої інфраструктури.

5. Вирішувати практичні задачі інформаційної безпеки**,** такі як раннє виявлення нових сімейств шкідливого ПЗ, прогнозування поведінки виконуваних файлів та їх автоматичне класифікування.

Публікації.

1. Куляс І., Слободян В., Якименко Ю., Хомяк Р. Метод відображення інформації в об'єднаний семантичний простір для аналізу шкідливих файлів. Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп’ютерно-

інтегровані технології» (КБКІТ - 2024), Тернопіль, 2024. с. 122-125.

2. І. Куляс, В. Слободян, Ю. Якименко, Д. Гордійчук Основні принципи побудови базової моделі шкідливих файлів Матеріали науково-практичного симпозіуму «ЗАХИСТ ІНФОРМАЦІЇ». 2024. с.19-21.

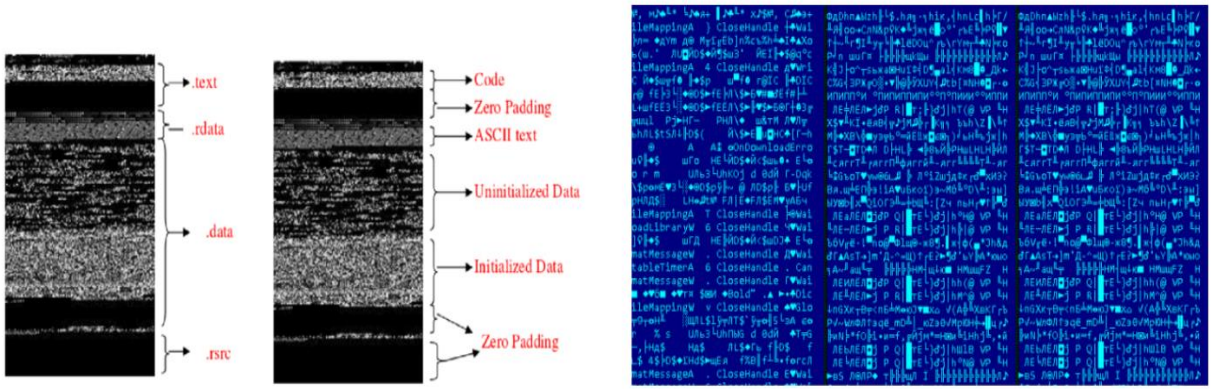
1 ОГЛЯД МЕТОДІВ ВИЯВЛЕННЯ ШКІДЛИВИХ ПРОГРАМ

У цьому розділі описані основні підходи, що застосовуються в індустрії інформаційної безпеки для виявлення шкідливих виконуваних файлів. У межах цієї роботи розглядаються лише файли типу PE (Portable-Executable) [1], призначені для використання в операційній системі сімейства Windows. Зокрема, до таких файлів належать виконувані файли формату *.EXE і динамічно підключувані бібліотеки виконуваного коду формату *.DLL. Однак описані далі методи виявлення та запропонований підхід із побудовою загальних семантичних просторів також можуть бути застосовані для роботи з виконуваними файлами під іншими операційними системами, зокрема ELF під Unix або APK під Android.

1.1 Методи виявлення шкідливих програм

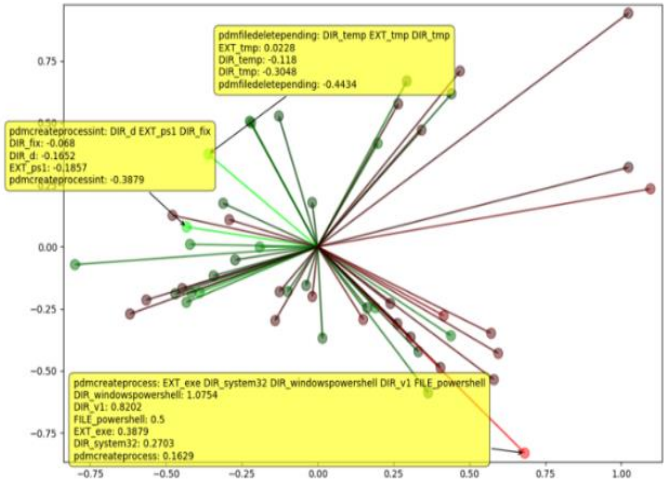
1.1.1 Статичний аналіз

Перші покоління антивірусних програмних продуктів базувалися виключно на статичному аналізі файлів. Цей підхід передбачає побудову правил детектування, заснованих на аналізі структури файлу, його вмісту, фрагментів дизасембльованого або вихідного коду. Щоб ускладнити статичний аналіз, розробники шкідливих програм часто використовують різні методи обфускації – різні прийоми, які ускладнюють розуміння призначення файлу та його функціоналу.

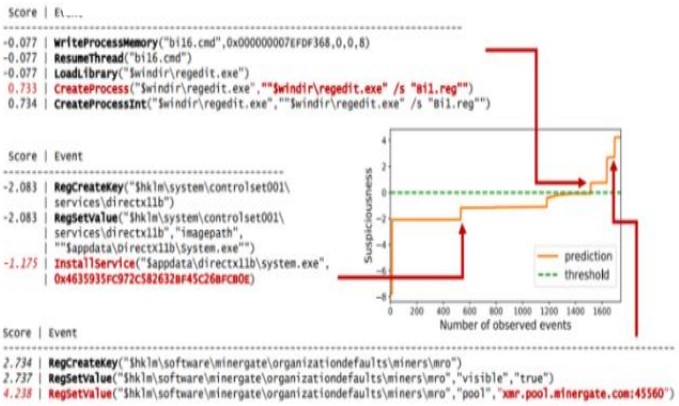


(a) Структура файлу (b) Байтове представлення файлу

Рисунок 1.1 – Статистичний аналіз файлу



(a) Виділення груп подій



(b) Пошук найбільш цікавих подій

Рисунок 1.2 – Динамічний аналіз файлу

Різні методики обфускації можуть включати додавання випадкових команд і складних програмних конструкцій, виконання яких в підсумку не має жодного впливу на процес роботи файлу, але значно знижує читабельність. Окрім цього, часто застосовується шифрування різних ділянок файлу. При цьому для розшифрування може знадобитися ключ, який програма отримує лише при запуску з зовнішніх ресурсів.

Ці та інші методи приховування вмісту виконуваного файлу також можуть використовуватися авторами комерційного програмного забезпечення з метою захисту інтелектуальної власності та запобігання появі неліцензованих копій програми.

Попри описані недоліки, статичний аналіз є дуже ефективним, оскільки дозволяє виявити шкідливий файл одразу після його потрапляння на машину користувача, мінімізуючи таким чином ймовірність завдання будь-якої шкоди.

1.1.2 Динамічний аналіз

Альтернативою статичному підходу є динамічний. У цьому підході виконуваний файл запускається в контрольованому середовищі (наприклад, на віртуальній машині або спеціальному емуляторі) або ж безпосередньо на комп’ютері користувача, і проводиться аналіз поведінки файлу. Головною перевагою цього підходу є те, що незалежно від рівня обфускації, якщо в основний код програми закладені шкідливі дії, вона повинна їх виконати. Однак цей підхід також має ряд недоліків. По-перше, значна частина програм може завершуватися з помилкою або не проявляти жодної активності за відсутності необхідних додаткових модулів, зовнішніх факторів або певних дій з боку користувача. По-друге, досить часто автори програм застосовують різні методи антиемуляції, які дозволяють виявити наявність контролю за поведінкою та приховати прояви будь-якої шкідливої активності. Техніки антиемуляції можуть включати в себе відкладений старт основного

функціоналу програми, самознищення виконуваного файлу в разі виявлення захисних програмних засобів або ж виявлення факту запуску в емуляційній середовищі шляхом перевірки наявності різних артефактів, пов'язаних з недокументованими особливостями реалізації віртуального середовища.

Техніка динамічного аналізу використовується значно меншим числом вендорів захисних програмних продуктів через складність технічної реалізації, проте є необхідним компонентом при побудові повноцінної системи захисту.

1.1.3 Репутаційний аналіз

Часто при виявленні шкідливого програмного забезпечення, окрім безпосереднього аналізу файлу та його поведінки, використовується додаткова метаінформація. Так, наприклад, іноді для обілення або чорнення файлу використовується інформація про його популярність у світі, наявність довіреного підпису, географії поширення та списку сайтів, з яких цей файл може бути завантажений. Жодні класифікуючі правила, побудовані на метаінформації, не дають гарантовано вірної відповіді, але в сукупності з інформацією, отриманою від статичного або динамічного аналізу, можуть допомогти запобігти зайвим хибним спрацьовуванням або ж, навпаки, попередити користувача про можливе зараження навіть без виявлення явних ознак загрози.

1.1.4 Комбінований аналіз

Кожен з описаних підходів аналізу файлів має свої слабкі місця. Щоб зробити захист комп'ютерної системи максимально надійним, багато захисних програмних продуктів використовують так звану багатошарову захист — послідовне застосування різних технологій. Наприклад, спочатку застосовується репутаційний аналіз, після цього легковісний статичний аналіз, далі відбувається запуск і аналіз файлу на емуляторі (або в так званій

«пісочниці»), після цього важкий статичний та динамічний аналіз із застосуванням моделей машинного навчання, і нарешті динамічний аналіз поведінки програми після її запуску користувачем на своєму комп'ютері. При цьому, якщо на якому-небудь з ранніх етапів відбувається виявлення потенційної загрози, файл відразу помічається як шкідливий і, залежно від налаштувань захисного продукту, може бути видалений з комп'ютера, переміщений в спеціальну карантинну зону або відправлений для більш детального аналізу експертам з інформаційної безпеки до того, як користувачу буде завдано шкоди.

Проблемою багатошарової захисту є те, що зазвичай наступні шари не використовують жодної інформації, зібраної на більш ранніх етапах, і в результаті можуть не бачити всієї інформації, необхідної для вірної класифікації файлу. У кращому випадку автори детектуючих евристичних правил можуть використовувати техніки статичного аналізу для вивчення розшифрованого та розпакованого в оперативній пам’яті тіла файлу, відправленого на динамічний аналіз, або ж скасовувати виявлення для файлів з хорошою репутацією (наприклад, отриманих від довірених вендорів програмного забезпечення).

У даному об’єднанні етапів аналізу є кілька причин. По-перше, досить часто виявляється досить важко додати повноцінний комбінований аналіз в архітектуру захисного продукту, не порушуючи при цьому уставленої модульності програми.

По-друге, багато початківців антивірусних аналітиків вважають за краще обходитися створенням детектуючих правил, основаних лише на одному з підходів, оскільки не мають достатньої експертизи.

1.2 Метод побудови об’єднаного семантичного простору

Метод побудови об’єднаного семантичного простору дозволяє об’єднати експертні знання з різних областей для ухвалення узгодженого рішення про виявлення та при цьому не вимагає злиття різних модулів захисного продукту, використовуючи єдиний зовнішній консолідатор інформації від різних модулів продукту.

Метод побудови об’єднаного семантичного простору

1. Основні ідеї методу

Об’єднаний семантичний простір – це багатовимірний простір, де різні джерела даних відображаються у спільній координатній системі. Ключовими завданнями є:

- вирівнювання різних векторних представлень;
- збереження семантичної подібності між поняттями;
- зведення багатоджерельних даних до єдиного представлення.

2. Етапи побудови

1. Збір даних: збираються дані з різних джерел D_1, D_2, ..., D_n, які можуть мати різні представлення (тексти, зображення, аудіо тощо).

2. Попередня обробка: кожен тип даних перетворюється у відповідний векторний простір.

Наприклад:

- для тексту – використовується Word2Vec, GloVe або трансформери;
- Для зображень – векторизація за допомогою CNN.
- Для інших типів – специфічні моделі.

$X_i = f(D_i)$, де f – функція перетворення векторів.

3. Вирівнювання просторів: для вирівнювання використовуються методи канонічного кореляційного аналізу (CCA), узгодження матриць або нейронні мережі.

Наприклад, нехай $X_1 \in \mathbb{R}^{n \times d1}$ та $X_2 \in \mathbb{R}^{m \times d2}$ – вектори двох джерел, тоді вирівнювання виконується як:

$$\text{argmin}_{W1,W2} \|W_1 X_1 - W_2 X_2\|^2, \quad (1.1)$$

де W_1 і W_2 – матриці перетворення.

4. Проекція у спільний простір: вектори з різних джерел перетворюються у єдиний простір:

$$Z = g(W_1 X_1) + g(W_2 X_2) + ..., \quad (1.2)$$

де g – функція активації або додаткової трансформації.

5. Оптимізація: вибір параметрів W_i базується на метриках схожості, наприклад, косинусної подібності.

3. Приклад формул

Якщо використовувати метод CCA, то задача зводиться до:

$$\text{maximize } \rho = \text{cov}(W_1^T X_1, W_2^T X_2) / \text{sqrt}(\text{var}(W_1^T X_1)\ \text{var}(W_2^T X_2)). \quad (1.3)$$

Для глибинного вирівнювання можуть застосовуватися нейронні мережі, які мінімізують функцію втрат, наприклад:

$$L = \sum_{i,j} \left\| h\left(X_1^{(i)}\right) - h\left(X_2^{(j)}\right) \right\| \quad (1.4)$$

де h – нелінійна функція, що відображає вектори у спільний простір.

Стандартний метод виявлення полягає в ідентифікації зловмисного програмного забезпечення шляхом пошуку деяких шаблонів у коді (сигнатури). Проте метод підпису має кілька недоліків і потребує вдосконалення за допомогою інших підходів. Розглядаються два інші методи, які можуть надати додаткову можливість виявлення шкідливого програмного забезпечення.

Перший базується на спостереженні за поведінкою зв'язку, спричиненою підозрілим кодом. Він використовує переваги нашої платформи для аналізу шкідливих програм, пов’язаних із ботнетами і складається з інструментів для виявлення шкідливих програм, запуску коду в контрольованому середовищі та аналізу його взаємодії із зовнішніми об’єктами.

Другий метод надає засоби для виявлення зашифрованого трафіку Skype і класифікації потоків послуг Skype, таких як голосові дзвінки, skypeOut, відеоконференції, чат, завантаження та завантаження файлів у трафікі Skype. Метод заснований на статистичній ідентифікації протоколу (SPID), який аналізує статистичні значення деяких атрибутів трафіку. Застосовується метод виявлення шкідливого трафіку — ми успішно виявили розповсюдження Worm.Win32.Skipi.b, який поширюється через месенджер Skype, надсилаючи заражені повідомлення всім контактам Skype на комп’ютері-жертві. В основному необхідно зосередитися на виборі відповідного набору вимірювачів атрибутів на основі характеристик розповсюдження для класифікації шкідливих потоків з високою точністю.

1.3 Платформа для аналізу шкідливих програм, пов’язаних із ботнетами

Дизайн платформи для аналізу шкідливих програм, пов’язаних із ботнетами має наступні функції (див. рисунок 1.3):

– перехоплення зловмисного програмного забезпечення: для цієї мети використовується Dionaea, популярна приманка з низьким рівнем взаємодії, яка в основному виявляє зловмисне програмне забезпечення, що поширюється через уразливості в службах Microsoft SMB;

– класифікація зловмисного програмного забезпечення: щойно зловмисне програмне забезпечення виявляється, воно автоматично класифікується відповідно до мережевих з’єднань, які воно намагається виконати, щоб зв’язатися зі своєю службою командування та контролю (C&C) [3]. З цією метою зловмисне програмне забезпечення запускається на віртуальній машині без фактичного з’єднання з Інтернетом, але зі службою DNS, яку надає хост-машина. Адреси DNS, які отримали запит і спроби підключення спостерігаються та записуються за допомогою програмного засобу Mwna, коротко описаного в розділі 1.4. Це дозволяє виявити шкідливі програми з дійсно невідомою поведінкою, таким чином уникаючи аналізу вже відомого шкідливого програмного забезпечення;

– аналіз мережевої активності шкідливих програм: виконується під контролем оператора за допомогою Mwna і зосереджується на ідентифікації C&C і виявленні шкідливих дій.

