УДК 681.3 http://doi.org/10.15407/pp2023.02.084

А.Е. Нафієв, А.М. Родіонов

СИСТЕМА ДИНАМІЧНОГО АНАЛІЗУ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ ІНТРОСПЕКЦІЇ ВІРТУАЛЬНИХ МАШИН ТА МЕТОДІВ МАШИННОГО НАВЧАННЯ

Автори шкідливого програмного забезпечення створюють все більш досконалі та витончені зловмисні програми, які майже неможливо виявити за допомогою статичного аналізу. Навіть у разі використання динамічного аналізу шкідливий файл може розпізнати, що його запускають у віртуальному середовищі, і змінити свій код. Тому метою цього дослідження є створення системи динамічного аналізу, в якій виконуваний файл не може розпізнати спостереження, за рахунок чого файл покаже всю свою природу. Для побудови такої системи використовується система Drakvuf, за допомогою якої ми отримуємо повний знімок виконаних дій .exe-файлу в операційній системі. Далі отримані дані обробляються запропонованими в цій статті методами і подаються алгоритму машинного навчання.

Ключові слова: динамічний аналіз, інтроспекція віртуальної машини, гіпервізор, виявлення шкідливого програмного забезпечення

Вступ

В історії людства війни є невід'ємною частиною природи хомо сапієнс. І як показує сучасна ситуація у світі, незважаючи на стрімкий технологічний прорив XXI століття, війна все ще притаманна сучасній людині. Однак із зростанням глобальної цифровізації світу тип війни також змінюється. Сьогодні ми можемо спостерігати, що чи не найважливішу роль відіграють кібератаки. Оскільки навіть один шкідливий файл може завдати шкоди критичній інфраструктурі. Тому це дослідження спрямоване на боротьбу зі шкідливим ПЗ, а саме на труднощі, пов'язані з його виявленням. Аналіз шкідливого програмного забезпечення можна виконувати як статичними, так і динамічними методами.

Статичний аналіз виконується шляхом аналізу коду та структури програми для визначення її функцій. Суть таких методів полягає в перевірці вмісту аналізованого об'єкта на наявність сигнатур уже відомих загроз. Однак використання тільки статичного аналізу для створення сигнатур стає все більш безперспективним, оскільки щороку з'являється величезна кількість нових зразків шкідливих програм, які можуть мутувати і змінюватися в процесі своєї роботи.

У процесі динамічного аналізу програма аналізується шляхом її запуску в реальному обчислювальному середовищі. Такий підхід дозволяє спостерігати за виконанням зразків шкідливого програмного забезпечення в ізольованому режимі і збирати поведінкові характеристики, які потім можуть бути використані для класифікації файлу. Тож динамічний аналіз є ефективнішим способом визначення функціональності досліджуваної програми. Однак успішність виявлення шкідливого файлу залежить від типу програмного агента, відповідального за збір даних під час динамічного аналізу. Тому автори шкідливого програмного забезпечення оснащують свої програми можливістю виявляти агента і обходити такі системи. В результаті шкідливий файл може розпізнати, що він працює в ізольованій системі, і затримати свій запуск, або змінити свою поведінку, щоб ввести в оману систему аналізу. Тому для систем динамічного аналізу шкідливого програмного забезпечення дуже важливо забезпечити реальне середовище виконання файлу з можливістю прихованого моні-

© А.Е. Нафієв, А.М. Родіонов, 2023

торингу, щоб приховати наявність патерну збору даних виконуваного файлу.

В цій роботі ми використовували систему аналізу шкідливого програмного забезпечення Drakvuf Sandbox [1], яка дозволяє відстежувати шкідливе програмне забезпечення на рівні користувача та ядра операційної системи без необхідності встановлення агента в гостьовій ОС. Система побудована на платформі віртуалізації Xen [2], використовує LibVMI API [3] та ядро DRAKVUF.

Більш детально описане в цій статті дослідження зосереджується на наступних пунктах:

1.Реалізувати процес збору та обробки поведінкових характеристик для набору різнотипних exe-файлів за допомогою системи Drakvuf.

2.На основі зібраних даних побудувати моделі машинного навчання для детекції шкідливих файлів та провести експерименти.

Система динамічного аналізу

Науковці досліджують різні способи динамічного аналізу для усунення недоліків статичного аналізу [4-9]. У дослідженні Jain та Nair [10] для отримання API-викликів було використано емульоване середовище. Для аналізу вони використовували QEMU. Зразки класифікували за допомогою методу розпізнавання образів. Автори роботи [11] запропонували суто динамічний підхід для класифікації доброякісних та шкідливих зразків. Трассировки API викликів виконуваних файлів проаналізовані шляхом їх виконання на віртуальній машині. Як інструменти трасування використовуються HookMe та Microsoft Detours.

Важливою відмінністю нашої роботи є використання методу динамічного аналізу з використанням інтроспекції віртуальної машини. Інтроспекція віртуальної машини (VMI) – це метод зовнішнього доступу до стану віртуальної машини в режимі паравіртуалізації (PV). Паравіртуалізація – це адаптація ядра виконуваної ОС до роботи спільно з гіпервізором. Це дозволяє досягти дуже високої ефективності за рахунок відсутності емуляції «реального заліза», простоти інтерфейсів і врахування наявності гіпервізора при виконанні системних викликів в коді ядра.

Гостьова операційна система взаємодіє з гіпервізором, який працює безпосередньо на апаратному рівні в найвищому кільці привілегій ядра процесора і керує всіма віртуальними машинами [12]. Така система дозволяє збирати дані про роботу навіть найскладнішого метаморфного шкідливого програмного забезпечення та відстежувати виконання процесів, файлові операції, системні виклики та трасування функцій ядра без необхідності встановлення агента в гостьовій системі.

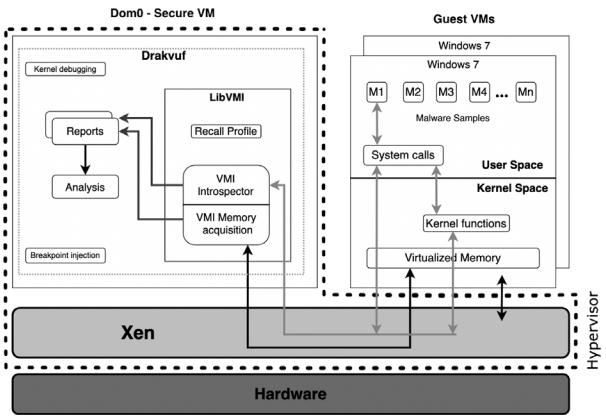

Рис. 1. Схема системи динамічного аналізу

Формування набору даних

Формування набору даних – це один із найважливіших етапів створення моделі машинного навчання. Для цього дослідження було зібрано набір даних, що включає 340 exe-файлів, з яких 170 є доброякісними, а 170 – шкідливими. Співвідношення шкідливих і доброякісних файлів 50/50 було обрано для більш репрезентативних результатів точності моделі. Представлено 7 типів шкідливих файлів: InstallCore (9), CryptoRansom (13), TheZoo (13), Zeus (15), Zbot (40), Zeroaccess (40), Winwebsec (40). Інфографіку розподілу можна побачити на Рис. 2. Шкідливі файли були взяті з різних сайтів: «virusshare.com», «malicia-project.com», «thezoo.morirt.com». Нешкідливі файли були взяті з папок встановлених додатків легального програмного забезпечення різних категорій. Також файли бралися з сайту «exefiles.com».

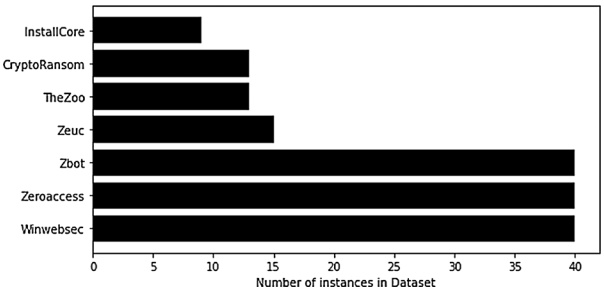

Рис. 2. Інфографіка розподілу типів файлів

Використовуючи Drakvuf sandbox, кожен exe-файл був запущений на віртуальній машині Windows 7. Після 5 хвилин роботи віртуальна машина знищується, і ми отримуємо .txt файл з результатом аналізу. На рис. 3 показано невеликий фрагмент такого файлу.