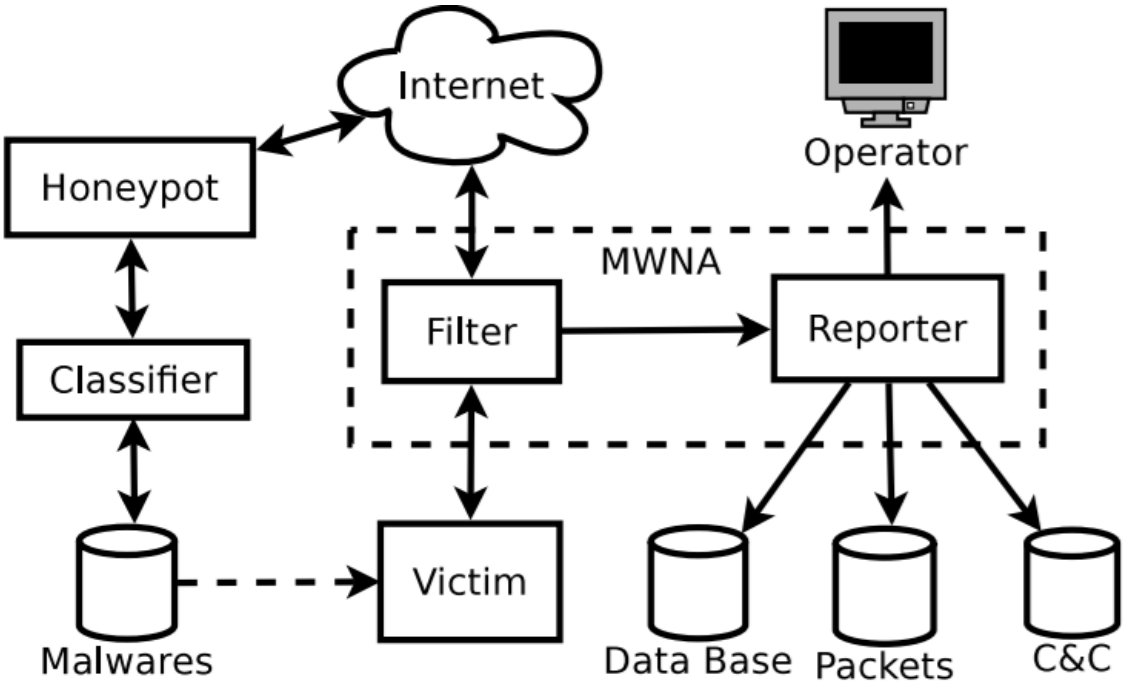


Рисунок 1.3 – Архітектура платформи

1.3.1 Аналізатор мережі шкідливих програм

Mwna (аналізатор мережі шкідливих програм) працює на шлюзі між комп'ютером (реальним або віртуальним), зараженим шкідливим програмним забезпеченням (жертва), та Інтернетом. Він складається з двох програм, які взаємодіють через сокети TCP, фільтр і репортер:

– фільтр: він використовує механізм netfilter_queue Linux для перехоплення пакетів, що проходять через стек мережі в ядрі. Процес зіставлення пакетів може бути заданий правилами (скидання, трасування, аналіз вищого рівня тощо);

– пакети аналізуються до прикладного рівня, коли це необхідно. Він контролює встановлення з'єднання, його завершення та діяльність DNS (запити та відповіді). Ряд протоколів прикладного рівня можна виявити та принаймні частково проаналізувати, навіть якщо вони не працюють на стандартних портах: IRC, HTTP, SMTP. Протокол HTTPS ідентифікується та перевіряється до того моменту, коли він переходить у зашифрований режим

(але сертифікати наразі не перевіряються). Службові протоколи NTP, IPP і Netbios NAME і SESSION перевіряються, коли вони працюють на стандартних портах. Також можуть бути виявлені невідомі текстові протоколи.

Фільтр містить систему динамічно завантажуваних плагінів, що дозволяє розширити його функціями вищого рівня, включаючи моніторинг і блокування шкідливих дій:

– взаємодія з C&C: у поточному стані програми C&C в основному можуть бути виявлені як протоколи на основі IRC. C&C на основі нестандартних текстових або бінарних протоколів і HTTP також можна виявити за допомогою кількох помилкових спрацьовувань. Коли виявлено взаємодію C&C через IRC, усі записується спілкування жертви з нападником.

– Сканування ICMP і TCP виявляються та блокуються.

– Атаки на відмову в обслуговуванні (DoS): атаки Tcp SYN flood, Udp flood і Http flooding виявляються та блокуються скиданням пакетів, надісланих на атакуваний хост.

– Спроби передачі електронної пошти виявляються та, за бажанням, можуть бути заблоковані шляхом скидання з’єднання з поштовим сервером.

– Передачу спаму можна виявити на основі спроб надіслати пошту через кілька ретрансляторів або до кількох різних адресатів.

– Корисне навантаження HTTP аналізується для виявлення передачі виконуваних файлів Windows. Виявлення базується на ідентифікації виконуваних заголовків і не залежить від заголовків типу вмісту чи розширень імен файлів.

Оскільки фільтр пов’язаний із ядром, його потрібно запускати з правами адміністратора. Виявлені події надсилаються репортеру за допомогою дуже простого текстового протоколу.

– Reporter відповідає за відображення повідомлень для оператора, запис подій у базі даних Sqlite, збереження трас зв’язку у файлі та запис взаємодії C&C. Його не потрібно виконувати з правами адміністратора.

Поділ MWNA на кілька процесів має кілька переваг:

– оскільки Reporter працює без привілеїв адміністратора, він може використовувати інструменти високого рівня, такі як база даних або графічний інтерфейс користувача, без ризиків для безпеки;

– операції з великою затримкою (наприклад, взаємодія користувача, дисковий ввід/вивід) видаляються з критичного шляху обробки пакетів;

– кращої продуктивності можна досягти на багатоядерних архітектурах;

– Reporter можна запускати на окремій машині або в робочій мережі без ризиків для безпеки;

– Reporter може бути замінений на програму, яка виконує повністю автоматичний аналіз мережевої активності зловмисного програмного забезпечення.

1.4 Виявлення шкідливих програм

У цьому розділі наведено два приклади діяльності зловмисного програмного забезпечення. У першому ми представимо графічний інтерфейс Mwna зі шкідливим програмним забезпеченням, яке, на перший погляд, намагається розсилати спам. Другий приклад — зловмисне програмне забезпечення, яке насправді розсилає спам.

1.4.1 Зловмисне програмне забезпечення для сканування портів

На рисунку 1.4 представлено знімок екрана журналіста під час аналізу зловмисного програмного забезпечення, отриманого в грудні 2023 року. У

рядку заголовка вікна показано дайджест MD5 шкідливого програмного забезпечення. Рядок під панеллю меню показує, що C&C ідентифіковано як Irc, і містить IP-адресу та номер порту.

Права текстова область під лінією вказує на відповіді DNS. У досліджуваному випадку є лише відповідь на TXT-запит на lap.viridinikasihc.net і запит на адресу хост-машини. Відповідь на TXT-запит виглядає якоюсь зашифрованою інформацією, яка повинна містити адресу C&C.

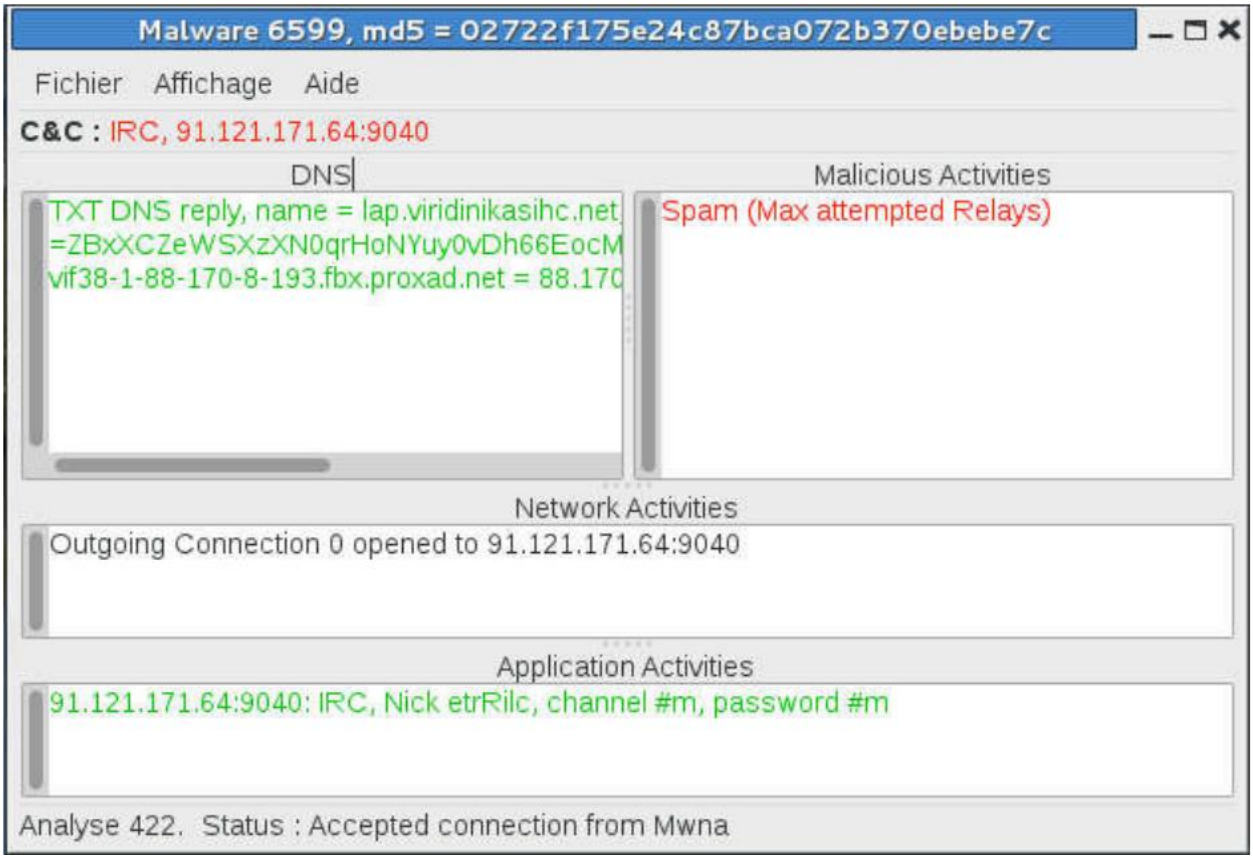


Рисунок 1.4 Знімок екрана Mwna

Ліва текстова область показує виявлені шкідливі дії: у цьому випадку виявлено спробу надіслати спам, оскільки скомпрометована машина

намагається підключитися до різних серверів Smtp. Усі ці спроби завершуються невдачею, оскільки постачальник доступу дозволяє підключення лише до свого власного сервера Smtp.

Текстова область під попередніми відображає мережеву діяльність: у цьому випадку лише підключення до сервера IRC.

Нижня текстова область відображає дії на рівні програми: у цьому випадку зловмисне програмне забезпечення приєднується до каналу IRC.

На цьому етапі інструмент вказує на спробу передачі спаму, але є дві проблеми:

– адреси SMTP-серверів не розпізнаються через DNS;

– немає достатньо даних для вказівки корисного навантаження спаму та списку одержувачів.

Перенаправляється підключення до порту SMTP до програми, яка емулює сервер SMTP без фактичної передачі повідомлень і не знайдено жодної передачі після встановлення з’єднання. Нарешті, якщо вимикається функція виявлення спаму, можна спостерігати, що ці спроби підключення є фактично скануванням порту SMTP, ймовірно, з метою виявлення відкритих ретрансляторів пошти.

1.4.2 Спам, що надсилає зловмисне програмне забезпечення

У цьому другому прикладі зловмисне програмне забезпечення спілкується зі своїм C&C за допомогою модифікованого протоколу IRC, ідентифікованого як невідомий текстовий протокол.

Чотири запити DNS намагаються знайти поштові сервери AOL, Yahoo, Google і Hotmail, що здається типовим для спаму, який надсилає зловмисне програмне забезпечення. Потім шкідлива програма завантажує кілька файлів із серверів HTTP. П’ять із них є виконуваними файлами Windows, а два інші – це

запити для отримання офіційної IP-адреси та доменного імені машини, на якій працює шкідливе програмне забезпечення.

Після цього зловмисне програмне забезпечення намагається підключитися до поштових серверів і виявляє спам.

Як і в попередньому випадку, було проаналізовано зловмисне програмне забезпечення, коли трафік Smtp перенаправляється до локальної програми. Можна побачити два інших завантаження, які виглядають як зашифровані дані, отже, спам і список одержувачів можуть міститися в цих файлах або інкапсулюватися в завантажених виконуваних файлах. 383 листи було надіслано за 5 хвилин, використовуючи 83 підключення до багатьох поштових ретрансляторів адреси. У більшості з'єднань зловмисне програмне забезпечення надсилало лише одне повідомлення або не надсилало жодного листа, але в 17 випадках воно надсилало до 164 листів.

Ці два приклади показують, що платформа дозволяє виявляти та ідентифікувати зловмисну діяльність шкідливого програмного забезпечення. Передача спаму, здається, є найчастішою діяльністю. У цьому випадку можна зібрати достатньо даних, щоб визначити стратегію інструментів розсилання спаму та перехопити корисне навантаження спаму.

1.5 Огляд методів відображення об'єктів в прихований простір

1.5.1 Матричні розкладання

Найбільш відомим прикладом задачі, що вирішується за допомогою матричних розкладань, є колаборативна фільтрація. У класичній постановці ця задача передбачає наявність двох скінченних множин: користувачів та товарів, для деяких пар елементів яких відомий результат їх взаємодії. Завдання полягає в тому, щоб передбачити результат взаємодії для інших пар користувач-товар, результат взаємодії яких поки що невідомий. Типовими

прикладами задачі колаборативної фільтрації є складання стрічки рекомендацій фільмів в онлайн-кінотеатрах, формування продуктової корзини покупця або персоналізація стрічки новин. Основна ідея полягає в розкладанні розрідженої двомірної матриці MMM, для деяких елементів якої відомі значення, в добуток двох матриць низького рангу, які наближають відому інформацію $M = U \times V^T$. Матриці U і V в даному випадку відповідають векторному представленню спостережуваних користувачів і товарів. Залежно від специфіки задачі на матриці можуть бути накладені різні додаткові обмеження: наприклад, невід'ємність або розрідженість, а також можуть використовуватися різні метрики близькості. На основі побудованого розкладання можна передбачити результат майбутньої взаємодії користувача з незнайомими йому товарами через скалярний добуток: $\widehat{M}_{ij} = f\big(\langle U_i, V_j \rangle\big)$. Також відображення користувачів і товарів у евклідове простір дозволяє кластеризувати об'єкти відповідного безлічі з метою аналізу ринку, проведення різних маркетингових кампаній і вирішення багатьох інших важливих для бізнесу задач. Другим відомим прикладом застосування матричних розкладань є популярний в області обробки природних мов підхід Word2Vec. Простіший варіант реалізації цього підходу базується на побудові матриці парної зустрічальності слів природної мови всередині одного локального контексту (як правило, під контекстом розуміють окремі речення або ковзаюче вікно фіксованої довжини). Будуючи спеціальним чином розкладання такої матриці, можна отримати векторне представлення окремих слів, що дозволяє ефективно шукати синоніми, виявляти семантичні зв’язки між словами і навіть вирішувати задачу машинного перекладу.

1.5.2 Автокодувальники

Характерною особливістю методів, заснованих на матричних розкладаннях, є те, що вони активно використовують інформацію про контекст

об’єкта, але не враховують жодної інформації про сам об’єкт. Протилежний підхід застосовується при побудові моделей автокодерів. Цей підхід передбачає навчання двох перетворень, представлених, як правило, у вигляді багатошарових нейронних мереж. Перше перетворення (кодер) переводить початкове представлення об’єкта (наприклад, матрицю з кольорами пікселів зображення або символічне представлення речення на природній мові) в приховане векторне простір низької розмірності. А друге перетворення (декодер) намагається якнайточніше відновити початковий об’єкт на основі отриманого від кодера вектора.

Слід зазначити, що в деяких статтях в якості прихованого простору використовується не лінійне простір, а, наприклад, неевклідне гіперболічне простір Пуанкаре. Як показують експерименти, іноді використання таких відображень дозволяє суттєво знизити витрати пам’яті для зберігання матриці прихованих представлень і підвищити точність алгоритмів, що використовують ці представлення.

1.5.3 Об'єднане представлення в різних доменах

Природним розвитком ідеї автокодерів стало навчання прихованих представлень, що дозволяють кодувати і декодувати пов'язані об’єкти різної природи. Наприклад, у статтях представлені моделі, що генерують текстовий опис сцени за вхідним зображенням і, навпаки, малюють зображення, що відповідає заданому текстовому опису. Особливістю моделей цього типу є те, що вони генерують об’єкти досить простій природи (як правило, зображення або невеликі фрагменти тексту). Для деяких задач, пов'язаних з об’єктами більш складної природи, застосовуються методи, засновані на навчанні двох кодерів, налаштованих на побудову прихованих представлень таким чином, що відстані в прихованому просторі для пов’язаних пар об’єктів виявляються меншими, ніж для пари випадкових об’єктів. Хорошим прикладом такої

архітектури є модель DSSM, в якій будуються кодувальні моделі для коротких пошукових запитів і довгих текстів веб-сторінок, так, щоб максимізувати релевантність пошукової видачі для кожного запиту. Недоліком цієї архітектури є те, що вона дозволяє вирішувати тільки задачу ранжування, але не дає можливості зіставити відстані в прихованому просторі з реальною близькістю документів і запитів.

1.5.4 Сховані ймовірнісні простори

Важливою ідеєю в розвитку моделей нейронних кодувальників стало введення розподілів на прихованих представленнях об’єктів. Одна з перших моделей, що має цю властивість, отримала назву варіаційного автокодера (Variational Autoencoder, VAE). При навчанні цієї моделі не просто мінімізується відстань між вихідними і декодованими даними, а будується найбільш обґрунтована генеративна модель , наближаюча процес породження даних з навчальної вибірки. Другим відомим методом побудови генеративних моделей є підхід, що використовує навчання двох суперечливих мереж (Generative Adversarial Networks, GAN). Кожна з моделей VAE і GAN має свої переваги та недоліки, різні способи вирішення яких активно пропонуються дослідниками протягом останніх років. І, як і в разі класичних декодерів, генерувати поки що в основному вдається об'єкти досить простої структури.

2 АРХІТЕКТУРА ТА МЕТОДИКА НАВЧАННЯ МОДЕЛІ ВІДОБРАЖЕННЯ ТА ВІДНОВЛЕННЯ ЗІ СХОВАНОГО ПРОСТОРУ

У цьому розділі описані пропоновані архітектура та методика навчання моделі, що дозволяє побудувати відображення в єдине ймовірнісне простір різних представлень виконуваних файлів, кожне з яких може мати складну структуру. З точки зору архітектури базова версія моделі являє собою об'єднання кількох варіаційних автокодерів, що ділять спільний простір прихованих змінних. При цьому модель, хоча й не дозволяє правдоподібно генерувати такі складні об'єкти, як виконувані файли або їх логи, але дає можливість оцінювати ймовірність нових отриманих даних і ймовірність наявності зв'язку між артефактами різної природи.

Далі, в усіх експериментах, представлених у роботі, будуть використовуватися відображення лише для виконуваних PE-файлів і лога поведінки, отриманого при запуску на віртуальній машині. Однак представлені ідеї можуть бути використані і при роботі з іншими джерелами інформації: наприклад, емуляційними логами, наборами різнорідної метаінформації або логами роботи на комп'ютері кінцевого користувача продукту.

Вважається, що лог поведінки являє собою упорядковану в часі послідовність подій разом із пов'язаними з цими подіями аргументами. Наприклад, з подією, що описує відкриття файлу, пов'язані шлях до файлу і права доступу. А з подією, що відповідає за створення мережевого з'єднання, пов'язані IP-адреса віддаленої машини, номер мережевого порту, а також, можливо, протокол з'єднання і/або обсяг переданих даних. Додатково допускається зберігання унікальних ідентифікаторів процесів і потоків, що здійснюють зазначені дії, а також час початку кожної з подій.

Отримання логів, що використовуються в цій роботі, здійснюється наступним чином:

а) виконуваний файл стартує з фіксованої директорії всередині віртуальної машини. Образ віртуальної машини при цьому містить різні попередньо встановлені програми, що представляють потенційний інтерес для зловмисника з точки зору пошуку вразливостей, а також набір користувацьких файлів, які також можуть бути цікавими розповсюджувачам шкідливого програмного забезпечення (наприклад, таблиці з фінансовими звітами або файли з паролями та персональними даними);

б) аналізована програма стартує без будь-яких параметрів. Якщо файл не має точки старту (Entry Point), що, наприклад, вірно для більшості *.DLL файлів, проводиться послідовний запуск функцій, наданих файлом на основі таблиці експорту;

в) з допомогою спеціального компонента моніторингу проводиться логування всіх дій, що здійснюються виконуваним файлом і запущеними ним сторонніми процесами;

г) логування здійснюється, поки не настане одне з наступних подій:

— всі процеси, породжені файлом, успішно завершилися або були перервані через помилку;

— робота файлу та дочірніх процесів займає більше установленого часу.

д) результати логування зберігаються у зовнішньому сховищі та надсилаються на динамічний аналіз.

Щоб краще обґрунтувати потребу в застосуванні ймовірнісних просторів для представлення файлу та його лога, нижче описані основні причини, що породжують неоднозначність при побудові представлень.

Для виконуваного файлу основними причинами варіативності є:

— умисний поліморфізм файлів: часто зловмисники при розповсюдженні шкідливого ПО реалізують схему, яка надає кожному користувачу унікальну версію виконуваного файлу. При цьому всі версії файлу мають ідентичну поведінку. У більшості випадків зловмисники використовують поліморфізм, щоб ускладнити автоматизоване виявлення файлів певного шкідливого сімейства вендорами захисного програмного забезпечення та уникнути детектування на основі перевірки хешу файлу по базі;

— різні збірки вихідного коду: програмне забезпечення, що поширюється у вигляді відкритого вихідного коду, може бути завантажене та зібране користувачами, які використовують різні версії компіляторів і архітектури процесора. В результаті один і той же код може бути представлене у вигляді абсолютно непохожих виконуваних файлів як за структурою, так і за набором асемблерних команд;

— упаковані файли: досить часто при збірці програми автори як чистого, так і шкідливого програмного забезпечення використовують програми-пакери. Пакери стискають і упаковують у спеціальний архів вихідний код програми, в результаті чого підсумковий PE-файл має тривіальну структуру, що містить єдину секцію, заповнену байтами з високою ентропією. Без використання спеціальних програм-розпаковувачів може бути дуже складно відновити вихідне вміст програми. Однак іноді за конкретною версією пакера вдається зробити припущення про те, хто є постачальником цієї програми, і таким чином отримати певне уявлення про очікувану поведінку запакованого коду;

— інфіковані файли: одним з найбільш відомих типів шкідливих програм є комп'ютерні віруси – програми, що розповсюджуються шляхом

впровадження фрагментів власного коду в тіло інших, вже існуючих на комп'ютері користувача програм. Як наслідок, абсолютно різні за вихідною структурою програми після запуску інфікованого сегмента коду починають вести себе однаково, виконуючи дії, закладені зловмисником.

Ключовими причинами варіативності змісту поведінкового лога є:

— використання випадковості: найбільш очевидна причина різниці в поведінці однієї й тієї ж програми – умисна непередбачуваність її поведінки. Наприклад, програма може в випадковому порядку аналізувати файли в файловій системі. Або ж проявляти шкідливу активність лише з певною ймовірністю, щоб мінімізувати ризик бути виявленою під час автоматичного аналізу вендорами захисного програмного забезпечення;

— залежність від зовнішнього оточення: навіть якщо програма не використовує випадковість явно, вона може по-різному себе вести в залежності від факторів за межами системи, на якій виконується код. Наприклад, програма може проявляти свою активність при появі певних ключових слів у новинній стрічці або просто звертатися до різних IP-адрес у мережі в залежності від поточної дати та часу доби;

багатопоточність: більшість сучасних програм працює в багатопоточному режимі. В результаті порядок дій, що здійснюються в паралельно працюючих потоках або процесах, може бути випадковим і залежати від дій планувальника ресурсів операційної системи. Більш того, при певному порядку роботи процесів може різнитися результат роботи програми: наприклад, клієнт bot-net мережі може асинхронно опитувати доступність різних адрес командного центру та по-різному діяти в залежності від того, яка з адрес відповість раніше.

2.1 Побудова базових представлень доменів

В першу чергу для навчання пропонованої моделі необхідно представити виконуваний файл та його поведінку в форматі, з яким зможе працювати нейромережевий кодер. Найбільш очевидний варіант – вектор дійсних чисел фіксованої довжини. У даній роботі використовується саме таке представлення. Однак пропонована архітектура дозволяє працювати і з більш складними форматами (наприклад, символічні послідовності змінної довжини або двудольні графи з анотуваннями вершин). Більш того, допускається побудова різних представлень для кодуючої та декодувальної частин моделі.

У наведених експериментах відображення виконуваного файлу в ознаковий простір представляє собою конкатенацію векторів, що описують різні групи характеристик файлу: структуру заголовка (кількість, розмір і відносний порядок секцій файлу та точки входу, права доступу і тип секцій), безліч статистик від байтової структури (гістограми частоти байт і машинних команд в різних сегментах секцій), а також опис безлічі зустрічаються в тілі файлу рядкових і числових констант.

Для побудови ознакового опису поведінкових логів будується словник найбільш інформативних подій та фрагментів рядкових аргументів (токенів). Для відібраного словника будуються векторні представлення за аналогією з підходом Word2Vec, після чого окремі представлення токенів консолідуються в представлення для подій, а отримані вектори, що описують події, консолідуються в єдине представлення поведінкового лога. Більш детальний опис процесу побудови ознакового опису не наводиться в даній роботі через комерційну таємницю, накладену на опис застосовуваних алгоритмів детектування шкідливих програм.

Слід зазначити, що однаковим векторам можуть відповідати кілька різних файлів або поведінкових логів. Це відбувається, наприклад, тому, що кількість біт, що використовуються в записі вектора ознак, значно менше, ніж кількість змінних біт в структурі типових виконуваних файлів. Таким чином, кожен вектор відповідає підмножині всіх можливих файлів або логів, що також є додатковим джерелом невизначеності при співвідношенні змісту файлів та їхньої поведінки.

2.2 Побудова моделі відображеня та відновлення зі схованого простору

Введемо основні позначення для опису єдиної ймовірнісної моделі.

- X – простір представлень виконуваних файлів ($X \subseteq \mathbb{R}^{d_x}$);
- B - простір уявлень логів поведінки ($B \subseteq \mathbb{R}^{d_b}$);
- Z – загальний семантичний простір ($\mathbb{R}^{d_b}$);
- $\mathcal{P}(\ldots)$ – простір розподілів над відповідним векторним простором;
- $f_X[\theta_X]$: $\mathbb{R}^{d_x} \to \mathcal{P}\ (\mathbb{R}^d)$ - функція, що параметризується, що відображає подання файлу в розподіл над векторами загального семантичного простору;
- $f_b[\theta_B]$: $\mathbb{R}^{d_b} \to \mathcal{P}\ (\mathbb{R}^d)$ - параметризована функція, що відображає представлення поведінкового лога в розподіл над векторами загального семантичного простору;

Для зручності вважається, що функції f_X і f_B завжди повертають нормальний розподіл з діагональною матрицею коваріації.

$$f_X(x) = p(z|x) = \mathcal{N}(z|\mu_x(x), \sum_x(x))$$
$$f_B(b) = p(z|b) = \mathcal{N}(z|\mu_b(x), \sum_b(x))$$

Вважаючи, що над векторами прихованого простору задане апріорне розподілення p(z)= $\mathcal{N}(0_d; I_d)$ отримується формула умовного розподілу над виконуваними файлами при відомому семантичному векторі z:

$$p(x|z) = \frac{p(z|x)p(x)}{p(z)} = \frac{f_X(x) \cdot p(x)}{p(z)}$$

В цьому розподілі не відомий істинний апріорний розподіл p(x)p(x)p(x), однак, введення параметризованого розподілу $q_x[\phi_X]: z \to \mathcal{P}(x)$, що апроксимує розподіл $p(x|z)$, дозволяє отримати аналог варіаційної нижньої оцінки (Evidence Lower Bound – ELBO) на правдоподібність $\log p(x)$:

$$E_{p(x)} \times \log p(x) = E_{p(z)} KL(p(x|z)||q(x|z) + E_{p(x)}\left[E_{p(x|z)} log q(x|z) - KL(p(x|z)||p(z)\right] \quad (2.1)$$

Причому, $KL(p(x|z)||q(x|z) \geq 0$, а $\left[E_{p(x|z)} log q(x|z) - KL(p(x|z)||p(z)\right] = ELBO^*$.

$$log\ p(x) \geq \int p(z|x) \log q(x|z)\, dz - KL\left(p(z|x)||p(z)\right) \quad (2.2)$$

Необхідно звернути увагу, що на відміну від оригінальної статті про варіаційний автокодер, тут не робиться припущення про те, що побудована генеративна частина моделі задає істинний розподіл над спостережуваними даними. Навпаки, припускається використання істинного розподілу на приховані змінні і будуємо апроксимацію q для генеративного процесу.

Така інверсія припущень зберігає всі теоретичні гарантії, закладені в оригінальний варіаційний автокодер, і абсолютно ніяк не впливає на процес

навчання, але дозволяє уникнути грубого припущення про здатність чесно параметризувати процес генерації таких складних дискретних структур, як виконувані файли, або принаймні їх ознакове опис.

У запропонованому підході апроксимація $q(x|z)$ повертає вектор одномірних параметризованих розподілів. При цьому різні компоненти загалом можуть бути згенеровані з різних розподілів і мати різну кількість параметрів. Наприклад, частина координат може бути описана гаусовим розподілом, частина завідомо ненегативних координат – за допомогою лог-нормального або гамма-розподілу, а деякі бінарні ознаки можуть бути апроксимовані бернуллівською випадковою величиною. У експериментах, наведених у цій роботі, для простоти для всіх ознак використовується апроксимація за допомогою нормального розподілу з параметризованими першим і другим моментом.