{"Plugin": "filetracer", "TimeStamp": "1661526556.283727", "PID": 1348, "PPID": 1928, "TID": 2800, "UserName": "SessionID", "UserId": 0, "ProcessName": "\\Device\\HarddiskVolume2\\Windows\\System32\\SearchProtocolHost.exe", "Method": "NtCreateFile", "EventUID": "0x5f3c5", "FileName": "\\??\\C:\\Program Files\\Microsoft Games\\Minesweeper\\minesweeper.exe", "FileHandle": "0x7c4", "ObjectAttributes": "OBJ_CASE_INSENSITIVE", "DesiredAccess": "FILE_READ_ATTRIBUTES | SYNCHRONIZE | GENERIC_READ", "FileAttributes": "(null)", "ShareAccess": "FILE_SHARE_READ", "CreateDisposition": "FILE_OPEN", "CreateOptions": "FILE_SYNCHRONOUS_IO_NONALERT | FILE_NON_DIRECTORY_FILE", "Status": "SUCCESS"}
{"Plugin": "filetracer", "TimeStamp": "1661526556.285202", "PID": 880, "PPID": 436, "TID": 2456, "UserName": "SessionID", "UserId": 0, "ProcessName": "\\Device\\HarddiskVolume2\\Windows\\System32\\svchost.exe", "Method": "NtReadFile", "EventUID": "0x5f3c6", "FileName": "\\Windows\\System32\\wbem\\Repository\\OBJECTS.DATA", "FileHandle": "0x8d0"}
{"Plugin": "filetracer", "TimeStamp": "1661526556.285994", "PID": 880, "PPID": 436, "TID": 2456, "UserName": "SessionID", "UserId": 0, "ProcessName": "\\Device\\HarddiskVolume2\\Windows\\System32\\svchost.exe", "Method": "NtReadFile", "EventUID": "0x5f3c8", "FileName": "\\Windows\\System32\\wbem\\Repository\\OBJECTS.DATA", "FileHandle": "0x8d0"}
{"Plugin": "filetracer", "TimeStamp": "1661526556.286624", "PID": 880, "PPID": 436, "TID": 2456, "UserName": "SessionID", "UserId": 0, "ProcessName": "\\Device\\HarddiskVolume2\\Windows\\System32\\svchost.exe", "Method": "NtReadFile", "EventUID": "0x5f3ca". "FileName": "\\Windows\\System32\\wbem\\Repository\\OBJECTS.DATA". "FileHandle": "0x8d0"}

Рис. 3. Фрагмент .txt файлу з результатом аналізу

Кожен рядок відповідає одній дії, яка відбувається в операційній системі під час виконання exe-файлу. Ця дія представлена у форматі типу даних dictionary у мові програмування python. Існує набір параметрів та їхніх значень. Нижче наведено приклад такої дії:

{«Plugin»: «filetracer»,
«TimeStamp»: «1657826673.189062»,
«PID»: 560,
«PPID»: 436,
«TID»: 572,
«UserName»: «SessionID»,
«UserId»: 0,
«ProcessName»: «\\Device\\
HarddiskVolume2\\Windows\\System32\\
svchost.exe»,
«Method»: «NtOpenFile»,
«EventUID»: «0x8c8»,
«FileName»: «\\??\\C:\\Windows\\
System32\\DriverStore\\en-US\\»,
«ObjectAttributes»: «OBJ_CASE_
INSENSITIVE»}

PID (ідентифікатор процесу) – номер для унікальної ідентифікації активного процесу.

ProcessName – ім'я, яке система використовує для ідентифікації процесу для користувача.

Method – містить функції з бібліотеки Windows ntdll.lib.

FileName – ім'я файлу, який використовується системою у певний момент часу.

В підсумку було зібрано 340 txt-файлів з результатами динамічного аналізу для кожного exe-файлу.

Виділення ознак

Тепер, коли ми маємо результати динамічного аналізу, постає фундаментальне завдання збору та обробки даних. На основі цих даних сформується фінальна матриця, яка буде надіслана алгоритму машинного навчання, а саме машині опорних векторів (SVM). Для аналізу було обрано три параметри (Method, FileName, ProcessName), які містять значущу інформацію про виконувані в операційній системі дії. Для формування ознак, на яких навчається SVM, використовується метод N-грам, де для побудови ланцюга Маркова замість 2-грам як базового представлення даних використовуються ймовірності переходів. У наступних підрозділах буде описано процес формування фінальної матриці на основі кожного з трьох параметрів.