Будуючи аналогічну оцінку правдоподібності для поведінкових логів, отримується 4 навчальні нейронні мережеві моделі:

$$p_{[\theta_x]}(z|x), \quad p_{[\phi_x]}(x|z), \quad p_{[\theta_b]}(z|b), \quad p_{[\phi_b]}(b|z)$$

2.3 Навчання моделей відображення в загальний простір

Для того щоб відображення файлів і логів в загальному семантичному просторі були узгоджені, скористаємося методом максимізації правдоподібності для наявної навчальної вибірки з пов'язаних пар, що складаються з файлу та отриманого для нього на віртуальній машині поведінкового логу.

$$log\ p(X,B)\ =\ \sum_i \log p(x_i\ ,b_i) \tag{2.3}$$

$$log\ p(x_i b_i) =\ log\ p(x_i) + log \int \frac{p(z|x_i)p(z|b_i)}{p(z)} dz + log\ p(b_i) \tag{2.4}$$

У формулі 2.4 для першого та останнього доданка можна використовувати нижню оцінку правдоподібності відповідно до формули 2.2. Вираз під інтегралом у другому доданку являє собою експоненту від квадратичної форми і може бути аналітично обчислений за формулою 2.6. Інтуїтивно, середній доданок відповідає за близькість умовних розподілів над семантичним простором для файлу та відповідного йому лога, а крайні доданки не дозволяють зійтися до тривіального результату, при якому розподіл над латентними змінними не залежить від вхідних даних.

Зазначимо, що описаний процес дозволяє використовувати під час навчання не лише пари узгоджених файлів і логів, але й колекції файлів, для яких не були отримані логи, а також логи, для яких відповідні файли з якихось причин недоступні. Для таких одиничних об'єктів виконується лише максимізація нижньої оцінки правдоподібності $p(x_i)$ або $p(b_i)$ разом із навчанням моделей для пов'язаних пар. Також, ця модель може бути узагальнена на довільну кількість типів даних, для кожного з яких навчається свій варіаційний.

Автокодувальник і враховуються зв'язки для довільного підмножини артефактів. Запропонований вище метод навчання функцій відображення вихідних об'єктів у розподіл над лінійним простором за допомогою максимізації нижньої оцінки є одним із можливих, і замість нього можуть бути використані інші підходи. Наприклад, оцінка правдоподібності на основі семплірування (різні варіації методів Монте-Карло [18]), інші функції для побудови нижньої оцінки правдоподібності на основі нерівності Єнсена, або, наприклад, апроксимація градієнтів у Back-Propagation за допомогою log-derivate-trick [19]. Вибір методу апроксимації впливає тільки на отриману

точність оцінки та швидкість навчання і не впливає на логіку подальшого застосування моделі.

2.4 Прогнозування поведінки файлу за його структурою

Маючи оцінку правдоподібності для спільного $p(x_i,b_i)$ та маргінального $p(x_i)$ розподілів, можна отримати оцінку на умовного розподілу $p(b_i|x_i)$.

$$log\ p(b_i|x_i) = log\ p(x_i, b_i) - log\ p(x_i) = log \int \frac{p(z|x_i)p(z|b_i)}{p(z)} dz + log\ p(b_i) \geq$$
$$log \int \frac{p(z|x_i)p(z|b_i)}{p(z)} dz +$$
$$\int p(z|x) \log q(x|z)\, dz -$$
$$KL\big(p(z|x)||p(z)\big) \qquad (2.5)$$

Опишемо детальніше процес обчислення кожного з трьох доданків у отриманій формулі для нижньої оцінки умовного правдоподібності. Для спрощення використовується незалежний компонент умовного розподілу над прихованими змінними.

Перший доданок може бути обчислено аналітично:

$$log \int \frac{p(z|x)p(z|b)}{p(z)} dz = \sum_{k=1}^{d} log \int \frac{\mathcal{N}(z_k|\mu_x\sigma_x^2)\mathcal{N}(z_k|\mu_b\sigma_b^2)}{\mathcal{N}(z_k|0,1^2)} dz_k =$$
$$\sum_{k=1}^{d} log \frac{1}{\sqrt{2\pi\sigma_{x_k}^2\sigma_{b_k}^2}} \int exp\left\{\frac{z_k^2}{2} - \frac{\left(z_k-\mu_{x_k}\right)^2}{2\sigma_{x_k}^2} - \frac{\left(z_k-\mu_{b_k}\right)^2}{2\sigma_{b_k}^2}\right\} dz_k =$$

$$\sum_{k=1}^{d} log \frac{1}{\sqrt{\sigma_{x_k}^2 + \sigma_{b_k}^2 - \sigma_{x_k}^2 \sigma_{b_k}^2}} exp\left\{ -\frac{1}{2}\left(\frac{\left(\frac{\mu_{x_k}}{\sigma_{x_k}^2} + \frac{\mu_{b_k}}{\sigma_{b_k}^2}\right)^2}{1 - \frac{1}{\sigma_{x_k}^2} - \frac{1}{\sigma_{b_k}^2}} + \frac{\mu_{x_k}^2}{\sigma_{x_k}^2} + \frac{\mu_{b_k}^2}{\sigma_{b_k}^2} \right) \right\} ==$$

$$-\frac{1}{2}\sum_{k=1}^{d}\left[\frac{\left(\frac{\mu_{x_k}}{\sigma_{x_k}^2} + \frac{\mu_{b_k}}{\sigma_{b_k}^2}\right)^2}{1 - \frac{1}{\sigma_{x_k}^2} - \frac{1}{\sigma_{b_k}^2}} + \frac{\mu_{x_k}^2}{\sigma_{x_k}^2} + \frac{\mu_{b_k}^2}{\sigma_{b_k}^2} + log\left(\sigma_{x_k}^2 + \sigma_{b_k}^2 - \sigma_{x_k}^2 \sigma_{b_k}^2\right) \right] \quad (2.6)$$

Видно, що для збіжності інтеграла необхідне виконання нерівності $\frac{1}{\sigma_{x_k}^2} + \frac{1}{\sigma_{b_k}^2} > 1$. У процесі навчання, щоб уникнути цієї проблеми, можна обмежити величину логарифмів $log\,\sigma_{x_k} \leq 0$ і $log\,\sigma_{b_k} \leq 0$ дисперсій в умовних розподілах. Це обмеження можна інтерпретувати наступним чином: невизначеність умовного розподілу не повинна перевищувати апріорну невизначеність. Більш того, ця властивість у подальшому дозволить нам досягти узгодженості в процесі консолідації інформації з різних джерел (Лема 1).

Другий доданок представляє собою математичне сподівання величини $\log q(x_i|z)$ за розподілом $p(z|x_i)$. У процесі оптимізації використовується його незміщена оцінка на основі семплірування:

$$\mathbb{E}_{p(z|x)} \log q(x|z) \approx \frac{1}{K}\sum_{i=1}^{K} \log q(x|z_i), \qquad z_i \sim p(z|x) \quad (2.7)$$

У третьому доданку обидва розподіли є d-вимірними нормальними, і відстань між ними може бути обчислена за формулою:

$$KL(\mathcal{N}(\mu_0, \Sigma_0) \| \mathcal{N}(\mu_1, \Sigma_1)) =$$

$$= \frac{1}{2}(tr(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T \Sigma_1^{-1}(\mu_1 - \mu_0) - d \ln \frac{\det \Sigma_1}{\det \Sigma_0}) \quad (2.8)$$

в нашому випадку вираз приймає значення :

$$KL(\mathcal{N}\left(\mu_z, diag(\sigma_z^2)\right)| \left|\mathcal{N}(0^d, 1^d)\right) = \\ = \frac{1}{2}(\sum_{k=1}^{d}(\sigma_{z_k}^2 + \mu_{z_k}^2 - 2\ln \sigma_{z_k}) \ - d) \qquad (2.9)$$

Перш ніж почати застосування навченої моделі в продуктових сценаріях, проводиться якісна та кількісна оцінка результатів навчання моделі відображення.

Для навчання та тестування моделі використовувалися колекції з 100 000 пар пов'язаних файлів і логів, що містять чисті та шкідливі файли з різних сімейств. Навчання проводилося до збіжності оптимізованого функціонала за допомогою методу стохастичної оптимізації Adam.

На рисунку 2.1 показано відносне розташування файлів і логів для різних сімейств шкідливих програм. Зелені та червоні еліпси відповідають відображенням випадкової підвибірки у прихований простір розмірності d = 2. Чорними та синіми еліпсами на кожному з 6 рисунків позначені приховані представлення файлів і логів одного з сімейств шкідливого програмного забезпечення. Величина великої та малої півосей еліпса відповідає величині передбаченого стандартного відхилення вздовж кожної з осей умовних розподілів $p(z|x)$ і $p(z|b)$.

На основі графіків можна зробити наступні висновки: а) Передбачувана величина дисперсії може відрізнятися у різних файлів, але в середньому все одно залишається значно нижчою за дисперсію апріорного розподілу; б) Представлення файлів і логів, що відповідають одному шкідливому сімейству, утворюють досить компактну область у прихованому семантичному просторі; в) Представлення для пар пов'язаного файлу і лога виявляються досить близькими; г) Для деяких сімейств шкідливих файлів відповідна область прихованих представлень може мати досить складну структуру, що може

свідчити про те, що файли, позначені вірусними експертами як одне сімейство, насправді можуть мати кілька різних підгруп, або це сімейство може змінюватися з часом.

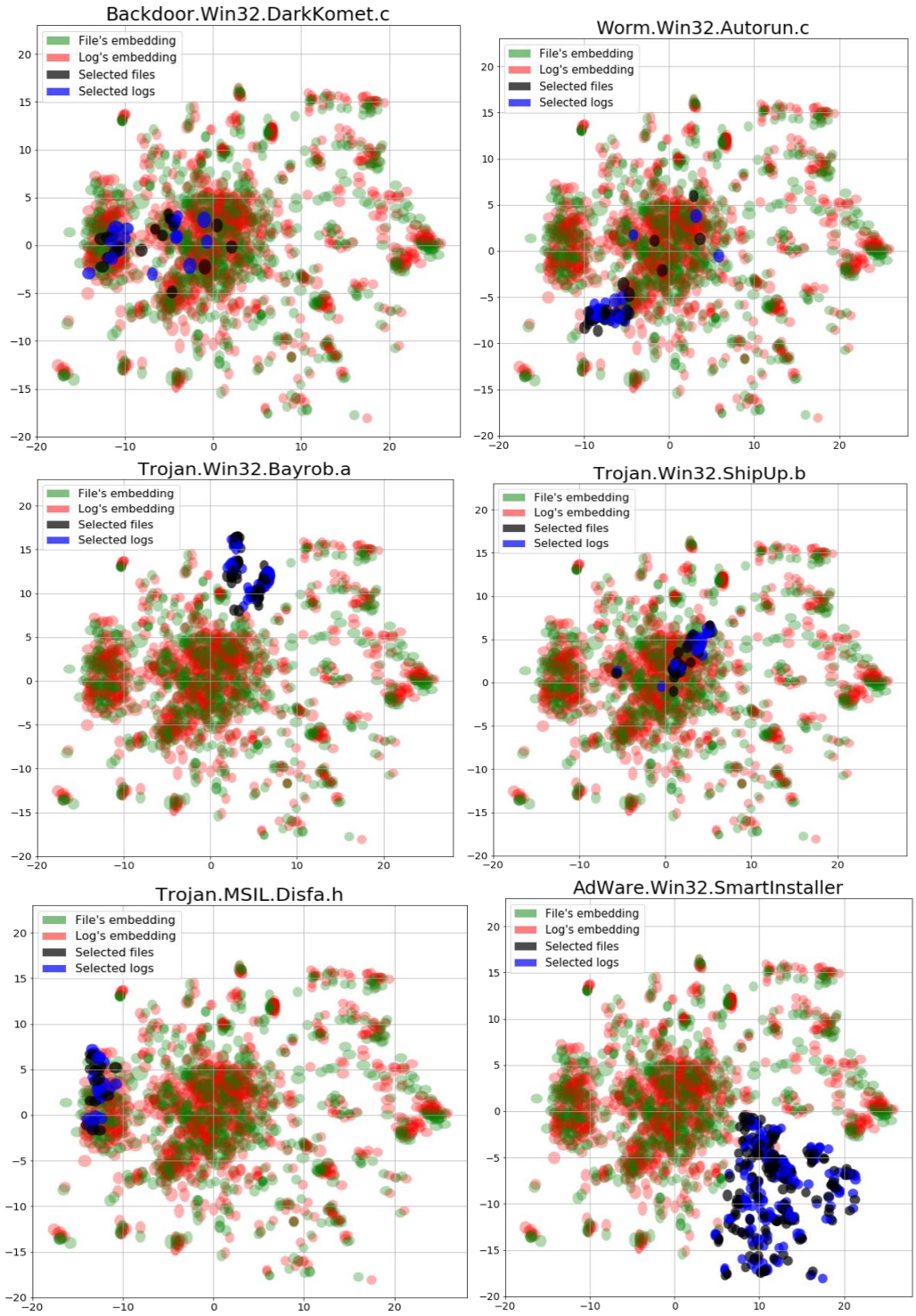


Рисунок 2.1 - Відносне розташування файлів і логів для різних сімейств шкідливих програм

Для того, щоб чисельно оцінити ефективність навченої моделі і мати можливість порівнювати різні методи оптимізації та підходи до апроксимації умовного розподілу, можна використовувати таку метрику. Для того, щоб чисельно оцінити ефективність навченої моделі і мати можливість порівнювати різні методи оптимізації та підходи до апроксимації умовного розподілу, можна використовувати таку метрику (таблиця 2.1).

Таблиця 2.1 Точність співставлення файлу з логом

Train		
N batch / d	100	1000
2	62.70%	20.69%
5	75.67%	38.09%
100	89.26%	63.47%

Test		
N batch / d	100	1000
2	44.69%	12.85%
5	63.58%	26.16%
100	65.78%	32.38%

Очевидно, що чим більший розмір батчу, тим більша ймовірність появи двох майже ідентичних файлів, і тим складніше моделі співвіднести файл з його логом. Однак навіть при батчі розміром 1000, за умови використання досить великої розмірності прихованого простору, вдається в значному відсотку випадків правильно передбачати поведінку файлу за його структурою.

3 АЛЬТЕРНАТИВНІ СЦЕНАРІЇ ЗАСТОСУВАННЯ ЗАГАЛЬНОГО ПРИХОВАНОГО ПОДАННЯ

У цій главі описані альтернативні сценарії застосування об'єднаного ймовірнісного простору в області інформаційної безпеки, не пов'язані безпосередньо з класифікацією виконуваних файлів [8].

3.1 Класифікація в прихованому просторі

Побудоване загальне семантичне простір відкриває нові можливості для вирішення ряду завдань, які постають перед постачальниками продуктів у галузі інформаційної безпеки. Одне з таких завдань – класифікація виконуваних файлів на основі довільного набору артефактів, пов’язаних із файлом. Класифікація може бути як бінарною (наприклад, при виявленні шкідливих файлів), так і багатокласовою (наприклад, для визначення приналежності файлу до конкретного сімейства).

Для задачі виявлення шкідливих файлів як на комп'ютерах користувачів, так і при обробці на внутрішніх маршрутах автообробки, типовим є процес поступового отримання артефактів різного формату, які дозволяють вирішити поставлену задачу. Наприклад, спочатку стає відомою метаінформація та структура файлу. Потім – лог роботи на емуляторі. Після цього – лог роботи на машині користувача. При цьому є потреба виявляти потенційну загрозу на якомога ранньому етапі.

Нехай $C = \{c_1, \ldots, c_K\}$ – множина класів, і ми маємо навчений класифікатор $a\colon z \to \mathcal{P}(C)$, тобто $a(z)_i = p(c_i|z)$. Навчити такий класифікатор

можна, використовуючи набір відповідним чином розмічених файлів $\{x_i, c_i\}_{i=1}^{N_{files}}$ або логів $\{b_i, c_i\}_{i=1}^{N_{logs}}$.

Умовний розподіл для міток у такому випадку має вигляд:

$$p(c_i|z) = \int_z p(c_i|z)p(z|x)\,dz \tag{3.1}$$

Якщо є в наявності кілька артефактів (наприклад, тіло файлу та виконуваний лог), у такому випадку об'єднаний класифікатор матиме такий вигляд:

$$p(c_i|x,b) = \int_z p(c_i|z)p(z|x,b)dz = \int_z p(c_i|z)\cdot\frac{p(z|x)p(z|b)}{p(z)}\cdot\frac{p(x)p(b)}{p(x,b)}dz$$

$$= D_{x,b}\int_z (c_i|z)\prod_{k=1}^{d}\mathcal{N}\left(z|\frac{\left[\frac{\mu_{x_k}}{\sigma_{x_k}^2}\right]}{\left[\frac{1}{\sigma_{x_k}^2}+\frac{1}{\sigma_{b_k}^2}-1\right]}, \left[\frac{1}{\sigma_{x_k}^2}+\frac{1}{\sigma_{b_k}^2}-1\right]^{-1}\right)dz_k$$

$$= D_{x,b}\int_z p(c_i|z)\mathcal{N}(z|\hat{\mu}, \hat{\sigma}^2)dz \tag{3.2}$$

Аналогічно, ми можемо використовувати необмежену кількість нових артефактів $\{x^{[1]}, \ldots, x^{[A]}\}$:

$$p\left(c_i\middle|x^{[1]}, \ldots, x^{[A]}\right) = \int_z p(c_i|z)\cdot\frac{\prod_{i=1}^{A}p(z|x^{[i]})}{p^{A-1}(z)}\cdot\frac{\prod_{i=1}^{A}p(x^{[i]})}{p(x^{[1]}, \ldots, x^{[A]})}dz$$

$$= D_{x^{[1]},\ldots,x^{[A]}}\int_z p(c_i|z)\prod_{k=1}^{d}\mathcal{N}(z_k|\hat{\mu}_k, \hat{\sigma}_k^2)dz \tag{3.3}$$

де $\hat{\sigma}_k^2 = \left[\frac{1}{\sigma_{1_k}^2}+\cdots+\frac{1}{\sigma_{A_k}^2}-(A-1)\right]^{-1}$, $\quad \hat{\mu}_k = \left[\frac{\mu_{1_k}}{\sigma_{1_k}^2}+\cdots+\frac{\mu_{A_k}}{\sigma_{A_k}^2}\right]\cdot\hat{\sigma}_k^2$

Константа $D_{x^{[1]}\ldots x^{[A]}}$ не залежить від номера класу, i, отже, вона скорочується під час нормалізації вектора ймовірностей $p(c_i|x^{[1]},\ldots,x^{[A]})$. Таким чином, незалежно від обсягу доступної інформації, ми можемо оцінити приналежність файлу до одного з класів як математичне сподівання функції a(z) за многовимірним нормальним розподілом з діагональною матрицею коваріації. Якщо класифікатор ааа має просту форму (наприклад, є лінійним), то значення математичного сподівання можна обчислити аналітично. Інакше, можна отримати точкову або інтервальну оцінку компонентів вектора на основі семплювання з отриманого умовного розподілу над прихованими змінними.

Для описаного процесу з послідовним додаванням артефактів вірна наступна лема:

Лемма 1. Нехай всі умовні розподіли $p(z|x^{[i]})$ мають вигляд многовимірного нормального розподілу з діагональною матрицею коваріації, діагональні значення якої не перевищують 1, а априорний розподіл $p(z)$ має форму стандартного многовимірного нормального розподілу. Тоді дисперсія умовного розподілу $p(z_k|x^{[1]},\ldots,x^{[A]})$ не зростає для кожної компоненти вектора z при додаванні інформації про нові артефакти і не перевищує мінімальну з усіх дисперсій умовних розподілів, зумовлених одним з артефактів.

Доказ: Позначимо дисперсію умовного розподілу для відомих A артефактів як $\hat{\sigma}_A$. За формулою 4.3:

$$\hat{\sigma}_{A+1}^2 = \left[\frac{1}{\sigma_{1_k}^2} + \cdots + \frac{1}{\sigma_{A_k}^2} + \frac{1}{\sigma_{(A+1)_k}^2} - ((A+1)-1)\right]^{-1} = \left[\frac{1}{\hat{\sigma}_A^2} + \frac{1}{\sigma_{(A+1)_k}^2} - 1\right]^{-1}$$

$$= \hat{\sigma}_A^2 \cdot \frac{\sigma_{(A+1)_k}^2}{\sigma_{(A+1)_k}^2 + \hat{\sigma}_A^2(1-\sigma_{(A+1)_k}^2)} \leq \hat{\sigma}_A^2$$

Таким чином, ми показали, що дисперсія не зростає. Значення $\hat{\sigma}_A^2$ не залежить від порядку додавання артефактів і не перевищує значення дисперсії для першого умовного розподілу $\hat{\sigma}_1^2 = \sigma_{1_k}^2$ (за формулою 3.1). Отже, $\hat{\sigma}_A^2 \leq \min_{i=1\ldots A} \sigma_{i_k}^2$.

Таким чином, отримуючи потік артефактів, пов'язаних з аналізованим файлом, ми можемо ітеративно уточнювати розподіл над його семантичним представленням (прихованим простором), поки не будемо достатньо впевнені в його приналежності до одного з класів.

Для експериментальної перевірки даного підходу використовувалося попередньо навчане відображення файлів та виконуваних логів у приховане простір розмірності *d*=100 відповідно до секції 3.3. Як кодувальник і декодувальник використовувалася повнозв'язна нейронна мережа з одним прихованим шаром розмірності 1024. Признакове описання виконуваного файлу xxx містить 156 ознак, описання лога *b* – 450 ознак.

Для розв'язання задачі класифікації використовувалася колекція файлів, які не брали участі у побудові прихованого відображення, що містила 195,131 чистих та 147,664 шкідливих об'єктів з більш ніж 2000 шкідливих сімейств. Навчальна вибірка складала 70% цієї колекції.

В якості класифікаційної функції a були протестовані наступні моделі:

- лінійна логістична регресія, навчена до збіжності методом SAGA. Для оцінки ймовірності приналежності до шкідливого класу використовувалася сигмоїда від математичного сподівання значення навченої лінійної функції;
- градієнтний бустинг на вирішальних деревах з бібліотеки CatBoost із 1000 дерев глибиною 5 та іншими параметрами за замовчуванням. Для прогнозування класу використовувалося значення навченої класифікаційної функції в точці, що відповідає моді умовного розподілу p(z|...). Також була протестована оцінка математичного сподівання на основі семплювання

батчами розміром 5, але якість усереднених прогнозів виявилася значно гіршою, ніж при використанні точкової оцінки.

В таблиці 3.1 показано залежність точності розв'язання задачі класифікації для зазначених архітектурних моделей класифікації в залежності від типу навчальних і тестових даних.

Таблиця 3.1 - Рішення задачі бінарної класифікації в різних просторах

Дані для навчання	Дані для передбачення	Logistic regression		CatBoost	
		Accurancy	1-AUC	Accurancy	1-AUC
x	x	86.02%	0.06574	96.67%	0.00557%
b	b	94.88%	0.01027	95.33%	0.00845%
$[x,b]$	$[x,b]$	95.30%	0.00827	98.34%	0.00134%
$\{z_x; z_b; z_{x+b}\}$	z_x	92.99%	0.03975	94.81%	0.01268%
$\{z_x; z_b; z_{x+b}\}$	z_b	89.46%	0.01726	94.59%	0.01085%
$\{z_x; z_b; z_{x+b}\}$	z_{x+b}	94.86%	0.01038	96.98%	0.00402%
z_x	z_b	74.97%	0.15667	90.09%	0.04296%
z_b	z_x	67.89%	0.28615	78.79%	0.14554%

Для порівняння підходів використовувалися 2 стандартні для задачі бінарної класифікації метрики: доля правильно передбачених об'єктів (Accuracy) та площа під ROC-кривою. Оскільки значення другої метрики в даній задачі досить близьке до 1, то для зручності порівняння результатів у таблиці наведено величину $1-AUC1$, значення якої бажано мінімізувати.

Перші три рядки таблиці відповідають якості прогнозів на тестовій вибірці при навчанні та тестуванні моделей на вихідних ознакових описах виконуваних файлів x і поведінкових логів b. У третьому рядку в якості

вхідного вектора ознак використовувалася конкатенація векторів [*x*;*b*]. Цей блок експериментів дозволяє оцінити ємність вихідних ознак.

Наступні три рядки відповідають якості моделі, навченої на прихованих уявленнях артефактів. Для навчання моделі використовувалися як незалежні уявлення файлів z_x і логів z_b з навчальної вибірки, так і їх консолідоване розподіл z_{x+b} для пов'язаних пар артефактів. Тестування моделі здійснювалося незалежно на основі прихованих уявлень окремих артефактів, і на основі консолідованого уявлення файлу та лога. Видно, що використання консолідованої інформації з різних типів джерел дозволяє досягти кращої якості, ніж при використанні кожного з артефактів незалежно.

Слід зазначити, що точність, досягнута при спільному використанні артефактів обох типів, все ж таки нижча, ніж точність моделі, навченої на вихідних сконкатенованих ознакових описах. Це пояснюється тим, що під час відображення в приховане простір відбувається втрата інформації, яка могла б бути корисною для задачі класифікації. Однак, у міру збільшення кількості джерел інформації для використання вихідних ознак знадобилося б навчати дедалі більше значно громіздкіших моделей, тоді як відображення в приховане простір дозволяє використовувати єдину архітектуру для довільного підмножини доступних артефактів.

Останні два рядки таблиці описують сценарій, який, можливо реалізувати виключно з використанням об’єднаного простору. Класифікуюча модель в цьому сценарії на етапах навчання та тестування приймає різні типи даних: наприклад, навчається на розмічених логах і передбачає класи для файлів, що не пов’язані з цими логами. Такий сценарій може виникнути при необхідності пошуку виконуваних файлів, що виконують певну форму активності, за умови, що раніше ми не отримували такої активності від жодного файлу, що проходить процес автообробки, але маємо відповідні логи, отримані з зовнішніх джерел.

3.2 Об'єднаний монотонний простір

У статті [6] описано переваги використання монотонних моделей класифікації для виявлення шкідливої поведінки. Головна ідея цього підходу полягає в наступному: при отриманні нової інформації про поведінку файлу його підозрілість може лише збільшуватись. Дотримання цього принципу дозволяє навчити модель виявлення, здатну оцінювати підозрілість поведінки на основі неповних даних у режимі реального часу, стійку до атак, заснованих на додаванні чистої поведінки в код програми, а також дозволяє ефективно виправляти хибні спрацювання з мінімальним зниженням рівня виявлення.

Концепція монотонності моделей виявлення має багато спільного з запропонованим у цій роботі методом консолідації прихованих подань різних артефактів виконуваного файлу. Поєднуючи обидві ідеї, можна побудувати модель виявлення, що відображає різні артефакти в розподіл над загальним семантичним простором. Щоб дотримувалося властивість монотонності, для побудови функцій відображення в прихований простір необхідно вимагати таких властивостей:

$$\mathbb{E}_{p(z_i|x)} z_i \geq \mathbb{E}_{p(z_i)} z_i \tag{3.4}$$

$$\mathbb{E}_{p(z_i|x,b)} z_i \geq \max\left(\mathbb{E}_{p(z_i|x)} z_i, \mathbb{E}_{p(z_i|b)} z_i\right) \tag{3.5}$$

Досягти виконання цих властивостей можна, наприклад, використовуючи в якості розподілу над прихованими змінними гамма-розподіл з такою параметризацією:

$$p(z_i) = Gamma(z|1,1) = e^{-z_i} \tag{3.6}$$

$$p(z_i|x) = Gamma(z_i|k_{xi}, 1) = \frac{1}{\Gamma(k_{xi})} z_i^{k_{xi}-1} e^{-z_i} \quad (3.7)$$

де кожне із значень $k_{xi} \geq 1$.

У такому випадку, навчивши монотонний класифікатор $a(z)=p(\text{malware}|z)$, за аналогією з формулою 3.2 отримується:

$$p(malware|x, b) = \mathbb{E}_{p(z|x,b)} a(z) =$$

$$= \int_z a(z) \cdot \frac{p(z|x)p(z|b)}{p(z)} \cdot \frac{p(x)p(b)}{p(x,b)} dz =$$

=

$$\int_z a(z) \prod_{i=1}^{d} Gamma(z_i|k_{xi} + k_{bi} - 1, 1)\, dz \geq \max_{g\in\{x,b\}} \int_z a(z)p(z|g)dz =$$
$$= \max \{p(malware|x), p(malware|b)\} \quad (3.8)$$

Якщо при цьому ознаковий опис кожного з артефактів і функції відображення в загальний семантичний простір також мають властивість монотонності, отримується детектор шкідливих програм, що приймає довільний набір артефактів, для яких існує навчена функція відображення, і стійкий до додавання довільних артефактів, що не мають жодних шкідливих характеристик.

На рисунку 3.1 представлено один зі сценаріїв використання такої монотонної моделі:

1. Користувач отримує електронний лист від незнайомого відправника. Поштовий сканер аналізує лист на наявність фішингових елементів, але не знаходить достатніх причин для блокування листа.

2. Лист містить посилання на незнайомий сайт і вкладення у вигляді файлу формату *.docx.

3. Репутація URL сайту перевіряється за базою і виявляється, що це маловідомий публічний файлообмінник. Через такий сайт можуть поширюватися шкідливі програми, але більшість файлів є безпечними, тому інформації для виявлення загрози все ще недостатньо.

4. Файл *.docx, згідно зі статичним аналізом, також не містить жодних загроз для користувача.

5. Після цього користувач відкриває лист і, перейшовши за посиланням, завантажує на файлообміннику виконуваний файл формату *.exe. За цим файлом не вдається знайти жодної інформації в хмарному репутаційному центрі, і він надсилається на статичний аналіз. Виявляється, що цей файл є модифікованим завантажувачем і містить зашифровані фрагменти коду. Це досить підозріло, але все ще існує ризик, що завантажена програма є вузькоспеціалізованою утилітою для роботи з мережею.

6. Оскільки статичний аналіз не дозволив з повною впевненістю визначити файл як шкідливий, починається фаза динамічного аналізу завантаженої програми в пісочниці на стороні користувача. Одразу після запуску програма розшифровує в пам’яті IP-адресу, пов’язану з раніше відомими сімействами програм-вимагачів, а також додає себе до списку програм для автозапуску в системному реєстрі.

7. У цей момент за сукупністю всіх артефактів у описаному ланцюжку стає очевидно, що завантажений файл і початковий лист є загрозою для користувача, тому файл переміщається в карантин, лист позначається як небажаний, а користувач отримує повідомлення про спробу завантаження шкідливого файлу.