Параметр «Method»

Першим кроком у створенні ознак є парсинг даних з усіх txt-файлів. Ми отримали 340 масивів наступного вигляду:

[NtOpenFile, NtCreateFile, …, NtWriteFile, NtQueryAttributesFile]

Виявилося, що в усіх txt-файлах параметр Method приймає лише 7 унікальних значень: NtReadFile, 0NtWriteFile, NtCreateFile, NtOpenFile, NtSetInformationFile, NtQueryAttributesFile, NtOpenDirectoryObject. Покажемо принцип формування фінальної матриці на прикладі послідовності значень з прикладу вищевказаного масиву. На основі такого масиву будується двовимірна квадратична матриця суміжності розміром 7 * 7, в якій

для кожної пари значень в матриці підраховується, скільки разів перше значення безпосередньо слідувало за другим:

	NtReadFile	NtWriteFile	NtCreateFile	NtOpenFile	NtSetInf...	NtQuer...	NtOpenDir...
NtReadFile	192033	4669	959	1535	63	406	406
NtWriteFile	4731	6549	41	81	49	15	15
NtCreateFile	1122	46	2319	3117	58	586	589
NtOpenFile	1401	143	3189	1070	30	293	293
NtSetInformationFile	70	52	40	26	1873	13	13
NtQueryAttributesFile	311	8	700	297	5	329	329
NtOpenDirectoryObject	5	1	16	11	0	11	11

Рис. 4. Матриця суміжності

Наш ланцюг Маркова можна представити у вигляді графа, в якому вершинами є стани процесу (всі можливі значення параметра Method), а ребрами – переходи між станами, причому на ребрі від i до j записується p_{ij} – ймовірність переходу від одного значення до іншого. В результаті отримуємо квадратичну матрицю переходів $P = \| p_{ij} \|$, на яку накладаються наступні умови:

$$p_{ij} \geq 0,$$

$$\forall i \sum_{j} p_{ij} = 1$$

Наша матриця переходів має наступний вигляд:

	NtReadFile	NtWriteFile	NtCreateFile	NtOpenFile	NtSetInf...	NtQuer...	NtOpenDir...
NtReadFile	0.961737	0.023383	0.004803	0.007688	0.000316	0.002033	0.000040
NtWriteFile	0.412439	0.571067	0.003575	0.007063	0.004273	0.001308	0.000174
NtCreateFile	0.154460	0.006333	0.319246	0.429102	0.007985	0.080672	0.002203
NtOpenFile	0.228250	0.023297	0.519550	0.174324	0.004888	0.047735	0.001792
NtSetInformationFile	0.033686	0.025024	0.019249	0.012512	0.901347	0.006256	0.001925
NtQueryAttributesFile	0.188143	0.004840	0.423472	0.179673	0.003025	0.199032	0.001815
NtOpenDirectoryObject	0.78125	0.015625	0.250000	0.171875	0.000000	0.171875	0.312500

Рис. 5. Матриця переходів

Далі ці матриці трансформуються у векторну форму і комплектуються у фінальну матрицю. Схему формування фінальної матриці можна побачити на рис. 6.

Таким чином, було отримано фінальну матрицю, сформовану на основі параметра Method.

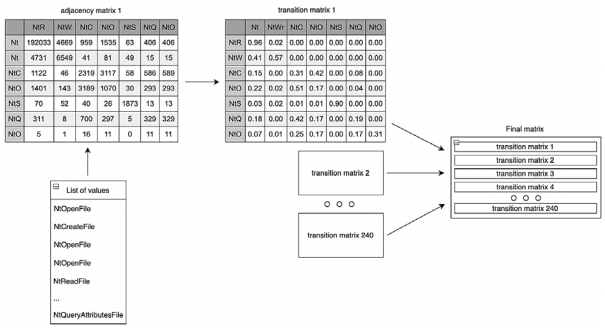

Рис. 6. Схема формування фінальної матриці

Параметр «FileName»