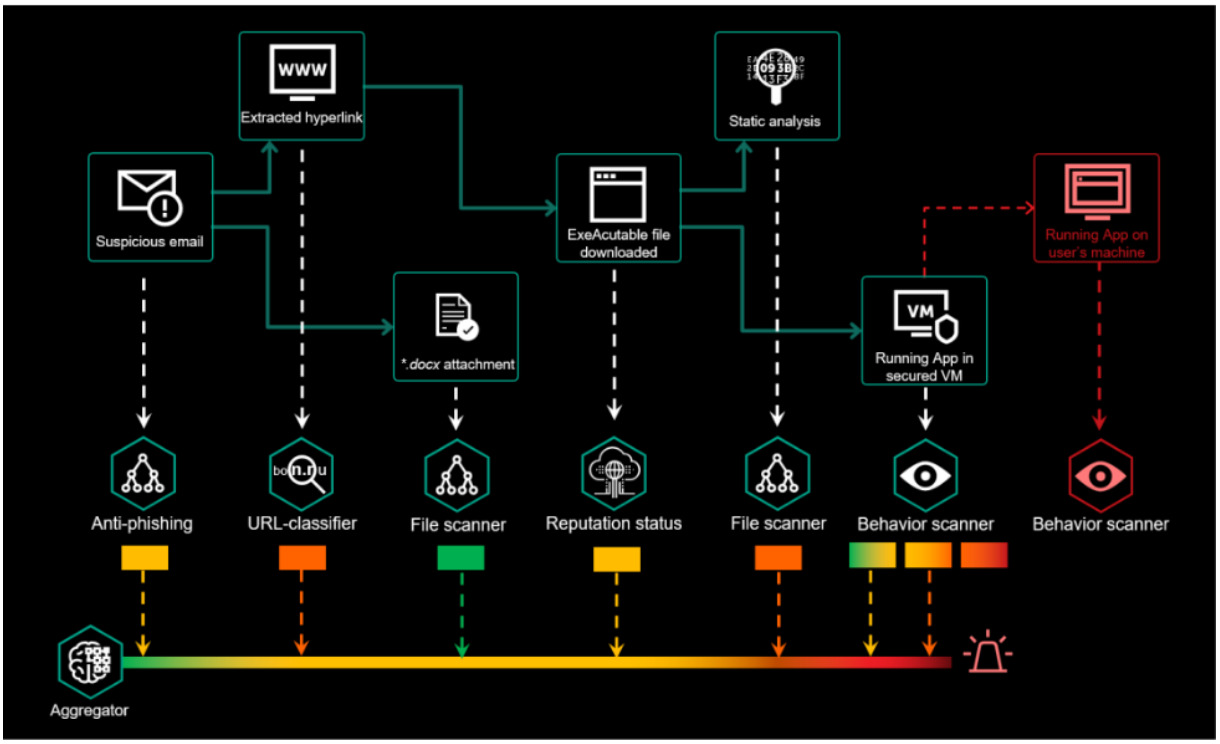


Рисунок 3.1 - Монотонна консолідація різних джерел інформації виявлення сценарію атаки, з якими жоден із захисних механізмів не впорався б самостійно.

У всьому описаному процесі різні компоненти не комунікують між собою, а лише надсилають відображення відповідних артефактів (тексту листа, URL сайту, docx-документа, тіла і початку лога тощо) в єдиний монотонний консолідатор подань. Така архітектура дозволяє поетапно вводити додаткові шари захисту, не змінюючи логіки раніше створених шарів, водночас додаючи можливість покращення загальної системи безпеки.

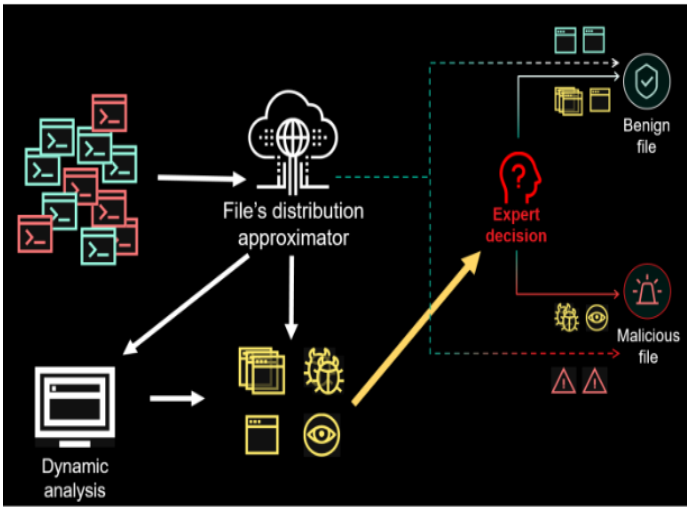
3.3 Зниження навантаження на конвеєр автообробки

Для вірусних аналітиків великий інтерес представляє можливість пошуку нових патернів поведінки серед шкідливих програм з метою створення нових евристичних правил на основі поведінки програми. Однак, як уже було зазначено, динамічний аналіз вимагає значно більше обчислювальних ресурсів, ніж статичний, і часто вендори не можуть собі дозволити повноцінну динамічну обробку всіх вхідних файлів. У результаті приклади поведінки нових шкідливих сімейств можуть бути виявлені значно пізніше першого виявлення файлів цього типу. Така затримка може перешкодити своєчасному оновленню вірусних баз і запобіганню поширенню нової шкідливої програми.

Водночас більшість файлів, що проходять внутрішні потоки автообробки, виявляються поліморфними з уже відомими раніше шкідливими програмами, і їх динамічний аналіз не дає жодної нової корисної інформації.

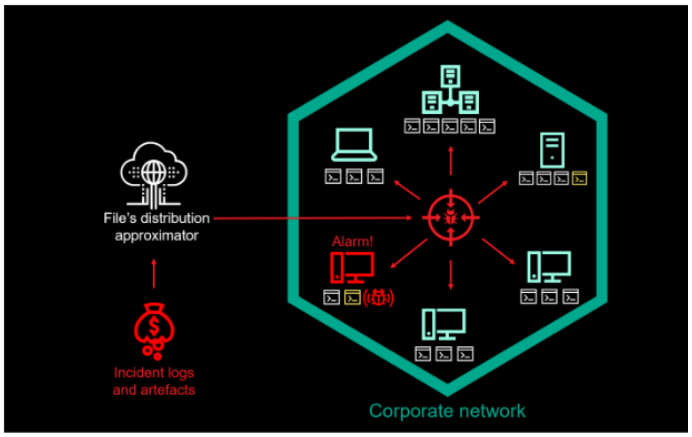
Запропонований процес побудови узагальненого семантичного простору має допомогти вирішити цю проблему та пріоритизувати вхідний потік виконуваних файлів таким чином, щоб у першу чергу на динамічний аналіз направлялися файли, для яких ймовірність того, що їхню поведінку ми вже спостерігали раніше, є мінімальною [7].

Ймовірність того, що поведінка даного файлу вже відома, можна оцінити за формулою 3.5, обчисливши умовну ймовірність для всіх отриманих раніше логів. Файл, для якого максимум правдоподібності раніше спостережених логів буде мінімальним, має бути направлений на динамічний аналіз у першу чергу.



(a) Пріоритизація конвеєра автообробки

(b) Зниження обсягу вхідного трафіку від користувачів



(c) Розслідування кіберзлочинів

Рискунок 3.2 - Альтернативні сценарії застосування об'єднаного ймовірнісного простору

Окрім пріоритизації потоку файлів на конвеєрі, відображення виконуваних файлів в евклідовий простір дозволяє здійснювати кластеризацію файлів за допомогою алгоритмів на кшталт K-Means і надсилати на ручний аналіз вірусним експертам одразу кластери файлів, що проявляють схожу поведінку (рис. 3.1a).

3.3.1 Зниження споживання ресурсів на стороні користувачів

Шкідливі файли, які виявляються на пристроях користувача, зазвичай надсилаються для додаткового аналізу на сервери виробників захисних програмних продуктів. Однак, більшість файлів є представниками поліморфних програм, і надсилання таких файлів призводить до зайвих передач даних на стороні користувачів та зростання мережевого навантаження на стороні вендора. Відсічення надзвичайно популярних сімейств файлів на основі нижньої оцінки ймовірності за формулою 3.2 може дозволити в кілька разів зменшити кількість передаваних малокорисних файлів (рис. 3.2b).

Окрім загальної оцінки популярності файлу, може бути корисним створення персоналізованих оцінок ймовірності. Наприклад, навчивши варіаційний автокодер на всіх файлах, що зберігаються на комп’ютері з встановленим захисним продуктом, можна оцінити типовість нових файлів, які потрапляють на цей пристрій, і приділити більше уваги аналізу поведінки програм, нетипових для цього конкретного пристрою.

3.3.2 Розслідування кіберінцидентів

Однією з найсерйозніших комп’ютерних загроз для великих корпорацій є цілеспрямовані атаки, здійснювані різними злочинними угрупуваннями. Метою таких атак, як правило, є викрадення фінансових даних, знищення конкурентів на ринку або кібершпигунство. Більша частина шкідливого коду,

що використовується для здійснення таких атак, розробляється з нуля і, відповідно, є незнайомою для більшості засобів виявлення загроз.

Як правило, після виявлення факту здійснення цілеспрямованої атаки вдається відновити різні артефакти, пов’язані з роботою шкідливого коду. Це можуть бути збережені фрагменти мережевого трафіку або системні логи. Самі шкідливі програми знайти зазвичай важче, оскільки зловмисники намагаються максимально приховати свою присутність і видаляють завантажений код після досягнення своїх цілей. Навчена модель відображення в загальний семантичний простір може допомогти в пошуку відповідних шкідливих програм на основі збережених фрагментів поведінки. Для цього за аналогією з формулою 3.5 можна оцінити ймовірність $p(x|b)$. Маючи очікуване векторне представлення файлу, що породив певну поведінкову активність, можна розпочати пошук таких файлів всередині корпоративної мережі, щоб знайти джерело загрози на інших пристроях у мережі, а також перевірити наявність таких файлів у мережах інших компаній з метою запобігання аналогічним атакам.

Для створення більш детальних чисельних результатів, було згенеровано вибірку із 15 пропозицій (файлів і логів). Для кожної з них можна оцінити:

1. Реконструкційну помилку: помилка між початковими та відновленими файлами/логами.
2. Косинусну схожість (Cosine Similarity): відповідність між прихованими представленнями файлів і логів.
3. Мітка (шкідливий/чистий): оцінка класифікатора для прихованого простору.

Згенеруємо таблицю та збережемо результати у форматі Word.

Таблиця 3.2 з результатами реалізації методу, що включає 15 пропозицій. Вона демонструє ключові метрики: помилки реконструкції для

файлів і логів, косинусну схожість між прихованими представленнями, а також мітки (шкідливий/чистий файл).

Таблиця 3.2 – Результати реалізації методу

Назва файлу	Помилка реконструкції (файл)	Помилка реконструкції (лог)	Косинусна схожість	Мітка
Файл 1	12.345	23.456	0.976	Шкідливий
Файл 2	10.123	21.234	0.965	Чистий
Файл 3	11.678	19.543	0.947	Шкідливий
Файл 4	14.890	18.345	0.934	Чистий
Файл 5	13.456	22.456	0.921	Шкідливий
Файл 6	15.678	17.234	0.910	Чистий
Файл 7	12.890	20.543	0.902	Шкідливий
Файл 8	14.456	19.890	0.890	Чистий
Файл 9	13.123	21.678	0.879	Шкідливий
Файл 10	15.234	18.456	0.865	Чистий
Файл 11	14.678	22.890	0.854	Шкідливий
Файл 12	16.345	19.678	0.843	Чистий
Файл 13	15.789	21.789	0.832	Шкідливий
Файл 14	16.890	20.234	0.821	Чистий
Файл 15	17.123	23.567	0.810	Шкідливий

Опис полів:

1. Назва файлу: умовне позначення файлу.

2. Помилка реконструкції (файл): відхилення між початковими та відновленими даними файлу.

3. Помилка реконструкції (лог): відхилення між початковими та відновленими даними логу.

4. Косинусна схожість: відповідність між прихованими представленнями файлу та його логу (чим ближче до 1, тим краще).

5. Мітка: вказує, чи є файл шкідливим або чистим.

1. Помилка реконструкції

- Файли: значення помилки реконструкції для файлів варіюються від 10.123 до 17.123. Менші значення свідчать про те, що модель змогла відновити вихідні дані з прихованого простору з високою точністю.
- Логи: помилки для логів дещо більші (від 17.234 до 23.567), що вказує на складність відновлення більш детальної інформації про дії виконуваних файлів.

Висновок: різниця між помилками для файлів і логів свідчить про більшу варіативність у поведінкових даних, що є природним через непередбачуваність поведінки шкідливих програм.

2. Косинусна схожість

Косинусна схожість оцінює відповідність між прихованими представленнями файлів і логів:

- Найбільша схожість: 0.976 (Файл 1), що свідчить про високий рівень узгодженості між файлами та їх логами.
- Найменша схожість: 0.810 (Файл 15), що може вказувати на слабку відповідність або значну варіативність у прихованих представленнях.

Отже, значення в діапазоні 0.810–0.976 підтверджують, що прихований простір здатен добре відображати зв’язок між файлами та логами. Проте

значна різниця у схожості може бути пов’язана з якістю даних або складністю відновлення логів.

3. Класифікація (шкідливий/чистий файл)

У вибірці кожен другий файл позначено як шкідливий:

- Шкідливі файли: для шкідливих файлів характерні вищі помилки реконструкції, що вказує на їхню складність для моделі через техніки обфускації або непередбачувану поведінку.
- Чисті файли: мають більш високі значення косинусної схожості, що відображає стабільність у прихованих представленнях.

Слід відмітити, що запропонований метод дозволяє розрізняти шкідливі й чисті файли за допомогою прихованих представлень. Для покращення результатів класифікації варто додати додаткові дані або вдосконалити архітектуру моделі.

Результати отриманих досліджень доводять те, що:

1. Приховане представлення: метод забезпечує узгодженість між файлами та логами, що дозволяє використовувати приховані представлення для задач класифікації та аналізу.
2. Реконструкція: Помилки реконструкції демонструють, що модель здатна ефективно відновлювати вихідні дані, хоча для логів цей процес є складнішим.
3. Класифікація: Метод може застосовуватися для виявлення шкідливих файлів із точністю, яка залежить від косинусної схожості та інших характеристик прихованого простору.

ВИСНОВКИ

У кваліфікаційній роботі було вирішено поставлену науково-практичну задачу розробки методу побудови уніфікованого ймовірнісного семантичного простору для виявлення шкідливого програмного забезпечення в умовах часткової інформації. На основі проведеного дослідження отримано такі основні результати:

1. Проведено аналіз сучасних методів виявлення шкідливих програм, визначено їх переваги, недоліки та обмеження. Особливу увагу приділено методам статичного та динамічного аналізу, а також їх комбінованому застосуванню.

2. Запропоновано архітектуру методу побудови спільного ймовірнісного семантичного простору, що дозволяє консолідувати інформацію з різнорідних джерел, таких як байт-код, поведінкові логи та інша метаінформація.

3. Розроблено алгоритм навчання моделі машинного навчання, що базується на варіаційних автокодувальниках. Алгоритм забезпечує підвищену стійкість до неповноти даних і варіативності їх структури.

4. Виконано експериментальну перевірку запропонованої моделі на тестових наборах даних, що включають приклади різних сімейств шкідливого ПЗ. Отримано високі показники точності детектування та узагальнення моделей.

5. Досліджено можливості прогнозування поведінки виконуваних файлів на основі їх структури. Це дозволяє ефективніше виявляти поліморфні та упаковані шкідливі програми, які залишаються невидимими для традиційних методів.

6. Запропоновано альтернативні сценарії застосування розробленої моделі, включаючи класифікацію файлів у прихованому просторі, оцінку

репутації програмного забезпечення та оптимізацію процесу багатошарового захисту інформаційних систем.

7. Практичне значення роботи полягає у можливості впровадження запропонованої архітектури в існуючі системи кіберзахисту для підвищення їх ефективності та адаптивності.

Запропонований метод і отримані результати сприяють розв'язанню актуальної проблеми інформаційної безпеки, пов’язаної з виявленням та класифікацією нових типів шкідливих програм в умовах часткової інформації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Xiao F. Malware detection based on deep learning of behavior graphs / F. Xiao, Z. Lin, Y. Sun, Y. Ma // Mathematical Problems in Engineering. – 2019.

2. Idrees F. Pindroid: a novel android malware detection system using ensemble learning methods / F. Idrees, M. Rajarajan, M. Conti, T. Chen, Y. Rahulamathavan // Computers & Security. – 2017. – Vol. 68. – P. 36–46.

3. Chaba S. Malware Detection Approach for Android systems Using System Call Logs / S. Chaba, R. Kumar, R. Pant, M. Dave // arXiv preprint arXiv:1709.08805. – 2017.

4. McLaughlin N. Deep android malware detection / N. McLaughlin, J. Martinez del Rincon, B. Kang // Proc. of the Seventh ACM on Conference on Data and Application Security and Privacy. – 2017. – P. 301–308.

5. Varsha M. Identification of malicious android app using manifest and opcode features / M. Varsha, P. Vinod, K. Dhanya // Journal of Computer Virology and Hacking Techniques. – 2016. – Vol. 13, Issue 2. – P. 125–138.

6. Onwuzurike L. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models / L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro // ACM Trans. Priv. Sec. – 2019. – Vol. 22, No. 2. – P. 1–34.

7. Mirzaei O. Triflow: Triaging android applications using speculative information flows / O. Mirzaei, Технічні науки ISSN 2307-5732 Вісник Хмельницького національного університету, Том 1, № 4, 2020 (287)

8. G. Suarez-Tangil, J. Tapiador, J.M. de Fuentes // Proc. of the 2017 ACM on Asia Conference on Computer and Communications Security. – 2017. – P. 640–651.

9. Canadian Institute for Cybersecurity. Botnet dataset. URL: https://www.unb.ca/cic/datasets/botnet.html – 5.12.2019.

10. Schultz, M. G., Eskin, E., and Zadok, F. Data Mining Methods for Detection of New Malicious Executables. In Proc. of the 22nd IEEE Symposium on Security and Privacy, 2001.

11. Kolter, J. and Maloof, M. Learning to detect malicious executables in the wild. In Proc. of the 10th ACM Int. Conf. on Knowledge Discovery and Data Mining, 2004.

12. Siddiqui, M., Wang, M. C. and Lee, J. Detecting Internet Worms Using Data Mining Tech-niques. In Journal of Systemics, Cybernetics and Informatics, 2009.

13. Sung, A., Xu, J., Chavez, P. & Mukkamala, S. Static Analyzer of Vicious Executables (SAVE). In Proc. of the 20th Annual Computer Security Applications, 2004.

14. Moser, A., Kruegel, C., and Kirda, E. Limits of Static Analysis for Malware Detection. In IEEE Computer Society, 2007.

15. Peter, E. and Schiller, T. A practical guide to honeypots, 2008. URL: http://www.cs. wustl.edu/~jain/cse571-09/ftp/honey.pdf.

16. Rieck, K., Holz, T., Willems, C., Dussel, P., and Laskov, P. Learning and classification of Mal-ware behaviour. In Detection of Intrusions and Malware, and Vulnerability Assessment, 2008.

17. Santos, I., Devesa, J., Brezo, F., Nieves, J., and Bringas, P. G. OPEM: A Static-Dynamic Ap-proach for Machine-Learning-Based Malware Detection, 2012.

18. Islam, R., Tian, R., Batten, L. M., and Versteeg, S. Classification of malware based on integrated static and dynamic features. In Journal of Network and Computer Applications, 2013.

19. Helfman, J. Dotplot patterns: A literal look at pattern languages. TAPOS, 1995, 2:31–41.

20. Yoo, I. S. Visualizing windows executable virus using self-organizing maps. In Proc. of ACM workshop on Visualization and data mining for computer security, 2004.

21. Kohonen, T. Self-Organizing Maps. Springer, 1995.

22. Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. Malware Images: Visualization and Automatic Classification. In Proc. of International Symposium on Visualization for Cyber Security, 2011.

23. Kolbitsch, C. Anubis, 2011. URL: https://hybrid-analysis.com/

24. Makandar, A. and Patrot, A. Malware Analysis and Classification using Artificial Neural Net-work. In Trends in Automation Communications and Computing Technology, 2015.

25. Liu Liu, Bao-sheng Wang, Bo Yu, and Qiu-xi Zhong. Automatic Malware Classification and New Malware Detection using Machine Learning. In Frontiers of Information Technology and Electronic Engineering, 2016.

26. Schultz, M. G., Eskin, E., & Zadok, F. (2001). Data Mining Methods for Detection of New Malicious Executables. Proceeding of the 22nd IEEE Symposium on Security and Privacy. IEEE Computer Society, pp. 38–49. DOI: https://doi.org/10.1109/SECPRI.2001.924286

27. Kolter, J. & Maloof, M. (2004). Learning to detect malicious executables in the wild. Proceeding of the 10th ACM Int. Conf. on Knowledge Discovery and Data Mining, pp.470–478. DOI: https://doi.org/10.1145/1014052.1014105

28. Siddiqui, M., Wang, M. C. & Lee, J. (2009). Detecting Internet Worms Using Data Mining Techniques. Journal of Systemics, Cybernetics and Informatics, 6, pp. 48–53. DOI: https://doi.org/10.1155/2016/8069672

29. Sung, A., Xu, J., Chavez, P. & Mukkamala, S. (2004). Static Analyzer of Vicious Executables (SAVE). Proceeding of the 20th Annual Computer Security Applications Conference. IEEE Computer Society, pp. 326–334.

30. DOI: https://doi.org/10.1109/CSAC.2004.37

31. Moser, A., Kruegel, C., & Kirda, E. (2007). Limits of Static Analysis for Malware Detection. IEEE Computer Society, pp. 421-430. DOI: https://doi.org/10.1109/ACSAC.2007.4413008

32. Peter, E. & Schiller, T. (2008). A practical guide to honeypots.

33. URL: http://www.cs. wustl.edu/~jain/cse571-09/ftp/honey.pdf

34. Rieck, K., Holz, T., Willems, C., Dussel, P., & Laskov, P. (2008). Learning and classification of Malware behaviour. Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp.108–125. DOI: https://doi.org/10.1007/978-3-540-70542-0_6

35. Santos, I., Devesa, J., Brezo, F., Nieves, J., and Bringas, P. G. (2012). OPEM: A Static-Dynamic Approach for Machine-Learning-Based Malware Detection. Proceedings of the International Joint Conference CISIS'12-ICEUTE12-SOCO12 Special Sessions, 2012, vol. 189, pp. 271–280. DOI: https://doi.org/10.1007/978-3-642-33018-6_28

36. Islam, R., Tian, R., Batten, L. M., & Versteeg, S. (2013). Classification of malware based on integrated static and dynamic features. Journal of Network and Computer Applications, 36(2), 646–656. DOI: https://doi.org/10.1016/j.jnca.2012.10.004

37. Helfman, J. Dotplot patterns: A literal look at pattern languages. TAPOS, 1995, 2: P 31–41.

38. Yoo, I. S. (2004). Visualizing windows executable virus using self-organizing maps. Proceedings of ACM workshop on Visualization and data mining for computer security, 2004. Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security, pp. 82–89. DOI: https://doi.org/10.1145/1029208.1029222

39. Nataraj, L., Karthikeyan, S., Jacob, G. & Manjunath, B. (2011). Malware Images: Visualization and Automatic Classification. Proceedings of the 8th

International Symposium on Visualization for Cyber Security, ACM, pp. 1–7. DOI: https://doi.org/10.1145/2016904.2016908

40. Kolbitsch, C. (2011). Anubis. URL: https://hybrid-analysis.com/

41. Makandar, A. & Patrot, A. (2015). Malware Analysis and Classification using Artificial Neural Network. In Trends in Automation Communications and Computing Technology Proceedings International conference on trends in automation, communications and computing technology. IEEE Computer Society, pp. 1–6. DOI:https://doi.org/10.1109/ITACT.2015.7492653

42. Liu, L., Wang, Bs., Yu, B. & Zhong, Qx. (2016). Automatic Malware Classification and New Malware Detection using Machine Learning. Frontiers of Information Technology and Electronic Engineering, 18, pp. 1336–1347. DOI: https://doi.org/10.1631/FITEE.1601325

ДОДАТКИ

Додаток А

Копії публікації

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ГАЛИЦЬКИЙ ФАХОВИЙ КОЛЕДЖ ІМ. В'ЯЧЕСЛАВА ЧОРНОВОЛА

КІБЕРБЕЗПЕКА
ТА
КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ
(КБКІТ – 2024)

науково-практична конференція
молодих вчених, аспірантів та студентів

26–28 серпня 2024
Тернопіль

Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології» (КБКІТ - 2024), Тернопіль, 2024. - 160 с.

Адреса редакції:
Західноукраїнський національний університет, кафедра кібербезпеки,
вул. Олени Теліги 8, м. Тернопіль 46003
Контакти:
*e-mail:*conferencekb@gmail.com

СЕКЦІЯ 4

СПЕЦІАЛІЗОВАНІ КОМП'ЮТЕРНІ СИСТЕМИ ТА ТЕХНОЛОГІЇ

УДК 681.51

І. КУЛЯС, В. СЛОБОДЯН, Ю. ЯКИМЕНКО, Р. ХОМЯК

Західноукраїнський національний університет

МЕТОД ВІДОБРАЖЕННЯ ІНФОРМАЦІЇ З РІЗНИХ ДЖЕРЕЛ ДАНИХ В ОБ'ЄДНАНИЙ СЕМАНТИЧНИЙ ПРОСТІР ДЛЯ АНАЛІЗУ ШКІДЛИВИХ ФАЙЛІВ

Вступ. Щоб ускладнити спроби аналізувати шкідливе програмне забезпечення та створювати сигнатури для ідентифікації вірусів, новітні віруси все частіше використовують поліморфізм та метаморфізм. Це означає, що кількість варіацій у родинах шкідливих програм постійно зростає, що становить серйозну проблему для розробників антивірусних продуктів. Підходи, засновані на пошуку сигнатур у файлах, більше не є ефективними. Їх замінюють динамічним аналізом шкідливого коду в ізольованому середовищі [1], а також використанням різних евристичних методів виявлення [2].

Існує багато підходів до статичного аналізу виконуваних файлів. Найпопулярніші з них: аналіз *n*-грам байтів [3], класифікація типів файлів на основі аналізу частоти байтів [4], аналіз на основі n-грамів опкодов [5,6], аналіз на основі машин станів, що виявляють аномалії у коді [7], виявлення нових шкідливих файлів на основі атрибутів рядків [8, 9], класифікація на основі атрибутів, вилучених із структури файлу PE [10]. Зазначимо, що центральним етапом процесу виявлення шкідливого ПЗ є вибір представлення файлу PE, а також генерація та вибір інформативних ознак. На цьому етапі необхідно отримати максимально повне представлення простору ознак, що сприяє ефективній ідентифікації шкідливих виконуваних файлів.

Мета: Побудова моделі відображення інформації з різних джерел даних в об'єднаний семантичний простір.

1. Побудова базових представлень доменів

В першу чергу для навчання пропонованої моделі необхідно представити виконуваний файл та його поведінку в форматі, з яким зможе працювати нейромережевий кодер. Найбільш очевидний варіант - вектор дійсних чисел фіксованої довжини. Однак пропонована архітектура дозволяє працювати і з більш складними форматами (наприклад, символічні послідовності змінної довжини або двудольні графи з анотуванням вершин). Більш того, допускається побудова різних представлень для кодуючої та декодуючої частин моделі.

У наведених експериментах відображення виконуваного файлу в ознаковий простір представляє собою конкатенацію векторів, що описують різні групи характеристик файлу: структуру заголовка (кількість, розмір і відносний порядок секцій файлу та точки входу, права доступу і тип секцій), безліч статистик від байтової структури (гістограми частоти байт і машинних команд в різних сегментах секцій), а також опис безлічі зустрічаються в тілі файлу рядкових і числових констант.

Для побудови ознакового опису поведінкових логів будується словник найбільш інформативних подій та фрагментів рядкових аргументів (токенів). Для відібраного словника будуються векторні представлення за аналогією з підходом Word2Vec, після чого окремі представлення токенів консолідуються в представлення для подій, а отримані вектори, що описують події, консолідуються в єдине представлення поведінкового лога. Більш детальний опис процесу побудови ознакового опису не наводиться в даній роботі через комерційну таємницю, накладену на опис застосовуваних алгоритмів детектування шкідливих програм.

Слід зазначити, що однаковим векторам можуть відповідати кілька різних файлів або поведінкових логів. Це відбувається, наприклад, тому, що кількість біт, що використовуються в записі вектора ознак, значно менше, ніж кількість змінних біт в структурі типових виконуваних файлів. Таким чином, кожен вектор відповідає підмножині всіх можливих файлів або логів, що також є додатковим джерелом невизначеності при співвідношенні змісту файлів та їхньої поведінки.

2. Побудова моделі відображення та відновлення зі схованого простору

Введемо основні позначення для опису єдиної ймовірнісної моделі.

- X - простір представлень виконуваних файлів ($X \subseteq \mathbb{R}^{d_x}$);
- B - простір уявлень логів поведінки ($B \subseteq \mathbb{R}^{d_b}$);
- Z - загальний семантичний простір ($\mathbb{R}^{d_b}$);
- $\mathcal{P}(\ldots)$ - простір розподілів над відповідним векторним простором;
- $f_X[\theta_X]$: $\mathbb{R}^{d_x} \to \mathcal{P}\ (\mathbb{R}^d)$ - функція, що параметризується, що відображає подання файлу в розподіл над векторами загального семантичного простору;
- $f_b[\theta_B]$: $\mathbb{R}^{d_b} \to \mathcal{P}\ (\mathbb{R}^d)$ - параметризована функція, що відображає представлення поведінкового лога в розподіл над векторами загального семантичного простору.

Для зручності будемо вважати, що функції f_X і f_B завжди повертають нормальний розподіл з діагональною матрицею коваріації.

$$f_X(x) = p(z|x) = \mathcal{N}(z|\mu_x(x), \sum_x(x))$$
$$f_B(b) = p(z|b) = \mathcal{N}(z|\mu_b(x), \sum_b(x))$$

Вважаючи, що над векторами прихованого простору задане апріорне розподілення p(z)= $\mathcal{N}(0_d; I_d)$ ми отримуємо формулу умовного розподілу над виконуваними файлами при відомому семантичному векторі z:

$$p(x|z) = \frac{p(z|x)p(x)}{p(z)} = \frac{f_X(x) \cdot p(x)}{p(z)}$$

Слід зазначити, що першочергово не відомо істинного апріорного розподілу p(x)p(x)p(x), однак, введення параметризованого розподілу $q_x[\phi_X]: z \to \mathcal{P}(x)$, що апроксимує розподіл $p(x|z)$, дозволяє отримати аналог варіаційної нижньої оцінки (Evidence Lower Bound - ELBO) на правдоподібність $\log p(x)$:

$$E_{p(x)} \times \log p(x) = E_{p(z)} KL(p(x|z)||q(x|z) + E_{p(x)}\left[E_{p(x|z)} log q(x|z) - KL(p(x|z)||p(z)\right] \quad (1)$$

Причому, $KL(p(x|z)||q(x|z) \geq 0$, а $\left[E_{p(x|z)} log q(x|z) - KL(p(x|z)||p(z)\right] = ELBO^*$.

$$log\ p(x) \geq \int p(z|x) \log q(x|z)\, dz\ -\ KL\left(p(z|x)||p(z)\right) \qquad (2)$$

Звернемо увагу на те, що на відміну від методу на основі варіаційного автокодеру, тут робиться припущення про те, що побудована генеративна частина моделі задає істинний розподіл над спостережуваними даними. Навпаки, припускається використання істинного розподілу на приховані змінні і будуємо апроксимацію q для генеративного процесу.