Після парсингу всіх .txt файлів виявилося, що існує 25440 унікальних значень параметра FileName. Можна було б побудувати фінальну матрицю так само, як і у випадку з параметром Method. Однак у такому випадку матриця суміжності буде дуже великою (25440 x 25440), а розмірність фінальної матриці буде (340 x 647 мільйонів), що занадто багато для потужностей сучасних комп'ютерів. Крім того, більшість комірок у такій матриці будуть нулями. Тому виникає задача вибору меншої кількості найбільш значущих ознак. Для формування набору таких ознак використовується метод, де беруться значення, які в сумі зустрічаються найбільше разів у всіх .txt файлах. Тож, було сформовано 3 фінальні матриці різної розмірності на основі 3 наборів значень параметра FileName. Перший набір містить 88 найпоширеніших значень. Другий – 326 і третій – 712.

Параметр «ProcessName»

Масив унікальних значень параметра ProcessName містить 312 елементів. Однак більшість з цих елементів – це значення, що містять ім'я .exe-файлу, який був запущений. Тобто таке значення зустрічається лише в одному .txt файлі з 340, а це означає, що воно не має особливої цінності у процесі навчання моделі машинного навчання. Тому із набору кількістю в 312 унікальних значень були видалені ті, що містять власні імена файлів. Отриманий масив налічував 65 елементів. На основі цього масиву для кожного .exe-файлу були побудовані матриці суміжності з розмірами (65 x 65) і одна фінальна матриця.

Всі три параметри

Після формування всіх фінальних матриць виникла ідея зробити модель, яка б враховувала інформацію не лише від одного параметра, а від усіх трьох одразу. Кожному .exe-файлу відповідає вектор числових значень, який описує природу файлу. З таких векторів і складається фінальна матриця. Щоб об'єднати три век-

тори з трьох різних матриць, ми можемо просумувати ці вектори разом. Тоді буде отримана одна, найбільш «повна» з точки зору інформації, підсумкова матриця. Для формування такої матриці були взяті наступні фінальні матриці: матриця на основі параметра Method з 7 ознаками, матриця на основі параметра ProcessName з 47 ознаками та матриця на основі параметра FileName з 88 ознаками. Як бачимо, розмірність трьох матриць різна, і для того, щоб підсумувати вектори, нам потрібно досягти однакової розмірності. Для цього було використано метод головних компонент, за допомогою якого матриці ProcessName та FileName було зведено до розмірності матриці Method.

Навчання та метрики

Наші моделі машинного навчання навчаються за допомогою алгоритму SVM з квадратичним експоненціальним ядром. Гіперпараметри алгоритму підбираються за допомогою крос-валідації. В експерименті навчальна вибірка налічувала 240 файлів, 120 безпечних і 120 шкідливих. Тестовий

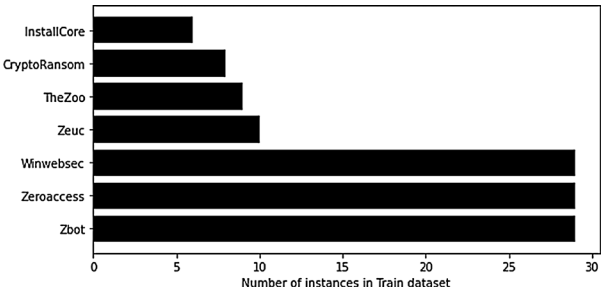

Рис. 7. Інфографіка розподілу типів файлів у навчальному наборі даних

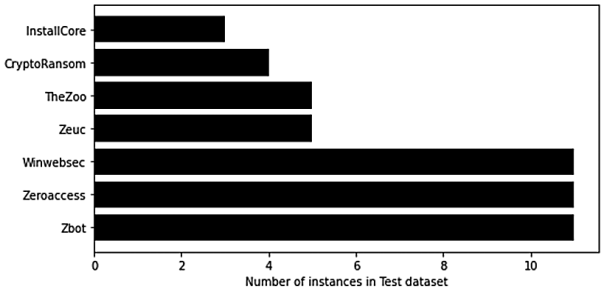

Рис. 8. Інфографіка розподілу типів файлів у тестовому наборі даних

набір включав 100 файлів, 50 безпечних і 50 шкідливих. Інфографіку наборів даних можна побачити на рис. 7 та рис. 8.

Тестові фінальні матриці були сформовані на основі ознак, отриманих із навчальної вибірки.

За оцінку точності досліджуваних моделей використовується F-score, який є спільною оцінкою таких метрик, як precision та recall. Також для аналізу моделей використовувалися графіки roc_auc та pr_auc кривих.

Результати

У таблиці 1 наведено результати точності всіх моделей. Усі моделі продемонстрували досить високі показники точності за всіма метриками. Найвище значення F-score має модель на основі параметра FileName з 376 ознаками. Найменше значеннння має модель FileName з 726 ознаками, але показники кривих roc_auc та pr_auc у цієї моделі знаходяться на дуже високому рівні. Варто зазначити, що модель на основі всіх трьох параметрів показала найвищі показники roc auc та precision-recall auc, а

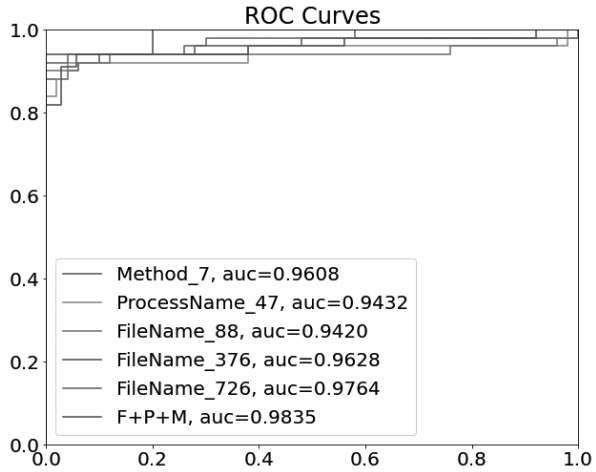

Рис. 9. ROC-крива

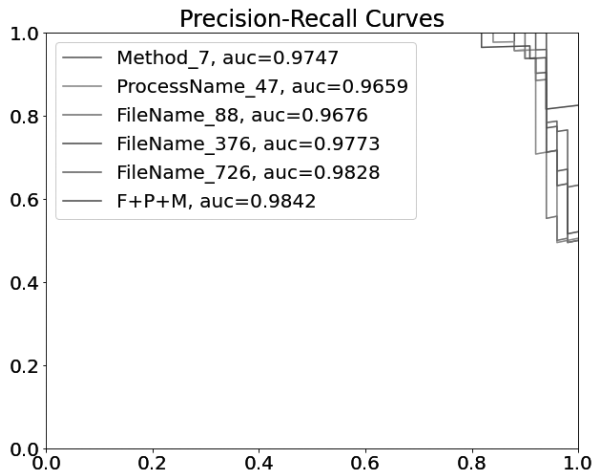

Рис. 10. Precision-Recall крива

Таблиця 1

Результати точності моделей машинного навчання

	F_score		Precision		Recall		pr_auc	roc_auc
	0	1	0	1	0	1		
Method_7	0.9230	0.9166	0.8888	0.9565	0.9600	0.8800	0.9747	0.0906
ProcessName_47	0.9411	0.9387	0.9230	0.9583	0.9600	0.9200	0.9659	0.9432
FileName_88	0.9320	0.9278	0.9056	0.9574	0.9600	0.9000	0.9676	0.9420
FileName_376	0.9607	0.9591	0.9423	0.9791	0.9800	0.9400	0.9773	0.9628
FileName_726	0.9183	0.9215	0.9375	0.9038	0.9000	0.9400	0.9828	0.9764
F88+P47+M7	0.9295	0.9230	0.9166	0.9375	0.9428	0.9090	0.9842	0.9835

за метрикою F-score посіла друге місце в рейтингу. На рис. 9 та рис. 10 можна побачити криві ROC та Precision-Recall.