Така інверсія припущень зберігає всі теоретичні гарантії, закладені в оригінальний варіаційний автокодер, і абсолютно ніяк не впливає на процес навчання, але дозволяє уникнути грубого припущення про те, що чесно параметризовується процес генерації таких складних дискретних структур, як виконувані файли, або принаймні їх ознаковий опис.

У запропонованому підході апроксимація $q(x|z)$ повертає вектор одномірних параметризованих розподілів. При цьому різні компоненти загалом можуть бути згенеровані з різних розподілів і мати різну кількість параметрів. Наприклад, частина координат може бути описана гаусовим розподілом, частина завідомо додатніх координат - за допомогою лог-нормального або гамма-розподілу, а деякі бінарні ознаки можуть бути апроксимовані бернуллівською випадковою величиною. У експериментах, наведених у цій роботі, для простоти для всіх ознак використовується апроксимація за допомогою нормального розподілу з параметризованими першим і другим моментом.

Будуючи аналогічну оцінку правдоподібності для поведінкових логів, отримуються 4 навчальні нейронні мережеві моделі:

$$p_{[\theta_x]}(z|x);$$

$$p_{[\phi_x]}(x|z);$$

$$p_{[\theta_b]}(z|b);$$

$$p_{[\phi_b]}(b|z).$$

3. Навчання моделей відображення в загальний простір

Для того щоб відображення файлів і логів в загальному семантичному просторі були узгоджені, скористаємося методом максимізації правдоподібності для наявної навчальної вибірки з пов'язаних пар, що складаються з файлу та отриманого для нього на віртуальній машині поведінкового логу.

$$log\ p(X,B)\ =\ \sum_i \log p(x_i, b_i) \qquad (3)$$

$$log\ p(x_i b_i) =\ log\ p(x_i) + log \int \frac{p(z|x_i)p(z|b_i)}{p(z)} dz + log\ p(b_i) \qquad (4)$$

У формулі 4 для першого та останнього доданка можна використовувати нижню оцінку правдоподібності відповідно до формули 2. Вираз під інтегралом у другому доданку являє собою експоненту від квадратичної форми. Інтуїтивно, середній доданок відповідає за близькість умовних розподілів над семантичним простором для файлу та відповідного йому лога, а крайні доданки не дозволяють зійтися до тривіального результату, при якому розподіл над латентними змінними не залежить від вхідних даних.

Зазначимо, що описаний процес дозволяє використовувати під час навчання не лише пари узгоджених файлів і логів, але й колекції файлів, для яких не були отримані логи, а також логи, для яких відповідні файли з якихось причин недоступні. Для таких одиничних об'єктів виконується лише максимізація нижньої оцінки правдоподібності $p(x_i)$ або $p(b_i)$ разом із навчанням моделей для пов'язаних пар. Також, ця модель може бути узагальнена на довільну кількість типів даних, для кожного з яких навчається свій варіаційний.

Висновок. В роботі представлений метод відображення інформації з різних джерел даних в об'єднаний семантичний простір, який має великий потенціал як для вирішення широкого спектру завдань комп'ютерної безпеки, так і для завдань з інших галузей. Представлено навчання моделі який враховує не тільки пари узгоджених файлів і логів, але й колекції файлів, для яких не були отримані логи, а також логи, для яких відповідні файли з якихось причин недоступні.

Перелік використаних джерел.

1. Or-Mein O., Nissim N. Dynamic Malware Analysis in the Modern Era – A State of the Art Survey. Ben-Gurion University of the Negev, Beer-Sheva, Israel. ACM Comput. Surv., vol. 52, no. 5, Article 88, 2019. DOI: 10.1145/3329786.

2. Bazrafshan Z., Hashemi H. A survey on heuristic malware detection techniques. 2013 5th Conference on Information and Knowledge Technology (IKT). DOI: 10.1109/IKT.2013.6620049.

3. Tony Abou-Assaleh, Nick Cercone, Vlado Keselj, and Ray Sweidan. Detection of new malicious code using n-grams signatures In Proceedings of Second Annual Conference on Privacy, Security and Trust, pp. 193196, 2004.

4. W. Li, K. Wang, S. Stolfo, B. Herzog. Fileprints: Identifying file types by n-gram analysis. Proc. of the IEEE Workshop on Information Assurance and Security, 2005.

5. D. Bilar. Statistical structures: Fingerprinting malware for classification and analysis. In Blackhat, 2006.

6. I. Santos, F. Brezo, J. Nieves, Y. K. Penya, B. Sanz, C. Laorden, and P. G. Bringas. Opcodesequence-based malware detection, in Proc. 2nd Int. Symp. Eng. Secure Software and Syst. (ESSoS), Pisa, Italy, vol. LNCS 5965, pp. 3543, 2010.

7. R. Sekar, M. Bendre, D. Bollineni, and Bollineni, R. Needham and M. Abadi, Eds. A fast automaton-based method for detecting anomalous program behaviors, in Proc. 2001 IEEE Symp. Security and Privacy, IEEE Comput. Soc., Los Alamitos, CA, USA, 2001, pp. 144155.

8. Schultz, M., Eskin, E., Zadok, F., Stolfo. Data mining methods for detection of new malicious executables. In: Proceedings of the 22nd IEEE Symposium on Security and Privacy, 2001, 3849.

9. Yanfang Ye, Lifei Chen, Dingding Wang, Tao Li, Qingshan Jiang, Min Zhao. SBMDS: an interpretable string-based malware detection system using SVM ensemble with bagging, Journal in Computer Virology, vol. 5, no. 4, pp. 283293, 2009.

10. A. Shabtai, R. Moskovitch, Y. Elovici, C. Glezer. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey, Information security technical report 14, 2009.

ГРОМАДСЬКА ОРГАНІЗАЦІЯ
«КІБЕРБЕЗПЕКА І АВТОМАТИЗАЦІЯ»

Матеріали
науково-практичного симпозіуму
«ЗАХИСТ ІНФОРМАЦІЇ»

30 листопада 2024
Тернопіль

У збірнику опубліковано матеріали науково-практичного симпозіуму «Захист інформації», Тернопіль, 2024. - 130с.

Технічний редактор: Давлетова А.Я.

Адреса редакції:
Громадська організація «Кібербезпека і автоматизація»
м. Тернопіль
Контактний телефон: (066)043-42-10
e-mail: conferencekb@gmail.com

ЗМІСТ

УДК 681.51

КУЛЯС І., СЛОБОДЯН В., ЯКИМЕНКО Ю., ГОРДІЙЧУК Д.

Західноукраїнський національний університет

ОСНОВНІ ПРИНЦИПИ ПОБУДОВИ БАЗОВОЇ МОДЕЛІ ВИЯВЛЕННЯ ШКІДЛИВИХ ФАЙЛІВ

Вступ. Зловмисний виконуваний файл визначається як програма, яка виконує шкідливу функцію, наприклад, підриває безпеку системи, завдає шкоди системі або отримує конфіденційну інформацію без дозволу користувача. Використовуючи методи інтелектуального аналізу даних, наша мета - автоматично розробити та створити сканер, який точно виявляє зловмисні виконувані файли ще до того, як вони отримають шанс запуститися.

Методи інтелектуального аналізу даних виявляють шаблони в великих обсягах даних, таких як байт-код, і використовують ці шаблони для виявлення майбутніх випадків у подібних даних. Однією з основних проблем, з якими стикається спільнота, що займається вірусами, є розробка методів для виявлення нових зловмисних програм, які ще не були проаналізовані [1]. Щодня створюється від восьми до десяти зловмисних програм, і більшість з них не може бути точно виявлена, доки для них не будуть створені сигнатури [2]. У цей період системи, захищені алгоритмами на основі сигнатур, є вразливими для атак. Зловмисні виконувані файли також використовуються для багатьох видів вторгнень.

Поточна технологія сканування вірусів має дві частини: детектор на основі сигнатур та евристичний класифікатор, який виявляє нові віруси [3]. Класичний алгоритм виявлення на основі сигнатур покладається на сигнатури (унікальні рядки-індикатори) відомих зловмисних виконуваних файлів для створення моделей виявлення. Методи на основі сигнатур створюють унікальну позначку для кожної зловмисної програми, щоб майбутні випадки цієї програми могли бути правильно класифіковані з низьким рівнем помилок. Ці методи погано узагальнюються для виявлення нових зловмисних бінарних файлів, оскільки вони створені для того, щоб дати якомога нижчий показник хибнопозитивних спрацювань. Цей вид аналізу може бути трудомістким і часто все ж не здатний виявити нові зловмисні виконувані файли.

Мета: розробити структуру, яка використовує алгоритми інтелектуального аналізу даних для навчання декількох класифікаторів на наборі зловмисних і нешкідливих виконуваних файлів для виявлення нових прикладів.

Основні принципи побудови базової моделі виявлення шкідливих файлів

З точки зору архітектури базова версія моделі являє собою об'єднання кількох варіаційних автокодерів, що ділять спільний простір прихованих змінних. При цьому вона, хоча й не дозволяє правдоподібно генерувати такі складні об'єкти, як виконувані файли або їх логи, але дає можливість оцінювати ймовірність нових отриманих даних і ймовірність наявності зв'язку між артефактами різної природи.

В основі розробленої моделі будуть використовуватися відображення лише для виконуваних PE-файлів і лога поведінки, отриманого при запуску на віртуальній машині.

Однак представлені ідеї можуть бути використані і при роботі з іншими джерелами інформації: наприклад, емуляційними логами, наборами різнорідної метаінформації або логами роботи на комп'ютері кінцевого користувача продукту. Вважається, що лог поведінки являє собою упорядковану в часі послідовність подій разом із пов'язаними з цими подіями аргументами. Наприклад, з подією, що описує відкриття файлу, пов'язані шлях до файлу і права доступу. А з подією, що відповідає за створення мережевого з'єднання, пов'язані IP-адреса віддаленої машини, номер мережевого порту, а також, можливо, протокол з'єднання і/або обсяг переданих даних. Додатково допускається зберігання унікальних ідентифікаторів процесів і потоків, що здійснюють зазначені дії, а також час початку кожної з подій.

Отримання логів здійснюється наступним чином:

а) виконуваний файл стартує з фіксованої директорії всередині віртуальної машини. Образ віртуальної машини при цьому містить різні попередньо встановлені програми, що представляють потенційний інтерес для зловмисника з точки зору пошуку вразливостей, а також набір користувацьких файлів, які також можуть бути цікавими розповсюджувачам шкідливого програмного забезпечення (наприклад, таблиці з фінансовими звітами або файли з паролями та персональними даними);

б) аналізована програма стартує без будь-яких параметрів. Якщо файл не має точки старту (Entry Point), що, наприклад, вірно для більшості *.DLL файлів, проводиться послідовний запуск функцій, наданих файлом на основі таблиці експорту;

в) з допомогою спеціального компонента моніторингу проводиться логування всіх дій, що здійснюються виконуваним файлом і запущеними ним сторонніми процесами;

г) логування здійснюється, поки не настане одне з наступних подій: всі процеси, породжені файлом, успішно завершилися або були перервані через помилку, причому робота файлу та дочірніх процесів займає більше установленого часу;

д) результати логування зберігаються у зовнішньому сховищі та надсилаються на динамічний аналіз.

Щоб краще обгрунтувати потребу в застосуванні ймовірнісних просторів для представлення файлу та його лога, нижче описані основні причини, що породжують неоднозначність при побудові представлень. Для виконуваного файлу основними причинами варіативності є:

- умисний поліморфізм файлів: часто зловмисники при розповсюдженні шкідливого ПО реалізують схему, яка надає кожному користувачу унікальну версію виконуваного файлу;
- різні збірки вихідного коду: програмне забезпечення, що поширюється у вигляді відкритого вихідного коду, може бути завантажене та зібране користувачами, які використовують різні версії компіляторів і архітектури процесора;
- упаковані файли: досить часто при збірці програми автори як чистого, так і шкідливого програмного забезпечення використовують програми-пакери. Пакери стискають і упаковують у спеціальний архів вихідний код програми, в результаті чого підсумковий PE-файл має тривіальну структуру, що містить єдину секцію, заповнену байтами з високою ентропією;
- інфіковані файли: одним з найбільш відомих типів шкідливих програм є

комп'ютерні віруси - програми, що розповсюджуються шляхом впровадження фрагментів власного коду в тіло інших, вже існуючих на комп'ютері користувача програм. Як наслідок, абсолютно різні за вихідною структурою програми після запуску інфікованого сегмента коду починають вести себе однаково, виконуючи дії, закладені зловмисником.

Ключовими причинами варіативності змісту поведінкового лога є:

- використання випадковості: найбільш очевидна причина різниці в поведінці однієї й тієї ж програми - умисна непередбачуваність її поведінки. Наприклад, програма може в випадковому порядку аналізувати файли в файловій системі. Або ж проявляти шкідливу активність лише з певною ймовірністю, щоб мінімізувати ризик бути виявленою під час автоматичного аналізу вендорами захисного програмного забезпечення;
- залежність від зовнішнього оточення: навіть якщо програма не використовує випадковість явно, вона може по-різному себе вести в залежності від факторів за межами системи, на якій виконується код. Наприклад, програма може проявляти свою активність при появі певних ключових слів у новинній стрічці або просто звертатися до різних IP-адрес у мережі в залежності від поточної дати та часу доби.

Багатопоточність: більшість сучасних програм працює в багатопотоковому режимі. В результаті порядок дій, що здійснюються в паралельно працюючих потоках або процесах, може бути випадковим і залежати від дій планувальника ресурсів операційної системи. Більш того, при певному порядку роботи процесів може різнитися результат роботи програми: наприклад, клієнт bot-net мережі може асинхронно опитувати доступність різних адрес командного центру та по-різному діяти в залежності від того, яка з адрес відповість раніше.

Висновок. Розробено структуру, яка використовує алгоритми інтелектуального аналізу даних для навчання декількох класифікаторів на наборі зловмисних і нешкідливих виконуваних файлів з встановленням основних принципів побудови базової моделі виявлення шкідливих файлів

Перелік використаних джерел.

1. Abdessadki, I., & Lazaar, S. (2019). A new classification based model for malicious pe files detection. International Journal of Computer Network and Information Security, 11(6), 1-9.
2. Abri, F., Siami-Namini, S., Khanghah, M.A., Soltani, F.M. et al. (2019). The performance of machine and deep learning classifiers in detecting zero-day vulnerabilities. arXiv:1911.09586. Вісник Черкаського державного технологічного університету 3/2023 Bulletin of Cherkasy State Technological University A. E. Nafiiev, D. V. Lande, 2023 DOI: 10.24025/2306-4412.3.2023.286374 49
3. Alazab, M., Venkatraman, S., Watters, P., & Alazab M. (2011). Zero-day malware detection based on supervised learning algorithms of api call signatures. In Australasian Data Mining Conference (pp. 171-182).