Обговорення

У цьому дослідженні ми створили 6 моделей машинного навчання: 3 моделі на основі параметра FileName, модель на основі параметра Method, модель на основі параметра ProcessName і модель, яка враховує всі три параметри одночасно. З усіх трьох параметрів оптимальним з точки зору здатності виявляти шкідливі файли виявився параметр FileName. А саме модель, що містить середню кількість ознак – 376. Вона показала найвищий показник F-score і майже найкращі значення кривих roc та precision-recall. Виходячи з цього, можна стверджувати, що не завжди модель з великою кількістю ознак покаже найкращий результат. Необхідно знайти оптимальний набір ознак, оскільки етап їх формування є чи не найважливішим етапом у процесі виявлення шкідливих файлів. Варто також відзначити високу ефективність моделі на основі трьох параметрів. Ідея сумування векторів трьох суміжних матриць виправдала свої очікування, оскільки показники кривих roc і precision-recall виявилися найвищими серед усіх моделей. Варто також відзначити, що, незважаючи на те, що модель на основі параметра Method має лише 7 ознак, результати точності виявилися на достатньому рівні серед інших моделей із більшою в рази кількістю ознак.

Висновки

У цій статті було проаналізовано набір exe-файлів за допомогою системи Drakvuf. Після аналізу були отримані txt-файли, що містять знімок дій, які виконував exe-файл в операційній системі. На основі цієї інформації для різних трьох параметрів було описано процес перетворення отриманих текстових даних у числові значення, на яких було проведено навчання алгоритму машинного навчання. Також була сформована додаткова модель, яка враховує всі три параметри. Отримані моделі показали досить хороші результати, що свідчить про можливість використання запропонованого алгоритму генерації моделей на реальній системі класифікації файлів.

Література

1. Tamas K Lengyel, Steve Maresca, Bryan D Payne, George D Webster, Sebastian Vogl, and Aggelos Kiayias. Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system. In The 30th Annual Computer Security Applications Conference, pages 386–395, 2014
2. Xen Project. Available at: xenproject.org
3. LibVMI. Available at: libvmi.com
4. Muhammad Ijaz, Muhammad Hanif Durad, Maliha Ismail, “Static and Dynamic Malware Analysis Using Machine Learning”, 2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST – 2019), January 2019
5. Matthew Nunes, Pete Burnap, Omer F. Rana, “Getting to the root of the problem: A detailed comparison of kernel and user level data for dynamic malware analysis”, Journal of Information Security and Applications, October 2019
6. Sudhir Kumar Rai, Ashish Mittal, Sparsh Mittal, “A Node-Embedding Features Based

Machine Learning Technique for Dynamic Malware Detection", IEEE Conference on Dependable and Secure Computing (DSC), April 2022
7. Hongwei Zhao, Mingzhao Li, Taiqi Wu, Fei Yang, "Evaluation of Supervised Machine Learning Techniques for Dynamic Malware Detection", International Journal of Computational Intelligence Systems, July 2018
8. David Escudero García, Noemí DeCastro-García, "Optimal Feature Configuration for Dynamic Malware Detection", Computers & Security, February 2021
9. Charles-Henry Bertrand Van Ouytsel, Axel Legay, "Malware Analysis with Symbolic Execution and Graph Kernel", April 2022
10. V.P.Nair et al.,"MEDUSA: Metamorphic Malware Dynamic analysis Using Signature from API", in 5th Int. Conf. on malicious and unwanted software, ACM, 2010
11. Ronghua Tian et al., "Differentiating Malware from Cleanware Using Behavioural Analysis", In Proc. of the 3rd Int. Conf. on Security of Inform. and Networks, SIN'10, IEEE, March 2010
12. Alfred Melvin G. Jaspher W. Kathrine, "A Quest for Best: A Detailed Comparison Between Drakvuf-VMI-Based and Cuckoo Sandbox-Based Technique for Dynamic Malware Analysis", Intelligence in Big Data Technologies—Beyond the Hype, January 2021

Одержано: 7.05.2023

Про авторів:

Алан Емінович Нафієв,
аспірант.
Кількість публікацій в українських виданнях – 1.
Кількість зарубіжнихих публікацій – 1.
https://orcid.org/0009-0004-8604-377X.

Андрій Миколайович Родіонов,
кандидат технічних наук, доцент.
Кількість публікацій в українських виданнях – 10.
Кількість зарубіжних публікацій – 0.
http://orcid.org/ 0000-0001-7284-9458.

Місце роботи авторів:

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»,
фізико-технічний інститут,
03056, м. Київ, Проспект Перемоги, 37.
Телефон: +38 (044) 236 70 98.
E-mail: Alan.nafiev@gmail.com,
andrey.rodionov@gmail.com